# أساسيات كيوت

إعداد و تحضير

م / أحمد البنا

المراجعة اللغوية

ا / محمد الدسوقى

# بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

**اقْرَأْ بِاسْمِ رَبِّكَ الَّذِي خَلَقَ (١) خَلَقَ الْإِنْسَانَ مِنْ عَلَقٍ (٢) اقْرَأْ وَرَبُّكَ الْأَكْرَمُ (٣) الَّذِي عَلَّمَ بِالْقَلَمِ (٤) عَلَّمَ الْإِنْسَانَ مَا لَمْ يَعْلَمْ (٥)**

سورة العلق

# مقدمـــــــــــــة

**كيوت** هى بنية تطوير عبر النظم ( Cross-Platform ) لصناعة وتطوير البرامج ذات الواجهة الرسومية، وهي أداة متعددة النظم يمكن تشغيلها على أكثر من نظام تشغيل مثل ويندوز( windows )، وماك ( Mac OSX )، ولينوكس ( Linux )، وبعض أنظمه الأجهزة المحموله مثل أنظمة أجهزة نوكيا سيمبيان ( Symbian )، ويندوز موبايل (Windows Mobile)، وايمبيدد لينوكس ( Embedded Linux ).

وبشكل أساسى تعتمد **كيوت** فى برمجتها علي لغة سي++ ، حيث أن جميع فصائل **كيوت** مكتوبة بالسي ++ ،وتأتى ميزة تعدد النظم كأداة تيسر على المبرمج أو المطور تشغيل التطبيق على أكثر من نظام، وهذا ما يعطيه فرصة جيدة فى تقليل تكلفة التشغيل وزيادة قاعدة التسويق للتطبيق، حيث يتم كتابة الكود مرة واحدة ثم ترجمته ( Compiling ) على أى نظام تشغيل بدون الحاجة إلى تعديل، وكلما تبحر المبرمج داخل **كيوت** سوف يكتشف مدى قابلية هذه الأداة للتطوير والهيكلة على حسب ما يريده .

## ما الهدف من كيوت ؟

الهدف الرئيسى هو إمكانية بناء تطبيق عابر للنظم بكتابة الكود مرة واحدة وتشغيله على أى نظام.

## فكرة و محتوى هذا الكتاب :

هذا الكتاب يتم التعامل معه على أساس أنه أداة تعليميه ذات منهج متسلسل يسهل على الدارس التعامل معه والتعلم منه، ويجب ملاحظة أن هذا الكتاب لا يعتبر مرجع شامل أو كامل **لكيوت**، لأن أداة المساعدة من **كيوت** هى بالفعل مرجع كامل وشامل، ولكن كيف تبدأ وما هى المبادئ الأساسية التى يجب أن تعرفها حتى يمكنك تعلم **كيوت** وفهمها فهماً جيداً فى أسرع وقت ممكن، هذا ما سوف نتناوله بالدراسة فى هذا الكتاب.

## لماذا هذا الكتاب؟

نظراً لإستحواذ شركة مايكروسوفت وهيمنة تطبيقاتها على معظم السوق العربية،
ونظراً لرؤيتنا أن **كيوت** هى إحدى وسائل الخلاص من هذه الهيمنة،
ونظراً لعدم وجود كتاب أو شرح وافى لهذه الأداة باللغة العربية،
فقد قمنا بإعداد هذا الكتاب والذى يعتبر بمثابة البوابة الرئيسية للدخول إلى عالم **كيوت**، والذى يتناول بداخله التعريف بالمبادىء الأساسية التى قامت عليها **كيوت**، كما سيتم أيضا شرح جميع إمكانيات **كيوت** شرحاً يسيراً مع أمثلة بسيطة .

## ماذا بعد الكتاب؟

عند الإنتهاء من هذا الكتاب سوف تكون قادراً على بدأ البرمجة المحترفة بإستخدام **كيوت**، وسيتكون لديك الإحساس المطلوب كى تستطيع التعامل بسلاسة مع الجديد من إصدارات **كيوت** أو مع المشاكل التى قد تواجهك.

تذكر أن هناك أكثر من طريق للوصول إلى المطلوب بالبرمجة، ولكن هناك دائماً الطريق الأفضل أو الأمثل، وهذا ما سنراه كثيرا داخل **كيوت**.

## ما يتطلب للدخول إلى عالم **كيوت** :

معرفه جيدة بـ C++ Object Oriented Programming
الحصول على Qt Nokia SDK Software ( مرفق مع هذا الكتاب )

..............................................................................

# فهرس الكتاب


**مقدمة** ........ ٢

**بنية وتكوين كيوت** ........ ٨

**وحدة الفصائل الأساسية غير الرسومية ( QtCore Module )** ........ ٩

فصيلة QDebug ........ ١٢

فصيلة QBitArray ........ ١٤

فصيلة QByteArray ........ ١٧

فصيلة QString ........ ٢٠

**الفصائل الحاوية ( Container Classes )** ........ ٢٣

فصيلة QList ........ ٢٤

فصيلة QVector ........ ٢٧

فصيلة QQueue ........ ٢٨

فصيلة QStack ........ ٣٠

فصيلة QStringList ........ ٣٢

فصيلة QSet ........ ٣٤

فصيلة QMap ........ ٣٦

فصيلة QMultiMap ........ ٣٩

**فصائل التكرار ( Iterator Classes )** ........ ٤١

Java Style Iterator ........ ٤٣

STL Style Iterator ........ ٤٤

foreach Keyword ........ ٥٠

فصيلة QDir ........ ٥٢

فصيلة QFileInfo ........ ٥٣

فصيلة QFile ........ ٥٦

فصيلة QTextStream ........ ٥٨

فصيلة QDataStream ........ ٦٠

فصيلة QVariant ........ ٦٢

**فصيلة QObject** ........ ٦٥

# فهرس الكتاب

**وحدة مكونات واجهة المستخدم الرسومية ( QtGui Module ) ........... ٨٣**
فصيلة QWidget ........ ٨٦
فصيلة QPainter ........ ١٠٣
فصيلة QDialog ........ ١٠٤
فصيلة QMainWindow ........ ١٠٨
**إدارة التخطيط ( Layout Managment ) ........ ١١٣**
**السحب و الإسقاط ( Drag and Drop ) ........ ١٢١**
**عرض البيانات - طريقة ( Model / View ) ........ ١٣١**
عرض البيانات كقائمة ........ ١٣٨
عرض البيانات داخل جدول ........ ١٤١
عرض البيانات على شكل شجرى ........ ١٤٤
**الرسم فى كيوت ( Graphics View Framework ) ........ ١٤٩**
فصيلة QGraphicsScene ........ ١٥٢
فصيلة QGraphicsView ........ ١٥٣
فصيلة QGraphicsItem ........ ١٥٤
**واجهة المستخدم المتحركة ( Animation Gui ) ........ ١٦١**
**وحدة فصائل برمجة الشبكات ( QNetwork Module ) ........ ١٦٩**
فصيلة QHostAddress ........ ١٧٢
فصيلة QHostInfo ........ ١٧٣
فصيلة QTcpServer ........ ١٧٧
فصيلة QTcpSocket ........ ١٨٧
فصيلة QUdpSocket ........ ١٩٥
**وحدة فصائل التعامل مع قواعد البيانات (QtSql Module ) ........ ٢٠٥**
**البرمجة الموازية ( Multithreaded Programming ) ........ ٢١٧**

..............................................

# فهرس الملحقات

**الملحقـــــات**

ملحق ( ١ ) إدارة الذاكرة ( Memory Managment ) .................................... ٢٢٨
ملحق ( ٢ ) هيكل بنية الفصائل فى ++C ........................................................ ٢٢٩
ملحق ( ٣ ) كيفية تنصيب كيوت ................................................................ ٢٣١
ملحق ( ٤ ) كيفية بدء تطبيق بواسطة الأداة QtCreator ................................ ٢٣٥
ملحق ( ٥ ) نماذج فارغة لفصائل QObject .................................................. ٢٣٦

...........................................................................

# فهرس الأمثلة


| Examples Name | Page | EX:No |
|---|---|---|
| QDebug Example | 13 | EX_1 |
| QBitArray Example | 15 | EX_2 |
| QByteArray Example | 18 | EX_3 |
| QString Example | 21 | EX_4 |
| QList Example | 25 | EX_5 |
| QVector Example | 27 | EX_6 |
| QQueue Example | 29 | EX_7 |
| QStack Example | 31 | EX_8 |
| QStringList Example | 33 | EX_9 |
| QSet Example | 35 | EX_10 |
| QMap Example | 37 | EX_11 |
| QMultiMap Example | 40 | EX_12 |
| Java_Iterator Example | 46 | EX_13 |
| STL_Iterator Example | 48 | EX_14 |
| foreach Example | 50 | EX_15 |
| QFileInfo Example | 54 | EX_16 |
| QFile Example | 57 | EX_17 |
| QTextStream Example | 58 | EX_18 |
| QDataStream Example | 60 | EX_19 |
| QVariant Example | 62 | EX_20 |
| QObject Example | 73 | EX_21 |
| QWidget Example | 87 | EX_22 |
| Car GUI Example | 94 | EX_23 |
| QPainter Example | 101 | EX_24 |
| QDialog Example | 105 | EX_25 |
| QMainWindow Example | 109 | EX_26 |
| Layout Example | 116 | EX_27 |
| Drag_Drop Example | 125 | EX_28 |
| ListView Example | 138 | EX_29 |
| TableView Example | 141 | EX_30 |
| TreeView Example | 144 | EX_31 |
| Graphics Example | 155 | EX_32 |
| Animation GUI Example | 164 | EX_33 |
| QHostInfo Example | 174 | EX_34 |
| QHostAddress Example | 174 | EX_34 |
| QTcpServer Example | 179 | EX_35 |
| QTcpSocket Example | 189 | EX_36 |
| QUdpSender Example | 196 | EX_37 |
| QUdpReceiver Example | 200 | EX_38 |
| Databse Example | 211 | EX_39 |
| QThread Example | 222 | EX_40 |

## بنية وتكوين كيوت

تتكون **كيوت** من مجموعه من الوحدات ( Modules )، وهذه الوحدات صممت للتعامل مع معظم أنواع التطبيقات، ويندرج تحت كل وحدة مجموعة كبيرة من الفصائل ( Classes ) لخدمة متطلبات هذه الوحدة.

ونوضح هذه الوحدات ( Modules ) فى الجدول التالى:

| Module | الوظيفـــــــــــــــــــــــة |
|---|---|
| **وحدات لتطوير التطبيقات العامة** | |
| QtCore | وحدة الفصائل الأساسية الغير رسومية. |
| QtGui | وحدة مكونات واجهة المستخدم الرسومية GUI . |
| QtMultimedia | وحدة الفصائل الخاصة بوظائف الوسائط المتعددة MultiMedia. |
| QtNetwork | وحدة فصائل برمجة الشبكات. |
| QtOpenGL | وحدة فصائل دعم OpenGL. |
| QtOpenVG | وحدة فصائل دعم OpenVG. |
| QtScript | وحدة فصائل تقييم وثائق **كيوت** Qt Scripts. |
| QtSql | وحدة الفصائل الخاصة بالتعامل مع قواعد البيانات DataBases. |
| QtSvg | وحدة فصائل لعرض محتوى ملفات ال SVG. |
| QtWebKit | وحدة فصائل لعرض وتحرير محتوى صفحات الويب Web content. |
| QtXml | وحدة فصائل للتعامل مع ملفات XML. |
| Phonon | وحدة فصائل للتعامل مع الوسائط المتعددة. |
| **وحدات خاصة بتطبيقات نظام ويندوز Windows OS** | |
| QAxContainer | وحدة للتحكم ب Windows ActiveX Control. |
| QAxServer | وحدة لكتابة خادم Server ل Windows ActiveX. |
| **وحدات خاصة بتطبيقات نظام يونكس UNIX OS** | |
| QtDBus | وحدة الفصائل الخاصة ب Inter-Process Communication. |

كيف نستخدم هذه الوحدات السابقة ؟ وما هى الوحدات التى يحتاجها المبرمج لتطبيقه؟

مثال : ما هى الوحدات المطلوبة لإنشاء تطبيق ذو واجهة رسومية و يتعامل مع قواعد البيانات و الشبكات؟

الوحدات المطلوبة : QtGui Module , QtSql Module , QtNetwork Module.

# QtCore Module

# وحدة

# الفصائل الأساسية

# غير الرسومية

## متطلبات هذه الوحدة

**إدراج**

```
#include <QtCore>
```

**داخل ملفات الكود**

**إدراج**

```
QT += core
```

**داخل ملف المشروع** project**.pro** .

تعتبر وحدة الفصائل الأساسية الغير رسومية واحدة من أهم وحدات ***كيوت*** ( Qt Modules )، حيث تحتوى على معظم الفصائل اللازمة للبنية التحتية لأى تطبيق.

أمثلة الفصائل الأساسية الغير رسومية :

فصائل معالجة البيانات:

هى فصائل تتعامل مع النصوص و تحوى البيانات ويتم التعديل فيها وتحويلها من صيغة لأخرى.

فصائل التعامل مع الملفات:

هى فصائل تتعامل مع الملفات من حيث الإنشاء والفتح والإغلاق بجانب كتابة و قراءة البيانات من الملفات.

فصيلة QObject:

هى تعتبر أهم فصيلة فى ***كيوت*** بل هى ما تميز ***كيوت***.

.......................................................................................

وسوف نقوم الآن بعرض مجموعة من أهم فصائل ***كيوت*** وبعض تطبيقات عليها.

ملحوظة :

إذا كنت تستخدم ***كيوت*** لأول مرة فيجب عليك أولاً الإطلاع على الآتى:

- الملحق رقم ( 3 ) : كيفية تنصيب ***كيوت*** صـ 223.
- الملحق رقم ( 4 ) : كيفية بدء تطبيق بواسطة الأداة QtCreator صـ 227.

.......................................................................................

# QDebug Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هى فصيلة تستخدم لإخراج البيانات إلى ملف أو وحدة أو شاشة العرض ( Console ).

**طريقة الإعـــلان ( Declaration ) :**

يتم شحن البيانات المراد إخراجها كالتالى :

```
qDebug(“ Value To Print ”);
```

أو

```
qDebug() << ”value = “ << 10;
```

**وظائف الفصيلة :**

يتم تمرير البيانات إلى qDebug عن طريق العامل ( << ) كما فى طريقة الشحن الثانية و نذكر بأنه يمكن تمرير أى بيانات إلى qDebug سواء كانت تلك البيانات نصية أو رقمية .

وتعتبر qDebug من الفصائل الهامة، وذلك لإستخدامها أثناء كتابة الكود للتأكد من صحة البيانات المراد التعامل معها.

.......................................................................

## QDebug Example

```
#include <QtCore>

int main()
{
   qDebug( “Hi Qt Developers !” ) ;

   qDebug() << “Hi Qt Developers !! again” ;

   qDebug() <<  ”10+20 = “  <<  10+20 ;

}
```

EXAMPLE NO 1

**شرح الكود السابق :**

مخرجات هذا الكود تظهر على الشاشة ( Console )

qDebug( “Hi Qt Developers !” );

طباعة **Hi Qt Developers ! إلى الشاشة** ( Console )

qDebug() << “Hi Qt Developers !! again”;

طباعة **Hi Qt Developers !! again** .

qDebug() <<  ”10+20 = “ << 10+20;

طباعة **10+20 = 30** .

........................................................................................

# QBitArray Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هى فصيلة تقوم بإنشاء مصفوفة عناصرها من نوع البت ( Bits )،حيث كل عنصر يحتوى على أحد القيمتين 0،1 True or False.

**طريقة الإعــلان ( Declaration ) :**

يتم الإعلان عن المصفوفة بأكثر من طريقة منها:

**QBitArray x(8);**

تم الإعلان عن المتغير x لإدارة مصفوفة حجمها ٨ عناصر من نوع البت ( Bits )، ويشحن كل عنصر بالقيمة 0 أو False كقيمة إفتراضية، ويمكن إنشاء مصفوفة تشحن عناصرها بالقيمة 1 أو True كقيمة إبتدائية إذا تم الإعلان عن المصفوفة بالطريقة التالية:

**QBitArray x(8,true);**

**وظائف الفصيلة :**

| الدالــــة | الوظيفــــــــــــة |
|---|---|
| at(int x) | تقوم الدالة بإرجاع قيمة العنصر x إذا كانت true or false. |
| isNull() | ترجع القيمة true إذا أعلنت المصفوفة دون أى عناصر. |
| isEmpty() | ترجع القيمة true إذا كانت عدد عناصر المصفوفة 0. |
| setBit(int x) | تحول قيمة العنصر رقم x القيمة true. |
| setBit(int x, bool t) | تحول قيمة العنصر رقم x القيمة t حيث t يمكن أن تكون true or false. |

مثال على الدالتين isNull , isEmpty لتوضيح الفرق بينهما :

**QBitArray x;**

**x.isNull() ---> will return true.**

**x.isEmpty() ---> will return true.**

**QBitArray x(0);**

**x.isNull() ---> will return false.**

**x.isEmpty() ---> will return true.**

## QBitArray Example

```
#include <QtCore>
int main()
{
    QBitArray x( 8 );
    QBitArray y( 8 , true );
    QBitArray r( 8 );
    x.setBit( 0 );
    x.setBit( 1 );
    x.setBit( 2 );
    x.setBit( 3 );
    x.setBit( 4 );

    y.setBit( 0 , false );
    y.setBit( 1 , false );
    y.setBit( 2 , false );
    r = x & y;
    r = x | y;
    r = x ^ y;
    r =  ~x;
    r =  y;
}
```

EXAMPLE NO 2

**شرح الكود السابق :**

QBitArray x( 8);

الإعلان عن المتغير **x** لإدارة مصفوفة حجم ٨ بت

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

QBitArray y( 8 , true );

الإعلان عن المتغير **y** لإدارة مصفوفة حجم ٨ بت

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

QBitArray r( 8 );

الإعلان عن المتغير **r** لإدارة مصفوفة حجم ٨ بت

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

x.setBit( 0 );

تحول العنصر رقم **0** مصفوفة **x** إلى القيمة True

| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

x.setBit( 1 );

تحول العنصر رقم **1** مصفوفة **x** إلى القيمة True

| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

x.setBit( 2 );

تحول العنصر رقم **2** مصفوفة **x** إلى القيمة True  | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

x.setBit( 3 );

تحول العنصر رقم **3** مصفوفة **x** إلى القيمة True  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

x.setBit( 4 );

تحول العنصر رقم **4** مصفوفة **x** إلى القيمة True  | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

y.setBit( 0 , false );

تحول العنصر رقم **0** مصفوفة **y** إلى القيمة False  | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

y.setBit( 1 , false );

تحول العنصر رقم **1** مصفوفة **y** إلى القيمة False  | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

y.setBit( 2 , false );

تحول العنصر رقم **2** مصفوفة **y** إلى القيمة False  | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

الآن أصبحت المصفوفتين **x , y** كالآتى:

X | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Y | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

r = x & y;  <-- AND Operator

المصفوفة r =  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

r = x | y;   <-- OR   Operator

المصفوفة r =  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

r = x ^ y;  <-- XOR Operator

المصفوفة r =  | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |

r =  ~x;     <-- NOT Operator

المصفوفة r =  | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

r =  y;      <-- Equal Operator

المصفوفة r =  | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

# QByteArray Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هى فصيلة تقوم بإنشاء مصفوفة عناصرها من نوع البايت ( Bytes )، حيث كل عنصر يحتوى على قيمة من 0 إلى 256 ، وهى إختيار جيد لتخزين مصفوفة حروف.

**طريقة الإعـــلان ( Declaration ) :**

يتم الإعلان عن المصفوفة بأكثر من طريقة منها:

**QBitArray x( “Test“ );**

تم الإعلان عن المتغير x لإدارة مصفوفة حجمها 4 عناصر هم T,E,S,T .

**QBitArray x( 5 , ’T’ );**

تم الإعلان عن المتغير x لإدارة مصفوفة حجمها 5 عناصر هم T,T,T,T,T .

**وظائف الفصيلة :**

| الدالـــــة | الوظيفـــــــــــــــــــــــــة |
|---|---|
| at(int x) | تقوم الدالة بإرجاع قيمة العنصر x. |
| left(int x) | تقوم بإرجاع مصفوفة من الحروف عددها **x** بداية من أول المصفوفة. |
| right(int x) | تقوم بإرجاع مصفوفة من الحروف عددها **x** بداية من آخر المصفوفة. |
| mid(int P , int L) | تقوم بإرجاع مصفوفة من الحروف عددها **L** بداية من العنصر رقم P. |
| trimmed() | تقوم بإرجاع مصفوفة الحروف وتقوم بإلغاء العناصر البيضاء إذا كانت فى آخر المصفوفة. |
| chop(int L) | تقوم بإلغاء عدد L عنصر من آخر المصفوفة. |
| truncate(int L) | تأخذ عدد L عنصر من أول المصفوفة و تلغى باقى العناصر و تكون هى المصفوفة الجديدة. |

................................................................................

## QByteArray Example

EXAMPLE NO 3

```
#include <QtCore>
int main()
{
    QByteArray ar( "Hello     " );
    QByteArray br( 5 , 'Q' );
    qDebug() << ar.trimmed();
    ar = ar.trimmed() + br;
    qDebug() << ar[ 0 ];
    qDebug() << ar.at( 1 );
    qDebug() << ar.left( 2 );
    qDebug() << ar.right( 7 );
    qDebug() << ar.mid( 3 , 4 );
    ar.chop( 7 );
    ar.truncate( 2 );
}
```

### شرح الكود السابق :

QByteArray ar( "Hello     " );

الإعلان عن المتغير **ar** لإدارة مصفوفة حجم 10 بايت

| H | e | l | l | o | | | | | |
|---|---|---|---|---|---|---|---|---|---|

QByteArray br( 5 , 'Q' );

الإعلان عن المتغير **br** لإدارة مصفوفة حجم 5 بايت

| Q | Q | Q | Q | Q |
|---|---|---|---|---|

qDebug() << ar.trimmed();

طباعة "**Hello**" بدون العناصر البيضاء

ar = ar.trimmed() + br;

المصفوفة **ar** =

| H | e | l | l | o | Q | Q | Q | Q | Q |
|---|---|---|---|---|---|---|---|---|---|

qDebug() << ar[ 0 ];

طباعة **H** هو العنصر **0** فى المصفوفة.

```
qDebug() << ar.at( 1 );
```

طباعة **e** هو العنصر رقم **1** فى المصفوفة.

```
qDebug() << ar.left( 2 );
```

طباعة **He** هما أول **2** عنصر من بداية المصفوفة.

```
qDebug() << ar.right( 7 );
```

طباعة **loQQQQQ** هم أول 7 عناصر من نهاية المصفوفة.

```
qDebug() << ar.mid( 3 , 4 );
```

طباعة **loQQ** هم أول **4** عناصر بداية من العنصر رقم **3** من بداية المصفوفة.

```
ar.chop( 7 );
```

المصفوفة **ar** = | H | e | l | بعد مسح آخر 7 عناصر.

```
ar.truncate( 2 );
```

المصفوفة **ar** = | H | e | بعد أخذ أول **2** عنصر و إعتبارهم المصفوفة الجديدة.

..........................................................................

# QString Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

تعد هذه الفصيلة من أهم فصائل ***كيوت*** غير الرسومية، حيث أنها الفصيلة الرئيسية للتعامل مع النصوص ( Strings ) سواء الثابت منها أو المتغير، كما تقوم هذه الفصيلة بتخزين الحروف كنص مستخدمة الحروف من نوع 16 بت ترميز ( UniCode )، وهى حالياً طريقة ترميز الحروف القياسية و المستخدمة على أوسع نطاق فى كل التطبيقات الحديثة.

**طريقة الإعــلان ( Declaration ) :**

يمكن شحنها بالنص مباشرة

**QString str( "string" );**

أو الإعلان عن الفصيلة ثم شحنها بعد ذلك

**QString str;**
**str = "string";**

**وظائف الفصيلة :**

تمتلك هذه الفصيلة عدد كبير من الدوال كدوال للبحث داخل النص و دوال للتحويل بين أنواع البيانات من رقمي إلى حرفي أو العكس، ومن رقم عشري إلى سداسي عشر أو العكس. ويمكننا القول أنها تمتلك كل مايلزم من دوال للتعامل مع جميع أشكال و أنواع النصوص.
وسوف نقوم بعرض بعض الدوال الرئيسية.

| الدالــــة | الوظيفــــــــــــــة |
|---|---|
| append(str) | تقوم بإضافة النص str فى آخر نص الفصيلة. |
| insert(int p , str) | تقوم بإضافة النص str إلى نص الفصيلة بعد عنصر رقم p. |
| remove(int P ,int L ) | تقوم بمسح عدد L حرف من نص الفصيلة بعد عنصر رقم p. |
| replace(str1,str2) | تقوم بإستبدال str2 مكان str1. |
| contains(str) | تقوم بإرجاع True فى حالة إحتواء نص الفصيلة على النص str. |
| startsWith(str) | تقوم بإرجاع True فى حالة بدء نص الفصيلة بالنص str. |
| endsWith(str) | تقوم بإرجاع True فى حالة إنتهاء نص الفصيلة بالنص str. |
| setNum(number) | تقوم بتحويل الرقم number إلى نص. |

هذه بعض الدوال البسيطة كبداية للتعارف على QString، ولكن المرجع الشامل يأتى مع مجموعة أدوات ***كيوت***.

## QString Example

```
#include <QtCore>

int main()
{
   QString str( “Hello Qt” );
   str += “ Developer”;
   str.append( “ 2010” );
   str.insert( 0 ,  “Hello !! “ );
   str.remove( 6 , 3 );
   str.replace( “2010”  , “2011” );
   qDebug() << str.contains( “Dev” );
   qDebug() << str.startsWith( “Hello” );
   qDebug() << str.startsWith( “Hi” );
   qDebug() << str.endsWith( “10” );
   qDebug() << str.endsWith( “11” );
   qDebug() << str.setNum( 2010.11 );
   qDebug() << QString::number( 10 , 2 );
   qDebug() << QString::number( 10 , 16 );
   QString strarg( “Hi %1 %2” );
   qDebug() << strarg.arg( “Qt”  , ”Developer” );
}
```

EXAMPLE NO 4

**شرح الكود السابق :**

QString str( “Hello Qt” );

إعلان عن المتغير **str** من النوع **QString** يحوى **Hello Qt.**

str += “ Developer”;

المتغير **str** يحوى **Hello Qt Developer.**

str.append( “ 2010” );

المتغير **str** يحوى **Hello Qt Developer 2010.**

str.insert( 0 ,  “Hello !! “ );

المتغير **str** يحوى **Hello !! Hello Qt Developer 2010.**

str.remove( 6 , 3 );

المتغير **str** يحوى **Hello Hello Qt Developer 2010.**

```
str.replace( “2010” , “2011” );
```

المتغير **str** يحوى **Hello Hello Qt Developer 2011**.

```
qDebug() << str.contains( “Dev” );
```

طباعة **True** نتيجة لإحتواء المتغير **str** على النص **Dev**.

```
qDebug() << str.startsWith( “Hello” );
```

طباعة **True** نتيجة لبدء المتغير **str** بالنص **Hello**.

```
qDebug() << str.startsWith( “Hi” );
```

طباعة **False** نتيجة لعدم بدء المتغير **str** بالنص **Hi**.

```
qDebug() << str.endsWith( “10” );
```

طباعة **False** نتيجة لعدم إنتهاء المتغير **str** بالنص **10**.

```
qDebug() << str.endsWith( “11” );
```

طباعة **True** نتيجة لإنتهاء المتغير **str** بالنص **11**.

```
qDebug() << str.setNum( 2010.11 );
```

تحويل **2010.11** إلى نص و طباعتها.

```
qDebug() << QString::number( 10 , 2 );
```

طباعة **1010** و هى تساوى القيمة **10** بالنظام **2** (الثنائى Binary).

```
qDebug() << QString::number( 10 , 16 );
```

طباعة **a** و هى تساوى القيمة **10** بالنظام **16** (السداسى عشر Hexadecimal).

```
QString strarg( “Hi %1 %2” );
qDebug() << strarg.arg( “Qt” , ”Developer” );
```

طباعة **Hi Qt Developer** حيث يتم التعويض عن 1% , 2% علي الترتيب ب **Qt** و **Developer** بنظام **Function Arguments**.

## الفصائل الحاوية (Container Classes):

هى مجموعة من الفصائل التى تقوم بتخزين البيانات كعناصر داخل قوائم، ولقد صممت هذه الفصائل لتصبح سريعة و آمنة فى حفظ و إستدعاء البيانات.

وللتعامل مع عناصر القوائم ( بالحفظ و الإستدعاء والتعديل) يتم إستخدام فصائل التكرار (Iterator Classes)، والتى سيتم شرحها بعد الإنتهاء من شرح الفصائل الحاوية.

ملخص عام للفصائل الحاوية:

| الفصيلـــــة | التعريـــــف بالفصيلـــــة |
|---|---|
| QList<T> | هى أكثر الفصائل إستخداماً، والتى تقوم بتخزين البيانات من النوع T لتصبح البيانات مفهرسة و مرتبة، وتتعامل تلك الفصيلة مع مؤشرات (Pointers) العناصر، وليس مع بياناتها ( Data ) فتصبح أكثر سرعة فى الإضافة والحذف و التبديل. |
| QVector<T> | تقوم بتخزين البيانات بشكل متتابع فى الذاكرة، وتتعامل تلك الفصيلة مع بيانات العناصر ( Data )، وليس مع مؤشراتها ( Pointers )، و بالتالى فعند إضافة أو مسح عنصر يتم إزاحة جميع بيانات القائمة التى تتبعه فتسبب بطئ ملاحظ إذا كانت القوائم كبيرة الحجم. |
| QQueue<T> | هى فصيلة ترث فصيلة QList و تعمل بنظام FIFO. |
| QStack<T> | هى فصيلة ترث فصيلة QVector و تعمل بنظام LIFO. |
| QSet<T> | هى فصيلة تخزن البيانات بنظرية المجموعات وبالتالى فلا يوجد ترتيب، ولا يمكن تكرار عنصر، ويمكن إجراء العمليات عليها كالتقاطع و الإتحاد. |
| QMap<Key,T> | تقوم بتخزين البيانات فى مصفوفة قاموس حيث أن لكل مفتاح (Key) قيمة تقابله (T) . و(key,T) يمكن أن تأخذ أى شكل من أنواع البيانات نصية كانت أو رقمية، وترتب البيانات تبعاً للمفتاح، ولا يمكن تكرار مفتاح مرتين. |
| QMultiMap<Key,T> | هى فصيلة ترث QMap وتسمح بتكرار المفاتيح، حيث يمكن إعطاء المفتاح (Key) أكثر من قيمة (T). |
| QHash<Key,T> | هى فصيلة تشبه فصيلة QMap تماماً و لكنها أسرع فى الوصول إلى البيانات، ولكن QHash تخزن البيانات بصور غير مرتبة أو مفهرسة. |
| QMultiHash<Key,T> | هى فصيلة ترث QHash وتسمح بتكرار المفاتيح. |

..................................................................................

# QList Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى الفصيلة الأساسية للفصائل الحاوية ( Container Classes )، والتى تقوم بتخزين قائمة من العناصر( يمكن أن تكون رقمية ، نصية أو مؤشرات لفصائل أخرى ).

**طريقة الإعــلان ( Declaration ) :**

يتم الإعلان عن نوع القائمة و اسمها بالطريقة التالية:

**QList<data type> name;**

**EX: QList<QString> strlist;**

تم الإعلان عن قائمة من النوع النصي( QString ) واسمها strlist ويمكن شحن هذه القائمة بقائمة من النصوص ( قائمة أسماء مثلاً). و يتم شحن عناصر القائمة بواسطة العامل (<<).

**EX: strlist << “name1”  << “name2” << “name2”;**

وبالتالى فعند طباعة القائمة strlist سوف نحصل على name1,nam2,nam3.

**وظائف الفصيلة :**

| الدالـــة | الوظيفـــــــــــــــــــــة |
|---|---|
| append( val ) | تقوم بإضافة عنصر جديد قيمته val إلى نهاية القائمة. |
| insert(int p , val) | تقوم بإضافة عنصر جديد قيمته val إلى القائمة بعد العنصر رقم p. |
| swap(int v1 ,int v2) | تقوم بتبديل أماكن عنصرين v2 و v1. |
| removeAt( int p ) | تقوم بمسح العنصر رقم p. |
| pop_back() | تقوم بمسح عنصر من نهاية القائمة. |
| pop_front() | تقوم بمسح عنصر من بداية القائمة. |
| push_back( val ) | تقوم بإضافة عنصر جديد قيمته val إلى نهاية القائمة. |
| push_front( val ) | تقوم بإضافة عنصر جديد قيمته val إلى بداية القائمة. |
| indexOf( val ) | تقوم بإرجاع مكان ( رقم المسلسل ) أول عنصر يساوى القيمة val. |
| value( int p ) | تقوم بإرجاع قيمة العنصر رقم p. |
| move(int v1,int v2) | تقوم بتحريك العنصر v1 إلى المكان v2. |
| prepend( val ) | تقوم بإضافة عنصر جديد قيمته val إلى بداية القائمة. |
| size() | تقوم بإرجاع عدد عناصر القائمة. |
| count( val ) | تقوم بإرجاع عدد مرات تكرار العنصر ذو القيمة val . |
| removeAll( val ) | تقوم بمسح جميع العناصر ذات القيمة val . |

## QList Example

```
#include <QtCore>

int main()
{
   QList<int> mylist;
   mylist << 0 << 1 << 2 << 3 << 5 << 7 << 6;
   mylist.append( 8 );
   mylist.insert( 4 , 4 );
   mylist.swap( 7 , 6 );
   mylist.removeAt( 8 );
   mylist.pop_back();
   mylist.pop_front();
   mylist.push_back( 7 );
   mylist.push_front( 0 );
   qDebug() << mylist.indexOf( 1 );
   qDebug() << mylist.value( 3 );
   mylist.move( 1, 5 );
   mylist.prepend( 0 );
   qDebug() << mylist.size();
   qDebug() << mylist.count( 0 );
}
```

EXAMPLE NO 5

**شرح الكود السابق :**

```
QList<int> mylist;
mylist << 0 << 1 << 2 << 3 << 5 << 7 << 6;
```

الإعلان عن المتغير **mylist** لإدارة قائمة عناصرها من النوع (int).

و شحنها بالعناصر.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 5 | 7 | 6 |

```
mylist.append( 8 );
```

إضافة عنصر قيمته **8** إلى نهاية القائمة.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 5 | 7 | 6 | 8 |

```
mylist.insert( 4 , 4 );
```

إضافة عنصر قيمته **4** بعد العنصر رقم **4**.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 4 | 5 | 7 | 6 | 8 |

```
mylist.swap( 7 , 6 );
```

التبديل بين عنصر رقم **7** و **6**.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
mylist.removeAt( 8 );
```

مسح العنصر رقم **8**.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
mylist.pop_back();
```

مسح آخر عنصر فى القائمة.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

```
mylist.pop_front();
```

مسح أول عنصر فى القائمة.

| No | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Value | 1 | 2 | 3 | 4 | 5 | 6 |

```
mylist.push_back( 7 );
```

إضافة عنصر قيمته **7** إلى آخر القائمة.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
qDebug() << mylist.indexOf( 1 );
```

طباعة **0** حيث أن أول مكان لعنصر يساوى القيمة **1** هو عنصر رقم **0**.

```
qDebug() << mylist.value( 3 );
```

طباعة **4** وهى قيمة العنصر رقم **3**.

```
mylist.move( 1 , 5 );
```

تحريك العنصر رقم **1** إلى المكان رقم **5**.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Value | 1 | 3 | 4 | 5 | 6 | 2 | 7 |

```
mylist.prepend( 0 );
```

إضافة عنصر قيمته **0** إلى أول القائمة.

| No | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Value | 0 | 1 | 3 | 4 | 5 | 6 | 2 | 7 |

```
qDebug() << mylist.size();
```

طباعة **8** وهى عدد عناصر القائمة.

```
qDebug() << mylist.count( 0 );
```

طباعة **1** و هى عدد مرات تكرار القيمة **0** داخل القائمة.

# QVector Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة مثل الفصيلة QList تماماً، ولكن الإختلاف هنا يكمن فى أسلوب الفصيلة فى التعامل مع البيانات فى الذاكرة. ( لفهم الإختلاف يمكنك الرجوع لجدول الفصائل الحاوية ص25 ).

**طريقة الإعــلان ( Declaration ) :**

يتم الإعلان عن نوع القائمة و اسمها بالطريقة التالية:

**QVector<data type> name;**

**EX: QVector<QString> strlist;**

تم الإعلان عن المتغير strlist لإدارة قائمة عناصرها من النوع النصي( QString )، ويمكن شحنها بقائمة من النصوص( قائمة أسماء مثلاً)، و يتم شحن عناصر القائمة بواسطة العامل (<<).

**EX: strlist << "name1" << "name2" << "name2";**

وبالتالى عند طباعة القائمة strlist سوف نحصل علي name1,nam2,nam3.

**وظائف الفصيلة :**

| الدالــــة | الوظيفـــــــــــــــــــة |
|---|---|
| append( val ) | تقوم بإضافة عنصر جديد قيمته val إلى نهاية القائمة. |
| insert(int p , val) | تقوم بإضافة عنصر جديد قيمته val إلى القائمة بعد العنصر رقم p. |
| remove( int p ) | تقوم بمسح العنصر رقم p. |
| remove(int p, int L ) | تقوم بمسح عدد عناصر L بدأً من العنصر p. |
| pop_back() | تقوم بمسح عنصر من نهاية القائمة. |
| pop_front() | تقوم بمسح عنصر من بداية القائمة. |
| push_back( val ) | تقوم بإضافة عنصر جديد قيمته val إلى نهاية القائمة. |
| push_front( val ) | تقوم بإضافة عنصر جديد قيمته val إلى بداية القائمة. |
| indexOf( val ) | تقوم بإرجاع مكان ( رقم المسلسل ) أول عنصر يساوى القيمة val. |
| value( int p ) | تقوم بإرجاع قيمة العنصر رقم p. |
| prepend( val ) | تقوم بإضافة عنصر جديد قيمته val إلى بداية القائمة. |
| size() | تقوم بإرجاع عدد عناصر القائمة. |
| count( val ) | تقوم بإرجاع عدد مرات تكرار العنصر ذو القيمة val . |

* المثال مرفق مع باقى الأمثلة وهو كمثال QList.

EXAMPLE
NO 6

# QQueue Class

**نسب الفصيلة :**

ترث فصيلة QList

**تعريف الفصيلة :**

هذه الفصيلة ترث جميع خصائص و دوال الفصيلة QList، وتتميز بطريقة ( FIFO ) فى إدارة عناصر القوائم فيمكن تشبيه القائمة بأنبوب مفتوح الطرفين كما هو مبين بالرسم.

**طريقة الإعــلان ( Declaration ) :**

يتم الإعلان و الشحن مثل فصيلة QList.

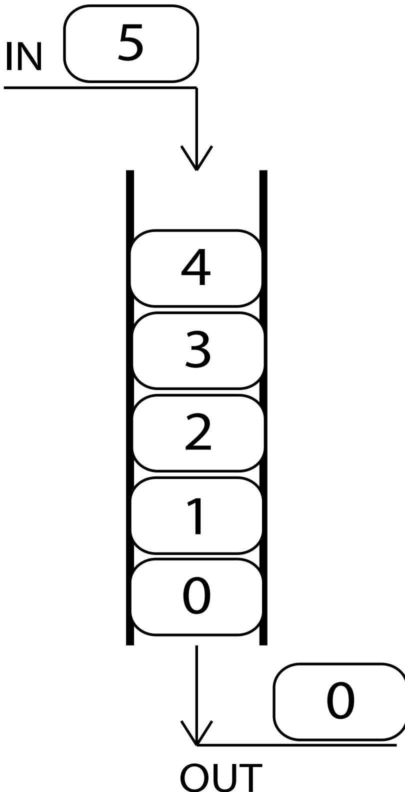
**QQueue<data type> name;**

**EX: QQueue<int> myqueue;**

تم الإعلان عن المتغير myqueue لإدارة قائمة عناصرها من النوع int .

**وظائف الفصيلة :**

| الدالـــــة | الوظيفـــــــــــــــــــــــة |
|---|---|
| dequeue() | تقوم بمسح أول عنصر من القائمة مع إرجاع قيمته. |
| enqueue( val ) | تقوم بإضافة عنصر إلى آخر القائمة مثل الدالة QList::append. |
| head() | تقوم بإرجاع العنصر على رأس القائمة و هو العنصر الأول وهو نفسه العنصر رقم 0. |




FIFO System

First In First Out

أول عنصر يدخل القائمة هو أول عنصر يخرج منها

## QQueue Example

```
#include <QtCore>

int main()
{
    QQueue<QString> myqueue;
    myqueue << "Ahmed" ;    //First In
    myqueue << "Osama" ;
    myqueue << "sami";      //Last  In

    qDebug() << myqueue;
    qDebug() << myqueue.dequeue();
    qDebug() << myqueue;
    qDebug() << myqueue.dequeue();
    qDebug() << myqueue;
    qDebug() << myqueue.head();
    myqueue.enqueue("Mahmoud");
    qDebug() << myqueue;
}
```

EXAMPLE NO 7

**شرح الكود السابق :**

```
QQueue<QString> myqueue;
myqueue << "Ahmed" ;
myqueue << "Osama" ;
myqueue << "sami";
```

الإعلان عن المتغير **myqueue** لإدارة قائمة عناصرها من النوع **QString**.
وشحنها بالعناصر **Ahmed , Osama , sami**.

```
qDebug() << myqueue.dequeue();
```

طباعة **Ahmed** حيث تم سحب العنصر الأول من القائمة.
وتصبح القائمة **Osama , sami**.

```
qDebug() << myqueue.dequeue();
```

طباعة **Osama** حيث تم سحب العنصر الأول من القائمة.
وتصبح القائمة **sami**.

```
myqueue.enqueue("Mahmoud");
```

إضافة **Mahmoud** إلى نهاية القائمة.
وتصبح القائمة **sami , Mahmoud**.

# QStack Class

**نسب الفصيلة :**

ترث فصيلة QVector

**تعريف الفصيلة :**

هذه الفصيلة ترث جميع خصائص و دوال الفصيلة QVector، وتتميز بطريقة (LIFO) فى إدارة عناصر القوائم فيمكن تشبيه القائمة بأنبوب مفتوح من طرف واحد كما هو مبين بالرسم.

**طريقة الإعــلان ( Declaration ) :**

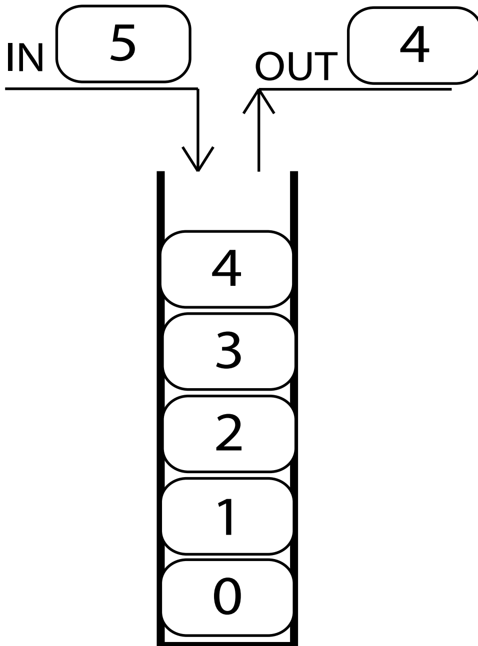
يتم الإعلان و الشحن مثل فصيلة QVector.

```
QStack<data type> name;
QStack<int> mystack;
```

تم الإعلان عن المتغير mystack لإدارة قائمة عناصرها من النوع int .

**وظائف الفصيلة :**

| الدالــــة | الوظيفـــــــــــــــــــــــة |
|---|---|
| pop() | تقوم بمسح آخر عنصر من القائمة مع إرجاع قيمته. |
| push( val ) | تقوم بإضافة عنصر إلى آخر القائمة قيمته val . |
| top() | تقوم بإرجاع العنصر على رأس القائمة و هو العنصر الأخير فى القائمة. |




LIFO System

Last In First Out

آخر عنصر يدخل القائمة هو أول عنصر يخرج منها

## QStack Example

```
#include <QtCore>

int main()
{
   QStack<QString> mystack;
   mystack << "AHmed" ;   //First In
   mystack << "Osama" ;
   mystack << "sami";     //Last  In

   qDebug() << mystack;
   qDebug() << mystack.pop();
   qDebug() << mystack;
   qDebug() << mystack.pop();
   qDebug() << mystack;
   qDebug() << mystack.top();
   mystack.push("Mahmoud");
   qDebug() << mystack;
}
```

EXAMPLE NO 8

**شرح الكود السابق :**

```
QStack<QString> mystack;
mystack << "AHmed" ;
mystack << "Osama" ;
mystack << "sami";
```

الإعلان عن المتغير **mystack** لإدارة قائمة عناصرها من النوع **QString**.

و شحنها بالعناصر **Ahmed , Osama , sami**.

```
qDebug() << mystack.pop();
```

طباعة **sami** حيث تم سحب العنصر الأخير من القائمة.

وتصبح القائمة **Ahmed , Osama**.

```
qDebug() << mystack.pop();
```

طباعة **Osama** حيث تم سحب العنصر الأخير من القائمة.

وتصبح القائمة **Ahmed** .

```
mystack.push("Mahmoud");
```

إضافة **Mahmoud** إلى نهاية القائمة.

وتصبح القائمة **Ahmed , Mahmoud**.

# QStringList Class

## نسب الفصيلة :

ترث فصيلة QList

## تعريف الفصيلة :

هذه الفصيلة ترث جميع خصائص ودوال الفصيلة QList، وهى فصيلة خاصة سابقة التعريف.

وتعرف كالتالى : **QList<QString>**

## طريقة الإعــلان ( Declaration ) :

```
QStringList mylist;
```

ويتم شحن العناصر بالعامل (<<).

```
mylist << "Osama" << "Mohamed" << "Samier";
```

## وظائف الفصيلة :

من أهم وظائف هذه الفصيلة هى عملية الوصل بين النصوص، حيث يمكن وصل قائمة من الكلمات وإخراجها كنص واحد.

وذلك كما سنرى فـى المثال التالى:

| الدالـــة | الوظيفــــــــــــة |
|---|---|
| sort() | تقوم بترتيب عناصر القائمة. |
| filter( QString str) | تقوم بإرجاع أعضاء الفصيلة التى تحتوى على النص str. |
| join( QString str) | تقوم بإرجاع نص يتكون من جميع عناصر القائمة بين كل عنصر النص str. |

..............................................................

## QStringList Example

EXAMPLE NO 9

```
#include <QtCore>

int main()
{
   QStringList mylist;
   mylist << "Osama" << "Mohamed" << "Ali" << "Samier" ;

   mylist.sort();

   qDebug() <<  mylist.filter("med");

   QStringList mylist2;

   QString str("Mahmoud,Ali,Said,Sami,Mohamed");

   mylist2 = str.split( " , " );

   str = mylist2.join( " - " );
}
```

**شرح الكود السابق :**

QStringList mylist;

mylist << "Osama" << "Mohamed" << "Ali" << "Samier" ;

الإعلان عن المتغير **mylist** لإدارة قائمة عناصرها من النوع (QString) بإستخدام QStringList.

و شحنها بالعناصر **Osama , Mohamed , Ali , Samier**.

mylist.sort();

ترتيب القائمة لتصبح كالتالى: **Ali,Mohamed,Osama,Samier**.

QStringList mylist2;

QString str("Mahmoud,Ali,Said,Sami,Mohamed");

الإعلان عن المتغير **mylist2** لإدارة قائمة عناصرها من النوع (QString) .

و الإعلان عن المتغير **str** من النوع QString.

mylist2 = str.split( " , " );

إدخال النص إلى القائمة كمجموعة عناصر يفصل بين كل عنصر ","."

str = mylist2.join( " - " );

وصل عناصر القائمة لإخراجه كنص واحد بين كل عنصر "-".

ويكون النص **str** كالتالى:"**Mahmoud-Ali-Said-Sami-Mohamed**" .

# QSet Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى إحدى الفصائل الحاوية، ولكنها تتعامل مع العناصر كمجموعة وليست كقائمة، فالعناصر تخزن بطريقة غير مرتبة، كما أن هذه المجموعة لاتسمح بتكرار تلك العناصر.

**طريقة الإعـــلان ( Declaration ) :**

**QSet<data type> name;**

**EX : QSet<int> myset;**

ويتم شحن العناصر بالعامل (<<).

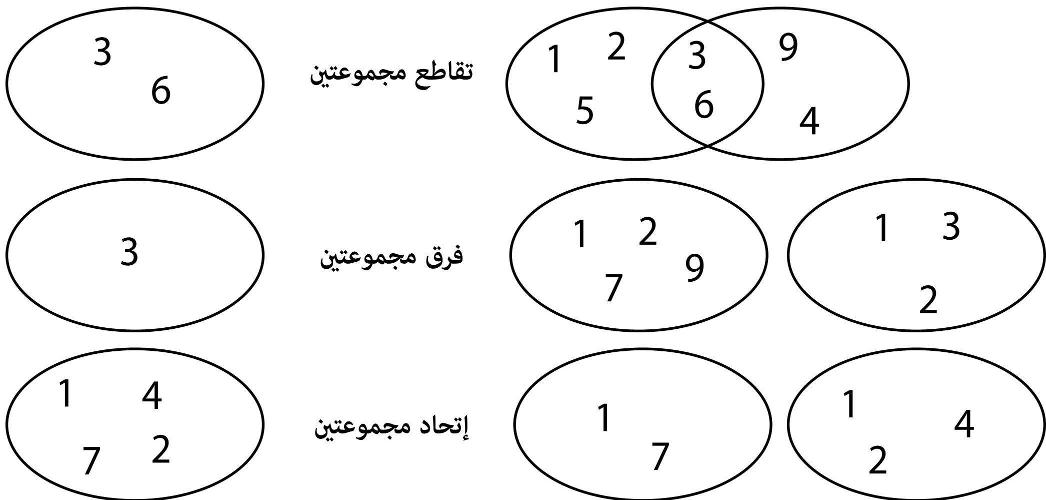
**myset << 1 << 2 << 3 << 4;**

**وظائف الفصيلة :**

من أهم وظائف هذه الفصيلة هى عمليات الإتحاد و التقاطع للمجموعات.

| الدالـــــة | الوظيفـــــــــــــــــــــة |
|---|---|
| intersect(QSet s) | تقوم بإرجاع المجموعة الناتجة عن التقاطع مع المجموعة s. |
| subtract(QSet s) | تقوم بإرجاع المجموعة الناتجة عن الفرق مع المجموعة s. |
| unite(QSet s) | تقوم بإرجاع المجموعة الناتجة عن الإتحاد مع المجموعة s. |

توضيح للعمليات على المجموعات من تقاطع و فرق و إتحاد.



## QSet Example

EXAMPLE NO 10

```
#include <QtCore>

int main()
{
   QSet<int> myset;
   myset << 1 << 2 << 3 << 2 << 5 << 1 << 3 << 1 << 2 << 3 << 5;

  QSet<int> myset2;
   myset2 << 5 << 7 << 2 << 9 << 8;

   qDebug() << myset.intersect( myset2 );
   qDebug() << myset2.subtract( myset );
   qDebug() << myset.unite( myset2 );
}
```

**شرح الكود السابق :**

```
QSet<int> myset;
myset << 1 << 2 << 3 << 2 << 5 << 1 << 3 << 1 << 2 << 3 << 5;
```

الإعلان عن المتغير **myset** لإدارة مجموعة وشحنها بالعناصر ونلاحظ أنه تم شحن المجموعة مثلاً بالعنصر **1** أكثر من مرة ولكن المجموعة **myset** لا تسمح بتكرار العناصر .

```
QSet<int> myset2;
myset2 << 5 << 7 << 2 << 9 << 8;
```

**.myset = (1,2,3,5) and myset2 = (5,7,2,8,9)**

```
qDebug() << myset.intersect( myset2 );
```

هذا الأمر يجعل المجموعة **myset** هى حاصل تقاطع المجموعتين **myset** و **myset2.**

**.myset = (2,5) and myset2 = (5,7,2,8,9)**

```
qDebug() << myset2.subtract( myset );
```

هذا الأمر يجعل المجموعة **myset2** هى حاصل فرق المجموعتين **myset** و **myset2.**

**myset = (2,5) and myset2 = (7,8,9)**

```
qDebug() << myset.unite( myset2 );
```

هذا الأمر يجعل المجموعة **myset2** هى حاصل إتحاد المجموعتين **myset** و **myset2.**

**myset = (2,5،7،8،9) and myset2 = (7,8,9)**

# QMap Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى إحدى الفصائل الحاوية والتى تقوم بتخزين عناصرها فى أزواج (pairs) على هيئة (مفتاح ، قيمة) (Key , Value)، وتسمى طريقة القاموس لأن القاموس يتكون من زوجين الكلمة ومعناها، والجدير بالذكر أن هذه الفصيلة لاتسمح بتخزين أكثر من قيمة لنفس المفتاح، ويمكن إعطاء المفتاح أكثر من قيمة بإستخدام فصيلة QMultiMap .

**طريقة الإعــلان ( Declaration ) :**

```
QMap<KEY data type ,VALUE data type> name;
EX : QMap<QString,int> mymap;
```

تم الإعلان عن المتغير mymap لإدارة قائمة مزدوجة :

نوع بيانات المفتاح ( نصى ) و القيمة ( رقمى ).

ويتم شحن عناصر القائمة بطريقتين كالتالى:

```
mymap.insert("good",1);
mymap["bad"] = 0;
```

**وظائف الفصيلة :**

| الدالـــــة | الوظيفـــــــــــــــة |
|---|---|
| key(value v) | تقوم بإرجاع أول مفتاح تقابله القيمة v. |
| keys(value v) | تقوم بإرجاع قائمة بالمفاتيح التى تقابلها القيمة v. |
| keys() | تقوم بإرجاع قائمة بجميع المفاتيح. |
| value(key k) | تقوم بإرجاع القيمة المقابلة للمفتاح k. |
| values() | تقوم بإرجاع قائمة بجميع القيم. |
| take(key k) | تقوم بإرجاع القيمة المقابلة للمفتاح k ثم مسح المفتاح k. |
| count() | تقوم بإرجاع عدد عناصر الفصيلة. |

.....................................................................

## QMap Example

```
#include <QtCore>
int main()
{
   QMap <QString , int > mymap;

   mymap["hadi"]    = 2;
   mymap["sami"]      = 2;

   mymap.insert( "ali" , 1 );
   mymap.insert( "hadi" , 1 );

   qDebug() << mymap.key( 1 );

   qDebug() << mymap.keys( 1 );

   qDebug() << mymap.keys();

   qDebug() << mymap.value( "sami" );

   qDebug() << mymap.values();

   qDebug() << mymap.take( "ali" );
}
```

EXAMPLE NO 11

**شرح الكود السابق :**

```
QMap <QString , int > mymap;
mymap["hadi"]    = 1;
mymap["sami"]    = 2;
mymap.insert( "ali" ,  1 );
mymap.insert( "hadi" , 1 );
```

الإعلان عن المتغير **mymap** لإدارة قائمة مزدوجة.
وشحنها بالعناصر.

| Key | Value |
|---|---|
| ali | 1 |
| hadi | 1 |
| sami | 2 |

نلاحظ أنه تم ترتيب عناصر القائمة تبعاً للمفتاح ( **key** ) ترتيباً أبجدياً، كما نلاحظ أن القيمة المقابلة للمفتاح ( **hadi** ) كانت( **2** ) ثم عند إضافته مرة أخرى بالقيمة الجديدة تم تغيير قيمته إلى ( **1** ) و لم يضاف كمفتاح جديد لأن الفصيلة لا تسمح بذلك.

```
qDebug() << mymap.key( 1 );
```

طباعة المفتاح **ali** لأنه أول مفتاح عنصر يقابله القيمة **1**.

```
qDebug() << mymap.keys( 1 );
```

طباعة المفاتيح **ali , hadi** وهى العناصر التى تقابلها القيمة **1**.

```
qDebug() << mymap.keys();
```

طباعة كل المفاتيح **ali , hadi , sami**.

```
qDebug() << mymap.value( "sami" );
```

طباعة القيمة **2** المقابلة للمفتاح **sami**.

```
qDebug() << mymap.values();
```

طباعة كل القيم **2،1،1**.

```
qDebug() << mymap.take( "ali" );
```

طباعة القيمة **1** المقابلة للمفتاح **ali**، ومسح المفتاح لتصبح القائمة كالتالى:

| **Key** | **Value** |
|---|---|
| hadi | 1 |
| sami | 2 |

.............................................................................

# QMultiMap Class

**نسب الفصيلة :**

ترث فصيلة QMap

**تعريف الفصيلة :**

هذه الفصيلة هى إحدى الفصائل الحاوية والتى ترث QMap، وتسمح بتخزين أكثر من قيمة لنفس المفتاح .

**طريقة الإعــلان ( Declaration ) :**

**QMultiMap<KEY data type ,VALUE data type> name;**

**EX : QMultiMap<QString,int> mymap;**

تم الإعلان عن المتغير mymap لإدارة قائمة مزدوجة :

نوع بيانات المفتاح ( نصى ) و القيمة ( رقمى ).

ويتم شحن عناصر القائمة بطريقتين كالتالى:

**mymap.insert( “good” , 1 );**

**mymap[“bad”] = 0;**

**وظائف الفصيلة :**

لها نفس وظائف و دوال الفصيلة QMap .

| الدالــــة | الوظيفــــــــــــــــــــــة |
|---|---|
| values(key k) | تقوم بإرجاع قائمة بجميع القيم المقابلة للمفتاح k. |
| count(key k) | تقوم بإرجاع عدد العناصر ذات المفتاح k. |

.........................................................................

## QMultiMap Example

```
#include <QtCore>
int main()
{
   QMultiMap <QString , int > myMmap,myMmap1,myMmap2;
   myMmap1.insert( "ali"     , 1 );
   myMmap1.insert( "sami"  , 1 );
   myMmap1.insert( "mohamed"  , 1 );
   myMmap2.insert( "sami"  , 2 );
   myMmap2.insert( "ali"     , 3 );
   myMmap2.insert( "sami"  , 1 );
   myMmap2.insert( "ali"     , 4 );
   myMmap = myMmap1 + myMmap2;
}
```

EXAMPLE NO 12

### شرح الكود السابق :

QMultiMap <QString , int > myMmap,myMmap1,myMmap2;

الإعلان عن المتغيرات **mymap,mymap1,mymap2** لإدارة القوائم.

```
myMmap1.insert("ali"      , 1 );
myMmap1.insert("sami"     , 1 );
myMmap1.insert("mohamed"  , 1 );
```

maymap1

| Key | Value |
|---|---|
| ali | 1 |
| mohamed | 1 |
| sami | 1 |

```
myMmap2.insert("sami"     , 2 );
myMmap2.insert("ali"      , 3 );
myMmap2.insert("sami"     , 1 );
myMmap2.insert("ali"      , 4 );
```

maymap2

| Key | Value |
|---|---|
| ali | 4 |
| ali | 3 |
| sami | 1 |
| sami | 2 |

```
myMmap = myMmap1 + myMmap2;
```

mymap

| Key | Value |
|---|---|
| ali | 4 |
| ali | 3 |
| ali | 1 |
| mohamed | 1 |
| sami | 1 |
| sami | 2 |
| sami | 1 |

## فصائل التكرار (Iterator Classes):

هذه الفصائل تقدم طريقة موحدة للقيام بالتعامل مع القوائم و عناصرها، ويوجد أسلوبين من أساليب الفصائل التكرارية هما : ( Java-Style- Iterators ), ( STL - Style -Iterators ).

### طريقة الجافا (Java Style Iterator):

طريقة الجافا هى طريقة جديدة فى ***كيوت*** وهى أكثر راحة من طريقة STL.

جدول فصائل التكرار(Iterator Classes):

| Containers | Read-only iterator | Read-write iterator |
|---|---|---|
| QList<T>, QQueue<T> | QListIterator | QMutableListIterator |
| QLinkedList | QLinkedListIterator<T> | QMutableLinkedListIterator |
| QVector<T>, QStack<T> | QVectorIterator<T> | QMutableVectorIterator<T> |
| QSet<T> | QSetIterator<T> | QMutableSetIterator<T> |
| QMap<Key, T>, QMultiMap<Key, T> | QMapIterator<Key, T> | QMutableMapIterator<Key, T> |
| QHash<Key, T>, QMultiHash<Key, T> | QHashIterator<Key, T> | QMutableHashIterator<Key, T> |

### طريقة (STL Style Iterator):

هى الطريقة القديمة المستخدمة منذ ***كيوت*** 2.0 وهى مفضلة فى حالة إذا ما كانت السرعة مطلوبة.

جدول فصائل التكرار(Iterator Classes):

| Containers | Read-only iterator | Read-write iterator |
|---|---|---|
| QList<T>, QQueue<T> | QList<T>::const_iterator | QList<T>::iterator |
| QLinkedList | QLinkedList<T>::const_iterator | QLinkedList<T>::iterator |
| QVector<T>, QStack<T> | QVector<T>::const_iterator | QVector<T>::iterator |
| QSet<T> | QSet<T>::const_iterator | QSet<T>::iterator |
| QMap<Key, T>, QMultiMap<Key, T> | QMap<Key, T>::const_iterator | QMap<Key, T>::iterator |
| QHash<Key, T>, QMultiHash<Key, T> | QHash<Key, T>::const_iterator | QHash<Key, T>::iterator |

وسوف نقوم بشرح الطريقتين على الفصيلة QList فقط، حيث أن جميع الفصائل تعمل بنفس الطريقة.

..............................................................................

# QListIterator Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى إحدى فصائل التكرار من نوع الجافا Java - Style - Iterator.

**طريقة الإعـــلان ( Declaration ) :**

**QListIterator<data type> name(QList);**

**EX:**

**QList<QString> mylist;**

**QListIterator<QString> iter(mylist);**

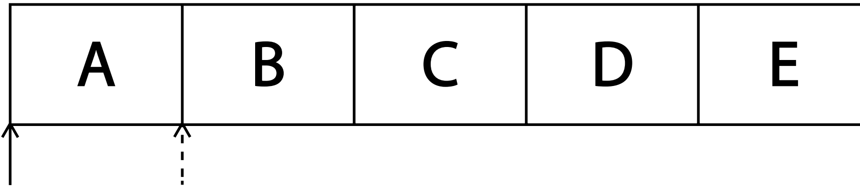
تم الإعلان عن المتغير mylist لإدارة قائمة عناصر من النوع QString.

و إعلان المتغير iter ليشير إلى عناصر القائمة mylist.

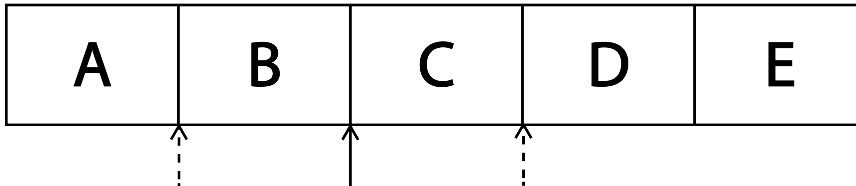
**وظائف الفصيلة :**

تقوم هذه الفصيلة بتعريف **مؤشر** يتنقل بين عناصر القائمة، ويقوم هذا المؤشر بإرجاع قيمة العنصر المطلوب، ويمكن إستخدام هذا المؤشر أيضاً لإجراء عملية معينة على كل عناصر القائمة.

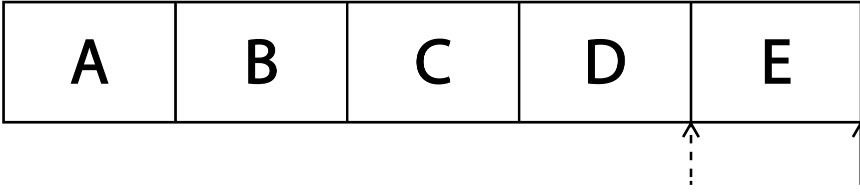
- طريقة الجافا تعتمد على وضع المؤشر بين عناصر القائمة و ليس على العنصر نفسه كما سنرى فى طريقة STL.




- عند بداية التعريف يقف المؤشر قبل أول عنصر فى القائمة، وبالتالى لا يكون أمامه إلا التحرك للأمام ( next ).




- فى حالة وجود المؤشر بين عنصرين يمكن تحريكه فى الإتجاهين، للأمام (next) أو للخلف (previous).




- فى حالة وصول المؤشر بعد آخر عنصر لا يكون أمامه إلا التحرك للخلف (previous).

| الدالـــــة | الوظيفــــــــــــــة |
|---|---|
| hasNext () | تقوم بإرجاع true فى حالة وجود عنصر بعد المؤشر. |
| hasPrevious () | تقوم بإرجاع true فى حالة وجود عنصر قبل المؤشر. |
| next () | تقوم بتحريك المؤشر خطوة للأمام مع إرجاع قيمة العنصر الذى عبر من فوقه المؤشر. |
| previous () | تقوم بتحريك المؤشر خطوة للخلف مع إرجاع قيمة العنصر الذى عبر من فوقه المؤشر. |
| peekNext () | تقوم بإرجاع قيمة العنصر بعد المؤشر دون تحريك المؤشر. |
| peekPrevious () | تقوم بإرجاع قيمة العنصر قبل المؤشر دون تحريك المؤشر. |
| toBack () | تقوم بوضع المؤشر بعد آخر عنصر فى القائمة. |
| toFront () | تقوم بوضع المؤشر قبل أول عنصر فى القائمة. |

.........................................................................

# STL Style Iterator

**طريقة الإعـــلان ( Declaration ) :**

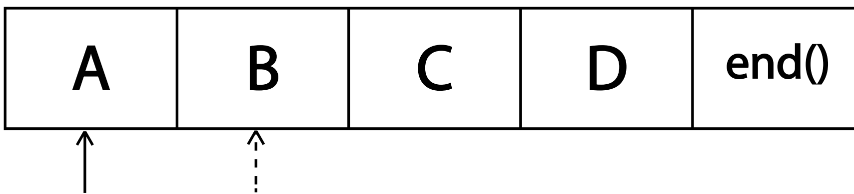
```
Container class<data type>::iterator name;
EX:
QList<QString> mylist;
QList<QString>::iterator iter;
```

تم الإعلان عن المتغير mylist لإدارة قائمة عناصر من النوع QString.
و إعلان المتغير iter ليشير لعناصر القائمة mylist.

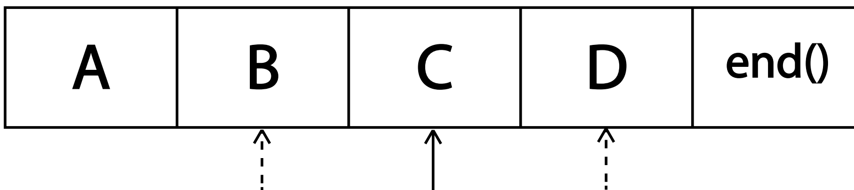
**طريقة العمل :**

تعمل هذه الفصيلة بتعريف **مؤشر** يتنقل على عناصر القائمة، ويقوم هذا المؤشر بإرجاع قيمة العنصر المطلوب، ويمكن إستخدام هذا المؤشر أيضاً لإجراء عملية معينة على كل عناصر القائمة.

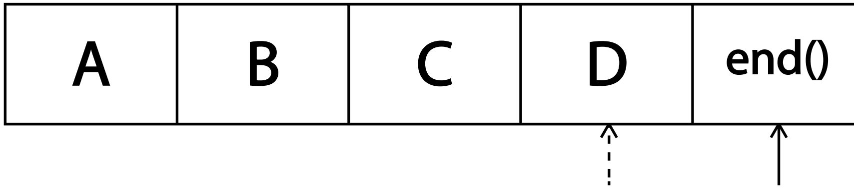
- هذه الطريقة على خلاف طريقة الجافا تعتمد على وضع المؤشر على عناصر القائمة و ليس بين العناصر.




- عند بداية التعريف يتم تعريف وضع المؤشر إذا كان على أول عنصر فلا يكون أمامه إلا التحرك للأمام (++).




- فى حالة وجود المؤشر بين عنصرين يمكن تحريكه فى الإتجاهين للأمام(++) أو للخلف (--).




- فى حالة وصول المؤشر عند آخر عنصر لا يكون أمامه إلا التحرك للخلف (--)، وهذه الطريقة تضع عنصر إفتراضى بعد نهاية عناصر القائمة هو( ()end ) وهو كعنصر ليس له قيمة و لكنه فقط للإستدلال على نهاية القائمة.

| الوظيفـــــــــــــــــة | الدالـــــة |
|---|---|
| تقوم بإرجاع قيمة العنصر الذى يقف عليه المؤشر. | *iterator |
| تقوم بوضع المؤشر على العنصر ()end. | end () |
| تقوم بوضع المؤشر على العنصر الأول. | begin () |
| تقوم بتحريك المؤشر خطوة للأمام. | ++ iterator |
| تقوم بتحريك المؤشر خطوة للخلف. | -- iterator |
| تقوم بتحريك المؤشر عدد n خطوة للأمام. | iterator += n |
| تقوم بتحريك المؤشر عدد n خطوة للخلف. | iterator -= n |
| تقوم بإرجاع عدد العناصر ما بين iterator و j. | iterator - j |

......................................................................

## QListIterator Example

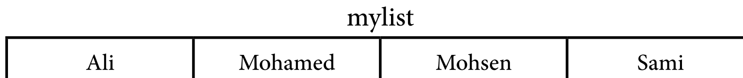
EXAMPLE NO 13

```
#include <QtCore>
int main()
{
   QList<QString> mylist;
   mylist << "Ali" << "Mohamed" << "Mohsen" << "Sami";
   QListIterator<QString> itr(mylist);
   while(itr.hasNext())
   qDebug() << itr.next();
   while(itr.hasPrevious())
   qDebug() << itr.previous();
   qDebug() << itr.peekNext();
   itr.toBack();
   qDebug() << itr.previous();
}
```

**شرح الكود السابق :**

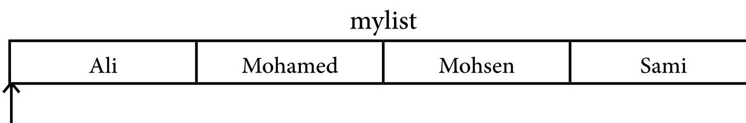
```
QList<QString> mylist;
mylist << "Ali" << "Mohamed" << "Mohsen" << "Sami";
```

الإعلان عن المتغير **mylist** لإدارة القائمة وشحنها.



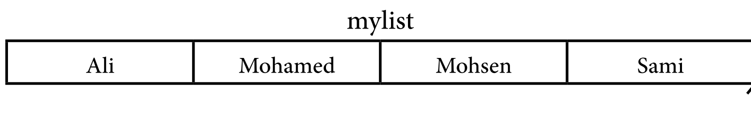

```
QListIterator<QString> itr(mylist);
```

الإعلان عن المتغير **itr** ليشير إلى عناصر القائمة **mylist**، ويبدأ المؤشر بالتواجد قبل أول عنصر.



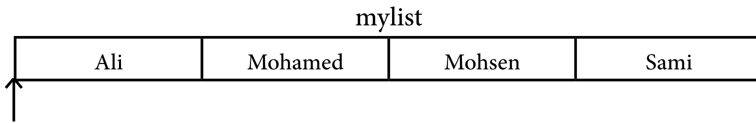

```
while(itr.hasNext())
qDebug() << itr.next();
```

طباعة جميع عناصر القائمة ويقف المؤشر بعد آخر عنصر.



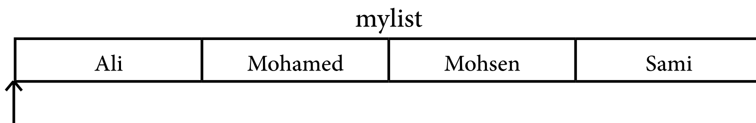
```
while(itr.hasPrevious())
qDebug() << itr.previous();
```

طباعة جميع عناصر القائمة من آخر إلى أول عنصر ويقف المؤشر قبل أول عنصر.



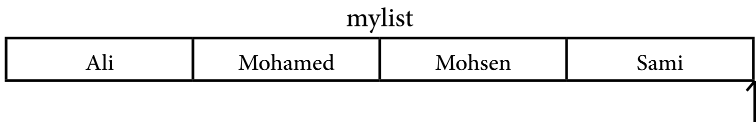

```
qDebug() << itr.peekNext();
```

طباعة العنصر (**Ali**) وهو العنصر التالى لمكان المؤشر، ولا يتحرك المؤشر من مكانه.



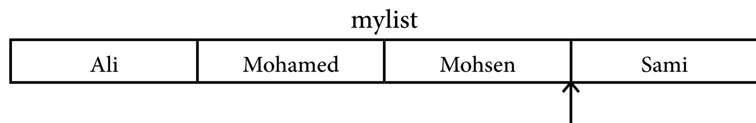

```
itr.toBack();
```

تحريك المؤشر إلى ما بعد آخر عنصر.




```
qDebug() << itr.previous();
```

طباعة العنصر (**Sami**) وهو العنصر السابق لمكان المؤشر، ويتحرك المؤشر خطوة للخلف.




............................................................................

## STL Iterator Example

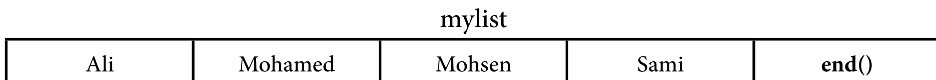
```
#include <QtCore>
int main()
{
   QList<QString> mylist;
   mylist << "Ali" << "Mohamed" << "Mohsen" << "Sami";

   QList<QString>::iterator itr;
   for( itr = mylist.begin() ; itr != mylist.end() ;  itr++)
      qDebug() << *itr;
   --itr;
   qDebug() << *( itr );
   qDebug() << *( --itr );
   qDebug() << *( --itr );
   qDebug() << *( ++itr );
   *(itr) = "Modified";
}
```

EXAMPLE NO 14

**شرح الكود السابق :**

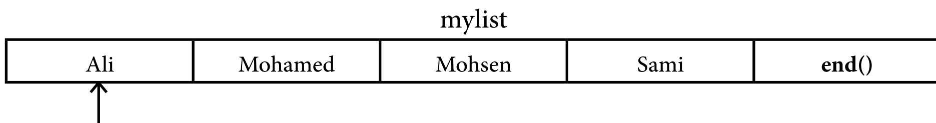
```
QList<QString> mylist;
mylist << "Ali" << "Mohamed" << "Mohsen" << "Sami";
```

الإعلان عن المتغير **mylist** لإدارة القائمة وشحنها.



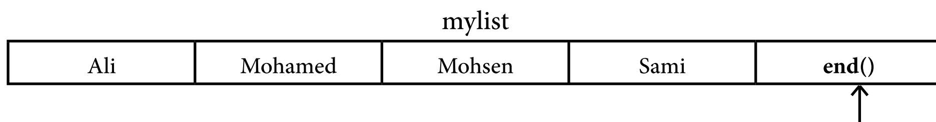

```
QList<QString>::iterator itr;
```

الإعلان عن المتغير **itr** ليشير إلى عناصر القائمة **mylist**، ويبدأ المؤشر بالقيمة 0 عند أول عنصر.



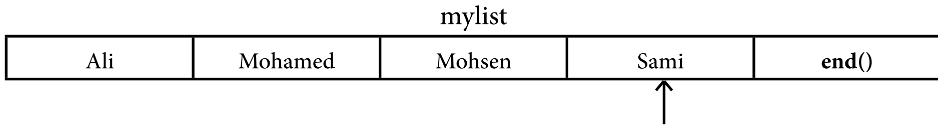

```
for(itr = mylist.begin() ; itr != mylist.end() ; itr++)
qDebug() << *itr;
```

طباعة جميع عناصر القائمة ويقف المؤشر عند آخر عنصر( () end ).



```
--itr;
```

تحريك المؤشر خطوة للخلف .

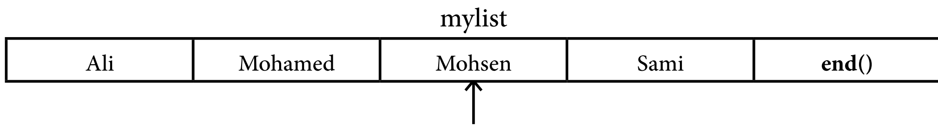



```
qDebug() << *(itr);
```

طباعة( **Sami** ).

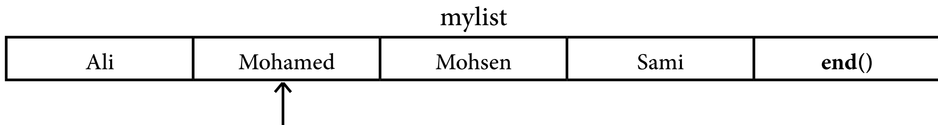
```
qDebug() << *(--itr);
```

تحريك المؤشر خطوة للخلف و يتم طباعة( **Mohsen** ).



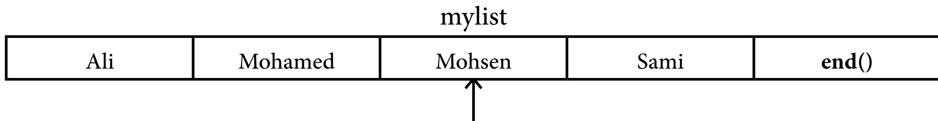

```
qDebug() << *(--itr);
```

تحريك المؤشر خطوة للخلف و يتم طباعة( **Mohamed** ).



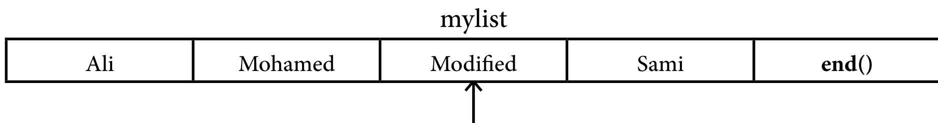

```
qDebug() << *(++itr);
```

تحريك المؤشر خطوة للأمام و يتم طباعة( **Mohsen** ).




```
*(itr) = "Modified";
```

تغيير قيمة العنصر المشار إليه بالقيمة ( **Modified** ).




.............................................................................

# foreach Keyword

**تعريف:**

هى مصطلح تأتى به ***كيوت*** مثل for و while و switch ، وتقوم **foreach** بتفريغ عناصر أى قائمة (القوائم التى تنتجها الفصائل الحاوية) داخل متغير من نفس نوع عناصر تلك القائمة، ومن ثم يمكن إجراء أى عملية على كل عنصر سواء كانت بالتعديل أو المسح... الخ.

صيغة الإستخدام كالآتى:

foreach (variable , container)

**QDir , QFileInfo Example**

```
#include <QtCore>
int main()
{
   QList<QString> mylist;
   mylist << "Mohamed" << "Mahmoud" << "Ali" << "Omar";
   foreach ( QString liststr , mylist )
      qDebug() << liststr;

   QMap<QString , int> mymap;
   mymap["ahmed"] = 1;
   mymap["mohamed"] = 2;
   mymap["osama"] = 3;

   foreach ( QString mapstr , mymap.keys() )
      qDebug() << mapstr;

   foreach ( int mapint , mymap.values() )
      qDebug() << mapint;
}
```

EXAMPLE NO 15

**شرح الكود السابق :**

```
QList<QString> mylist;
mylist << "Mohamed" << "Mahmoud" << "Ali" << "Omar";
```

الإعلان عن المتغير **mylist** لإدارة قائمة ذات عناصر من النوع **QString** و شحنها.

```
foreach ( QString liststr ,  mylist )
  qDebug() << liststr;
```

الإعلان عن المتغير **liststr** من النوع **QString.**

وتقوم **foreach** بشحن المتغير **liststr** بالقيم من القائمة **mylist** بالتتابع ثم طباعتها.

```
QMap<QString , int> mymap;
mymap["ahmed"] = 1;
mymap["mohamed"] = 2;
mymap["osama"] = 3;
```

الإعلان عن المتغير **mymap** لإدارة قائمة تحتوى على مفتاح من النوع **QString** وقيمة من النوع **int**.

```
foreach ( QString mapstr ,  mymap.keys() )
   qDebug() << mapstr;
```

الإعلان عن المتغير **mapstr** من النوع **QString** .

وتقوم **foreach** بشحن المتغير **mapstr** بالمفاتيح من **mymap** بالتتابع ثم طباعتها.

```
foreach ( int mapint ,  mymap.values() )
   qDebug() << mapint;
```

الإعلان عن المتغير **mapint** من النوع **int**.

وتقوم **foreach** بشحن المتغير **mapint** بالقيم من **mymap** بالتتابع ثم طباعتها.

هناك أيضا مصطلح جديد تأتى به ***كيوت*** وهو forever حيث يقوم بعمل تكرار لانهائى ويتم إستخدامه بالشكل التالى:

```
forever{
...
}
```

كل ما بداخل القوسين { } يتم تنفيذه بشكا لانهائى.

..............................................................................

# QDir Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة تستخدم لمعالجة أسماء مسارات الملفات، وهنا نجد أن **كيوت** تستخدم العلامة ( / ) كفاصل بين المجلدات، وكما نعلم فإن نظام ويندوز يستخدم العلامة ( \ )، ولكن **كيوت** تتعامل مع نظام التشغيل دون تغيير كود البرمجة.

وسوف يوضح المثال التالى كيفية تعامل **كيوت** مع هذا الوضع.

**1: QDir("/home/user/Documents") use in mac or linux.**

**2: QDir("C:/Documents and Settings") use in windwos.**

فى الحالة الثانية ستقوم **كيوت** بتحويل العلامة ( / ) إلى العلامة ( \ ) تلقائياً ليتلائم مع النظام وتصبح النتيجة بهذا الشكل C:\Documents and Settings.

**طريقة الإعـــلان ( Declaration ) :**

**QDir name("dir name");**

**EX: QDir mydir("C:\\windows");**

**وظائف الفصيلة :**

| الدالـــــة | الوظيفـــــــــــــــــــــــــــة |
|---|---|
| cd(dir name) | تقوم بتحويل المسار إلى مسار dir name. |
| mkdir(dir name) | تقوم بإنشاء مجلد باسم dir name. |
| mkpath(dir path) | تقوم بإنشاء مجلدات متتابعة dir path. |
| absolutePath() | تقوم بإرجاع المسار الحالى. |
| entryInfoList(atrribute) | تقوم بإرجاع قائمة الملفات ذات خصائص معينة attribute. |

..................................................................................

# QFileInfo Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة تختص بالتعامل مع الملف أو المجلد الواحد من حيث الاسم ، المساحة ، المسار ، صلاحية الوصول إليه و إلى آخره من تلك البيانات.

وللتعامل مع قائمة ملفات سوف نستخدم الفصيلة QFileInfoList التى ترث الفصيلة QFileInfo وهى معرفة بـQList<QFileInfo>.

**طريقة الإعــلان ( Declaration ) :**

**QFileInfo name(“filepath/filename”);**

**EX: QDir mydir(“c:\\mypath\\myfile.txt“);**

**وظائف الفصيلة :**

| الدالـــــة | الوظيفــــــــــــــــة |
|---|---|
| cd(dir name) | تقوم بتحويل المسار إلى مسار dir name. |
| mkdir(dir name) | تقوم بإنشاء مجلد باسم dir name. |
| mkpath(dir path) | تقوم بإنشاء مجلدات متتابعة dir path. |
| absolutePath() | تقوم بإرجاع المسار الحالى. |
| entryInfoList(atrribute) | تقوم بإرجاع قائمة الملفات ذات خصائص معينة attribute. |

ونذكر هنا أن ما قدمناه هو نبذة بسيطة عن خصائص الفصائل التى تتعامل مع الملفات والمجلدات على نظم التشغيل المختلفة ، ولمزيد من المعرفة لهذه الفصائل وخصائصها يمكن مراجعة أداة المساعدة من ***كيوت***.

وفى المثال التالى سوف نقوم بشرح QDir, QFileInfo , QFileInfoList لنعطى صورة مبسطة عن كيفية التعامل مع هذه الفصائل.

..........................................................................................

## QDir , QFileInfo Example

```
#include <QtCore>

int main()
{
   QDir mydir;
   QFileInfoList flist;
   mydir.cd(mydir.currentPath());
   qDebug() << mydir.absolutePath();
   mydir.mkdir( "test" );
   mydir.cd( "test" );
   mydir.mkdir( "1" );
   mydir.cd( "1" );
   mydir.mkdir( "2" );
   mydir.cdUp();
   mydir.cd( ".." );
   mydir.mkpath( "test2/1/2" );
   flist = mydir.entryInfoList(QDir::AllEntries ,QDir::Size);
   QString d;
   qDebug() << "FileName" << "\t\t" << "Size" << " \t " << "Type\n";
   foreach( QFileInfo f  ,  flist )
   {
      if ( f.isDir() )        d = "DIR";
      else d = "file";
      qDebug() << f.fileName() << "\t\t" << f.size() << " \t " << d;
   }
}
```

EXAMPLE NO 16

**شرح الكود السابق :**

```
QDir mydir;
QFileInfoList flist;
```

الإعلان عن المتغير **mydir** و قائمة ملفات **flist**.

```
mydir.cd(mydir.currentPath());
```

تحويل مسار **mydir** إلى مسار التطبيق الحالى.

```
qDebug() << mydir.absolutePath();
```

طباعة المسار الحالى.

```
mydir.mkdir("test");
```

إنشاء مجلد **test**.

```
mydir.cd("test");
```

الدخول لمجلد **test**.

```
mydir.mkdir("1");
```

إنشاء المجلد **1**.

```
mydir.cd("1");
```

الدخول للمجلد **1**.

```
mydir.mkdir("2");
```

إنشاء المجلد **2**.

```
mydir.cdUp();
```

الخروج من المجلد 1 إلى المجلد test.

```
mydir.cd("..");
```

الخروج لمجلد التطبيق حيث **cdUP()** تساوى **cd(".."**) وهو الخروج من المجلد الحالى إلى المجلد الأعلى.

```
mydir.mkpath("test2/1/2");
```

إنشاء المسار **test2 / 1 /2**.

```
flist = mydir.entryInfoList(QDir::AllEntries ,QDir::Size);
```

شحن قائمة الملفات **flist** بملفات المسار الحالى بترتيب المساحات **QDir::Size**.

```
foreach(QFileInfo f , flist)
  {
    if (f.isDir()) d = "DIR";
    else d = "file";
    qDebug() << f.fileName() << "\t\t" << f.size() << " \t " << d;
  }
```

المرور على الملفات لتحديد الملف من المجلد وطباعته.

# QFile Class

## نسب الفصيلة :

ترث فصيلة QIODevice

## تعريف الفصيلة :

هذه الفصيلة تستخدم للتعامل مع الملفات من حيث الإنشاء ، القراءة و الكتابة.

## طريقة الإعـــلان ( Declaration ) :

```
QFile myfile("myfile.txt");
```

## وظائف الفصيلة :

| الدالــــة | الوظيفـــــــــــــــــــة |
|---|---|
| open(option) | تقوم بفتح الملف حسب الخيارات option (للكتابة أو القراءة و الكتابة و القراءة). |
| read(length) | تقوم بقراءة عدد حروف length من الملف. |
| seek(length) | تقوم بوضع مؤشر البيانات فى مكان يبعد عن بداية الملف بمقدار length حرف. |
| readLine() | تقوم بقراءة سطر من البيانات. |
| write(data) | تقوم بكتابة البيانات data داخل الملف. |

............................................................................

## QFile Example

```
#include <QtCore>

int main()
{
   QFile File("test.txt");

   File.open(QIODevice::ReadWrite | QIODevice::Text);

   File.write( "Hi Qt Developer !!\n" );

   File.seek( 0 );

   qDebug() << File.readLine();

   File.close();
}
```

EXAMPLE NO 17

**شرح الكود السابق :**

```
QFile File("test.txt");
```

الإعلان عن المتغير **File** لإدارة الملف **test.txt**.

```
File.open(QIODevice::ReadWrite | QIODevice::Text);
```

فتح الملف للقراءة و الكتابة كنص و ليس كبيانات.

```
File.write( "Hi Qt Developer !!\n" );
```

كتابة **!! Hi Qt Developer** داخل الملف.

```
File.seek( 0 );
```

وضع مؤشر الحركة داخل الملف عند **0** أى عند أول الملف .

```
qDebug() << File.readLine();
```

قراءة سطر وطباعته على الشاشة.

```
File.close();
```

إغلاق الملف.

# QTextStream Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة تقوم بقراءة و كتابة النصوص ( Text )، وتقوم بهذا العمل على QIODevice أو QByteArray أو QString. وسوف نرى فى المثال التالى كيفية تعامل فصيلة QTextStream مع فصيلة QFile التى ترث فصيلة QIODevice .

**طريقة الإعــلان ( Declaration ) :**

```
QTextStream mystream(QIODevice);
QTextStream mystream(QByteArray);
QTextStream mystream(QString);
```

**وظائف الفصيلة :**

يتم التعامل على البيانات بالعاملين (<<) ، (>>) المختصين بإدخال و إخراج البيانات.

**QTextStream Example**

EXAMPLE NO 18

```
#include <QtCore>

int main()
{
   QFile File("test.txt");

   File.open(QIODevice::ReadWrite | QIODevice::Text);

   QTextStream fstr( &File );

   fstr << "Hi Qt Developer !!\n";

   fstr << 65;

   fstr.seek( 0 );

   qDebug() << fstr.readLine();

   qDebug() << fstr.readLine();
}
```

**شرح الكود السابق :**

```
QFile File("test.txt");
File.open(QIODevice::ReadWrite | QIODevice::Text);
```

الإعلان عن المتغير **File** لإدارة الملف **test.txt**. وفتحه للقراءة و الكتابة.

```
QTextStream fstr( &File );
```

الإعلان عن المتغير **fstr** كمدخل ومخرج بيانات من وإلى المتغير **File**.

```
fstr << "Hi Qt Developer !!\n";
```

إدخال **!! Hi Qt Developer** كبيانات نصية للملف.

```
fstr << 65;
```

إدخال **65** كبيانات نصية للملف.

```
fstr.seek( 0 );
```

وضع مؤشر البيانات عند **0** أى عند بداية الملف.

```
qDebug() << fstr.readLine();
qDebug() << fstr.readLine();
```

إستخراج البيانات بواسطة **readLine** و طباعتها على الشاشة.

..............................................................................

# QDataStream Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة تقوم بقراءة و كتابة البيانات، وتقوم بهذا العمل على QIODevice أو QByteArray، وسوف نرى فى المثال التالى كيفية تعامل فصيلة QDataStream مع فصيلة QFile التى ترث فصيلة QIODevice .

**طريقة الإعــلان ( Declaration ) :**

```
QDataStream mystream(QIODevice);
QDataStream mystream(QByteArray);
```

**وظائف الفصيلة :**

يتم التعامل على البيانات بالعاملين (<<) ، (>>) المختصين بإدخال و إخراج البيانات.

**QDataStream Example**

EXAMPLE NO 19

```
#include <QtCore>
int main()
{
   QFile File("test.txt");
   File.open(QIODevice::ReadWrite);

   QDataStream fdata(&File);

   fdata << QString("Hi Qt Developer !!");
   fdata << 65;

   fdata.device()->seek( 0 );

   QString str;
   int a;
   fdata >> str  >> a ;

   qDebug() << str << a ;

   File.close();
}
```

**شرح الكود السابق :**

```
QFile File("test.txt");
File.open(QIODevice::ReadWrite | QIODevice::Text);
```

الإعلان عن المتغير **File** لإدارة الملف **test.txt**. وفتحه للقراءة و الكتابة.

```
QDataStream fdata(&File);
```

الإعلان عن المتغير **fdata** كمدخل ومخرج بيانات من المتحكم **File**.

```
fdata << QString("Hi Qt Developer !!");
```

إدخال **Hi Qt Developer !!** كبيانات للملف.

```
fstr << 65;
```

إدخال **65** كبيانات للملف.

```
fdata.device()->seek( 0 );
```

وضع مؤشر البيانات عند **0** أى عند بداية الملف.حيث **fdata.device()** تعود بالقيادة لفصيلة **QFile** التى تحوى الدالة **seek** .

```
QString str;
int a;
fdata >> str  >> a ;
qDebug() << str << a ;
```

إستخراج البيانات بواسطة العامل (<<) و طباعتهـا على الشاشــــة، ونلاحظ خروج البيانـــات بنفس نوعهـا فالنصى يخرج كنص و الرقمى يخرج كرقم.

..............................................................................

# QVariant Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة يمكنها تحويل المتغير من أى نوع إلى آخر، كالتحويل من حرفى إلى عددى ومن أعداد صحيحة إلى أعداد حقيقية.

**طريقة الإعــلان ( Declaration ) :**

QVariant myvar(any data type);

**وظائف الفصيلة :**

تحتوى الفصيلة على جميع الدوال لتحويل البيانات من نوع لآخر.

**QVariant Example**

EXAMPLE NO 20

```
#include <QtCore>
int main()
{
  QVariant v( 100 );
  int x =  v.toInt() + 10;
  QString str = v.toString();
  double z =  v.toDouble() * 0.2234;

  qDebug() << v.typeName() ;

  qDebug() << x ;
  qDebug() << str;
  qDebug() << z ;

  QVariant v1( "100" );
  x = v1.toInt() + 10;
  str = v1.toString();
  z = v1.toDouble() * 0.2234;

  qDebug() << v1.typeName() ;

  qDebug() << x ;
  qDebug() << str;
  qDebug() << z;
}
```

**شرح الكود السابق :**

```
QVariant v( 100 );
```

الإعلان عن المتغير **v** من النوع QVariant وشحنه بالقيمة **100**.

```
int x = v.toInt() + 10;
```

الإعلان عن المتغير **x** من النوع **int** و يساوى **110**، حيث تم تحويل **v** إلى **int**.

```
QString str = v.toString();
```

الإعلان عن المتغير **str** من النوع **QString** و يساوى **"100"**، حيث تم تحويل **v** إلى **QString**.

```
double z = v.toDouble() * 0.2234;
```

الإعلان عن المتغير **z** من النوع **double** و يساوى **22.34** ، حيث تم تحويل **v** إلى **Double**.

```
qDebug() << v.typeName() ;
```

يتم طباعة نوع المتغير **v** وتكون **int** نظراً لشحنه بقيمة عددية عند الإعلان عنه فى أول الكود.

```
QVariant v1( "100" );
```

الإعلان عن المتغير **v1** وشحنه بالقيمة "100".

```
x = v1.toInt() + 10;
str = v1.toString();
z = v1.toDouble() * 0.2234;
```

هذا الكود يعطى نفس النتائج السابقة.

```
qDebug() << v1.typeName() ;
```

يتم طباعة نوع المتغير **v** وتكون **QString** نظراً لشحنه بقيمة نصية عند الإعلان عنه.

..........................................................................

والجدير بالذكر أن وحدة الفصائل الأساسية غير الرسومية ( QtCore Module ) تحتوى على أكثر من 150 فصيلة، أما ما تم شرحه سابقاً هى فقط أهم فصائل هذه الوحدة.

وسوف نتعرض الآن إلى فصيلة QObject والتى تعتبر أهم فصيلة فى ***كيوت***، وهى ما تعطى لكل فصائل ***كيوت*** طابعها و مميزاتها الخاصة.

Notes

# QObject Class

# QObject Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى أهم فصيلة فى ***كيوت***، حيث أنها هى الفصيلة الرئيسية المكونة لمعظم فصائل ***كيوت***، ولذلك تحتاج هذه الفصيلة الكثير من الإهتمام.

وتمتلك هذه الفصيلة عدة خصائص خاصة تميز الفصيلة التى ترثها، وسيتم شرح هذه المميزات شرحاً مفصلاً فيما بعد، حيث أن فهم وإستيعاب هذه الفصيلة يمثل ما يوازى ٪70 من قوة ***كيوت***.

تنظم هذ الفصيلة نفسها على شكل شجرى حيث أن:

- لكل فرع أصل واحد ( parent ).
- والأصل يمكن أن يكون له عدة فروع ( children ).
- وأى فرع ( child ) من هذه الفروع يمكن أن يكون أصل ( parent ) لفروع أخرى.
- وإذا تم مسح الأصل يتم إزالة كل ما يتبعه من فروع.

والملاحظ أن كل كائن ( object ) يرث الفصيلة ( QObject ) يصبح له اسم ويمكننا معرفة أصله و فروعه واسم الفصيلة التى يتبعها.

يمكن أيضاً للفصيلة الوارثة لفصيلة ( QObject ) أن تستقبل أحداث ( events ) مثل الوقت ( timer ) وأى من المدخلات مثل أحداث لوحة المفاتيح ( keyboard Events ).

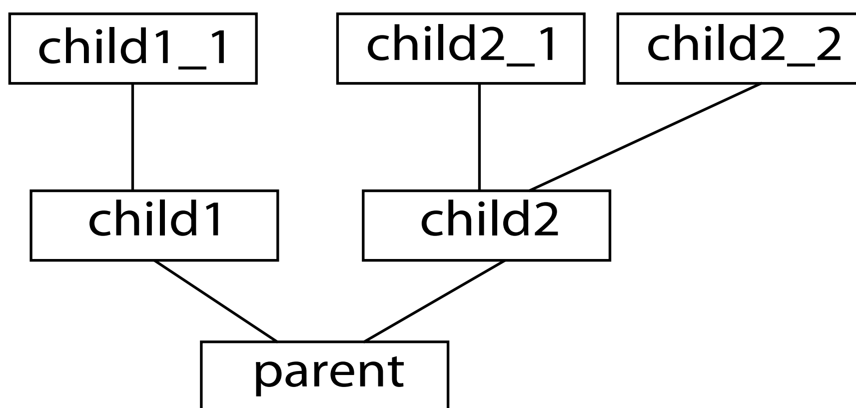
تستخدم فصيلة ( QObject ) ميكانيكة ( Signals & Slots ) للربط بين دوال الفصائل وبعضها وهى من أهم خصائص هذه الفصيلة.

تأتى ( QObject ) ببعض الماكرو المستخدمة فى تكوين الفصيلة مثل Q_CLASSINFO .Q_SIGNALS , Q_SLOTS , Q_PROPERTY , Q_OBJECT

جميع الفصائل الوارثة لفصيلة ( QObject ) تتعامل مع خاصية الترجمة التى تقدمها ***كيوت*** والتى تسمح بسهولة ترجمة واجهة الإستخدام إلى أية لغة.

تنظم فصيلة ( QObject ) نفسها على شكل شجرى، حيث أن كل فصيلة أب ( parent ) يمكن أن يتفرع منها أكثر من فصيلة ابن ( child ) فتصبح بالشكل التالى:

```
QObject *parent = new QObject();
QObject *child1 = new QObject(parent);
QObject *child2 = new QObject(parent);
QObject *child1_1 = new QObject(child1);
QObject *child2_1 = new QObject(child2);
QObject *child2_2 = new QObject(child2);
```




ونلاحظ هنا التسلسل الشجرى حيث أن :

parent هو الأب ( الأصل ) للأبناء ( الفروع ) child1 , child2 .

child1 هو الأب ( الأصل ) للابن ( الفرع ) child1_1.

child2 هو الأب ( الأصل ) للأبناء ( الفروع ) child2_1 , child2_2.

وكما نعلم أنه فى حالة مسح أى QObject يتم مسح كل الأبناء ( الفروع ) التى تليه.

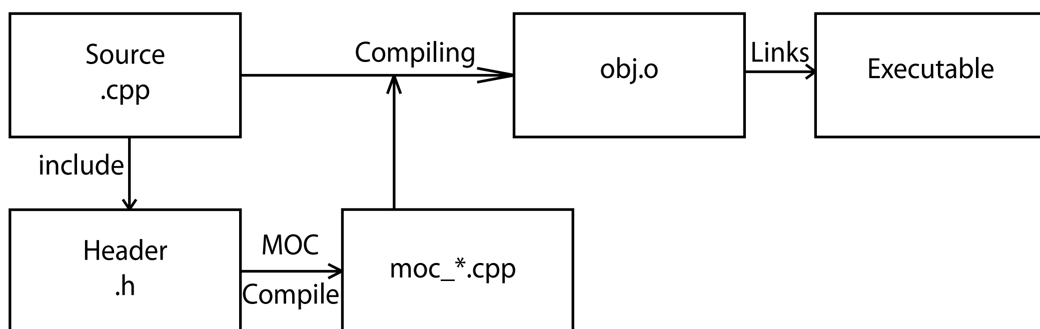
فبالتالى إذا تم مسح child2 يتم تلقائياً مسح ما يتبعه من فروع وهم child2_1 , child2_2.

..................................................................................

وضعت **كيوت** عدة قواعد لأى فصيلة ترث QObject، حيث أن كل فصيلة ترث فصيلة ( QObject ) لها تعاريف وبيانات خاصة بها ( meta object ) وهذه التعاريف تشمل الآتى:

| | |
|---|---|
| - اسم الفصيلة | Class Name |
| - الوراثة | Inheritance |
| - خصائص الفصيلة | Properties |
| - دوال الإرسال والإستقبال | Signals & Slots |
| - معلومات عامة | Class Info |

ويتم وضع هذه البيانات داخل تعريف الفصيلة كماكرو ( Macro )، ونظراً لأن مترجم سي++ لا يمكنه التعامل مع الماكرو مباشرة، تستخدم **كيوت** مترجم ( Compiler ) إضافى بجانب مترجم سى++ لترجمة الوظائف الخاصة **بكيوت**.




هذا المترجم الإضافى ( moc ) Meta Object Compiler يقوم بالبحث فى ملفات (headers) عن الفصائل الوارثة لفصيلة ( QObject ).ثم يقوم بتوليد كود للماكرو الخاصة ( slots , signals ، etc ) ويضع الكود فى ملف moc_classname.cpp حتى يمكن ترجمته بمترجم سى++.

و تتم هذه العملية كالآتى :

يبدأ moc بالبحث فى الملفات (headers) عن الماكرو Q_OBJECT.

- فإذا لم يحتوى تعريف الفصيلة على الماكرو Q_OBJECT فإن المترجم moc لن ينظر إليها، وبالتالى لن يولد الكود الخاص (moc_classname.cpp)، وعند ترجمة التطبيق سنحصل على خطأ بالترجمة.

- أما إذا وجد الماكرو Q_OBJECT سيبدأ بالبحث عن باقى الماكرو مثل Q_PROPERTY , Q_SIGNALS , Q_SLOTS لتوليد الأكواد الخاصة بها.

ولكى نستطيع فهم هذه الفصيلة سنبدأ بشرح كيفية تكوين فصيلة ترث QObject.

## تكوين فصيلة ترث QObject :

| | |
|---|---|
| class className : public QObject | <----------- QObject اسم الفصيلة وإعلان ميراثها لـ |
| { | |
| Q_OBJECT | <--Q_OBJECT ماكرو |
| Q_CLASSINFO(....) | <-- Q_CLASSINFO ماكرو |
| Q_PROPERTY(.....) | <-- Q_PROPERTY ماكرو |
| public: | <-- منطقة المتغيرات و الدوال العامة |
| constructor(QObject *parent, ...); | <-- المنشئ و هو ينشئ الفصيلة فى الذاكرة |
| setter functions(); | <-- الدوال اللازمة لوضع قيم المتغيرات الموجودة فى المنطقة الخاصة |
| getter functions(); | <-- الدوال اللازمة لجلب قيم المتغيرات الموجودة فى المنطقة الخاصة |
| | |
| private: | <-- منطقة المتغيرات و الدوال الخاصة |
| variables; | <-- توضع معظم المتغيرات فى المنطقة الخاصة ويتم التعامل معها بدوال setter و getter. |
| ..... | |
| | |
| protected: | <-- منطقة المتغيرات و الدوال المحمية |
| ... | |
| Q_SLOTS: | <-- Q_SLOTS ماكرو |
| slots functions; | <-- منطقة الدوال المستقبلة للاشارة |
| | |
| Q_SIGNALS: | <-- Q_SIGNALS ماكرو |
| signals function; | <-- منطقة الدوال المرسلة للاشارة |
| } | |

وكما نرى فى أجزاء الفصيلة تم إضافة أكثر من ماكرو، وسوف نقوم بتوضيح وظيفة كل ماكرو وكيفية كتابته فى الجدول التالى.

### Q_OBJECT

هو الماكرو المسؤول عن إخبار ( meta object compiler ) بأن هذه الفصيلة ترث فصيلة QObject وبالتالى فهى بحاجة إلى ترجمة بواسطة ( moc ).
ويتم كتابته فى أول الفصيلة دون أى علامات إضافية مثل ( ; )

### Q_CLASSINFO

هو ماكرو لإضافة معلومات نصية عن الفصيلة، وكمثال يمكن إضافة اسم المبرمج أو موضوع الفصيلة، ويتم كتابته كالتالى:

```
Q_CLASSINFO("Programmer" , " Ali")
Q_CLASSINFO("about" , " drawing")
```

### Q_PROPERTY

كإتجاه سائد فى **كيوت** لا يتم التعامل مباشرة مع متغيرات الفصيلة، ولكن يتم من خلال دالتين:
واحدة لوضع قيمة ( Setter Function ) والأخرى لجلب قيمة ( Getter Function ).
Q_PROPERTY هو ماكرو لإضافة خاصية للفصيلة تبنى على الدوال Setter , Getter.
وتكتب بالطريقة التالية :

Q_PROPERTY( **Var_datatype** *propertyName* **READ** *Getter* **WRITE** *Setter* )

```
class classname : QObject
{
Q_OBJECT
Q_PROPERTY(int num_pro READ number WRITE setNumber)
public:
int number()
void setNumber(int);
private:
int num;
};
```

ومن ثم يمكن إستخدامها بواسطة الدالة setProperty( *propertyName , Variable Value* )

### Q_SLOTS & Q_SIGNALS

يتم كتابة الماكرو مثل public , private , protected داخل تعريف الفصيلة.

ونذكر أن الماكرو Q_OBJECT يجب وجوده فى تعريف الفصيلة، أما بالنسبة لأى ماكرو آخر فيتم إستخدامه حسب الحاجة إليه .

## ميكانيكية Q_SIGNALS and Q_SLOTS

هذه الميكانيكية شائعة الإستخدام فى تصميم الواجهات الرسومية، وهى طريقة للربط ما بين دوال الفصائل المختلفة، ويتم الربط بين هذه الدوال كالتالى :

بين دالة إرسال ( Signal Function ) ودالة إستقبال ( Slot Function ).

أو بين دالة إرسال ( Signal Function ) ودالة إرسال ( Signal Function ).

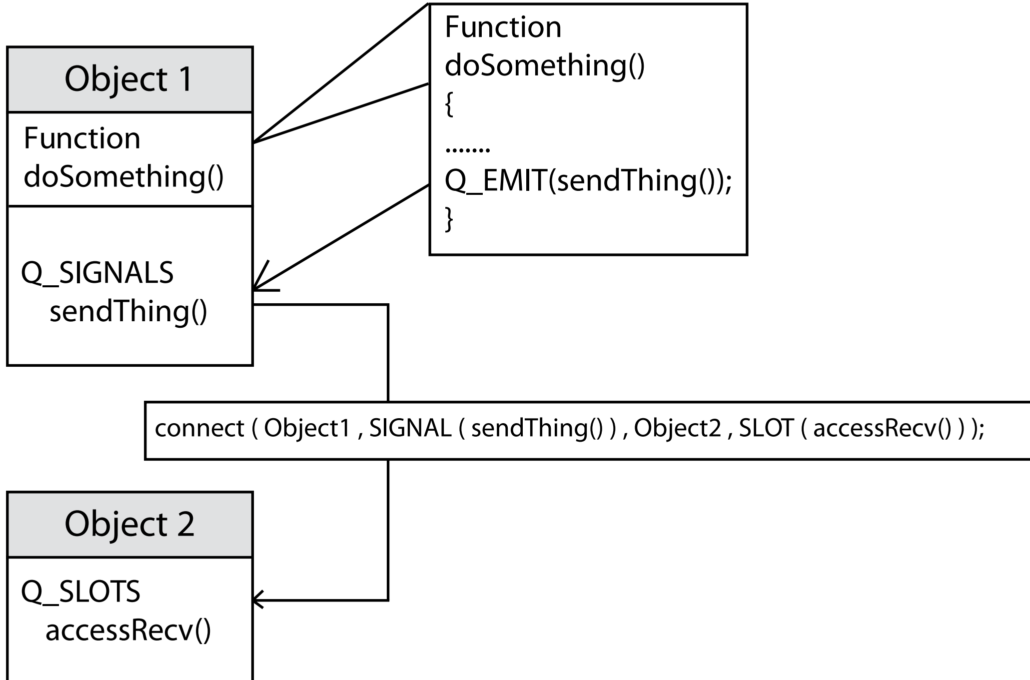
**دالة إرسال الإشارة :** و هى دالة يعلن عنها داخل تعريف الفصيلة فقط وليس لها كود، وتعلن هذه الدالة داخل النطاق Q_SIGNALS.

- ويتم إشعال دالة إرسال الإشارة بواسطة الماكرو ( Q_EMIT ( Signal Function.

**دالة إستقبال الإشارة :** و هى دالة عادية ولكن يعلن عنها داخل النطاق Q_SLOTS.

**يتم الربط بواسطة الأمر :**

connect( obj1 , SIGNAL(signal function) , obj2 , SLOT(slot function));




**مثـــــال :** إذا فرضنا وجود فصيلة Container تقوم بتسجيل الأسماء، وهناك فصيلة Watcher تقوم بمتابعة عمل الفصيلة الأولى من إضافة عناصر جديدة، فيتم إنشائها فى هذه الحالة كالتالى.

إنشاء فصيلة container ترث QObject.

EXAMPLE NO 21

**container.h**

```
#include <QObject>
#include <QStringList>
class container : public QObject           <---   QObject ترث container
{
   Q_OBJECT                                <---   Q_OBJECT    الماكرو
public:
   explicit container(QObject *parent = 0); <---  دالة إنشاء الفصيلة
   void addItem(QString);                  <---   strlist  دالة إدخال اسم للقائمة

Q_SIGNALS:                                 <---   Q_SIGNALS    الماكرو
  void ItemAdded(QString);                 <---   دالة لإرسال إشارة قيمتها من النوع QString
                                                  دالة الإرسال ليس لها كود
public Q_SLOTS:                            <---   Q_SLOTS    الماكرو
                                           <---   لايوجد دوال مستقبلة للإشارة
private:
   QStringList strlist;                    <---   المتغير strlist الذى يحوى الأسماء

};
```

**container.cpp**

```
#include "container.h"

container::container(QObject *parent) :
   QObject(parent)
{                                          <---   كود دالة الإنشاء
}

void container::addItem(QString str)       <---   strlist  كود دالة إدخال اسم للقائمة
{
   this->strlist << str;                   <---   إدخال القيمة str للقائمة strlist
   Q_EMIT(ItemAdded(str));                 <---   Q_EMIT    الماكرو
}
```

سيقوم الماكرو Q_EMIT بإشعال دالة الإرسال ItemAdded لتبث إشارة قيمتها str، وتتم هذه العملية فى كل مرة تستخدم الدالة addItem.

ونلاحظ هنا أنه لا يوجد كود لدالة الإرسال ItemAdded، فهى تعلن داخل تعريف الفصيلة فقط.

إنشاء فصيلة watcher ترث QObject.

watcher.h

```
#include <QObject>

class watcher : public QObject
{
   Q_OBJECT
public:
   explicit watcher(QObject *parent = 0);

Q_SIGNALS:

public Q_SLOTS:                              <---      Q_SLOTS       الماكرو
   void printstr(QString);                   <---      دالة لإستقبال الاشارة
};
```

يمكن إستخدام الدالة printstr كأى دالة أخرى، ولكن وضعها فى نطاق الماكرو Q_SLOTS يعطيها إمكانية الإستجابة لإشارات دوال الإرسال ( signal function ) وذلك عند ربطهما بأمر connect.

watcher.cpp

```
#include "watcher.h"
#include <QDebug>

watcher::watcher(QObject *parent) :
   QObject(parent)
{
}

void watcher::printstr(QString str)
{
   qDebug() << "new Item was added to Container : " << str;
}
```

يتم كتابة الدالة printstr بطريقة عادية، ويمكن إستخدامها بإحدى الطريقتين :

الأولـى : إستدعائها بالطريقة العادية بتمرير المتغير str إلى الدالة printstr .

الثانيـة : التعامل معها كدالة إستقبال، وتلقيها إشارة قيمتها من نفس النوع QString من دالة إرسال.

إنشاء فصيلة watcher ترث QObject.

main.cpp

```
#include <QtCore/QCoreApplication>
#include “container.h“
#include “watcher.h“

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    container *cont = new container;          <---    إنشاء دالة container تسمى cont
    watcher *watch = new watcher;             <---    إنشاء دالة watcher تسمى watch
    QObject::connect(cont , SIGNAL(ItemAdded(QString)) , watch , SLOT(printstr(QString)));

    cont->addItem(“Mohamed 1“);
    cont->addItem(“Mohamed 2“);
    cont->addItem(“Mohamed 3“);
    return a.exec();
}
```

ماذا يحدث ؟

connect( cont , SIGNAL( ItemAdded(QString) ) , watch , SLOT( printstr(QString) ) );

ربط دالة الإرسال **ItemAdded** من الفصيلة **cont**

بدالة الإستقبال **printstr** من الفصيلة **watch.**

سيحدث الآتى :

- كلما إستدعيت الدالة **addItem** من الفصيلة **cont** ستقوم بإدخال الاسم إلى القائمة **strlist.**
- ثم يتم تشغيل دالة الإرسال **ItemAdded** بواسطة **Q_EMIT** لتبث إشارة قيمتها الاسم الجديد الذى أدخل للقائمة.
- وبواسطة الأمر **connect** فإن الدالة **printstr** من الفصيلة **watch** ستقوم بإستقبال الإشارة وتأخذ قيمتها لتجرى العمليات عليها ( وهى طباعتها ).

فيكون ناتج التطبيق السابق:

```
new Item was added to Container :  “Mohamed 1“
new Item was added to Container :  “Mohamed 2“
new Item was added to Container :  “Mohamed 3“
```

### قواعد تحكم دوال الإرسال و الإستقبال signals and slots:

يمكن فهم هذه القواعد بسهولة، حيث الهدف هنا هو ربط دالة إرسال ( signal ) بدالة إستقبال ( slot )، فدالة الإرسال تبث بيانات( عدد و نوع )، ودالة الإستقبال تتلقى بيانات ( عدد ونوع )، لذلك يتوقف الربط الصحيح على عدد و نوع البيانات التى تبث و تستقبل.

على سبيل المثال:

دالة إرسال تبث عدد 1 بيان من النوع int <--- signal function : setdata(int)

دالة إستقبال تستقبل عدد 1 بيان من النوع int <--- slot function : execdata(int)

وبالتالى يمكن ربطهما، حيث أن بيانات الدالتين متكافئتين من حيث النوع و العدد.

### حالات الربط بين دوال الإرسال و دوال الإستقبال :

الجدير بالذكر أن الربط الصحيح للدوال يتوقف على دالة الإستقبال.

الحالة الأولـى : يمكن الربط بينهما إذا تكافأت البيانات تماماً كما فى المثال السابق.

الحالة الثانية : إذا كانت دالة الإرسال تبث عدد بيانات أكثر فتأخذ دالة الإستقبال ما تريد وتترك الباقى، ولكن لا يمكن حدوث العكس، وإليكم بعض الأمثلة.

| Signals | Slots | connect |
|---|---|---|
| setdata(int) | execdata(int) | مسموح |
| setdata(int,QString) | execdata(QString,int) | غير مسموح |
| setdata(int,QString) | execdata(int,QString) | مسموح |
| setdata() | execdata() | مسموح |
| setdata(int) | execdata() | مسموح |
| setdata(int,int) | setdata(int) | مسموح |
| setdata(int) | setdata(int,int) | غير مسموح |
| setdata() | execdata(int) | غير مسموح |

وسنعطى الآن مثال على فصيلة كاملة ترث فصيلة QObject.

مثـــال : نريد إنشاء فصيلة للسيارات تحتوى على كلاً من اسم السيارة ، الموديل ، أقصى سرعة للسيارة ، إمكانية تشغيل الموتور، تحديد سرعة حركة السيارة.

وبالتالى معرفة وضع السيارة فى حالة تشغيل الموتور أو إيقافه، وأيضاً معرفة السرعة الحالية.

وسوف يساعدنا ملئ النموذج التالى قبل البدء فى كتابة الكود.

| ClasName | *car* | Inherits | *QObject* |
|---|---|---|---|

## Macros , Constructors and Destructors

| Level | Type | Function member |
|---|---|---|
| *NON* | *Macro* | *Q_OBJECT* |
| *NON* | *Macro* | *Q_CLASSINFO("Author" , "QT-Developer")* |
| *NON* | *Macro* | *Q_PROPERTY(QString carname READ name WRITE setName);* |
| *NON* | *Macro* | *Q_PROPERTY(QString carmodel READ model WRITE setModel)* |
| *NON* | *Macro* | *Q_PROPERTY(int carspeed READ speed WRITE setSpeed)* |
| *public* | *Contructor* | *explicit car(QObject *parent = 0);* |
| *public* | *Contructor* | *explicit car(QString name, QObject *parent = 0);* |

## Variables

| Level | Type | Name | Info |
|---|---|---|---|
| *private* | *QString* | *car_name;* | |
| *private* | *QString* | *car_model;* | |
| *private* | *int* | *car_speed;* | |
| *private* | *int* | *car_maxspeed;* | |
| *private* | *bool* | *car_enginestatus;* | |

## Setters and Getter Function members

| Setters Functions | | | Getters Functions | | |
|---|---|---|---|---|---|
| **Level** | **Type** | **Function** | **Level** | **Type** | **Function** |
| *public* | *void* | *setName(QString);* | *public* | *QString* | *name();* |
| *public* | *void* | *setModel(QString);* | *public* | *QString* | *model();* |
| *public* | *void* | *setMaxSpeed(int);* | *public* | *int* | *speed();* |
| *public* | *void* | *StartEngine();* | *public* | *int* | *maxSpeed();* |
| *public* | *void* | *StopEngine();* | *public* | *bool* | *isEngineOn();* |
| | | | *public* | *QString* | *EngineStatus();* |

**Slots and Signals**

| Q_SLOTS | | | Q_SIGNALS | | |
|---|---|---|---|---|---|
| **Level** | **Type** | **Function** | **Level** | **Type** | **Function** |
| *Q_SLOTS* | *void* | *setSpeed(int);* | *Q_SIGNALS* | *void* | *SpeedChanged(int);* |
| | | | *Q_SIGNALS* | *void* | *Enginetoggled(bool);* |
| | | | *Q_SIGNALS* | *void* | *MaxSpeed(int);* |

عند تفريغ النموذج السابق ستصبح الفصيلة بالشكل التالى( car.h ):

```
#include <QObject>
class car : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("Author" , "QT-Developer")
    Q_PROPERTY(QString carname READ name WRITE setName)
    Q_PROPERTY(QString carmodel READ model WRITE setModel)
    Q_PROPERTY(int carspeed READ speed WRITE setSpeed)
public:
    explicit car(QObject *parent = 0);
    explicit car(QString name, QObject *parent = 0);
    void setName(QString);
    void setModel(QString);
    void setMaxSpeed(int);
    void StartEngine();
    void StopEngine();
    QString name();
    QString model();
    int  speed();
    int  maxSpeed();
    bool    isEngineOn();
    QString EngineStatus();
Q_SIGNALS:
    void SpeedChanged(int);
    void Enginetoggled(bool);
    void MaxSpeed(int);
public Q_SLOTS:
    void setSpeed(int);
private:
    QString car_name;
    QString car_model;
    int     car_maxspeed;
    int     car_speed;
    bool    car_enginestatus;
};
```

الماكرو

تعريف منشئ الفصيلة

الدوال اللازمة لوضع قيم المتغيرات Setters

الدوال اللازمة لجلب قيم المتغيرات Getters

دوال الإرسال

دوال إستقبال للإشارة

المتغيرات الخاصة بالفصيلة

**شرح الكود السابق :**

سوف نقوم بشرح كيفية كتابة الماكرو وكيفية الإستفادة منها.

Q_OBJECT

**Q_OBJECT** ماكرو ليستدل المترجم **moc** على أن الفصيلة ترث فصيلة **QObject**.

Q_CLASSINFO

هو ماكرو يوضع فيه المعلومات الأساسية عن الفصيلة مثل:

Q_CLASSINFO("Author" , "QT-Developer")

Q_CLASSINFO("Web" , "www.carclass.net")

Q_PROPERTY

ماكرو يوضع به نوع المتغير( Variable Type ) ودالة جلب القيمة ( Getter Function ) و دالة وضع القيمة ( Setter Function ).

وتأخذ الشكل التالى:

Q_PROPERTY(DataType PropertyName READ GetterFunction WRITE SetterFunction)

سنشرح فيما بعد أهمية Q_PROPERTY .

وتطبيقاً فى المثال السابق نرى :

Q_PROPERTY(QString carname READ name WRITE setName)

Q_PROPERTY(QString carmodel READ model WRITE setModel)

Q_PROPERTY(int carspeed READ speed WRITE setSpeed)

Q_SIGNALS

جميع الدوال تحت الماكرو **Q_SIGNALS** تكون من النوع **void**، وهذه الدوال لا يكتب لها تفصيل فى ملف الكود **classname.cpp** .

وهى دوال خاملة، فقط هى تبث إشارة إذا تم إشعالها بواسطة الماكرو **Q_EMIT** .

Q_SLOTS

جميع الدوال تحت الماكرو **Q_SLOTS** هى دوال عادية يمكن إستخدامها كأى دالة أخرى، ولكن ما يميزها أنها تستطيع إستقبال إشارات دوال الماكرو **Q_SIGNALS**.

.................................................................

الكود التفصيلى ( car.cpp ) للفصيلة car

```
#include "car.h"

car::car(QObject *parent) :
    QObject(parent)
{
    this->car_speed = 0;
    this->car_enginestatus = false;
}
```

كود دالة المنشئ

```
car::car(QString name, QObject *parent):QObject(parent)
{
    this->car_speed = 0;
    this->car_enginestatus = false;
    this->car_name = name;
}
```

كود دالة المنشئ

```
void car::setName(QString name)
{
    this->car_name = name;
}
```

دالة Setter : لإعطاء اسم للسيارة

```
void car::setModel(QString model)
{
    this->car_model = model;
}
```

دالة Setter : لإعطاء موديل السيارة

```
void car::setMaxSpeed(int max)
{
    this->car_maxspeed = max;
    Q_EMIT MaxSpeed(max);
}
```

دالة Setter : لتحديد أقصى سرعة

نلاحظ وجود الماكرو Q_EMIT لتشعل دالة الإشارة MaxSpeed ------------>
لتبث إشارتها بالقيمة max

```
void car::setSpeed(int speed)
{
    if(speed < 0)speed = 0;
    if(speed > this->car_maxspeed) speed = this->car_maxspeed;
    if(this->car_speed == speed)return;
    if(!this->isEngineOn())speed=0;
    this->car_speed = speed;
    Q_EMIT SpeedChanged(speed);
}
```

دالة Setter : لوضع السرعة الحالية

Q_EMIT

<---------------------------- لتشعل دالة الإرسال SpeedChanged

لتبث إشارتها بالقيمة speed

```
void car::StartEngine()
{
   if(this->car_enginestatus == true) return;
       this->car_enginestatus = true;
       Q_EMIT Enginetoggled(this->car_enginestatus);
}
```

دالة Setter : لتشغيل المحرك

Q_EMIT

<------ تشعل دالة الإرسال Enginetoggled

لتبث إشارتها بالقيمة car_enginestatus

```
void car::StopEngine()
{
    if(this->car_enginestatus == false) return;
    else{
        this->setSpeed(0);
        this->car_enginestatus = false;
        Q_EMIT Enginetoggled(this->car_enginestatus);
    }
}
```

دالة Setter : لإيقاف المحرك

Q_EMIT

<------ تشعل دالة الإرسال Enginetoggled

لتبث إشارتها بالقيمة car_enginestatus

```
QString car::name()
{
    return this->car_name;
}
```

دالة Getter : لإرجاع اسم السيارة

```
QString car::model()
{
    return this->car_model;
}
```

دالة Getter : لإرجاع موديل السيارة

```
int  car::maxSpeed()
{
   return this->car_maxspeed;
}
```

دالة Getter : لإرجاع قيمة أقصى سرعة

```
int  car::speed()
{
   return this->car_speed;
}
```

دالة Getter : لإرجاع قيمة السرعة الحالية

```
bool    car::isEngineOn()
{
   return this->car_enginestatus;
}
```

دالة Getter : لإرجاع حالة المحرك    bool : true , false

```
QString car::EngineStatus()
{
   QString status;
   if(this->isEngineOn()) status = "Engine On";
   else status = "Engine Off";
   return status;
}
```

دالة Getter : لإرجاع حالة المحرك كنص QString

لقد رأينا طريقة **signals and slots** فى الربط مابين دوال الفصائل الوارثة لفصيلة **QObject**، ويوجد أيضاً طريقة أخرى للربط ما بين دوال الفصائل، وهى عن طريق وضع دالة إستقبال باسم يحتوى على دالة الإرسال واسم فصيلتها، وتوضع هذه الدالة داخل النطاق **slots** أو **Q_SLOTS**.

وتقوم هذه الدالة بإستقبال البيانات التى تبثها دالة الإرسال المكتوبة فى اسم الدالة كالآتى:

اسم الفصيلة    دالة الإرسال

```
void on_<object name>_<signal name>(<signal parameters>);
EX : void on_PushBotton_clicked();
```

وتعرف هذه الطريقة بـ **connectSlotsByName**، وسنقوم بشرحها بصورة أفضل عند التعامل مع الواجهات الرسومية.

# QtGui Module

# وحدة

# مكونات واجهة

# المستخدم الرسومية

## متطلبات هذه الوحدة

**إدراج**

```
#include <QtGui>
```

**داخل ملفات الكود**

**إدراج**

```
QT += gui
```

**داخل ملف المشروع** project**.pro** .

قبل البدأ بدراسة فصائل الواجهات الرسومية GUI، من الممكن أن نثير بعض الأسئلة: ماهى مكونات الواجهة الرسومية؟ وما هى خصائص تلك المكونات؟ وهل هذه المكونات سابقة التجهيز أم يتم برمجتها؟

مكونات الواجهة الرسومية : هى على سبيل المثال (Buttons , Line Edit , Radio Buttons )، بجانب كل ما هو رسم على الشاشة و يتفاعل مع أحداث ( Events )، مثل الضغط على الفأرة أو أحد مفاتيح لوحة المفاتيح.

وبالتالى يمكن القول أن جميع مكونات الواجهة الرسومية ينحدر من فصيلة واحدة ألا وهى فصيلة **QWidget**، والتى تعتبر البنية الأساسية لمكونات الواجهة الرسومية، وهى عبارة عن مساحة مستطيلة ترسم على الشاشة وتتفاعل مع الأحداث ( Events ).

فصيلة **QWidget** هى فصيلة ترث فصيلتين( QObject , QPaintDevice )، فهى ترث كل خصائص و مميزات QObject، كما رأينا سابقاً و ترث أيضاً خصائص QPaintDevice الرسومية.

وبالتالى فعند دراسة خصائص **QWidget** جيداً، نكون قد قمنا بتعريف الخصائص العامة لجميع مكونات الواجهة الرسومية، و كيفية التحكم بها.

..........................................................................

# QWidget Class

**نسب الفصيلة :**

ترث فصيلة QPaintDevice , QObject

**تعريف الفصيلة :**

تعتبر هذه الفصيلة هى أهم فصيلة فى وحدة مكونات واجهة المستخدم الرسومية ( Gui module )، حيث أنها هى الفصيلة الرئيسية المكونة لكل فصائل ***كيوت*** الرسومية، وهى عبارة عن مساحة مستطيلة ترسم على الشاشة وتتفاعل مع الأحداث ( Events ).

**ملحوظة :**

عند تعريف الفصيلة QWidget سوف نلاحظ قيمة لابد من تمريرها فى بداية التعريف.

مثال على ذلك :

دالة الإنشاء QWidget( QWidget *parent = 0, Qt::WindowFlags f = 0 )

عند الإعلان عن أى فصيلة QWidget يتم شحنها بقيمتين للمتغيرين parent , f .

المتغير parent إذا لم يتم شحنه بقيمة، تصبح قيمته 0 كقيمة إفتراضية.

وفى حالة إذا كانت قيمة المتغير parent تساوى 0،

فإن الفصيلة QWidget تمثل نافذة مستقلة ( window ).

وإذا تم شحن المتغير parent بفصيلة widget كأب،

فإن الفصيلة QWidget تصبح بمثابة ابن للفصيلة الأب widget.

mywidget نافذة مستقلة --------> QWidget mywidget(0);

mywidget نافذة مستقلة --------> QWidget mywidget;

QWidget mywidget2( &mywidget );

mywidget2 هى كائن رسومى جديد يرسم بداخل mywidget، ويكون ابن لـ mywidget .

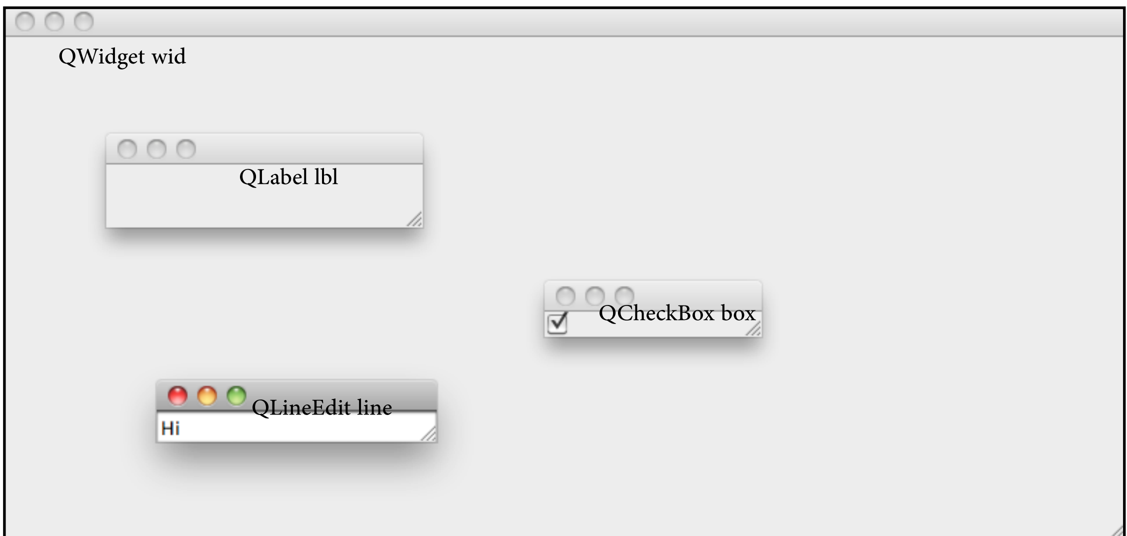
وكل ما يتم فعله على الأب يحدث لجميع أبناءه، والعكس ليس بصحيح، بمعنى أنه إذا ما تم إخفاء النافذة الأب parent يختفى جميع أبناءه children، وكذلك أيضاً فى حالة الإلغاء والتعطيل، وسيقوم المثال التالى بتوضيح ذلك بصورة أكبر.

main.cpp

EXAMPLE NO 22

```
#include <QtGui/QApplication>
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QCheckBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWidget wid;
    QLabel lbl;
    QLineEdit line;
    QCheckBox box;
    wid.show();
    lbl.show();
    line.show();
    box.show();
    return a.exec();
}
```

نتيجة هذا الكود : يتم عرض كل مكون رسومى كأنه نافذة مستقلة.



```
#include <QtGui/QApplication>
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QCheckBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWidget wid;
    QLabel lbl(&wid);
    QLineEdit line(&wid);
    QCheckBox box(&wid);
    wid.show();
    return a.exec();
}
```

فى هذا الكود يتم رسم كلاً من lbl,line,box بداخل wid، حيث أن wid هى النافذة الوحيدة والباقى يعتبر مكونات رسومية أبناء لـ wid.

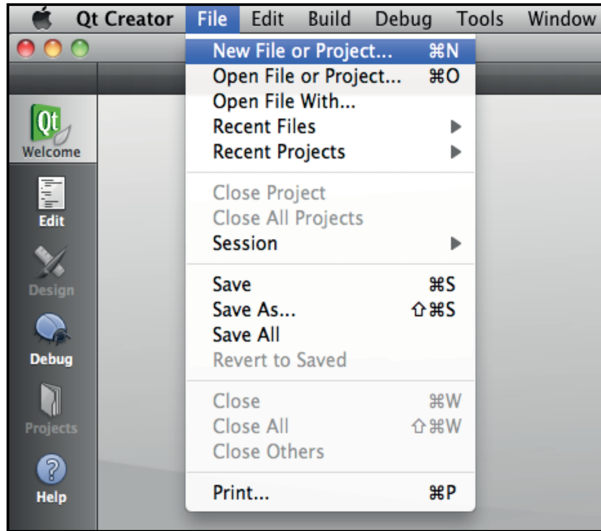
ولابد إذاً من شرح كيفية عمل ***كيوت*** فى الواجهات الرسومية، وما هى الأسس التى بنيت عليها. ونذكر أيضاً أنه عند التطبيق الفعلى سيتم إستخدام الأداة Qt Designer لرسم كل أجزاء التطبيق وتسميتها وإعطاءها الخواص الخاصة بها، كما سيتم الربط بين دوال الفصائل الرسومية من خلال نافذة signals and slots الموجودة بـ Qt Designer .

وبالنظر إلى فصيلة QWidget نجد أن لها كم هائل من الدوال التى تتحكم فى خصائصها الرسومية، فيوجد دوال لتحديد طول وعرض النافذة ومكانها فى إحداثيات الشاشة ( setGeometry )، ودوال لإظهار النافذة أو إخفائها ( show , hide ) بجانب دوال أخرى كثيرة.

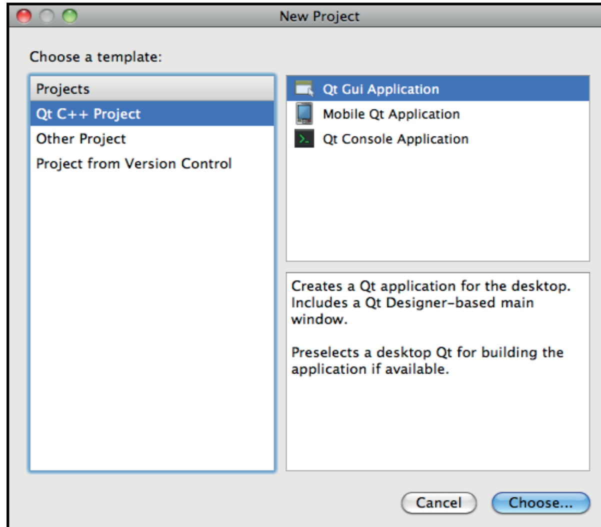
وسنبدأ الآن بإنشاء أول تطبيق ذو واجهة رسومية، وهذا التطبيق ينحصر فى إنشاء QWidget وعرضها على الشاشة، وذلك لمعرفة كيفية عمل الأداة Qt Creator فى تكوين الأكواد الخاصة بالتطبيق.

نبدأ أولاً بفتح Qt Creator.

- نختار New File or Project من القائمة File.

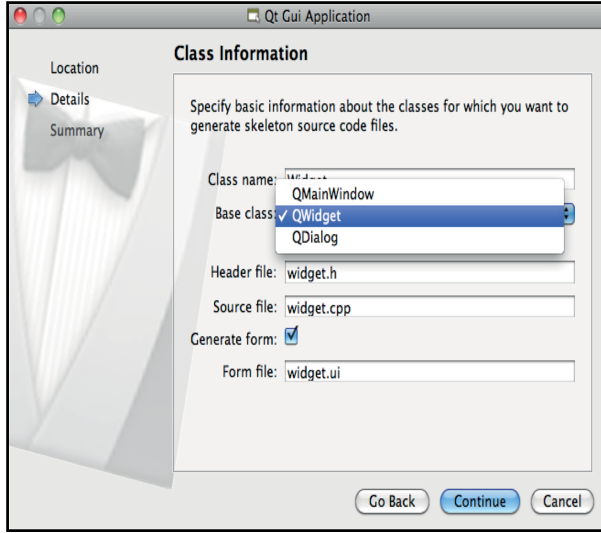



- و نختار Qt c++ Project و Qt Gui Application من القائمة Project.




- ثم تظهر نافذة لإعطاء اسم للتطبيق وسنقوم بتسمية التطبيق test.

- من الواجهة class information : نختار QWidget من قائمة base class .




- بمجرد الإنتهاء من هذه الخطوات وبضغط Runسيقوم التطبيق بإظهار نافذة خالية.

-المطلوب هنا هو معرفة ما قام به Qt Creator، وما هى الملفات والأكواد التى أنتجها تلقائياً.

**الملفات التى أنتجها Qt Creator هى:**

```
Project
 |---test.pro              <--- ملف لتعريف الخواص الخاصة بالتطبيق
 |---Forms                 <--- توضع هنا النماذج الرسومية
 |      |----widget.ui     <---Qt Designer نموذج نافذة يمكن تعديله بواسطة الأداة
 |
 |---Headers
 |      |----widget.h      <--- ملف تعريف الفصيلة
 |
 |---Source
        |----main.cpp      <--- ملف التطبيق الأساسى
        |----widget.cpp    <--- ملف كود تفصيلى للفصيلة
```

سيقوم Qt Creator بإنشاء مجلد ( Folder ) باسم التطبيق فى المكان الذى سبق إختياره أثناء إعداد التطبيق، ففى المثال السابق سيقوم بإنشاء مجلد اسمه test ويوضع فيه الملفات السابق ذكرها.

الملف test.pro :

هو ملف يحتوى على أسماء الوحدات ( Modules ) المطلوب التعامل معها مثل (gui , network )، كما يحتوى على أسماء الملفات الخاصة بالمشروع.

الملف widget.ui :

هو ملف توصيف النافذة و ما بها من كائنات رسومية.

الملف main.cpp :

هو ملف كود المشروع الرئيسى.

الملف widget.h :

هو ملف كود تعريف الفصيلة الرسومية widget.

الملف widget.cpp :

هو ملف كود تفصيل لتعريف الفصيلة الرسومية widget.

عند الضغط على build all يحدث الآتى :

- يتم ترجمة widget.ui إلى ملف كود اسمه ui_widget.h بواسطة المترجم uic.
ويقوم uic بإنتاج فصيلة مستقلة تحتوى على جميع الكائنات الرسومية الموجودة ما عدا النافذة الرئيسية، وتعرف هذه الفصيلة داخل namespace.

- يقوم المترجم moc بالبحث عن جميع الماكرو الخاص بـ QObject و إنتاج ملف كود اسمه moc_widget.cpp.

- يتم ترجمة الملفات السابقة لإنتاج ملف تشغيل التطبيق.

الملخص : تم إنتاج ملفين تلقائياً هما ( ui_widget.h , moc_widget.cpp ).

ولنرى الآن ما بداخل main.cpp , widget.cpp , widget.h.

## main.cpp

```
#include <QtGui/QApplication>
#include “widget.h“

int main(int argc, char *argv[])      <------ الدالة Main الرئيسية
{
    QApplication a(argc, argv);       <------ تعريف الفصيلة التى تقوم بإدارة التطبيق

    Widget w;                         <------ تعريف فصيلة Widget وهى ترث من الفصيلة QWidget
                                              وكما نرى فأنها ليس لها أب وبالتالى فهى نافذة مستقلة.
    w.show();                         <------ دالة لإظهار النافذة على الشاشة

    return a.exec();                  <------ دالة التكرار لضمان إستمرار التشغيل حتى إيقاف التطبيق
}
```

## widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>

namespace Ui {
    class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = ٠);
    ~Widget();

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H
```

هما فصيلتين مختلفتين:

الأولى : داخل namespace وهى فصيلة مستقلة تحوى دالتين فقط، وتعريفها موجود داخل الملف ui_widget.h. والدالتين هما:

setupUi(QWidget *);

retranslateUi(QWidget *Widget);

setupUi الدالة المختصة بالإنشاء والوصول إلى جميع الكائنات الرسومية الموجودة داخل النافذة الرئيسية، وتستدعى كالتالى:

Ui::Widget

retranslateUi الدالة المختصة بترجمة نصوص الكائنات الرسومية

..............................................

الثانية : هى فصيلة النافذة الرئيسية، وهى ترث فصيلة QWidget.

ui مؤشر للفصيلة Ui::Widget

**widget.cpp**

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::Widget)
{
   ui->setupUi(this);
}

Widget::~Widget()
{
   delete ui;
}
```

لإنشاء الكائنات الرسومية داخل النافذة الرئيسية، وبدونها تظهر النافذة خالية.

لنلقى نظرة على الملف ui_widget.h

( كود هذا الملف معروض بشكل غير كامل، وما نعرضه هنا هى فقط الأجزاء الهامة، وذلك بغرض فهم ما يحدث داخل ***كيوت***، ولكن عند التطبيق ينتج هذا الملف تلقائيا، ولا يتم التعديل فيه من الداخل. )

```
class Ui_Widget
{
public:
   void setupUi(QWidget *Widget) {
          ......
   }

   void retranslateUi(QWidget *Widget) {
          .......
   }
};
```

تعريف فصيلة Ui_Widget
تحتوى على الدالتين retranslateUi , setupUi
setupUi : تستخدم لعرض المكونات الرسومية الخاصة بالمشروع.
retranslateUi : تستخدم لوضع الكلمات المطلوب ترجمتها فى الواجهة الرسومية.

```
namespace Ui {
   class Widget: public Ui_Widget {};
}
```

تعريف الفصيلة Ui::Widget التى ترث الفصيلة Ui_Widget وبالتالى فإنها ترث الدالتين retranslateUi , setupUi .
و الفصيلة Ui::Widget هى المستخدمة فى widget.h

بعد التوضيح السابق لكيفية إنشاء تطبيق ذو واجهة رسومية ومكوناته، سنقوم الآن بشرح مثال لفهم signals and slots جيداً، وربطهما بمكونات الواجهة الرسومية.

سنقوم بإستخدام الفصيلة car التى قمنا بشرحها فى نهاية الفصل السابق QObject ليكون لدينا ملفات الكود التالية:

main.cpp <----- الملف الرئيسى

car.h <----- ملف تعريف الفصيلة car

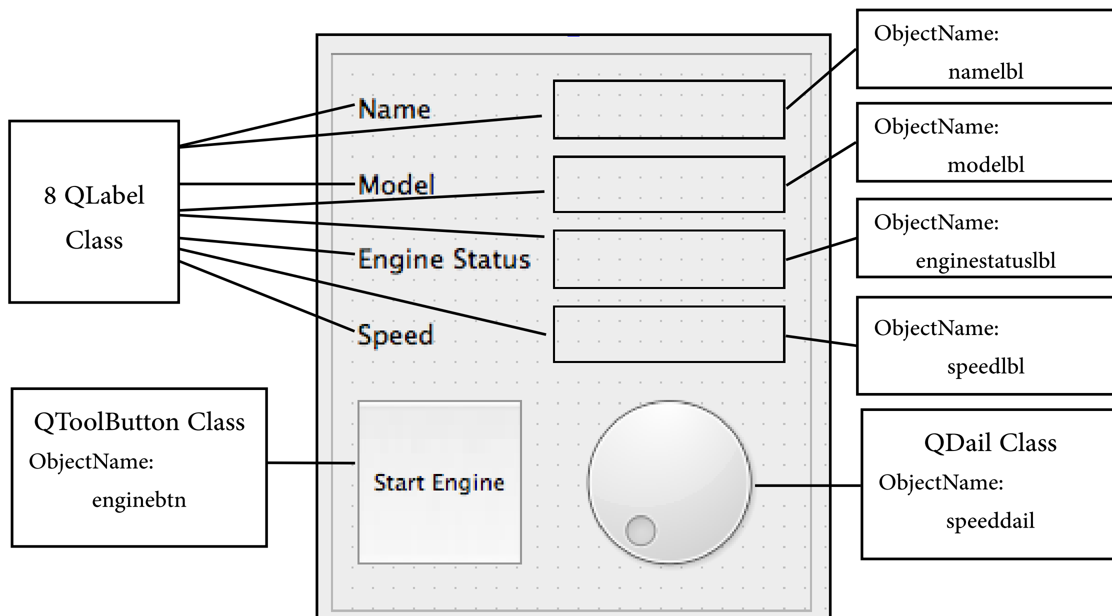
car.cpp <----- ملف الكود التفصيلى للفصيلة car

الفصيلة car_gui هى فصيلة رسومية ترث QWidget وتحوى بعض الكائنات الرسومية.

car_gui.h <----- ملف تعريف الفصيلة car_gui

car_gui.cpp <----- ملف الكود التفصيلى للفصيلة car_gui

نقوم بالخطوات العادية لإنشاء تطبيق رسومى باسم ( EX_QObject_car )، ثم نبدأ برسم الآتى بـ Qt Designer.



car_gui.h

EXAMPLE NO 23

```
#ifndef CAR_GUI_H
#define CAR_GUI_H

#include <QWidget>
#include "car.h"                    <-------- car إستدعاء لتعريف الفصيلة

namespace Ui {
    class car_gui;                  <-------- Ui::car_gui تعريف الفصيلة
}

class car_gui : public QWidget
{
    Q_OBJECT

public:
    explicit car_gui(QWidget *parent = 0);
    ~car_gui();
    car *toyota;

public Q_SLOTS:
    void setCarSpeed(int);
    void setCarEngine(bool);        } دوال مستقبلة للإشارة

private:
    Ui::car_gui *ui;
};

#endif // CAR_GUI_H
```

(car_gui تعريف الفصيلة)

ملحوظة هامة :

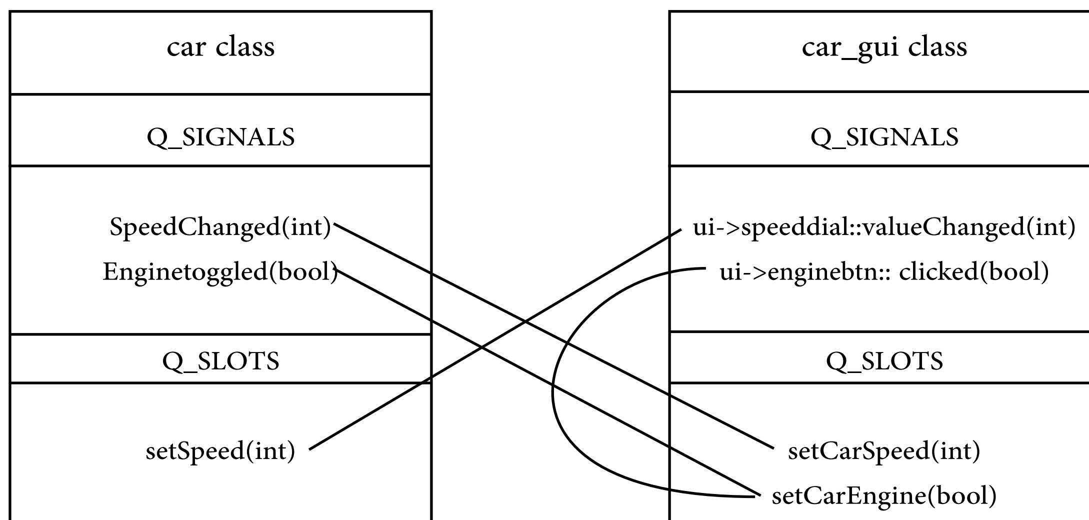
هذا الملف ينشأ تلقائيا نتيجة لإنشاء تطبيق رسومى جديد، ولكن تم إضافة بعض الأكواد اللازمة، وسنضع مستطيل حول الأكواد التى قمنا بإضافتها.

car_gui.cpp

```
#include "car_gui.h"
#include "ui_car_gui.h"
#include "car.h"

car_gui::car_gui(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::car_gui)
{
    QString str;
    ui->setupUi(this);
    toyota = new car(this);
    toyota->setProperty("carname" , "TOYOTA");
    toyota->setModel("2005");
    toyota->setMaxSpeed(200);
    ui->speeddial->setMaximum(toyota->maxSpeed());
    ui->namelbl->setText(toyota->name());

    ui->modelbl->setText(toyota->property("carmodel").toString());

    ui->enginestatuslbl->setText(toyota->EngineStatus());

    str.setNum(toyota->speed());
    ui->speedlbl->setText(str);

    connect(toyota , SIGNAL(SpeedChanged(int)), this , SLOT(setCarSpeed(int)));
    connect(toyota ,SIGNAL(Enginetoggled(bool)),this,SLOT(setCarEngine(bool)));

    connect(ui->speeddial,SIGNAL(valueChanged(int)),toyota,SLOT(setSpeed(int)));
    connect(ui->enginebtn,SIGNAL(clicked(bool)) , this,SLOT(setCarEngine(bool)));
}

car_gui::~car_gui()
{
    delete ui;
}
```

```
void car_gui::setCarSpeed(int speed)
{
    QString str;
    str.setNum(speed);
    ui->speedlbl->setText(str);
    ui->speeddial->setValue(speed);
}

void car_gui::setCarEngine(bool status)
{
    if (status == true)
    {
        toyota->StartEngine();
        ui->enginestatuslbl->setText(toyota->EngineStatus());
        ui->enginebtn->setText("Stop  Engine");
    }
    else
    {
        toyota->StopEngine();
        ui->enginestatuslbl->setText(toyota->EngineStatus());
        ui->enginebtn->setText("Start Engine");
    }
}
```

رسم عمليات الإرتباط ( connect ) بين signals and slots.



ونرى فى المثال السابق كيف أن الإرتباط بين النافذة car_gui والكائنات التى تحويها تتم بواسطة Signals and Slots، وكما ذكرنا فى نهاية الشرح لخواص الفصيلة QObject والفصائل الوارثة لها أنه يمكن الربط بين الدوال بطريقة أخرى وهى طريقة connectSlotsByName، والتى تتم بواسطة إنشاء دالة إستقبال slot بالصيغة التالية :

void on_<object name>_<signal name>(<signal parameters>);

ولعمل ذلك فى المثال السابق سنقوم بالتعديلات الآتية:

فى ملف car_gui.h :

```
class car_gui : public QWidget
{
.....
public Q_SLOTS:
.....
void on_mycarname_SpeedChanged(int);
...
}
```

إضافة الدالة
on_mycarname_SpeedChanged(int)
داخل النطاق Q_SLOTS
حيث mycarname سيكون اسم الكائن المشتق من الفصيلة car

فى ملف car_gui.cpp :

```
car_gui::car_gui(QWidget *parent) :QWidget(parent),ui(new Ui::car_gui)
{
....
   toyota = new car(this);
   toyota->setObjectName("mycarname");
...
 //  connect(toyota , SIGNAL(SpeedChanged(int)), this , SLOT(setCarSpeed(int)));
....
  QMetaObject::connectSlotsByName(this);
}
```

mycarname اسم الكائن المشتق من الفصيلة car.

إلغاء الأمر connect الخاص بدالة الإرسال SpeedChanged

لابد من تشغيل هذه الدالة التى تقوم بالبحث عن الدوال التى على الصيغة.
on_<object name>_<signal name>(<signal parameters>).

```
void car_gui::on_toyota_SpeedChanged(int i )
{
   setCarSpeed( i);
}
```

ستقوم بنفس عمل الدالة connect التى تم إلغائها.

## دوال الأحــــــــــداث

لقد قمنا بتعريف الفصيلة QWidget بأنها عبارة عن مساحة مستطيلة ترسم على الشاشة، وتتفاعل مع الأحداث ( Events ).

والسؤال هنا هو : ماهى دوال الأحداث؟ وكيف يتم تعريفها بالفصيلة QWidget ؟

دوال الأحداث ( Event Function ) :

هى الدوال الخاصة بإستقبال أى أحداث مثل الرسم داخل النافذة ، الضغط على مفتاح من لوحة المفاتيح ، الضغط على مفتاح الفأرة ، تحريكها ، حدوث خاصية ( drag and drop ) أو أى حدث آخر.

تعرف دوال الأحداث بالفصيلة بخطوتين :

١- توضع الدالة فى ملف تعريف الفصيلة ( class.h ) داخل النطاق المحمى protected.

٢- يتم تفصيل الكود ( implement code ) فى ملف كود الفصيلة ( class.cpp ).

- مثال للدالة (* keyPressEvent(QKeyEvent.

والتى ستقوم بفعل معين عند الضغط على مفتاح shift.

١- فى الملف class.h :

```
class classname
{
.....
protected:
void keyPressEvent(QKeyEvent *);
};
```

٢- فى الملف class.cpp :

```
void classname::keyPressEvent(QKeyEvent *key)
{
if(key->key() == Qt::Key_Shift)
	{
		do somthing.....
	}
}
```

## دالة الفلتر للأحداث ( Event Filter Function ) :

هى دالة تقوم بعمل فلتر لأحداث معينة، والسماح بتمرير تلك الأحداث أو عدم تمريرها. وتعرف دوال الفلتر للأحداث بالفصيلة بثلاث خطوات:

١- يتم تشغيل الدالة installEventFilter للكائن المراد فلترة أحداثه.

٢- توضع الدالة فى ملف تعريف الفصيلة ( class.h ) داخل النطاق المحمى protected.

٣- يتم تفصيل الكود ( implement code ) فى ملف كود الفصيلة ( class.cpp ).

مثال:

١- فى الملف class.cpp:

```
classname:: classname(..)        {
ui->textEdit->installEventFilter(this);
}
```

تشغيل فلتر الأحداث للكائن textEdit

٢- فى الملف class.h :

```
class classname
{
.....
protected:
 bool eventFilter(QObject *, QEvent *);
};
```

نلاحظ أن الدالة من النوع bool وليست من النوع void، كما يمرر لها قيمتين هما (الكائن و نوع الحدث)، وعند تشغيل الدالة بواسطة العملية رقم ١ فإن جميع الأحداث التى تخص الكائن تمر مسبقا على الفلتر، فإذا عاد الفلتر بالقيمة true منع الحدث أو false مرر الحدث

٣- فى الملف class.cpp :

```
bool classname::eventFilter(QObject *obj, QEvent *event)    {
   if (obj == ui->textEdit)      {
      if (event->type() == QEvent::KeyPress)  {
         QKeyEvent *ke = static_cast<QKeyEvent *>(event);
         if (ke->key() == Qt::Key_A)   return true;
         else     return false;
      }
   }
      return QWidget::eventFilter(this,event);
}
```

إذا كان الكائن هو textEdit فسوف تراجع أحداثه و غير ذلك يمر.

الحدث KeyPress هو ما نريده بالفلتر و غير ذلك يمر

إذا كان الحدث هو الضغط على مفتاح A فسوف يرجع القيمة true، وبالتالى يمنع الحدث ولا تستطيع كتابة الحرف A داخل الكائن textEdit،
و بالضغط على أى مفتاح آخر ترد القيمة false أى تمر دون الفلتر.

## الرسم داخل QWidget

وكما ذكرنا أن الرسم داخل الفصيلة QWidget هو حدث يسمى paintEvent،
وبالتالى للرسم داخل الفصيلة QWidget يتم تعريف الحدث Event ( دوال الأحداث )كما قمنا بشرحه سابقاً، ثم نستخدم للرسم الفصيلة QPainter،
وسيقوم المثال التالى بتوضيح كيفية الرسم داخل الفصيلة QWidget .

سنقوم بإنشاء تطبيق رسومى يرث الفصيلة QWidget،
وإعلان دالة الحدث paintEvent داخل النطاق protected فى الملف widget.h،
وإعلان دالة الرسم QPainter داخل دالة الحدث paintEvent فى الملف widget.cpp.

**widget.h**

EXAMPLE NO 24

```
#include <QWidget>

class Widget : public QWidget
{
   Q_OBJECT
  ......
protected:
   void paintEvent(QPaintEvent *);          <------ الإعلان عن دالة الحدث paintEvent
  ......
};
```

**widget.cpp**

```
void Widget::paintEvent(QPaintEvent *p)          <------ دالة الحدث paintEvent
{
QPainter pp(this);          <------ الإعلان عن المتغير pp لإدارة فصيلة من النوع QPainter
QPen pen;                   <------ الإعلان عن المتغير pen لإدارة فصيلة من النوع QPen
QColor color(10,100,100);   <------ الإعلان عن المتغير color لإدارة فصيلة من النوع QColor
pen.setColor(color);        <------ إستخدام اللون color للقلم pen

pp.setPen(pen);             <------ إستخدام القلم pen للرسام pp
pp.save();                  <------ حفظ وضع الإحداثيات
```

widget.cpp

```
pp.translate(100,100);            <------  نقل الإحداثيات ( 0 ، 0 ) إلى الإحداثيات (100 ، 100)
pp.drawLine(0,0,100,0);           <------  رسم خط من ( 0 ، 0 ) إلى ( 100 ، 0 )
pp.translate(100,0);              <------  نقل الإحداثيات إلى( 100 ، 0)
pp.rotate(90);                    <------  دوران الإحداثيات 90 درجة
pp.drawLine(0,0,100,0);           <------  رسم خط من ( 0 ، 0 ) إلى ( 100 ، 0 )
pp.translate(100,0);
pp.rotate(90);
pp.drawLine(0,0,100,0);
pp.translate(100,0);
pp.rotate(90);
pp.drawLine(0,0,100,0);
pp.restore();                     <------  إسترجاع وضع الإحداثيات

QBrush br;                        <------  إشتقاق br من الفصيلة QBrush
br.setColor(color);               <------  إستخدام اللون color للرشاش br
br.setStyle(Qt::Dense7Pattern);   <------  إختيار شكل الرشاش
pp.fillRect(0,0,100,100,br);      <------  رش مساحة المربع بواسطة الرشاش br

pen.setWidth(3);                  <------  وضع سُمك القلم Pen Width عند القيمة 3
pp.setPen(pen);                   <------  إعادة إستخدام الرسام للقلم pen
pp.drawEllipse(100,0,100,100);    <------  رسم دائرة
pp.setRenderHint(QPainter::Antialiasing , true);   <------  تشغيل خاصية منع التعريجات
pp.drawEllipse(200,0,100,100);    <------  رسم دائرة
}
```

ويجب ملاحظة الفرق بين الدائرتين فى المثال السابق، وتأثير خاصية منع التعريجات Antialiasing على الدائرة الثانية.

وسنقوم الآن بشرح الفصيلة QPainter وبعض خصائصها.

# QPainter Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة هى الفصيلة المسؤولة عن الرسم داخل QWidget.

**طريقة الإعــلان ( Declaration ) :**

```
QPainter  mypaint( QPaintDevice * device );
```

يمكن لفصيلة QPainter التعامل والرسم على أى فصيلة ترث فصيلة QPaintDecice، وفصيلة QWidget هى إحدى هذه الفصائل.

**وظائف الفصيلة :**

تحتوى هذه الفصيلة على عدد كبير من الوظائف :

- إعدادات الرسم ( Settings ) :

اللون والخط ( نوع وسُمك ) وخلفية الرسم.

- أشكال الرسم ( Drawing ) :

رسم الكلمات والنقط والخطوط والمربعات والدوائر ورسم المسارات لتكوين أشكال معقدة.

- دقة الرسم ( Rendering Quality ) :

تقدم الفصيلة دوال لتحديد دقة الرسم، ومن أهمها خاصية مضاد التعرجات Antialiasing.

- تحريك الإحداثيات ( Coordinate Transformations ):

التعامل مع الإحداثيات مثل مقياس الرسم ( Scale )، والدوران ( Rotate )، والإنتقال ( Translate )، كما يمكننا حفظ الوضع الحالى للإحداثيات بالدالة ( save ) ، ثم عمل عمليات معينة على الإحداثيات من نقل أو دوران، ثم إسترجاع وضع الإحداثيات المحفوظة بواسطة الدالة ( restore )، ذلك بالإضافة إلى وظائف أخرى كثيرة.

# QDialog Class

**نسب الفصيلة :**

ترث فصيلة QWidget

**تعريف الفصيلة :**

ترث هذه الفصيلة خصائص فصيلة QWidget، وتضيف عليها بعض الخصائص الجديدة لتجعل منها نافذة لها صفات خاصة، وهذه الفصيلة تقوم بفتح نافذة خلال التطبيق، ويكون الغرض منها إجراء عملية قصيرة وصغيرة لمساعدة التطبيق، وهناك نوعان من النوافذ :
نوافذ مشروطة، ونوافذ غير مشروطة ( Modal Dialogs , Modeless Dialogs ).

النوافذ المشروطة ( Modal Dialogs ) :

هى نوافذ تتوقف عند ظهورها جميع نوافذ التطبيق منتظرة القيمة المردودة أو الراجعة منها، و تكون القيمة المردودة بالإيجاب Accepted أو بالرفض Rejected.

ومثال على هذه النوافذ :

نافذة فتح ملف ( Open File ) ، نافذة الموافقة أو الرفض على العمليات التى يقوم بها التطبيق، والتى تحتوى هذه المفتاحين ( Ok , Cancel ).

النوافذ غير المشروطة ( Modeless Dialogs ):

هى نوافذ يتم ظهورها بجانب النوافذ الأخرى للتطبيق، ولا ينتظر التطبيق منها رد سواء كان بالإيجاب أو بالرفض.

مثال على هذه النوافذ :

نوافذ البحث ( Find ) ، الإستبدال ( Replace ) فى تطبيقات معالجة النصوص.

**طريقة الإعـــلان ( Declaration ) :**

تتم طريقة التعريف.

```
QDialog        mydialog( QWidget * parent = 0);
```

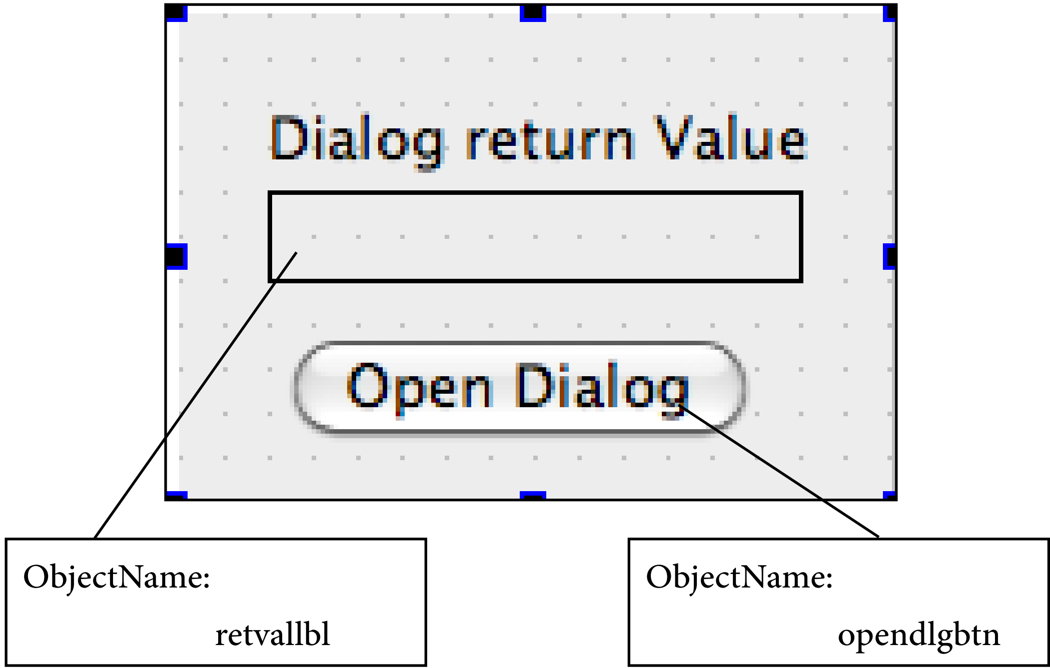
**وظائف الفصيلة :**

فى النوافذ المشروطة يتم إرجاع القيمة عن طريق الدالة ()exec، وهذه القيمة قد تكون بالموافقة Accepted أو بالرفض Rejected.

- يتم إرجاع القيمة Accepted عند الضغط على مفتاح Ok أو ضغط Enter من لوحة المفاتيح.
- يتم إرجاع القيمة Rejected عند الضغط على مفتاح Cancel أو ضغط ESC من لوحة المفاتيح.

مثال: نريد فتح نافذة Qwidget تسمح بفتح نافذة مشروطة Modal Dialog وإستقبال نص منها يكتب بداخلها.

- نقوم بفتح مشروع جديد بالأداة QtCreator من النوع widget.



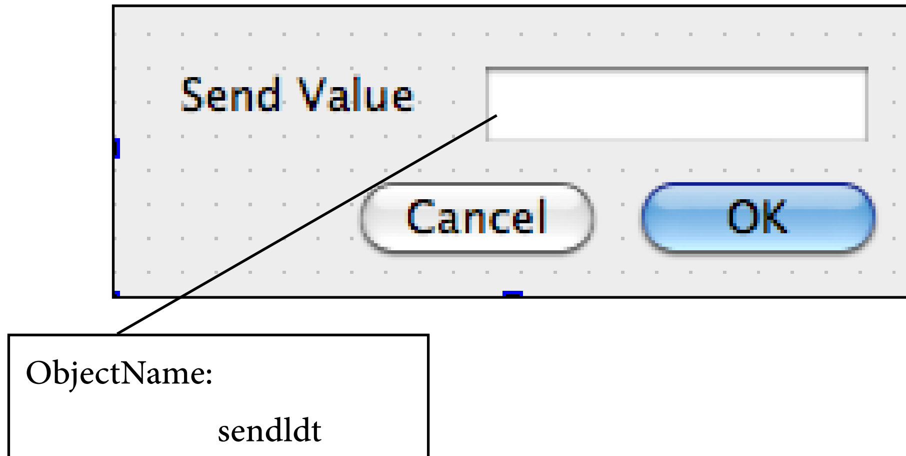

- نقوم بإضافة فصيلة رسومية جديدة كالآتى:

إختيار newfile or project من القائمة file،

ثم إختيار Qt Designer Form Class من القائمة Files And Classes،

ثم إختيار Dialog with Buttons،

ثم تسمية الفصيلة **myDialog**.



ستقوم الأداة Qt Creator بتوليد الأكواد اللازمة، وسنذكر هنا فقط الأكواد المضافة داخل الملفات.

ملف widget.h------->

EXAMPLE NO 25

```
...
#include “mydialog.h“
...
class Widget : public QWidget
{
.....
private slots:
    void on_opendlgbtn_clicked();
...
};
```

الدالة on_opendlgbtn_clicked هى دالة الإستقبال
لدالة الإرسال clicked الخاصة بالكائن opendlgbtn

ملف widget.cpp------->

```
void Widget::on_opendlgbtn_clicked()
{
    myDialog mydlg(this);
    if(mydlg.exec() == true)
        ui->retvallbl->setText(mydlg.retval());
}
```

عند الضغط ( clicked ) على المفتاح ( opendlgbtn ) يتم تنفيذ الكود الداخلى للدالة.

```
myDialog mydlg(this);
```

الإعلان عن المتغير mydlg لإدارة فصيلة myDialog.

```
if(mydlg.exec() == true)
    ui->retvallbl->setText(mydlg.retval());
```

الدالة mydlg.exec تساوى true عند الضغط على ok، و false عند الضغط على cancel.

```
class myDialog : public QDialog                <-------mydialog.hملف
{
..
public:
...
   QString retval();
...
};
```

```
QString myDialog::retval()                     <-------mydialog.cppملف
{
   return this->ui->sendldt->text();
}
```

دالة **retval** لإرجاع القيمة التى ستكتب داخل الكائن sndldt الذى يرث الفصيلة QLineEdit.

.........................................................................

المثال السابق يوضح كيفية إنشاء فصيلة QDialog كنافذة مشروطة Modal Dialog.
ولإنشاء فصيلة QDialog كنافذة غير مشروطة Modeless Dialog، لن نقوم بإستعمال الدالة exec() حيث يمكن كتابة الكود كالآتى:

```
void Widget::on_opendlgbtn_clicked()          <-------widget.cpp ملف
{
   myDialog *mydlg =  new myDialog (this);
   mydlg->show();
}
```

.........................................................................

# QMainWindow Class

**نسب الفصيلة :**

ترث فصيلة QWidget

**تعريف الفصيلة :**

تتعامل هذه الفصيلة مع الفصائل الآتية:

- شريط القوائم Menu Bar.

يشتق من الفصيلة QMenuBar، وهى ترث QWidget.

- شريط الأدوات Tool Bar.

يشتق من الفصيلة QToolBar، وهى ترث QWidget.

- شريط الحالة Status Bar.

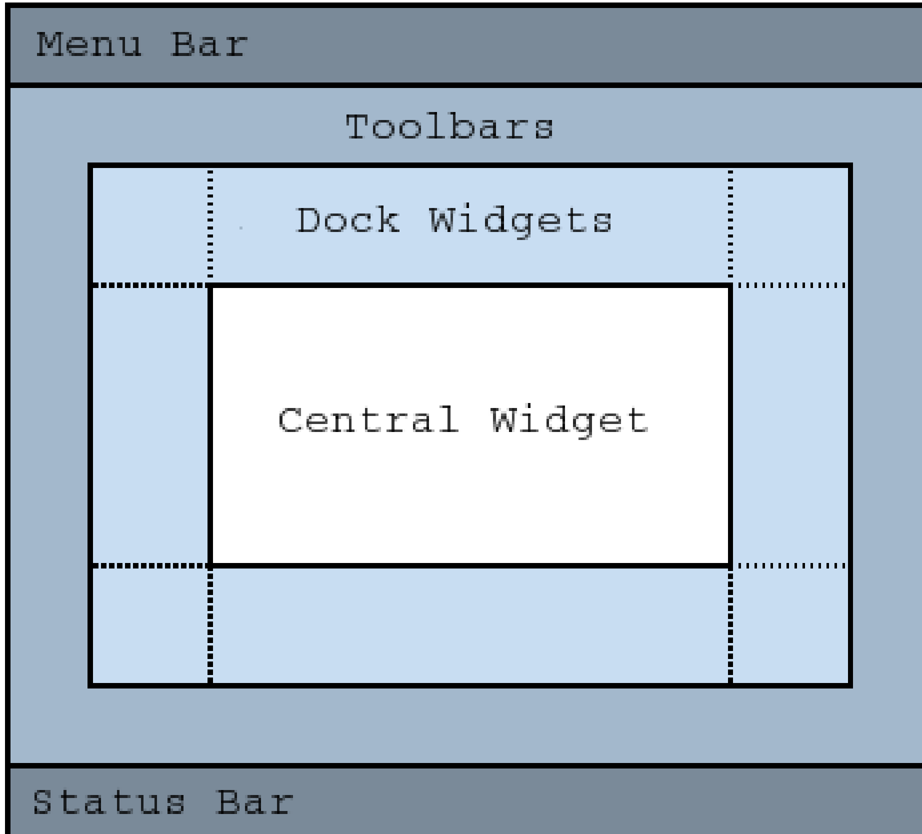
يشتق من الفصيلة QStatusBar، وهى ترث QWidget.

- سطح مكتب مساعد Dock Widget.

يشتق من الفصيلة QDockWidget، وهى ترث QWidget.

- سطح مكتب التطبيق Central Widget.

يشتق من فصيلة QWidget.

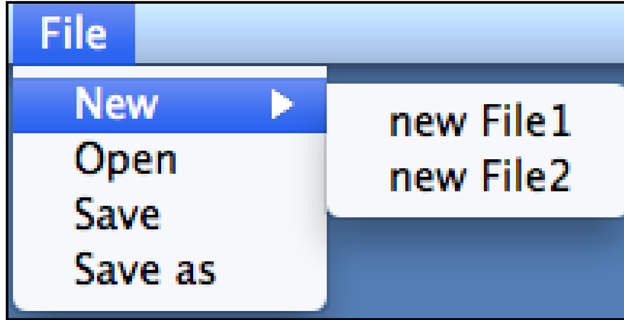


## شريط القوائم Menu Bar

يشتق من الفصيلة QMenuBar، وهى ترث الفصيلة QWidget، ويحتوى شريط القوائم على مجموعة من العناصر أو على قوائم يتفرع منها عدة عناصر، حيث أن القوائم والقوائم المتفرعة ترث فصيلة QMenuBar، والعناصر ترث فصيلة QAction.

مثال : إنشاء قائمة File تحتوى على عناصر Open , Save , Save as ، وتحتوى على قائمة متفرعة New، التى تحتوى على عناصر new File1 , new File2.

لتكون القائمة بالشكل التالى:




EXAMPLE NO 26

إنشاء العناصر

```
actionNew_File1 = new QAction(MainWindow);
actionNew_File2 = new QAction(MainWindow);
actionOpen = new QAction(MainWindow);
actionSave = new QAction(MainWindow);
actionSave_as = new QAction(MainWindow);
```

إنشاء القوائم

```
menuBar = new QMenuBar(MainWindow);
menuFile = new QMenu(menuBar);
menuNew = new QMenu(menuFile);
MainWindow->setMenuBar(menuBar);
```

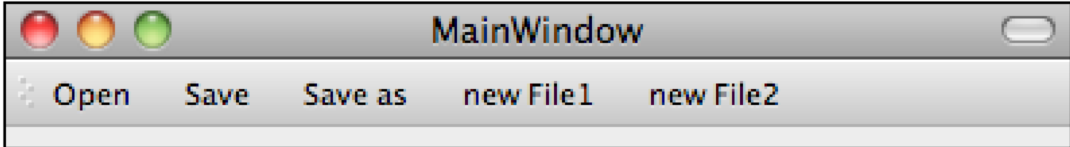
إضافة العناصر داخل القوائم

```
menuBar->addAction(menuFile->menuAction());
menuFile->addAction(menuNew->menuAction());
menuFile->addAction(actionOpen);
menuFile->addAction(actionSave);
menuFile->addAction(actionSave_as);
menuNew->addAction(actionNew_File1);
menuNew->addAction(actionNew_File2);
```

## شريط الأدوات Tool Bar

يشتق من الفصيلة QToolBar، وهى ترث الفصيلة QWidget، ويحتوى شريط الأدوات على عناصر ترث فصيلة QAction، حيث يمكن إضافة نفس العناصر التى تحتويها القوائم إلى شريط الأدوات، ويمكن إظهار هذه العناصر فى شكل أيقونات أو كتابة.

ليكون شريط الأدوات بالشكل التالى:




```
actionNew_File1 = new QAction(MainWindow);
actionNew_File2 = new QAction(MainWindow);
actionOpen = new QAction(MainWindow);
actionSave = new QAction(MainWindow);
actionSave_as = new QAction(MainWindow);
```

إنشاء العناصر

```
mainToolBar = new QToolBar(MainWindow);
MainWindow->addToolBar(Qt::TopToolBarArea, mainToolBar);
```

إنشاء شريط الأدوات

```
mainToolBar->addAction(actionOpen);
mainToolBar->addAction(actionSave);
mainToolBar->addAction(actionSave_as);
mainToolBar->addAction(actionNew_File1);
mainToolBar->addAction(actionNew_File2);
```

إضافة العناصر داخل شريط الأدوات

## شريط الحالة Status Bar

يشتق من الفصيلة QStatusBar، وهى ترث الفصيلة QWidget، ويظهر فى أسفل النافذة وتعرض به الرسائل الخاصة بحالة التطبيق، وتعرض هذه الرسائل بواسطة الدالة showMessage، وتمسح بواسطة الدالة clearMessage.

مثال على ذلك : إظهار حالة حفظ البيانات، أو إستدعائها أثناء تشغيل التطبيق.

### سطح المكتب المساعد Dock Widget

يشتق من الفصيلة QDockWidget وهى ترث الفصيلة QWidget، ويضاف إليها أى فصيلة ترث فصيلة QWidget ، ويمكن وضعها فى أى إتجاه من الإتجاهات الأربعة للنافذة MainWindow.

Qt::LeftDockWidgetArea

Qt::RightDockWidgetArea

Qt::TopDockWidgetArea

Qt::BottomDockWidgetArea

وتعرف كالآتى:

```
QDockWidget *docw = new QDockWidget(MainWindow);
MainWindow->addDockWidget(Qt::RightDockWidgetArea , docw);
```

### سطح مكتب التطبيق Central Widget

يشتق من فصيلة QWidget، وهو صفحة التطبيق الرئيسية، و يتم إضافة الكائن QWidget إلى النافذة MainWindow داخل Central widget عن طريق الدالة

```
mainwindow->setCentralWidget(QWidget *mywidget);
```

### وهناك نوعين من النوافذ الرئيسية mainwindow :

- نافذة ذات وثيقة واحدة ( SDI ( Single Document Interface.
- نافذة متعددة الوثائق ( MDI ( Multiple Document Interface.

### النافذة ذات الوثيقة الواحدة SDI :

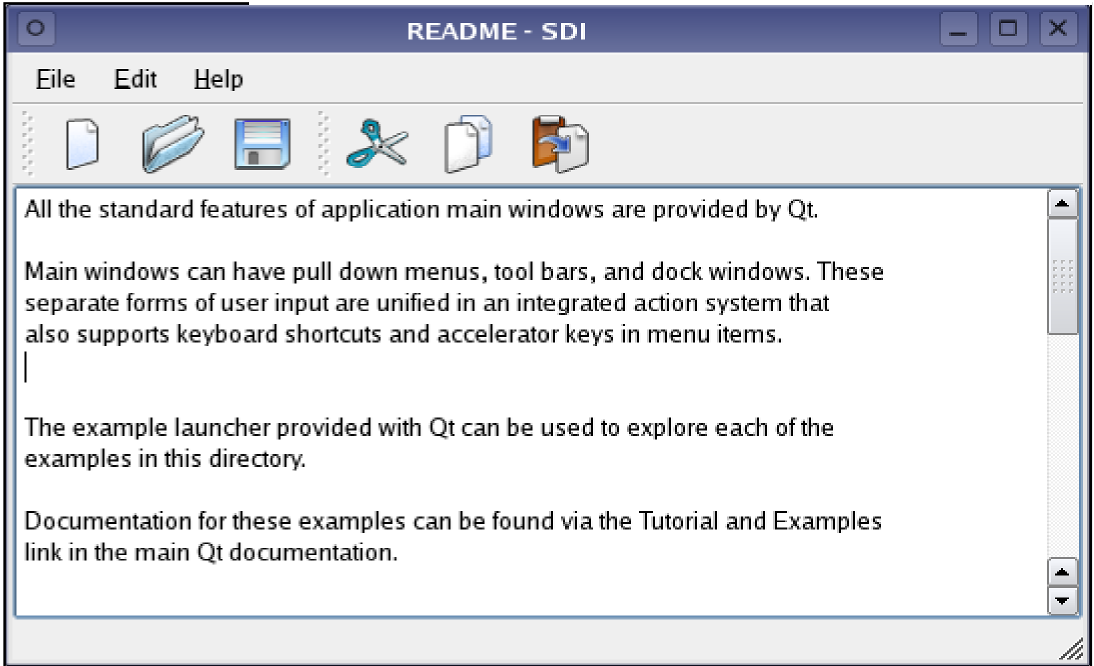
هى نافذة تطبيق تتعامل مع ملف أو وثيقة واحدة فقط، وتنشأ عن طريق وضع أى فصيلة منحدرة من QWidget كسطح مكتب CentralWidget للتطبيق.

### النافذة متعددة الوثائق MDI :

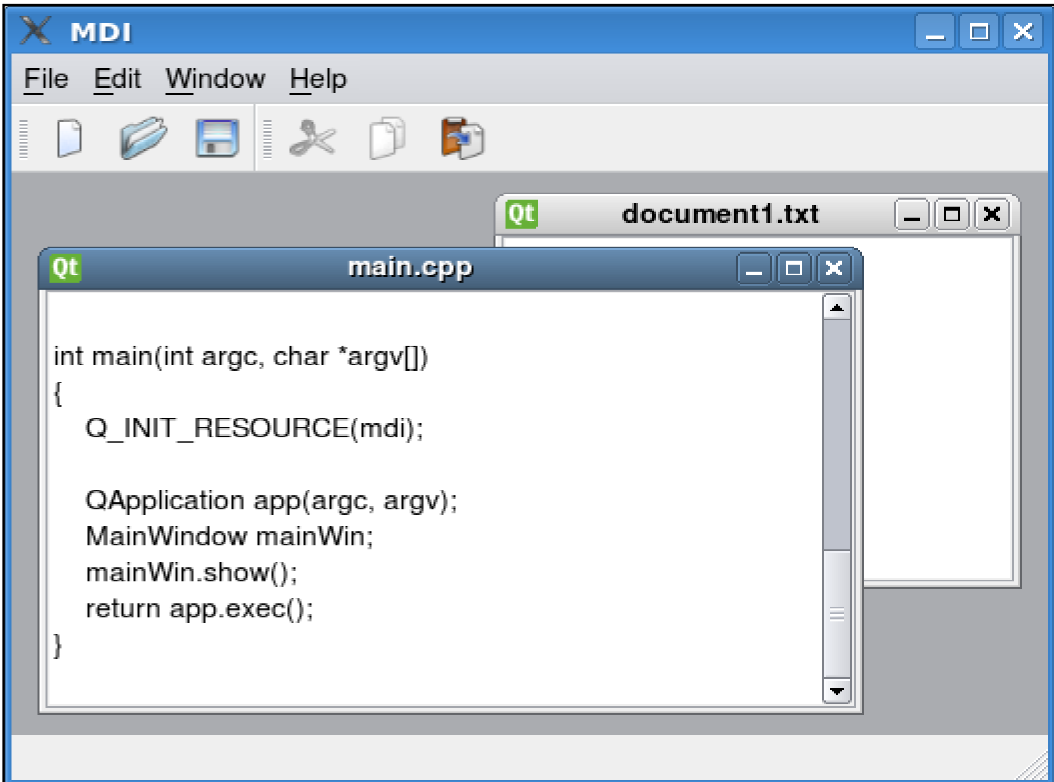
هى نافذة تطبيق تتعامل مع أكثر من ملف أو وثيقة على نفس سطح المكتب، وتنشأ عن طريق وضع فصيلة QMdiArea كسطح مكتب CentralWidget للتطبيق.

ملحوظة : الأكواد فى الأمثلة السابقة كانت لتوضيح كيفية بناء النوافذ الرئيسية MainWindows ، ولكن يمكن بناء النافذة ووضع كل مميزاتها من خلال الأداة Qt Designer.

نافذة ذات وثيقة واحدة SDI ( Single Document Interface )




نافذة متعددة الوثائق MDI ( Multiple Document Interface )



# إدارة التخطيط

# Layout Managment

المقصود بإدارة التخطيط هو كيفية تخطيط النافذة من حيث أماكن وضع الكائنات الرسومية والمسافات بينها، وماذا يحدث عند تكبير نافذة التطبيق بحجم الشاشة أو عند تصغيرها، وبمعنى آخر هو كيفية ضبط الكائنات الرسومية على واجهة التطبيق الرسومية GUI، ويتم التحكم فى قياس وصلاحية ووضعية الكائن الرسومى على الواجهة الرسومية من خلال :

- قياسات الكائن الرسومى Object Geometry.
- التخطيط Layout.
- صلاحيات مقاييس الكائن Size Policy.

## أولاً قياسات الكائن الرسومى Object Geometry:

لكل فصيلة QWidget ( الفصيلة الأساسية للمكونات الرسومية ) عدة دوال مختصة بقياسات الكائن الرسومى من إرتفاع ، عرض ، أصغر إرتفاع ، أصغر عرض ، أكبر إرتفاع وأكبر عرض، وتحسب القياسات بالنقطة ( Pixel ) على الشاشة.

دالة وضع قياس الكائن طول وعرض.

```
setGeometry ( int x, int y, int w, int h )
```

x , y إحداثيات بداية رسم الكائن على الشاشة

w عرض الكائن

h إرتفاع الكائن

```
setMaximumHeight ( int maxh )
```

maxh أقصى إرتفاع للكائن

```
setMaximumWidth ( int maxw )
```

maxw أقصى عرض للكائن

```
setMinimumHeight ( int minh )
```

minh أقل إرتفاع للكائن

```
setMinimumWidth ( int minw )
```

minw أقل عرض للكائن

## ثانياً التخطيط Layout:

يتم تخطيط الواجهة الرسومية للمحافظة على شكل التطبيق و إظهاره بأفضل صورة، فدائماً ما تتغير كلاً من المسافات و ترتيب الكائنات الرسومية، وذلك عند تمدد أو إنكماش النافذة التى تحتويهم، ولذلك قامت ***كيوت*** بعمل عدة فصائل لتقوم بضم الكائنات الرسومية داخل تخطيط معين، وطرحت ***كيوت*** أربعة نظم للتخطيط :

- تخطيط أفقى Horizontal Layout.
- تخطيط رأسى Vertical Layout.
- تخطيط شبكى Grid Layout.
- تخطيط على شكل إستمارة Form Layout.

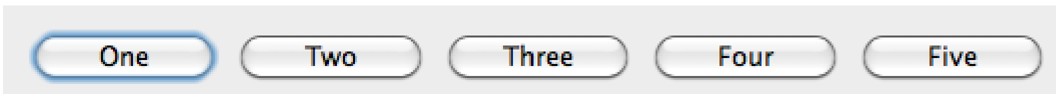
## تخطيط أفقى Horizontal Layout :

تتم عملية التخطيط الأفقى بإستخدام الفصيلة QHBoxLayout، ثم إضافة المكونات الرسومية لها كعناصر لتقوم بترتيبهم ترتيباً أفقياً على مسافات متساوية.

مثال:

EXAMPLE NO 27

```
QWidget *window = new QWidget;                          <------- window إنشاء النافذة
QPushButton *button1 = new QPushButton("One");          }
QPushButton *button2 = new QPushButton("Two");          }
QPushButton *button3 = new QPushButton("Three");        }  إنشاء المفاتيح
QPushButton *button4 = new QPushButton("Four");         }
QPushButton *button5 = new QPushButton("Five");         }

QHBoxLayout *layout = new QHBoxLayout;                  <------- layout إنشاء المخطط الأفقى
layout->addWidget(button1);     }
layout->addWidget(button2);     }
layout->addWidget(button3);     }  وضع المفاتيح داخل المخطط الأفقى
layout->addWidget(button4);     }
layout->addWidget(button5);     }
window->setLayout(layout);      <-------window وضع المخطط الأفقى layout داخل النافذة
window->show();
```



### تخطيط رأسى Vertical Layout :

تتم عملية التخطيط الرأسى بإستخدام الفصيلة QVBoxLayout، ثم إضافة المكونات الرسومية لها كعناصر لتقوم بترتيبهم ترتيباً رأسياً على مسافات متساوية.
مثال: نفس المثال السابق مع إستبدال الفصيلة QHBoxLayout بالفصيلة QVBoxLayout.

### تخطيط شبكى Grid Layout :

تتم عملية التخطيط الشبكى بإستخدام الفصيلة QGridLayout، ثم إضافة المكونات الرسومية لها كعناصر لتقوم بترتيبهم فى صفوف و أعمدة على مسافات متساوية.
ولذلك يتم إدخال الكائنات الرسومية لها مع رقم الصف و رقم العمود.

```
addWidget ( QWidget * widget, int row, int column);
```

### تخطيط على شكل إستمارة Form Layout :

تتم عملية التخطيط على شكل إستمارة بإستخدام الفصيلة QFormLayout، ثم إضافة المكونات الرسومية لها كعناصر لتقوم بترتيبهم على شكل إستمارة فى صفوف، و لها أكثر من طريقة لإدخال العناصر، ومن هذه الطرق :

```
addRow ( const QString & labelText, QWidget * field );
setWidget ( int row, ItemRole role, QWidget * widget );
```

فصائل التخطيط Layout Classes ليست لديها فقط إمكانية إحتواء الكائنات الرسومية، ولكن يمكنها إحتواء عدة فصائل تخطيط أخرى، فمثلاً يمكن إنشاء مخططين أفقيين لترتيب مجموعتين من المفاتيح أفقياً ثم إنشاء مخطط رأسى و نضم له المخططين الأفقيين.

```
QHBoxLayout *H1_layout = new QHBoxLayout;
QHBoxLayout *H2_layout = new QHBoxLayout;
QVBoxLayout *V_layout = new QHBoxLayout;
V_layout ->addLayout(H1_layout);
V_layout ->addLayout(H2_layout);
```

بمجرد أن يصبح الكائن ضمن فصيلة تخطيط Layout تقوم الفصيلة بحسابات معينة ينتج عنها ما يسمى بالحجم المفترض SizeHint للكائن، والذى يستخدم من قِبل Size Policy.

### ثالثاً صلاحيات مقاييس الكائن Size Policy:

تتمثل هذه الصلاحيات فى قدرة الكائن على التمدد، الإنكماش، أو تثبيت قياساته، وخصصت الفصيلة QSizePolicy لتكوين الصلاحيات المطلوبة، ومن ثم إعطاء هذه الصلاحيات للكائن أو لأكثر من كائن عن طريق الدالة :

```
setSizePolicy ( QSizePolicy )
```

### لتكوين الصلاحية المطلوبة بواسطة QSizePolicy:

```
QSizePolicy sizePolicy;
```

إعلان المتغير **sizePlolicy** لإدارة فصيلة من النوع **QSizePolicy**.

```
sizePolicy. setHorizontalPolicy ( Policy policy )
```

تقوم هذه الدالة بوضع الصلاحية **policy** فى الإتجاه الأفقى للكائن.

```
sizePolicy. setVerticalPolicy ( Policy policy )
```

تقوم هذه الدالة بوضع الصلاحية **policy** فى الإتجاه الرأسى للكائن.

```
sizePolicy.setHorizontalStretch(uchar stretchFactor);
```

تقوم هذه الدالة بوضع معامل التمدد **stretchFactor** فى الإتجاه الأفقى للكائن.

```
sizePolicy.setVerticalStretch(uchar stretchFactor);
```

تقوم هذه الدالة بوضع معامل التمدد **stretchFactor** فى الإتجاه الرأسى للكائن.

ما هى الصلاحية policy ؟ وما هو معامل التمدد stretchFactor ؟

.........................................................................

## الصلاحيات Policies:

تعمل هذه الصلاحيات فقط عندما يصبح الكائن ضمن فصيلة تخطيط Layout، فبمجرد وضع الكائن داخل فصيلة تخطيط تقوم الفصيلة بحسابات معينة ينتج عنها ما يسمى بالحجم المفترض SizeHint للكائن، وبناءاً على هذا الحجم المفترض من إدارة التخطيط تتحرك تلك الصلاحيات من خلال إلتزامها بهذا الحجم المفترض SizeHint.

| الصلاحيـــــــــة | الشـــــــــــــــــــــــرح |
| --- | --- |
| QSizePolicy::Fixed | الحجم المفترض SizeHint هو الحجم الإلزامى،<br>وغير مسموح للكائن أن يتضخم أو ينكمش عن الحجم المفترض SizeHint. |
| QSizePolicy::Minimum | الحجم المفترض SizeHint هو الحجم الأصغر،<br>ويمكن للكائن أن يتمدد، وغير مسموح له أن يقل حجمه عن الحجم المفترض SizeHint. |
| QSizePolicy::Maximum | الحجم المفترض SizeHint هو الحجم الأكبر،<br>ويمكن للكائن أن ينكمش، وغير مسموح له أن يزيد حجمه عن الحجم المفترض SizeHint. |
| QSizePolicy::Preferred | الحجم المفترض SizeHint هو الحجم الأفضل،<br>ويمكن للكائن أن ينكمش أو يتمدد ويظل مقبولاً، وفى الغالب لا حاجة لجعله أكبر من الحجم المفترض SizeHint. ( هذا هو الإختيار الإفتراضى لكل QWidget) |
| QSizePolicy::Expanding | الحجم المفترض SizeHint هو الحجم المعقول،<br>ويمكن للكائن أن ينكمش ويظل مقبولاً ، ويمكنه أيضاً إستخدام مساحة إضافية للتمدد، لذلك يجب أن يترك للكائن مساحة قدر المستطاع. |
| QSizePolicy::MinimumExpanding | الحجم المفترض SizeHint هو الحجم الأصغر و الكافى،<br>ويمكن للكائن إستخدام مساحة إضافية للتمدد، لذلك يجب أن يترك للكائن مساحة قدر المستطاع. |
| QSizePolicy::Ignored | الحجم المفترض SizeHint يتم تجاهله،<br>سيحاول الكائن إستخدام أكبر مساحة ممكنة للتمدد. |

........................................................................

معامل التمدد stretchFactor:

القيمة الإفتراضية لمعامل التمدد ( stretchFactor ) تساوى 0، وبالتالى فعند محاولة تمديد نافذة التطبيق سوف تتمدد كل الكائنات الرسومية بنفس المقياس.

أما إذا كانت قيمة معامل التمدد ( stretchFactor ) للكائن X تساوى 1،
وقيمة المعامل للكائن Y تساوى 2،
فعند التمدد دائماً ما سيأخذ الكائن Y ضعف المساحة التى يأخذها الكائن X.

**الجدير بالذكر أنه يمكن القيام بجميع ما سبق من خلال الأداة Qt Designer .**

Notes

# السحب و الإسقاط

# Drag and Drop

خاصية السحب و الإسقاط Drag And Drop هى خاصية تتيح نقل المعلومات الخاصة بالكائنات الرسومية أو النصية بين تطبيق و آخر أو داخل التطبيق الواحد، حيث يتم الإمساك بالكائن ( object ) المراد سحبه بواسطة الفأرة ( mouse )، ثم تحريكه إلى مكان الكائن ( object ) المراد إسقاطه عليه.
و تتطلب هذه العملية الآتى:

- موافقة الكائن المراد سحبه على إجراء عملية السحب ( Drag ).
- موافقة الكائن المراد الإسقاط عليه على إجراء عملية الإسقاط ( Drop ).

ويمكننا ملاحظة أن هذه الخاصية تستخدم بالفعل بين الكثير من البرامج بسهولة تامة، وتستخدم أيضاً على أنظمة تشغيل مختلفة، والسبب فى ذلك أن خاصية السحب و الإسقاط تقوم بنقل البيانات بهيكلة ثابتة و موحدة بين التطبيقات، فمعظم أنظمة التشغيل تتعامل مع البيانات بنظام MIME ( Multipurpose Internet Mail Extensions ).

ولأن MIME يعتبر هو النظام القياسى، فإن ***كيوت*** تتعامل به فى نقل البيانات داخل خاصية السحب و الإسقاط، ونذكر أيضاً أن لكل نظام تشغيل طريقته الخاصة فى التعامل مع نقل البيانات، لذا أنتجت ***كيوت*** فصيلتين للتعامل مع النظام ( ويندوز ) و النظام ( ماك )، وتقوم الفصيلتين بالتحويل ما بين الأنظمة الخاصة لنقل البيانات و النظام MIME.

تتم عملية السحب و الإسقاط على ثلاثة مراحل:

- سحب الكائن بالضغط عليه بالفأرة.
- تحريك الكائن متجهاً إلى مكان الإسقاط.
- إسقاط الكائن أو إلغاء العملية.

نلاحظ من الخطوات السابقة أن جميع العمليات تتم وفقاً للأحداث Event، وبما أن الفصيلة الأساسية للكائنات الرسومية QWidget عبارة عن مساحة مستطيلة ترسم على الشاشة وتتفاعل مع الأحداث ( Events )، فإن عملية السحب والإسقاط ما هى إلا واحدة من هذه الأحداث التى تتعامل معها QWidget.

..........................................................................

الأحداث التى تخص عملية السحب و الإسقاط :

| الحــــدث | فصيلـة الحــدث | الإختصــــــــــــــــاص |
|---|---|---|
| dragEnterEvent | QDragEnterEvent | يحدث عند بدأ عملية السحب و الإسقاط. |
| dragLeaveEvent | QDragLeaveEvent | يحدث عند إلغاء عملية السحب و الإسقاط. |
| dragMoveEvent | QDragMoveEvent | يحدث أثناء عملية السحب و الإسقاط. |
| dropEvent | QDropEvent | يحدث عند إنتهاء عملية السحب و الإسقاط. |

ما هو النظام MIME؟

هى طريقة تستخدم لنقل البيانات من و إلى البريد الإلكترونى، وهى تعتمد على إرسال رأس البيانات إذا كانت نصية ( text/plain ) أو صورة ( image/jpeg )...إلخ ومن ثم إرسالها، ولقد إعتمدت ***كيوت*** هذا النظام فى نقل البيانات لخاصية السحب و الإسقاط، ونؤكد مرة أخرى أن هذه العملية تقوم بسحب و إسقاط البيانات فقط.

لمزيد من المعلومات عن MIME :

http://www.iana.org/assignments/media-types/

ما هى خطوات سحب كائن رسومى و إسقاطه فى مكان آخر؟

الإجابة :

- عند السحب :

نأخذ من الكائن كل ما نحتاجه من بيانات،

ثم نمرر هذه البيانات بواسطة النظام MIME.

- عند الإسقاط نتعامل مع البيانات بإحدى الطريقتين:

نقوم بإنشاء كائن جديد يحمل البيانات الممررة من النظام MIME ومسح الكائن القديم،

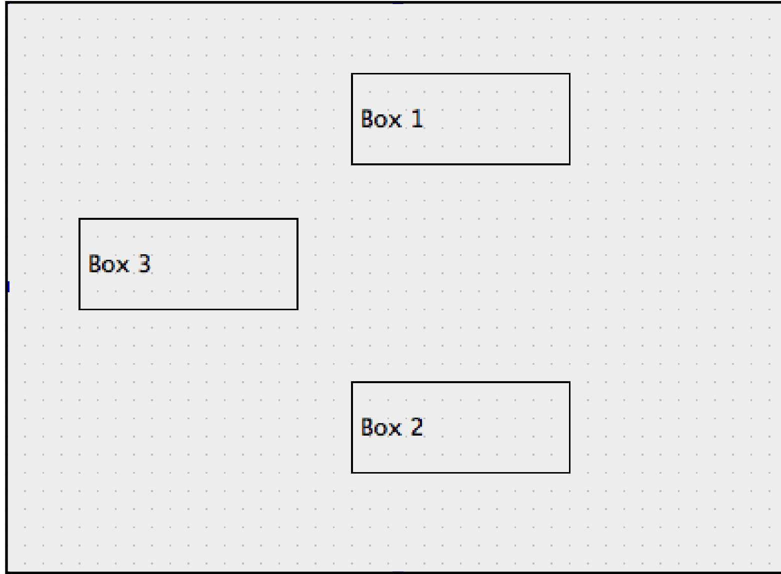
أو نقوم بتعديل الكائن القديم وفقا لوضع الإسقاط الجديد.

..........................................................................

مثال لفهم و تطبيق خاصية السحب و الإسقاط Drag and Drop.

المطلوب عمل نافذة بها ثلاثة كائنات QLabel، ويمكن سحب أى منهم و إسقاطه فى مكان أخر.

سنبدأ تطبيق رسومى QWidget ونرسم التالى فى Qt Designer.




حيث :

Box 1:

Class : QLabel.

ObjectName : label.

Box 2:

Class : QLabel.

ObjectName : label_2.

Box 3:

Class : QLabel.

ObjectName : label_3.

.........................................................................

## widget.h

EXAMPLE NO 28

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QLabel>

namespace Ui {
   class Widget;
}

class Widget : public QWidget
{
   Q_OBJECT

public:
   explicit Widget(QWidget *parent = 0);
   ~Widget();
   QWidget *DragedLabel;        <------   مؤشر للكائن QLabel الذى يتم سحبه
  QLabel *DropedLabel;          <------   مؤشر للكائن QLabel الذى يتم إسقاطه

private:
   Ui::Widget *ui;

protected:
   void mousePressEvent(QMouseEvent *);          <------  لبرمجة ما يحدث عند الضغط على الفأرة
   void dragEnterEvent(QDragEnterEvent *);       <------  لبرمجة ما يحدث عند بداية عملية السحب
   void dropEvent(QDropEvent *);                 <------  لبرمجة ما يحدث عند عملية الإسقاط
};

#endif // WIDGET_H
```

## widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
#include <QMouseEvent>
#include <QStringList>
Widget::Widget(QWidget *parent) : QWidget(parent),  ui(new Ui::Widget)
{
   ui->setupUi(this);
   this->setAcceptDrops(true);              <------ هذه الدالة ضرورية لجعل حدث السحب و الإسقاط متاح لهذه النافذة
}
Widget::~Widget()
{
   delete ui;
}

void Widget::mousePressEvent(QMouseEvent *event)
{
   if (event->button() == Qt::LeftButton)
      {
         DragedLabel = childAt(event->pos());
         if(!DragedLabel) return;
         if (DragedLabel->inherits("QLabel") == true)
            {
               QString labelName = DragedLabel->objectName() ;
               QString labelText = static_cast<QLabel*> (DragedLabel)->text();
               QString transferText = labelName + "---" + labelText;

               QDrag *drag = new QDrag(this);
               QMimeData *mimeData = new QMimeData;

               mimeData->setText(transferText);
               drag->setMimeData(mimeData);
               drag->exec();
            }
      }
}
```

```
void Widget::dragEnterEvent(QDragEnterEvent *event)
{
    if (event->mimeData()->hasFormat("text/plain"))
        event->acceptProposedAction();
}

void Widget::dropEvent(QDropEvent *event)
{
    QStringList strlist;
    strlist = event->mimeData()->text().split("---");

    DropedLabel = this->findChild<QLabel *>(strlist[0]);
    DropedLabel->setGeometry(  event->pos().x()-75,event->pos().y()-25,150,50  );
    DropedLabel->setText(strlist[1]);

    event->acceptProposedAction();
}
```

شرح الكود السابق

void Widget::mousePressEvent(QMouseEvent *event)

هى الدالة المسؤولة عن حدث الضغط على الفأرة.

if (event->button() == Qt::LeftButton)

فى حالة الضغط على الزر الأيسر للفأرة.

DragedLabel = childAt(event->pos());

**DragedLabel** سيكون المؤشر للكائن الإبن الموجود عند **event->pos()**، حيث أن الدالة **event->pos()** تقوم بإرجاع الإحداثيات **X,Y** لمكان ضغط الفأرة على الشاشة، وتقوم الدالة **childAt** بإيجاد الكائن الإبن ( ابن بالنسبة للنافذة ) الموجود بتلك الإحداثيات.

if(!DragedLabel) return;

نقوم بإختبار المؤشر **DragedLabel**، فإذا كان خاوياً ينهى الحدث ولا يفعل شئ.

```
if (DragedLabel->inherits("QLabel") == true)
```
نقوم بإختبار المؤشر **DragedLabel**، إذا كان يشير إلى كائن يرث الفصيلة **QLabel**.

```
QString labelName = DragedLabel->objectName() ;
QString labelText = static_cast<QLabel*> (DragedLabel)->text();
QString transferText = labelName + "---" + labelText;
```
تجهيز البيانات التى سيتم إرسالها بواسطة **MIME**، حيث يتم إرسال نص يحتوى على كلاً من ( اسم الكائن والنص الموجود بداخله )، وعلى سبيل المثال وبإستخدام الكود السابق:
عند سحب الكائن **label_2** سيصبح النص المجهز للإرسال **"label_2---Box 2"** .

```
QDrag *drag = new QDrag(this);
```
الإعلان عن المتغير **drag** لإدارة فصيلة سحب **QDrag**.

```
QMimeData *mimeData = new QMimeData;
```
الإعلان عن المتغير **mimeData** لإدارة فصيلة **QMimeData**.

```
mimeData->setText(transferText);
```
إدخال النص المراد إرساله لمؤشر الفصيلة **mimeData**. وهو من النوع **text/plain**.

```
drag->setMimeData(mimeData);
```
إدخال مؤشر الفصيلة **mimeData** لمؤشر الفصيلة ليتم إستعمال **mimeData** عند عملية السحب.

```
drag->exec();
```
تنفيذ عملية السحب.

.......................................................................................................

```
void Widget::dragEnterEvent(QDragEnterEvent *event)
```
هى الدالة المسؤولة عن حدث ( بدأ السحب ) .

```
if (event->mimeData()->hasFormat("text/plain"))
    event->acceptProposedAction();
```
إذا كانت نوع البيانات المنقولة بواسطة **MIME** من النوع **text/plain**.
يتم الموافقة على إجراء العملية.

```
void Widget::dropEvent(QDropEvent *event)
```
هى الدالة المسؤولة عن حدث ( إنتهاء عملية السحب و الإسقاط ) .

```
QStringList strlist;
```
الإعلان عن المتغير **strlist** لإدارة فصيلة من النوع **QStringList**.

```
strlist = event->mimeData()->text().split("---");
```
الدالة **event->mimeData()->text()** تقوم بإستقبال البيانات المرسلة .

ولنفرض أن  النص المرسل **"label_2---Box 2"** سيتم فصله بـ **split("---")** إلى :

**strlist[0]= label_2** وهو يمثل اسم الكائن ( **objectname** ).

**strlist[1]= Box 2** وهو يمثل النص الموجود داخل الكائن ( **objecttext** ).

```
DropedLabel = this->findChild<QLabel *>(strlist[0]);
```
**DropedLabel**  يشير إلى الكائن الذى يحمل الأسم ( **objectname** )  المرسل من **MIME**.

```
DropedLabel->setGeometry(  event->pos().x()-75,event->pos().y()-25,150,50  );
```
تغيير إحداثيات الكائن ووضعه فى الإحداثيات **X,Y** الخاصة بمكان حدث الإسقاط.

```
DropedLabel->setText(strlist[1]);
```
إضافة النص للكائن.

```
event->acceptProposedAction();
```
يتم الموافقة على إتمام العملية.

.......................................................................

# عرض البيانات

# طريقة نموذج / عرض

# Model/View programmig

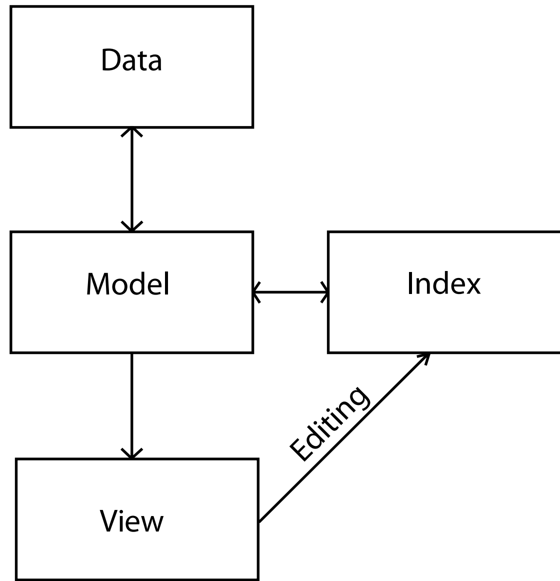
طريقة نموذج /عرض ( Model / View Programming ) هى طريقة عرض البيانات من خلال فصائل الواجهة الرسومية ل**كيوت**.

فى الإصدارات السابقة ل**كيوت** ( ما قبل الإصدار 4.4 ) كانت **كيوت** تقوم بعرض البيانات من خلال فصائل تقوم بتخزين البيانات وعرضها أيضاً فى نفس الوقت، و منذ الإصدار 4.4 قامت **كيوت** بتغيير الميكانيكية الخاصة بعرض البيانات وطبقت طريقة نموذج / عرض.

تعتمد هذه الطريقة فى الأساس على الفصل ما بين الفصائل التى تقدم نموذج البيانات والفصائل التى تقوم بعرضها، وهذا الفصل يتيح حرية التعامل مع الفصائل التى تقدم نموذج البيانات، حيث يمكن عرض نفس البيانات على أكثر من فصيلة عرض دون الحاجة إلى نسخ البيانات داخل كلاً منها، مما يؤدى إلى تقليل إستهلاك الذاكرة.

تكوين الهيكل الأساسى لطريقة نموذج / عرض :

- النماذج Models.
- العرض Views.
-المؤشر QModelIndex.



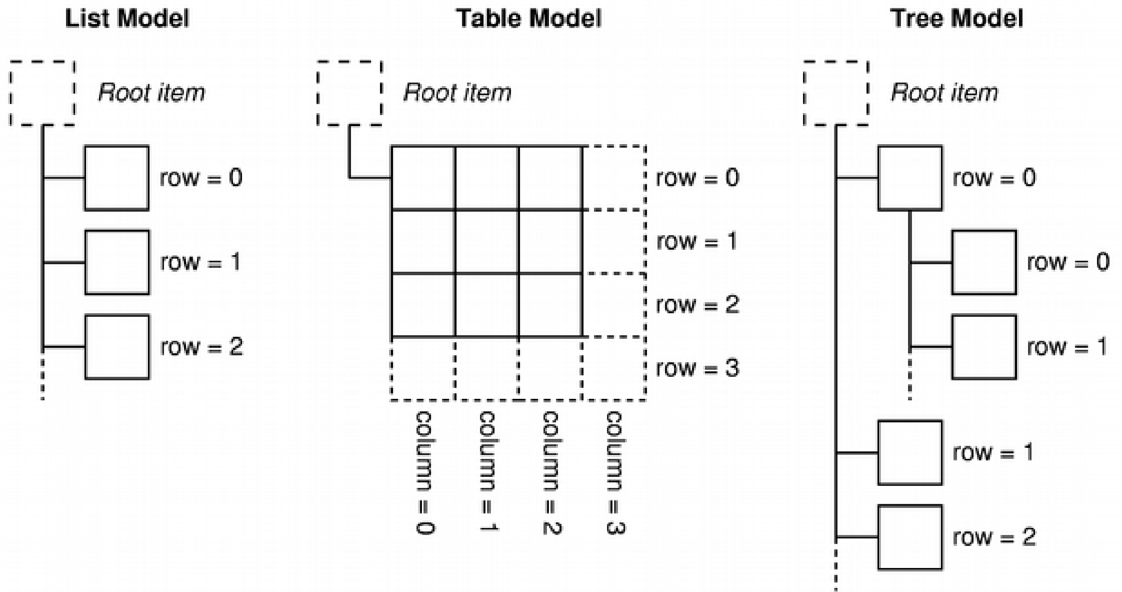
ميكانيكية العمل تتم عن طريق تحليل البيانات، وإختيار فصيلة النموذج السليم لإحتوائها، ثم تقوم فصائل العرض بالربط مع فصيلة النموذج لعرض البيانات، ويتم الربط بإستخدام دوال الإرسال و الإستقبال ( Signals and Slots ) .

## النماذج Models:

الفصيلة QAbstractItemModel هى فصيلة البنية الأساسية المكونة لفصائل النماذج، والتى تقوم بجلب البيانات سواء من قاعدة بيانات أو مباشرة من ملف أو من أى وحدة إدخال بيانات، ثم وضعها فى نموذج أو شكل معين.

وكمثال على ذلك نقوم بالتعامل مع ثلاثة نماذج للبيانات :

- نموذج القائمة List Model.
- نموذج الجدول Table Model.
- نموذج الشجرة Tree Model.




فإذا كان لدينا بيانات مثل قائمة أسماء فيمكن وضعها فى نموذج القائمة، أما إذا كان لدينا بيانات مثل أسماء منتجات ، أسعار، تاريخ إنتاج فيمكن وضعها فى نموذج الجدول،

وتعتبر بيانات الملفات والمجلدات هى أشهر أمثلة لنموذج الشجرة.

وتأتى **كيوت** بمجموعة فصائل جاهزة ترث الفصيلة QAbstractItemModel لتتعامل مع معظم أشكال البيانات ومنها:

**QStringListModel :**

وتستسخدم لحفظ وإدارة قائمة من العناصر النصية QString.

**QStandardItemModel :**

وتستخدم لحفظ وإدارة البيانات التى تتخذ شكل جدول أو شجرى، ويستخدم معها فصيلة QStandardItem لإدارة العناصر داخل الجدول.

**QFileSystemModel :**

وهى فصيلة خاصة تقدم البيانات الخاصة بملفات و مجلدات النظام.

QSqlQueryModel, QSqlTableModel, and QSqlRelationalTableModel :

وهى مجموعة فصائل لإدارة البيانات من قواعد البيانات ( SQL ).

ماذا إذا كان شكل البيانات المراد التعامل معها لاتأخذ أى من الأشكال السابقة ؟

فى هذه الحالة يمكن عمل النموذج ( Model ) الخاص بك عن طريق توليد فصيلة ترث الفصيلة QAbstractItemModel، وبها شكل أو هيكل البيانات الذى تريد.

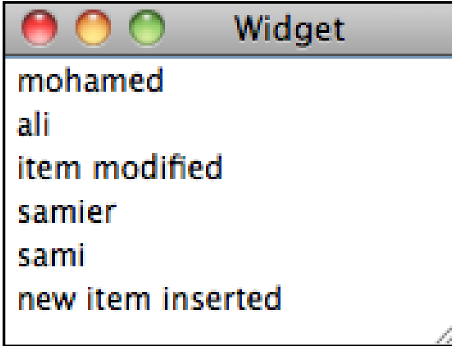
..............................................................................

## العـــرض Views:

الفصيلة QAbstractItemView هى فصيلة البنية الأساسية المكونة لفصائل العرض ، والتى تقوم بالربط مع فصائل النماذج ( Models ) لجلب البيانات، وتقوم كذلك بعرضها على شاشة المستخدم، وتقدم لنا **كيوت** فصائل مختلفة لعرض البيانات منها:

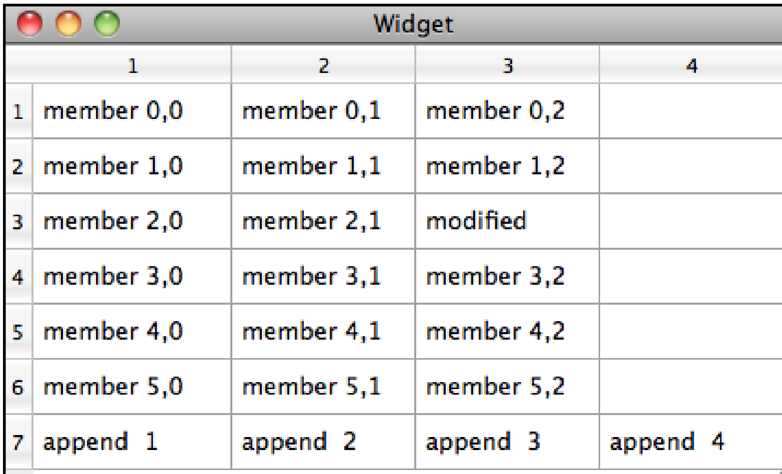
### QListView :

تقوم بعرض قائمة عناصر.



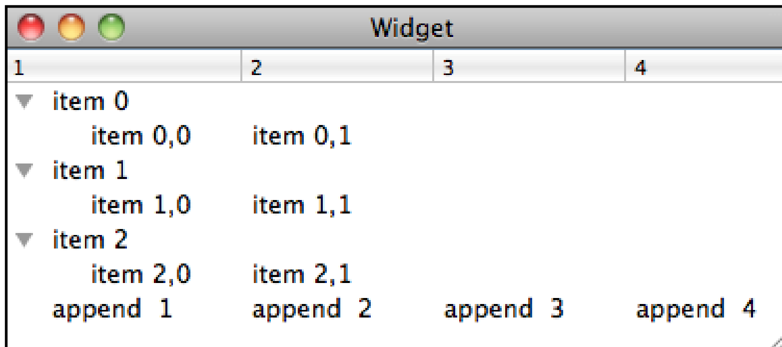

### QTableView :

تقوم بعرض عناصر فى نموذج جدول.




### QTreeView :

تقوم بعرض عناصر فى نموذج شجرى.

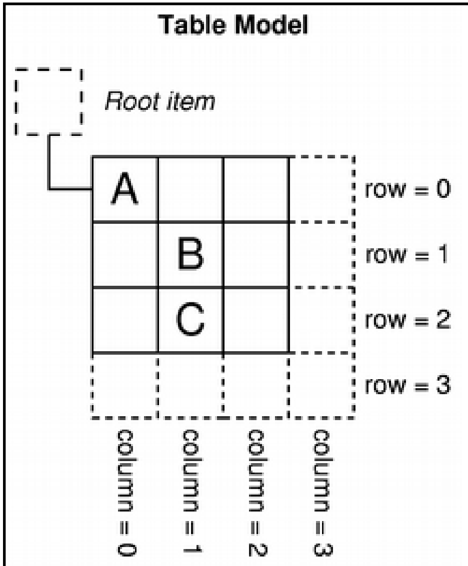


المؤشر QModelIndex :

الفصيلة QModelIndex هى فصيلة تقوم بالإشارة إلى عنصر ما داخل النموذج Model، وذلك للوصول إلى العنصر المطلوب لتغيير قيمته أو مسحه أو للقيام بأى إجراء آخر. وتتعامل هذه الفصيلة مع جميع فصائل النماذج Models Classes، ويحدد مكان العنصر بثلاث قيم ( السطر Row والعمود Column و التفرع root ) ، حيث QModelIndex() تمثل الفرع الرئيسى، ونقتبس هذا الجزء من الملفات المساعدة الآتية مع ***كيوت***.

تعرف بالصيغة التالية

```
QModelIndex index = model->index( Row , Column , root );
```




```
QModelIndex indexA =
        model->index(0, 0, QModelIndex());
```

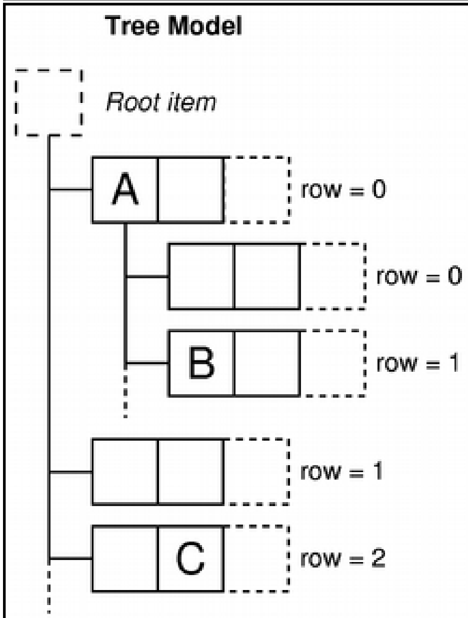
indexA تشير إلى المكان A وهو بالسطر 0 و العمود 0.

```
QModelIndex indexB =
        model->index(1, 1, QModelIndex());
```

indexB تشير إلى المكان B وهو بالسطر 1 و العمود 1.

```
QModelIndex indexC =
        model->index(2, 1, QModelIndex());
```

indexC تشير إلى المكان C وهو بالسطر 2 و العمود 1.




```
QModelIndex indexA =
        model->index(0, 0, QModelIndex());
```

indexA تشير إلى المكان A وهو بالسطر 0 و العمود 0.

```
QModelIndex indexC =
        model->index(2, 1, QModelIndex());
```

indexC تشير إلى المكان C وهو بالسطر 2 و العمود 1.

```
QModelIndex indexB =
        model->index(1, 0, indexA);
```

indexB تشير إلى المكان B وهو بالسطر 1 و العمود 0 من التفرع indexA.

لتطبيق بعض الأمثلة البسيطة على طريقة نموذج/عرض سنقوم بفتح مشروع جديد على Qt Creator من النوع QWidget لكل مثال،
وسنقوم بالتعديل فقط فى ملف widget.cpp.

أولاً : تكوين وعرض قائمة عناصر: widget.cpp

EXAMPLE NO 29

```
#include “widget.h“
#include “ui_widget.h“
#include <QStringListModel>
#include <QListView>
#include <QStandardItem>
#include <QModelIndex>

Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
   ui->setupUi(this);
   QStringList strlist;
   strlist << “mohamed“ << “ali“ << “ahmed“ << “samier“ << “sami“;

   QStringListModel *mymodel = new QStringListModel;
   mymodel->setStringList(strlist);

   QListView *myview = new QListView(this);
   myview->setModel(mymodel);

   QModelIndex myindex;

   mymodel->insertRow(mymodel->rowCount());

   myindex = mymodel->index(mymodel->rowCount()-1);

   mymodel->setData(myindex,“new item inserted“);

   myindex = mymodel->index(2);
   mymodel->setData(myindex, “item modified“);
}
```

شرح الكود السابق

```
QStringList strlist;
strlist << “mohamed“ << “ali“ << “ahmed“ << “samier“ << “sami“;
```

إنشاء قائمة تضم عناصر من النوع **QString**.

```
QStringListModel *mymodel = new QStringListModel;
```

الإعلان عن المتغير **mymodel** لإدارة فصيلة **QStringListModel**،
وهى تقدم نموذج مصمم لإحتواء قائمة عناصر .

```
mymodel->setStringList(strlist);
```

إدخال قائمة العناصر **strlist** للنموذج **mymodel**.

| 0 | mohamed |
|---|---|
| 1 | ali |
| 2 | ahmed |
| 3 | samier |
| 4 | sami |

```
QListView *myview = new QListView(this);
```

الإعلان عن المتغير **myview** لإدارة فصيلة **QListView**، وهى مصممة لعرض نموذج قائمة عناصر .

```
myview->setModel(mymodel);
```

توجيه النموذج **mymodel** إلى كائن عرض النموذج **myview** ليقوم بعرضه.

```
QModelIndex myindex;
```

الإعلان عن المتغير **myindex** لإدارة فصيلة **QModelIndex**، وهى تقوم بالإشارة إلى العناصر .

```
mymodel->insertRow(mymodel->rowCount());
```

إضافة سطر جديد فى آخر النموذج **mymodel**.

| 0 | mohamed |
|---|---|
| 1 | ali |
| 2 | ahmed |
| 3 | samier |
| 4 | sami |
| 5 | |

```
myindex = mymodel->index(mymodel->rowCount()-1);
```

**myindex** يشير إلى آخر سطر بالنموذج **mymodel**.

| | |
|---|---|
| 0 | mohamed |
| 1 | ali |
| 2 | ahmed |
| 3 | samier |
| 4 | sami |
| 5 | |

myindex → 5

```
mymodel->setData(myindex,"new item inserted");
```

وضع البيان **new intem inserted** فى المكان المشار إليه بواسطة **myindex**، وهو آخر سطر بالنموذج.

| | |
|---|---|
| 0 | mohamed |
| 1 | ali |
| 2 | ahmed |
| 3 | samier |
| 4 | sami |
| 5 | new item inserted |

myindex → 5

```
myindex = mymodel->index(2);
```

تغيير **myindex** ليشير إلى السطر رقم **2** بالنموذج ، ويعتبر السطر الثالث لأن العد يبدأ من 0 ، 1 ، 2 ،..

| | |
|---|---|
| 0 | mohamed |
| 1 | ali |
| 2 | ahmed |
| 3 | samier |
| 4 | sami |
| 5 | new item inserted |

myindex → 2

```
mymodel->setData(myindex, "item modified");
```

وضع البيان **item modified** فى المكان المشار إليه بواسطة **myindex** وهو السطر الثالث بالنموذج.

| | |
|---|---|
| 0 | mohamed |
| 1 | ali |
| 2 | item modified |
| 3 | samier |
| 4 | sami |
| 5 | new item inserted |

myindex → 2

ثانياً : تكوين وعرض عناصر فى جدول: widget.cpp

EXAMPLE NO 30

```
#include "widget.h"
#include "ui_widget.h"
#include <QStandardItemModel>
#include <QTableView>
#include <QStandardItem>
#include <QModelIndex>

Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
   ui->setupUi(this);

   QList<QStandardItem*> itemlist;
   itemlist << new QStandardItem("append  1");
   itemlist << new QStandardItem("append  2");
   itemlist << new QStandardItem("append  3");
   itemlist << new QStandardItem("append  4");
   QStandardItemModel *mymodel = new QStandardItemModel;
   mymodel->setColumnCount(3);

   for(int i = 0 ; i < 6 ;i++)
  {
      for(int j = 0 ; j < mymodel->columnCount() ;j++)
       {
         mymodel->setItem(i,j,new QStandardItem(   QString("member %0,%1").arg(i).arg(j)   ));
       }
   }

   mymodel->appendRow(itemlist);
   QTableView *myview = new QTableView(this);
   myview->setGeometry(0,0,500,300);
   myview->setModel(mymodel);
   QModelIndex myindex;
   myindex = mymodel->index(2,2);
   mymodel->setData(myindex,"modified");
}
```

شرح الكود السابق

```
QList<QStandardItem*> itemlist;
itemlist << new QStandardItem("append 1");
itemlist << new QStandardItem("append 2");
itemlist << new QStandardItem("append 3");
itemlist << new QStandardItem("append 4");
```

إنشاء قائمة تضم عناصر من النوع **QStandardItem**.

```
QStandardItemModel *mymodel = new QStandardItemModel;
```

الإعلان عن المتغير **mymodel** لإدارة فصيلة **QStandardItemModel**،
وهى تقدم نموذج مصمم لإحتواء عناصر داخل جدول أو على شكل شجرى.

```
mymodel->setColumnCount(3);
```

وضع عدد أعمدة النموذج وهى **3** أعمدة.

```
QListView *myview = new QListView(this);
```

الإعلان عن المتغير **myview** لإدارة فصيلة **QListView**، وهى مصممة لعرض نموذج قائمة عناصر.

```
for(int i = 0 ; i < 6 ;i++)          {
 for(int j = 0 ; j < mymodel->columnCount() ; j++ ) {
mymodel->setItem( i , j
            ,new QStandardItem( QString("member %0,%1").arg(i).arg(j)  ));
    }
}
```

كود تكرارى لوضع العناصر بالترتيب داخل كل خلية،
حيث يتكون لدينا جدول من **6** صفوف و **3** أعمدة و إضافة عنصر لكل خلية.

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | member 0,0 | member 0,1 | member 0,2 |
| 2 | member 1,0 | member 1,1 | member 1,2 |
| 3 | member 2,0 | member 2,1 | member 2,2 |
| 4 | member 3,0 | member 3,1 | member 3,2 |
| 5 | member 4,0 | member 4,1 | member 4,2 |
| 6 | member 5,0 | member 5,1 | member 5,2 |

```
mymodel->appendRow(itemlist);
```

إضافة سطر فى آخر النموذج يحتوى على قائمة العناصر **itemlist**، ونلاحظ هنا أن عدد أعمدة الجدول ثلاثة، و عدد العناصر بالقائمة **itemlist** أربعة، وبالتالى فعند الإضافة سيقوم النموذج بإضافة عمود رابع تلقائياً ليحتوى العنصر الرابع من القائمة **itemlist**.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | member 0,0 | member 0,1 | member 0,2 | |
| 2 | member 1,0 | member 1,1 | member 1,2 | |
| 3 | member 2,0 | member 2,1 | member 2,2 | |
| 4 | member 3,0 | member 3,1 | member 3,2 | |
| 5 | member 4,0 | member 4,1 | member 4,2 | |
| 6 | member 5,0 | member 5,1 | member 5,2 | |
| 7 | append 1 | append 2 | append 3 | append 4 |

```
QTableView *myview = new QTableView(this);
```

الإعلان عن المتغير **myview** لإدارة فصيلة **QTableView**، وهى مصممة لعرض عناصر داخل جدول.

```
myview->setGeometry(0,0,500,300);
```

لتحديد حجم رسم الجدول على الشاشة.

```
myview->setModel(mymodel);
```

توجيه النموذج **mymodel** إلى كائن عرض النموذج **myview** ليقوم بعرضه.

```
QModelIndex myindex;
```

الإعلان عن المتغير **myindex** لإدارة فصيلة **QModelIndex** وهى تقوم بالإشارة إلى العناصر .

```
myindex = mymodel->index(2,2);
```

توجيه **myindex** ليشير إلى العنصر الموجود بالصف **2** و العمود **2** داخل النموذج **mymodel** .

```
mymodel->setData(myindex,"modified");
```

تغيير قيمة العنصر المشار إليه بواسطة **myindex** للقيمة **modified**.

ثالثاً : تكوين وعرض عناصر على شكل شجرى: widget.cpp

EXAMPLE NO 31

```
#include "widget.h"
#include "widget.h"
#include "ui_widget.h"
#include <QStandardItemModel>
#include <QTreeView>
#include <QStandardItem>
#include <QModelIndex>
#include <QDebug>

Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
    ui->setupUi(this);
    QList<QStandardItem*> itemlist;
    itemlist << new QStandardItem("append  1");
    itemlist << new QStandardItem("append  2");
    itemlist << new QStandardItem("append  3");
    itemlist << new QStandardItem("append  4");
    QStandardItemModel *mymodel = new QStandardItemModel;
    mymodel->setColumnCount(2);
    QModelIndex myindex ;
    for(int i = 0 ; i < 3 ;i++){
        QStandardItem *item = new QStandardItem(   QString("item %0").arg(i) ) ;
        mymodel->setItem(i,item);
        item->setColumnCount(2);
        item->insertRow(0,new QStandardItem);
        for(int j = 0 ; j < item->columnCount() ;j++){
            myindex = mymodel->index(0,j,item->index());
            mymodel->setData(myindex,QString("item %0,%1").arg(i).arg(j) );
         }
    }
   mymodel->appendRow(itemlist);
    QTreeView *myview = new QTreeView(this);
    myview->setGeometry(0,0,500,300);
    myview->setModel(mymodel);
}
```

## شرح الكود السابق

```
QList<QStandardItem*> itemlist;
itemlist << new QStandardItem("append  1");
itemlist << new QStandardItem("append  2");
itemlist << new QStandardItem("append  3");
itemlist << new QStandardItem("append  4");
```

إنشاء قائمة تضم عناصر من النوع **QStandardItem**.

```
QStandardItemModel *mymodel = new QStandardItemModel;
```

الإعلان عن المتغير **mymodel** لإدارة فصيلة **QStandardItemModel**،
وهى تقدم نموذج مصمم لإحتواء عناصر داخل جدول أو على شكل شجرى .

```
mymodel->setColumnCount(2);
```

وضع عدد أعمدة النموذج وهى عدد 2 عمود

| 1 | 2 |
|---|---|
| | |

```
QModelIndex myindex;
```

الإعلان عن المتغير **myindex** من فصيلة **QModelIndex**، وهى تقوم بالإشارة إلى العناصر .

```
for(int i = 0 ; i < 3 ;i++){
QStandardItem *item = new QStandardItem(   QString("item %0").arg(i) ) ;
```

الإعلان عن المتغير **item** لإدارة فصيلة **QStandardItem**، وهى تمثل عنصر جديد.

```
mymodel->setItem(i,item);
```

يتم وضع العنصر **item** داخل النموذج فى القائمة الرئيسية فى السطر **i**.

| 1 | 2 |
|---|---|
| item 0 | |

```
item->setColumnCount(2);
```

يتم تحديد عدد **2** عمود للعنصر **item**.

```
item->insertRow(0,new QStandardItem);
```

إضافة سطر فرعى من سطر العنصر **item**.

```
for(int j = 0 ; j < item->columnCount() ; j++ ){
	myindex = mymodel->index(0,j,item->index());
```

توجيه **myindex** لتشير إلى العنصر عند السطر **0** ، العمود **j** المتفرعين من العنصر **item**.

```
	mymodel->setData(myindex,QString("item %0,%1").arg(i).arg(j) );
```

وضع القيمة المطلوبة عند العنصر المشار إليه بواسطة **myindex**.

```
  }

}
```

| 1 | 2 |
|---|---|
| item 0 | |
| └ item 0,0 —— | item 0,1 |

بعد نهاية العملية التكرارية السابقة نحصل على الآتى:

| 1 | 2 |
|---|---|
| item 0 | |
| └ item 0,0 —— | item 0,1 |
| item 1 | |
| └ item 1,0 —— | item 1,1 |
| item 2 | |
| └ item 2,0 —— | item 2,1 |

mymodel->appendRow(itemlist);

إضافة سطر فى آخر النموذج يحتوى على قائمة العناصر **itemlist** ، ونلاحظ هنا أن عدد أعمدة الجدول إثنان، و عدد العناصر بالقائمة **itemlist** أربعة، وبالتالى فعند الإضافة سيقوم النموذج بإضافة عمودين ثالث و رابع تلقائياً ليحتوى العنصر الرابع من القائمة **itemlist**.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| item 0 | | | |
| └── item 0,0 ——— | item 0,1 | | |
| item 1 | | | |
| └── item 1,0 ——— | item 1,1 | | |
| item 2 | | | |
| └── item 2,0 ——— | item 2,1 | | |
| append 1 | append 2 | append 3 | append 4 |

QTreeView *myview = new QTreeView(this);

الإعلان عن المتغير **myview** لإدارة فصيلة **QTreeView**، وهى مصممة لعرض عناصر على شكل شجرى.

myview->setGeometry(0,0,500,300);

لتحديد حجم رسم الجدول على الشاشة.

myview->setModel(mymodel);

توجيه النموذج **mymodel** إلى كائن عرض النموذج **myview** ليقوم بعرضه.

..........................................................................

هناك بعض الفصائل المرتبطة بعرض البيانات بأسلوب نموذج / عرض والتى لم نتطرق لها بالشرح، مثل فصيلة QAbstractItemDelegate، وفصيلة QStyledItemDelegate، وهى تقوم بعرض وتعديل البيانات رسومياً، ويوجد لها أمثلة عديدة داخل QtDemo.

Notes

# الرسم فى كيوت

# Graphics View Framework

## الرسم فى **كيوت** Graphics View Framework :

ما نتحدث عنه الآن هى الطريقة التى تعتمدها **كيوت** فى الرسم ثنائى الأبعاد 2D ، وتعتمد هذه الطريقة على نفس إسلوب نموذج / عرض ( Model / View ) فى إدارة العناصر، ولنقوم بالرسم ثنائى الأبعاد بهذه الطريقة يلزمنا معرفة ميكانيكية العمل والفصائل المطلوب التعامل معها.

هناك ثلاث فصائل أساسية لا غنى عنها لتكوين هيكل متكامل للرسم ثنائى الأبعاد:

-فصيلة المشهد ( QGraphicsScene Class ) :

هى الفصيلة المسؤولة عن إحتواء و إدارة العناصر الرسومية.

-فصيلة العرض ( QGraphicsView Class ) :

هى الفصيلة المسؤولة عن عرض المشهد الكلى أوجزء منه.

-فصيلة العناصر الرسومية ( QGraphicsItem Class ) :

هذه الفصيلة يتم التعامل معها كعناصر داخل الفصيلة QGraphicsScene.

الصورة الكلية هى المشهد العام Scene، ويتم التعامل معه من خلال الفصيلة QGraphicsScene، وأى رسم بداخلها هو عنصر Item، ويتم التعامل معه من خلال الفصيلة QGraphicsItem، وشاشة العرض للمشهد هى View، ويتم التعامل معه من خلال الفصيلة QGraphicsView، ويجب ملاحظة أن المشهد العام scene هو موجود بالذاكرة، ويتم عرضه من خلال View، ويمكن أيضاً تحريك شاشة العرض View لعرض أى جزء من المشهد Scene.

# QGraphicsScene Class

**نسب الفصيلة :**

ترث فصيلة QObject

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن إحتواء وإدارة العناصر الرسومية،
وتمتاز هذه الفصيلة بالخواص الآتية :

- قدرتها السريعة فى إدارة أعداد كبيرة جداً من العناصر الرسومية قد تصل إلى عدة ملايين من العناصر.

- قيامها بنقل الأحداث ( نقر الفأرة أو لوحة المفاتيح ) لكل عنصر رسومى.

- قيامها بإدارة حالة العنصر الرسومى من تحديد و إختيار.

**طريقة الإعـــلان ( Declaration ) :**

**QGraphicsScene scene;**

**وظائف الفصيلة :**

تحتوى هذه الفصيلة على مجموعة كبيرة من الدوال الخاصة بإضافة العناصر الرسومية، مثل إضافة خط addLine ، مستطيل addRect ، أو شكل بيضاوى addEllipse.

كما أن بها دالة addItem التى يمكن عن طريقها إضافة أى عنصر رسومى منحدر من الفصيلة QGraphicsItem للمشهد Scene.

وبها أيضاً الدالة ( item( x , y  التى تقوم بإسترجاع العنصر الرسومى الموجود بالإحداثيات ( x , y )، ودوال أخرى كثيرة لإدارة العناصر الرسومية الموجودة داخل المشهد Scene.

..........................................................................................

# QGraphicsView Class

**نسب الفصيلة :**

ترث فصيلة QAbstractScrollArea
و QAbstractScrollArea ترث QFrame
و QFrame ترث QWidget

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن عرض المشهد الكلى أو جزء منه، وهى بمثابة شاشة العرض لفصيلة المشهد Scene، حيث أن فصيلة المشهد تكون فى ذاكرة الجهاز ولا تعرض على الشاشة، ويتم عرضها من خلال هذه الفصيلة QGraphicsView، وبالتالى يمكن لهذه الفصيلة عرض كامل المشهد أو عرض جزء منه، وذلك على حسب المساحة المخصصة للعرض.

**طريقة الإعــلان ( Declaration ) :**

**QGraphicsView view;**

**وظائف الفصيلة :**

من الدوال الأساسية الخاصة بهذه الفصيلة الدالة setScene، وهى الدالة الخاصة بوضع فصيلة المشهد داخل فصيلة العرض.

ودوال أخرى هدفها هو التحكم التام فى جميع خصائص عرض المشهد من دوران شاشة العرض أو عمل تكبير أو تصغير للمشهد.

..............................................................................

# QGraphicsItem Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هذه الفصيلة يتم التعامل معها كعناصر داخل الفصيلة QGraphicsScene.

**طريقة الإعــلان ( Declaration ) :**

**QGraphicsItem item;**

**وظائف الفصيلة :**

هذه الفصيلة هى فصيلة البنية الأساسية لأى عنصر رسومى يمكن وضعه داخل فصيلة المشهد Scene، ولا تستقبل فصيلة المشهد أى نوع آخر من العناصر.

وقد قدمت ***كيوت*** عدة فصائل مشتقة من الفصيلة QGraphicsItem للرسومات الأساسية مثل:

- فصيلة QGraphicsLineItem ترث فصيلة QGraphicsItem : لرسم الخطوط.
- فصيلة QGraphicsRectItem ترث فصيلة QGraphicsItem : لرسم المستطيلات.
- فصيلة QGraphicsTextItem ترث فصيلة QGraphicsItem : لرسم الحروف.

وأهمية هذه الفصيلة تأتى عندما يقوم المبرمج بعمل فصيلة الرسم الخاصة به، ويتم إضافتها كعنصر لفصيلة المشهد.
فيمكن مثلا إنتاج فصيلة ترث فصيلة QGraphicsItem، وتقوم برسم سيارة، وبالتالى يمكن إضافة رسم السيارة للمشهد والتعامل معه كعنصر.
وتتعامل هذه الفصيلة مع العنصر من دوران أو تكبير أو تصغير، فهى تقوم بالعمل على العنصر فقط ولا دخل لها بالمشهد أو إحداثيات المشهد.

.............................................................................

لتطبيق بعض الأمثلة البسيطة على الرسم سنقوم بفتح مشروع جديد على Qt Creator من النوع QWidget .

وسنقوم بالتعديل فقط فى ملف widget.cpp.

widget.cpp

EXAMPLE NO 32

```
#include "widget.h"
#include "ui_widget.h"
#include <QtGui>

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    QGraphicsScene *scene = new QGraphicsScene(0,0,2000,2000,this);
    scene->addRect(QRectF(0, 0, 100, 100));
    scene->addRect(QRectF(1800, 1800, 100, 100));

    QGraphicsView *view1 = new QGraphicsView(scene,this);
    view1->setGeometry(0,0,250,250);
    view1->setSceneRect(0,0,200,200);

    QGraphicsView *view2 = new QGraphicsView(scene,this);
    view2->setGeometry(251,0,250,250);
    view2->setSceneRect(1800,1800,200,200);

    QGraphicsView *view3 = new QGraphicsView(scene,this);
    view3->setGeometry(502,0,250,250);

    QGraphicsItem *item = scene->itemAt(50, 50);
    item->rotate(10);
}
```

شرح الكود السابق

```
QGraphicsScene *scene = new QGraphicsScene(0,0,2000,2000,this);
```

الإعلان عن المتغير **scene** لإدارة فصيلة **QGraphicsScene**، وهى عبارة عن المشهد الذى سيتم العمل عليه، وقد تم إدخال مساحة المشهد وهى عرض **2000** نقطة و إرتفاع **2000** نقطة.

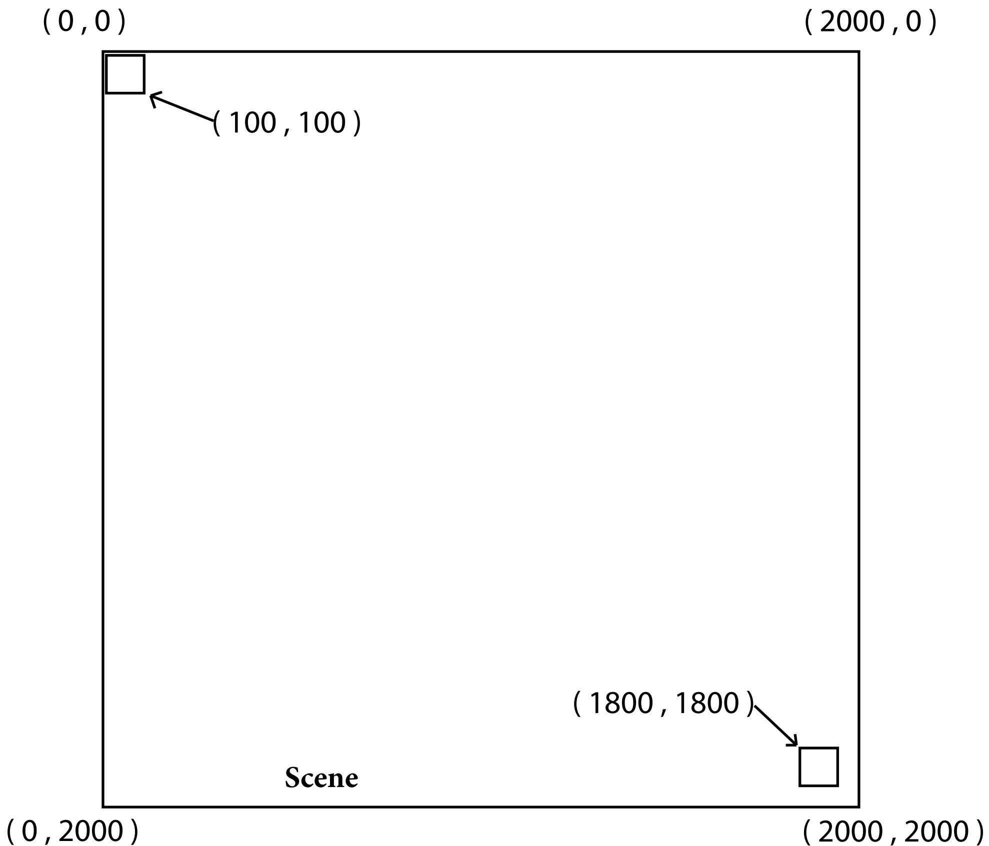
```
scene->addRect(QRectF(0, 0, 100, 100));
```

إضافة مربع للمشهد يبدأ من الإحداثيات ( **0** ، **0** ) وعرضه **100** نقطة وإرتفاعه **100** نقطة.

```
scene->addRect(QRectF(1800, 1800, 100, 100));
```

إضافة مربع للمشهد يبدأ من الإحداثيات ( **1800** ، **1800** ) وعرضه **100** نقطة وإرتفاعه **100** نقطة.

ولا ننسى أننا لا نرى المشهد على الشاشة إلا من خلال فصيلة العرض **QGraphicsView**، وهذا المشهد يتم تكوينه فى ذاكرة الجهاز، وسيكون المشهد المفترض كالتالى:

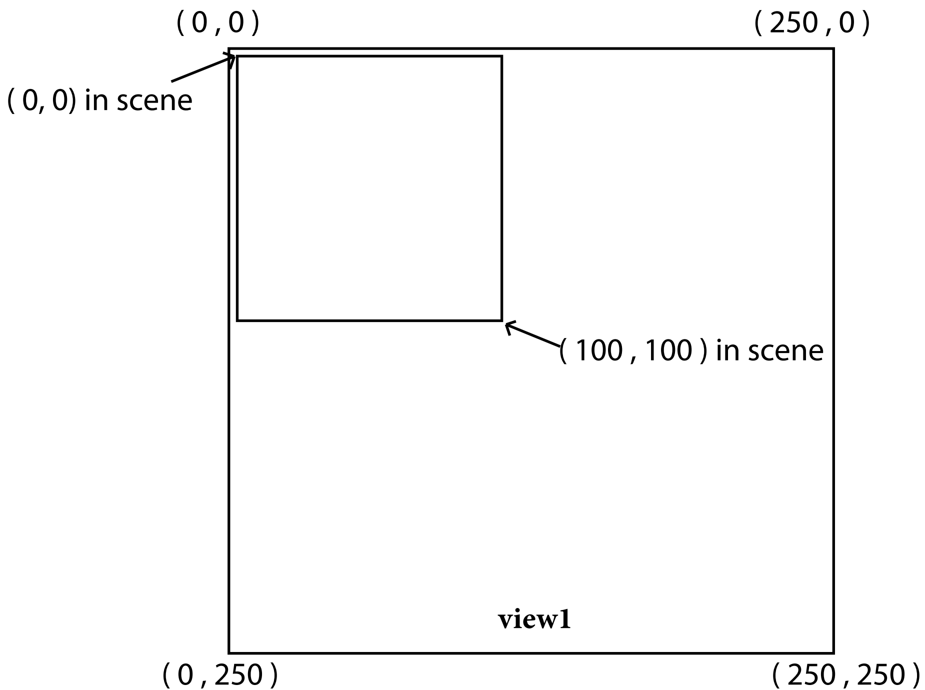


```
QGraphicsView *view1 = new QGraphicsView(scene,this);
```
الإعلان عن المتغير **view1** لإدارة فصيلة **QGraphicsView** لتكون بمثابة شاشة عرض للمشهد **scene**.

```
view1->setGeometry( 0 , 0 , 250 , 250 );
```
تحديد مساحة شاشة العرض **view1** ( عرض **250** نقطة ، إرتفاع **250** نقطة ) ووضعها بالإحداثيات ( **0** ، **0** ) على النافذة الرئيسية **widget**.

```
view1->setSceneRect( 0 , 0 , 200 , 200 );
```
تحديد المساحة المطلوب عرضها من المشهد، وهى من الإحداثيات ( **0** ، **0** ) حتى عرض **200** نقطة وإرتفاع **200** نقطة.

عند التشغيل ستظهر شاشة العرض **view1** وبها المربع المرسوم فى أعلى يسار المشهد.



```
QGraphicsView *view2 = new QGraphicsView(scene,this);
```

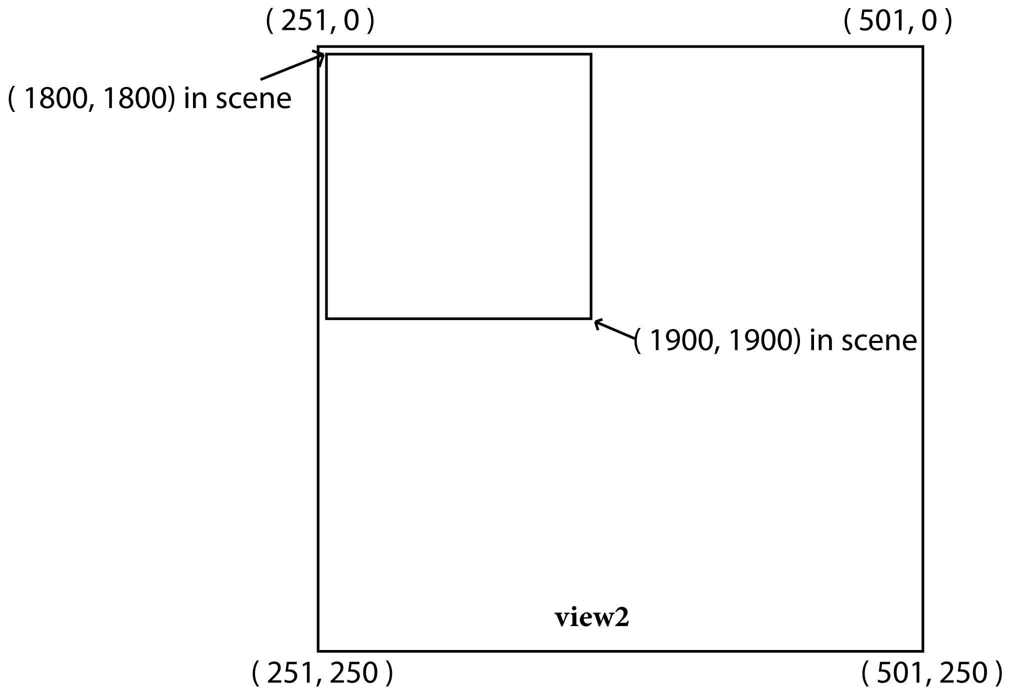
الإعلان عن المتغير **view2** لإدارة فصيلة **QGraphicsView** لتكون بمثابة شاشة عرض للمشهد **scene**.

```
view2->setGeometry( 251 , 0 , 250 , 250 );
```

تحديد مساحة شاشة العرض **view2** ( عرض **250** نقطة ، إرتفاع **250** نقطة ) ووضعها بالإحداثيات ( **0** ، **251** ) على النافذة الرئيسية **widget**، لتظهر بجانب نافذة العرض **view1**.

```
view2->setSceneRect( 1800 , 1800 , 200 , 200 );
```

تحديد المساحة المطلوب عرضها من المشهد، وهى من الإحداثيات ( **1800** ، **1800** ) حتى عرض **200** نقطة وإرتفاع **200** نقطة.



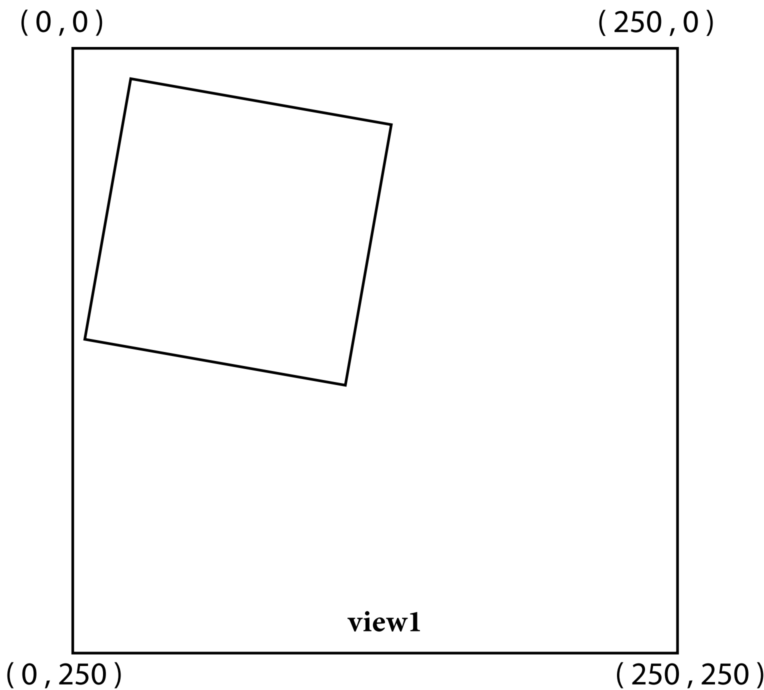
```
QGraphicsView *view3 = new QGraphicsView(scene,this);
```
الإعلان عن المتغير **view3** لإدارة فصيلة **QGraphicsView** لتكون بمثابة شاشة عرض للمشهد **scene**.

```
view3->setGeometry( 502 , 0 , 250 , 250 );
```
تحديد مساحة شاشة العرض **view3** ( عرض **250** نقطة ، إرتفاع **250** نقطة ) ووضعها بالإحداثيات ( **0** ، **502** ) على النافذة الرئيسية **widget**، لتظهر بجانب نافذة العرض **view2**.

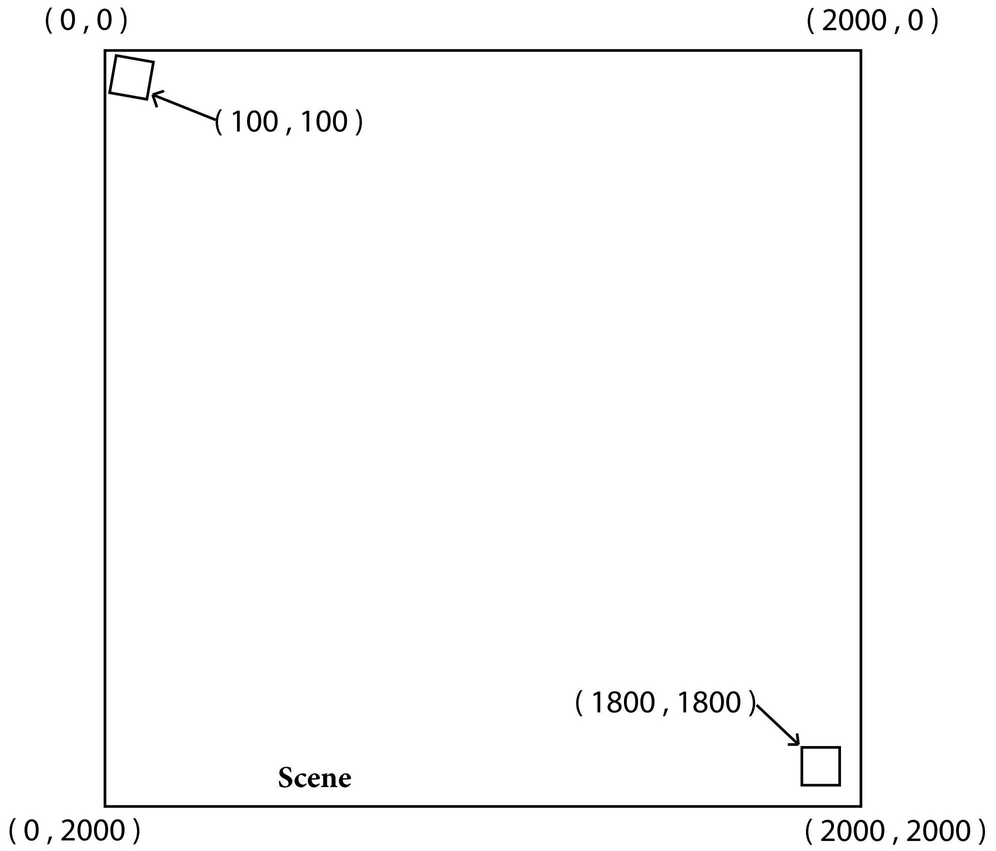
نلاحظ هنا أنه لم يتم تحديد المساحة المطلوب عرضها من المشهد، وبالتالى ستقوم **view3** بعرض المشهد كاملاً، وبما أن مساحة المشهد ( **2000** ✗ **2000** )، ومساحة شاشة العرض ( **250** ✗ **250** )فقط ، فسيظهر فى شاشة العرض شريط أفقى و شريط رأسى لتحريك المشهد داخل النافذة.

```
QGraphicsItem *item = scene->itemAt( 50 , 50 );
```
الإعلان عن المتغير **item** لإدارة فصيلة **QGraphicsItem**، والتى تقوم بإلتقاط العنصر الموجود فى نقطة إحداثيات المشهد ( **50** ، **50** )، وهو المربع الموجود أعلى يسار المشهد.

```
item->rotate( 10 );
```
يتم دوران العنصر بمقدار **10** درجات و بالتالى سنرى هذا الشكل داخل **view1**.



ويكون الشكل النهائى للمشهد **scene** داخل الذاكرة كالتالى :



# واجهة الإستخدام المتحركة

# Animation GUI Framework

واجهة الإستخدام المتحركة Animation GUI Framework :

واجهة الإستخدام المتحركة هى واحدة من مميزات ***كيوت*** الفريدة، فهذه الميزة تعمل لإضفاء الحياة على الواجهة الرسومية، فعملها ينصب بشكل أساسى على خصائص الكائنات الرسومية من مفاتيح وصناديق نصوص ( Buttons , Text Box ) ...الخ.

وتحتاج هذه الخاصية فى عملها للبيانات الآتية:

1 - الكائن الرسومى المطلوب العمل عليه ( الكائن لابد أن يرث QObject ) .

2 - خاصية الكائن الرسومى المطلوب العمل عليها ( Q_PROPERTY ).

3 - تحديد مدة الحركة و البداية والنهاية، ويمكن إضافة مفاتيح keyframe فى وسط المشهد.

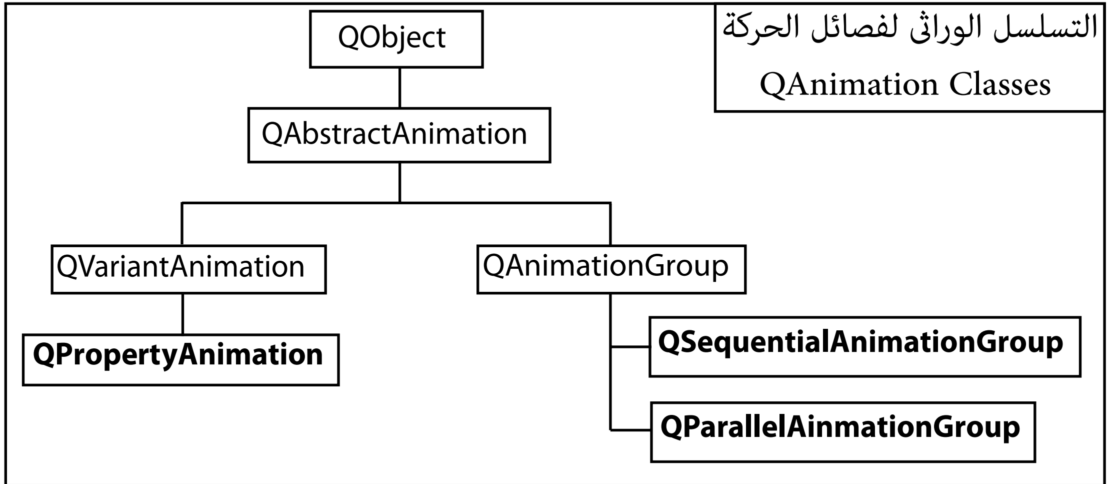
-إمكانية تحريك أكثر من كائن رسومى بإحدى الطريقتين:

- الطريقة التتابعية :

أى إمكانية تحريك الكائنات الرسومية بشكل تتابعى، فلا يبدأ الكائن حركته إلا بإنتهاء حركة الكائن السابق له.

- الطريقة المتوازية :

أى إمكانية تحريك الكائنات الرسومية بشكل متوازى، حيث تبدأ جميع الكائنات حركتها فى نفس الوقت.




وسوف يتم التعامل مباشرة مع الفصائل الآتية:

**QPropertyAnimation** : هى الفصيلة المسؤولة عن تحديد الكائن الرسومى وتحديد حركته.

**QSequentialAnimationGroup** : تستخدم لتحريك الكائنات الرسومية بشكل تتابعى.

**QParallelAnimationGroup** : تستخدم لتحريك جميع الكائنات الرسومية بالتوازى زمنياً.

مثال : لدينا مفتاحين QPushButton هما B1 , B2 تم وضعهما عند الإحداثيات ( 0 ، 0 ) .
المطلوب :
- وضع نموذج الحركة للمفتاح B1 من الإحداثيات ( 0 ، 0 ) إلى الإحداثيات ( 150 ، 150 ) مع زيادة عرض و إرتفاع المفتاح ثم رده إلى الإحداثيات ( 0 ، 0 ) مرة أخرى خلال 5 ثوان.
- وضع نموذج الحركة للمفتاح B2 من الإحداثيات ( 0 ، 0 ) إلى الإحداثيات ( 250 ، 250 ) مع زيادة عرض و إرتفاع المفتاح خلال 3 ثوان.
- تحريك المفتاحين B1 , B2 متتاليين زمنياً مرة و متوازيياً زمنياً مرة أخرى.

سنقوم بفتح مشروع جديد على Qt Creator من النوع QWidget،
وسنقوم بالتعديل فقط فى ملف widget.cpp.

widget.cpp

EXAMPLE NO 33

```
Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
    ui->setupUi(this);
    QPushButton *B1 = new QPushButton("Button B1",this);
    QPushButton *B2 = new QPushButton("Button B2",this);

    QPropertyAnimation *B_anim1 = new QPropertyAnimation(B1, "geometry");
    B_anim1->setDuration(5000);
    B_anim1->setKeyValueAt(0, QRect(0, 0, 150, 30));
    B_anim1->setKeyValueAt(0.8, QRect(150, 150, 200, 60));
    B_anim1->setKeyValueAt(1, QRect(0, 0, 150, 30));

    QPropertyAnimation *B_anim2 = new QPropertyAnimation(B2, "geometry");
    B_anim2->setDuration(3000);
    B_anim2->setStartValue(QRect(0, 0, 150, 30));
    B_anim2->setEndValue(QRect(250, 250, 200, 60));

    QSequentialAnimationGroup *S_group = new QSequentialAnimationGroup;
    S_group->addAnimation(B_anim1);
    S_group->addAnimation(B_anim2);
    S_group->start();
}
```

شرح الكود السابق

```
QPushButton *B1 = new QPushButton("Button B1",this);
QPushButton *B2 = new QPushButton("Button B2",this);
```

الإعلان عن المتغيرين **B1 , B2** لإدارة فصيلة **QPushButton** و إدراجهما داخل النافذة **widget**.

```
QPropertyAnimation *B_anim1 = new QPropertyAnimation(B1, "geometry");
```

الإعلان عن المتغير **B_anim1** لإدارة فصيلة **QPropertyAnimation** والتى تحدد نموذج الحركة للمفتاح **B1** وسيقوم بالتعامل مع الخاصية **geometry** الخاصة بالمفتاح **B1**.

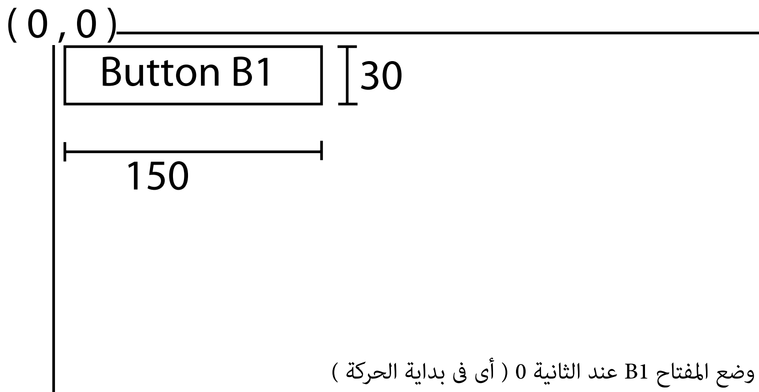
```
B_anim1->setDuration(5000);
```

يتم تحديد الفترة الزمنية لكامل الحركة بواسطة الدالة **setDuration**.
ويتم إدخال قيمة الزمن بالمللى ثانية، ففى هذا المثال **5000** مللى ثانية تساوى **5** ثوان.

الدالة **setKeyValueAt ( qreal real, const QVariant & )** تستقبل هذه الدالة قيمتين
القيمة **QVariant** وهى القيمة التى تمرر للخاصية **geometry** الخاصة بالمفتاح **B1**.
القيمة **real** وهى تكون ما بين **0** ، **1** حيث **0** هو بداية الحركة و **1** هو نهاية الحركة.

```
B_anim1->setKeyValueAt( 0 , QRect( 0 , 0 , 150 , 30 ));
```

يتم وضع المفتاح **B1** عند الإحداثيات ( **0** ، **0** ) وعرضه **150** نقطة وإرتفاعه **30** نقطة.
-- عند الزمن : ( **5000** ✗ **0** ) = **0** مللى ثانية / **1000** = **0** ثانية ( بداية الحركة ).

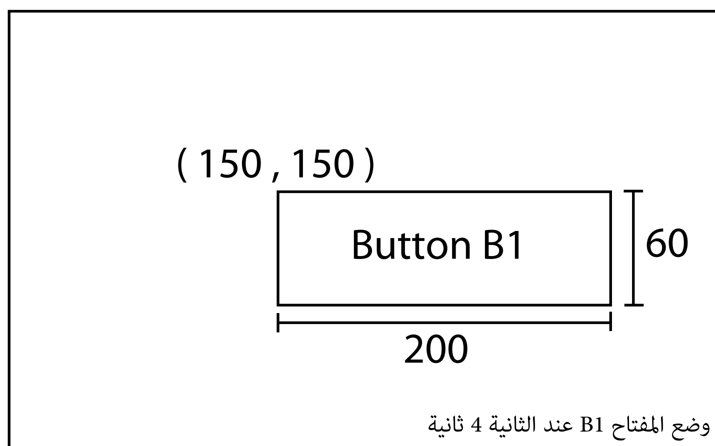


وضع المفتاح B1 عند الثانية 0 ( أى فى بداية الحركة )

```
B_anim1->setKeyValueAt( 0.8 , QRect( 150 , 150 , 200 , 60 ));
```

يتم وضع المفتاح **B1** عند الإحداثيات ( **150** ، **150** ) وعرضه **200** نقطة وإرتفاعه **60** نقطة .

-- عند الزمن : ( **5000** ✗ **0.8** ) = **4000** مللى ثانية / **1000** = **4** ثانية.

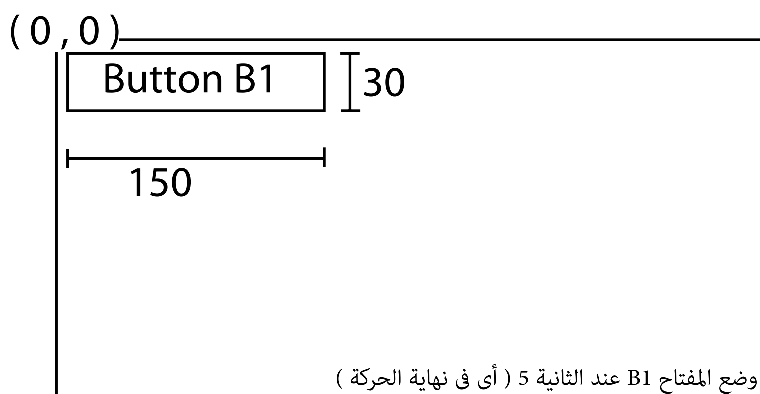


وضع المفتاح B1 عند الثانية 4 ثانية

```
B_anim1->setKeyValueAt( 1 , QRect( 0 , 0 , 150 , 30 ));
```

يتم وضع المفتاح **B1** عند الإحداثيات ( **0** ، **0** ) وعرضه **150** نقطة وإرتفاعه **30** نقطة

-- عند الزمن : ( **5000** ✗ **1** ) = **5000** مللى ثانية / **1000** = **5** ثانية ( إنتهاء الحركة )



وضع المفتاح B1 عند الثانية 5 ( أى فى نهاية الحركة )

```
QPropertyAnimation *B_anim2 = new QPropertyAnimation(B2, “geometry“);
```

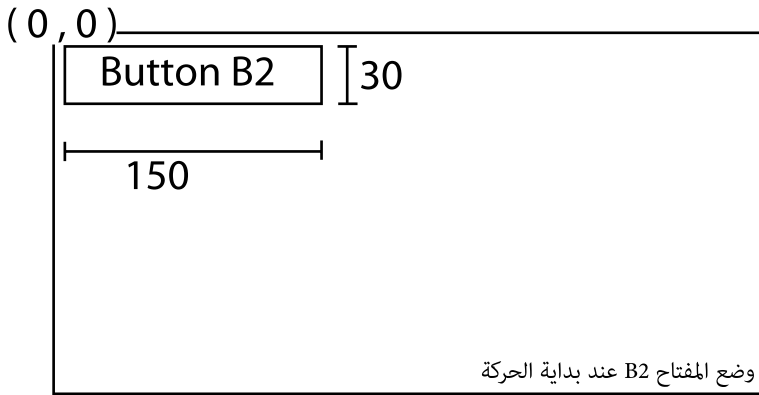
الإعلان عن المتغير **B_anim2** لإدارة فصيلة **QPropertyAnimation**، والتى تحدد نموذج الحركة للمفتاح **B2** وسيقوم بالتعامل مع الخاصية **geometry** الخاصة بالمفتاح **B2**.

```
B_anim2->setDuration(3000);
```

يتم تحديد الفترة الزمنية لكامل الحركة **3** ثانية.

```
B_anim2->setStartValue(QRect(0, 0, 150, 30));
```

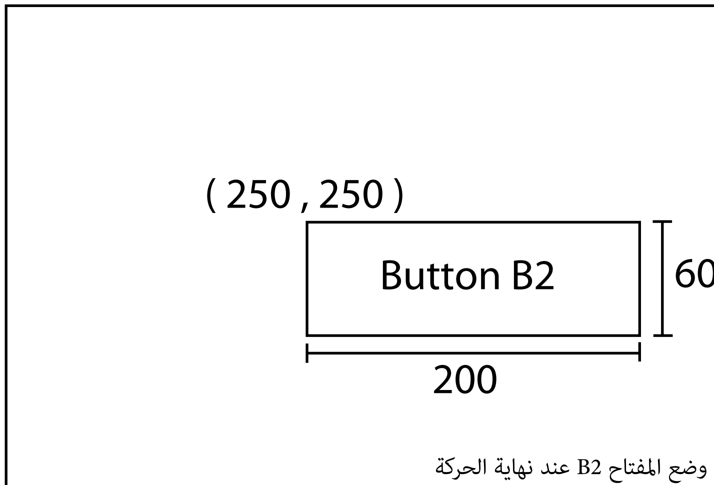
يتم وضع المفتاح **B1** عند الإحداثيات ( **0** ، **0** ) وعرضه **150** نقطة وإرتفاعه **30** نقطة فى بداية الحركة.



وضع المفتاح B2 عند بداية الحركة

```
B_anim2->setEndValue(QRect(250, 250, 200, 60));
```

وضع المفتاح **B1** عند الإحداثيات ( **250** ، **250** ) وعرضه **200** نقطة وإرتفاعه **60** نقطة فى نهاية الحركة.



وضع المفتاح B2 عند نهاية الحركة

```
QSequentialAnimationGroup *S_group = new QSequentialAnimationGroup;
```
الإعلان عن المتغير **S_group** لإدارة فصيلة **QSequentialAnimationGroup**، والتى تقوم بعرض الكائنات الرسومية بتتابع زمنى.

```
S_group->addAnimation(B_anim1);
```
إضافة نموذج الحركة **B_anim1** إلى منظم الحركة **S_group**.

```
S_group->addAnimation(B_anim2);
```
إضافة نموذج الحركة **B_anim2** إلى منظم الحركة **S_group**.

```
S_group->start();
```
بدأ عرض الحركة.

النتيجة هى حركة المفتاح **B1** أولاً وتستغرق **5** ثوانى ثم تبدأ حركة **B2** وتستغرق **3** ثوانى.

للحركة المتوزية زمنياً يمكننا تغيير سطر واحد فقط فى الكود السابق لنحصل على نموذجين حركة يبدءا معاً زمنياً وينهى المفتاح **B2** حركته قبل المفتاح **B1** بفارق **2** ثانية.

للحركة المتوازية يتم تغيير سطر الكود

```
QSequentialAnimationGroup *S_group = new QSequentialAnimationGroup;
```
بالسطر

```
QParallelAnimationGroup *S_group = new QParallelAnimationGroup;
```

..........................................................................

# QtNetwrok Module

# وحدة

# فصائل برمجة الشبكات

## متطلبات هذه الوحدة

**إدراج**

**#include <QtNetwork>**

**داخل ملفات الكود**

**إدراج**

**QT += network**

**داخل ملف المشروع project.pro** .

برمجة الشبكات هى من أهم العناصر الأساسية لأى تطبيق، ونلاحظ فى الوقت الحالى أن معظم التطبيقات يتم ربطها بشبكة الإنترنت مثل التطبيقات التى تتفاعل مع المواقع الخادمة، وذلك لإستكشاف تحديثات التطبيق.

ولقد قدمت **كيوت** كل الأدوات اللازمة لإجراء الإتصالات ونقل البيانات بين الشبكات، وأنتجت **كيوت** عدة فصائل خاصة للتعامل مع برمجة الشبكات منها :

فصيلة QHostAddress :

هى الفصيلة المسؤولة عن التعامل مع رقم العنوان الشبكى IP Address.

فصيلة QHostInfo :

هى الفصيلة المسؤولة عن :

الإستدلال عن رقم العنوان الشبكى ( IP Address ) باسم النطاق ( HostName )،

أو الإستدلال عن اسم النطاق ( HostName ) برقم العنوان الشبكى ( IP Address ).

فصيلة QTcpSocket :

هى الفصيلة المسؤولة عن التعامل مع الربط بأجهزة الخوادم Servers ونقل البيانات عبر الشبكة بطريقة TCP/IP.

فصيلة QTcpServer :

هى الفصيلة المسؤولة عن التعامل مع الربط بأجهزة العملاء Clients وإدارة الروابط بينهم ونقل البيانات عبر الشبكة بطريقة TCP/IP.

فصيلة QUdpSocket :

هى الفصيلة المسؤولة عن التعامل مع نقل البيانات عبر الشبكة بطريقة UDP/IP.

والجدير بالذكر أن **كيوت** قد قدمت فصائل أخرى عديدة، ولكننا سنكتفى بتقديم هذه الفصائل السابقة والتى تكفى لتوضيح ميكانيكية **كيوت** لبرمجة الشبكات بشكل تام.

# QHostAddress Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن التعامل مع رقم العنوان الشبكى IP Address،
ومن مزاياها:
- دعم التعامل مع العناوين من النوع IPv4 , IPv6.
- القيام بإستقبال العنوان IP بأى صيغة وتحويله لصيغة أخرى.

**وظائف الفصيلة :**

الدالة setAddress وتستخدم لإدخال العنوان الشبكى

ويمكن إدخال العنوان كنص

```
setAddress (“127.0.0.1“);
```

أو كرقم

```
setAddress (34322342334);
```

الدالة toIPv4Address وتستخدم لتحويل العنوان الشبكى من نصى إلى رقمى.

```
QHostAddress myip(“192.168.1.1”);
qDebug() << myip.toIPv4Address();
// this print 3232235777
```

الدالة toString وتستخدم لتحويل العنوان الشبكى من رقمى إلى نصى.

```
QHostAddress myip(3232235777);
qDebug() << myip.toString();
// this print 192.168.1.1
```

.........................................................................

# QHostInfo Class

**نسب الفصيلة :**

فصيلة مستقلة

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن :

البحث عن رقم العنوان الشبكى ( IP Address ) باسم الجهاز المضيف ( HostName )،
أو البحث عن اسم الجهاز المضيف ( HostName ) برقم العنوان الشبكى ( IP Address ).

**وظائف الفصيلة :**

تقوم الفصيلة بالإستدلال والبحث عن العنوان الشبكى باسم الجهاز المضيف ( HostName ) بطريقتين:

**الطريقة الأولى :** بإستخدام الدالة lookupHost - وكمثال على ذلك:

- لإيجاد رقم العنوان الشبكى للجهاز المضيف qt.nokia.com

```
QHostInfo::lookupHost("qt.nokia.com",
                      this, SLOT(printResults(QHostInfo)));
```

- لإيجاد اسم الجهاز المضيف للعنوان الشبكى 4.2.2.1.

```
QHostInfo::lookupHost("4.2.2.1",
                      this, SLOT(printResults(QHostInfo)));
```

وتمتاز هذه الطريقة بالتعامل بنظام دوال الإرسال و الإستقبال ( Signals and Slots )، وبالتالى يقوم التطبيق بأداء باقى المهام إلى حين حدوث إرسال إشارة بوجود نتائج لعملية البحث.

**الطريقة الثانية :** بإستخدام الدالة fromName - وكمثال على ذلك:

- لإيجاد رقم العنوان الشبكى للجهاز المضيف qt.nokia.com

```
QHostInfo info = QHostInfo::fromName("qt.nokia.com");
```

ولكن هذه الطريقة تحدث تجميد للتطبيق إلى حين رجوع الدالة بنتائج، لذلك لا يفضل إستخدامها فى التطبيقات ذات الواجهة الرسومية منعاً لحدوث تجميد لشاشة التطبيق، ولكن يمكن إستخدامها فى تطبيقات الشاشة Console.

مثال : لتوضيح وظائف QHostInfo , QHostAddress

سنقوم بفتح مشروع جديد على Qt Creator من النوع QWidget،

وسنقوم بالتعديل فى ملف widget.cpp , widget.h.

EXAMPLE NO 34

widget.h

```
#include <QWidget>
#include <QtNetwork>                          <---------          إضافة الكود
...
class Widget : public QWidget
{
...
public Q_SLOTS:                               <---------          إضافة الكود
  void printResults(QHostInfo info);          <---------          إضافة الكود
};
```

widget.cpp

```
#include <QtNetwork>
Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
   ui->setupUi(this);


       QHostAddress ip;
       ip.setAddress("192.168.1.1");
       qDebug() << ip.toString();
       qDebug() << ip.toIPv4Address();
       QHostInfo::lookupHost("www.yahoo.com",
                       this, SLOT(printResults(QHostInfo)));
       QHostInfo::lookupHost("69.147.125.65",
                       this, SLOT(printResults(QHostInfo)));
}
void Widget::printResults(QHostInfo info)
{
   foreach (QHostAddress addr , info.addresses())
   qDebug() << addr.toString();
   qDebug() << info.hostName();
}
```

شرح الكود السابق

```
QHostAddress ip;
```

الإعلان عن المتغير **ip** لإدارة الفصيلة **QHostAddress**.

```
ip.setAddress("192.168.1.1");
```

إعطاء **ip** القيمة **192.168.1.1** وهى رقم عنوان شبكى تم إدخاله لـ **ip** كنص.

```
qDebug() << ip.toString();
```

طباعة محتوى العنوان الشبكى الموجود بـ **ip** كنص .

النتيجة : **"192.168.1.1"**

```
qDebug() << ip.toIPv4Address();
```

طباعة محتوى العنوان الشبكى الموجود بـ **ip** كرقم .

النتيجة : **"3232235777"**

```
QHostInfo::lookupHost("www.youtube.com",
                    this, SLOT(printResults(QHostInfo)));
```

تقوم **lookupHost** بالبحث عن جميع أرقام العناوين الشبكية ( **IP Addresses** ) الخاصة باسم المضيف **www.youtube.com** ، وعندما يجد أى عنوان يقوم بتمرير البيانات للدالة المستقبلة **printResults** لتقوم بطبعها.

```
QHostInfo::lookupHost("209.85.225.136",
                    this, SLOT(printResults(QHostInfo)));
```

تقوم **lookupHost** بالبحث عن اسم المضيف ( **Host Name** ) صاحب العنوان الشبكى **209.85.225.136** ، وعندما يجد أى عنوان يقوم بتمرير البيانات للدالة المستقبلة **printResults** لتقوم بطبعها.

........................................................................

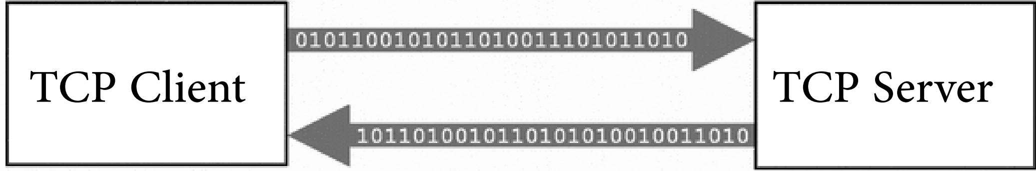
## نقل البيانات عبر الشبكات

تقدم **كيوت** أشهر طريقتين لنقل البيانات عبر الشبكات وهما :

1- TCP - Transmission Control Protocol.

2- UDP - User Datagram Protocol.

## أولا : طريقة TCP :

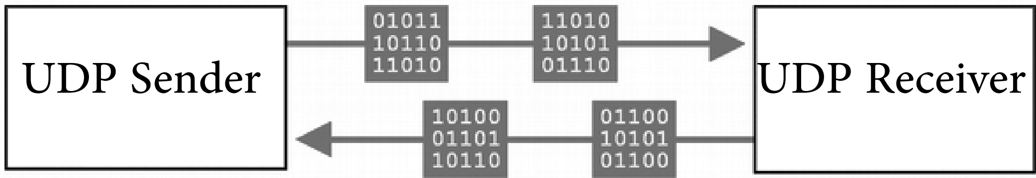



لإتمام عملية الإتصال لابد من وجود جهاز مستقبل Server و جهاز مرسل Client، ويتم نقل البيانات بين الطرفين بطريقة مسلسلة، حيث أن البيانات المرسلة تأخذ أرقام تسلسل ، وعند فقد أى جزء من البيانات يتم إرساله مرة أخرى تلقائياً، وهى تعتبر طريقة آمنة لنقل البيانات، كما يتم من خلالها تحديد الجهاز المرسل و الجهاز المستقبل.

وهذه الطريقة تستخدم فى HTTP , FTP.

وتستعمل **كيوت** فى هذه الطريقة الفصيلتين QTcpServer , QTcpSocket.

## ثانياً : طريقة UDP :




فى هذه الطريقة لايتم الإتصال بين جهازين، ولكن تقوم UDP Sender ببث البيانات إلى نطاق معين، ويمكن لكل من هم فى هذا النطاق إستقبال البيانات (Broadcasting Sender) ، وترسل البيانات فى بلوكات ثابتة الحجم ( تكون فى معظم الأحيان أقل من 512 بايت )، وفى حالة فقد بيانات خلال الإستقبال UDP Receiver لا يتم إعادة إرسالها, وذلك لعدم وجود ترقيم لبلوكات البيانات المرسلة, وهذه الطريقة تصلح لبث بيانات صغيرة الحجم مثل أرقام الطقس أو الساعة ، ونذكر أيضاً أن طريقة UDP أسرع فى نقل البيانات من TCP.

وتستعمل **كيوت** فى هذه الطريقة الفصيلة QUdpSocket.

# QTcpServer Class

**نسب الفصيلة :**

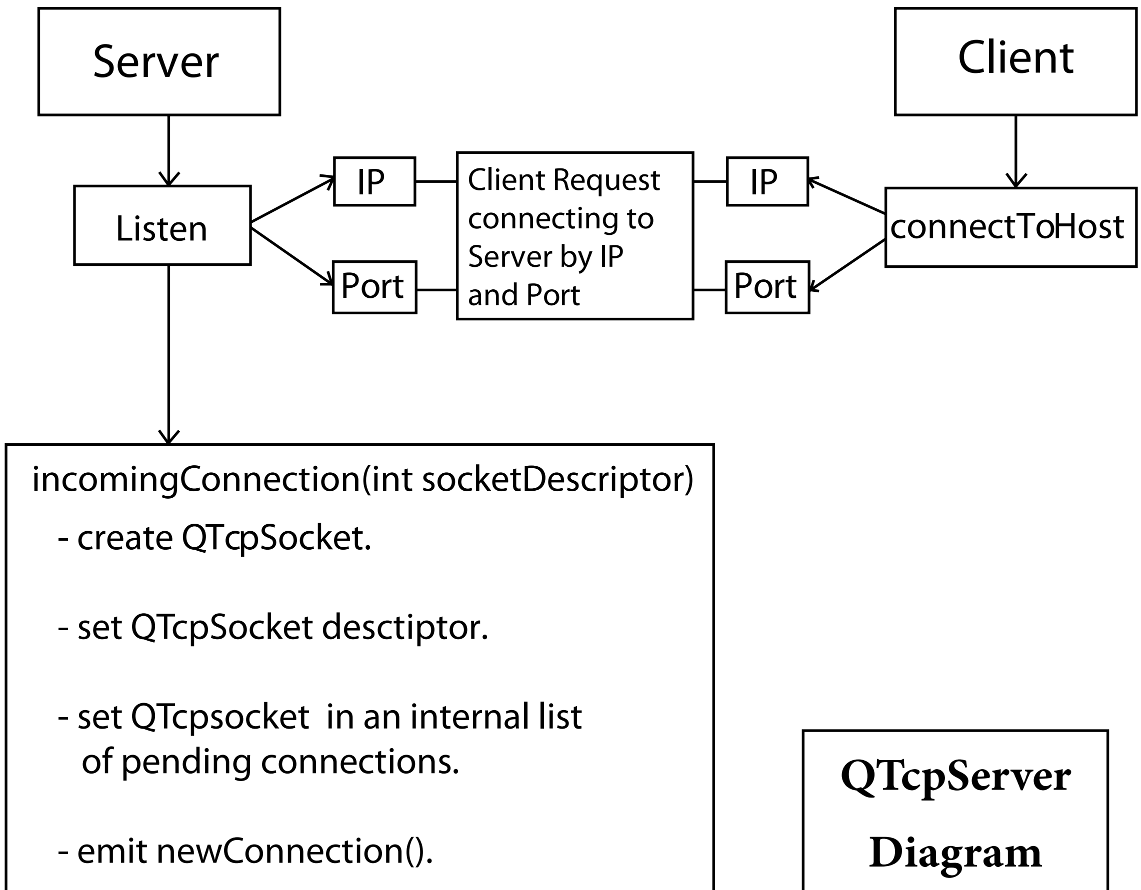
ترث فصيلة QObject

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن التعامل مع وصلات الإتصال بأجهزة العملاء Clients، وإدارة الإتصالات بينهم، ونقل البيانات عبر الشبكة بطريقة TCP/IP.

**ماذا يحدث داخل الفصيلة ؟**

**وكيف تقوم الفصيلة بتنظيم الربط مع المستخدمين؟**



يبدأ عمل الفصيلة بالدالة listen التى تأخذ المتغيرات Port , IP ، ثم تنتظر الفصيلة حتى يقوم أى جهاز عميل client بطلب الإتصال مع الجهاز server .

فإذا قام جهاز عميل **client** بطلب الإتصال مع الجهاز **server** عن طريق رقم العنوان الشبكى **IP** ورقم مدخل و مخرج البيانات **Port**، تبدأ الفصيلة العمل بالخطوات الآتية:

1 - تحدد الفصيلة رقم لإدارة عملية الإتصال socket descriptor، وهذا الرقم هو الذى يدل على عملية الإتصال.

2 - تقوم بتمرير رقم عملية الإتصال socket descriptor للدالة incomingConnection وتشغيلها تلقائياً.

3 - تحتوى الدالة incomingConnection على كود يقوم بالآتى :
- إنشاء فصيلة QTcpSocket كفصيلة إتصال لتقوم بإدارة عملية الإتصال.
- تمرير رقم عملية الإتصال socket descriptor للفصيلة.
- وضع الفصيلة داخل قائمة داخلية للإتصالات تستدعى بالدالة nextPendingConnection.
- تشغيل دالة الإرسال newConnection.

هذا ما يحدث داخل الفصيلة، ولكن عند كتابة كود التطبيق فكل ما نحتاجه هو:

الدالة listen :

وهى تقوم بإنتظار أى طلب إتصال، وعند الإتصال تشعل دالة الإرسال newConnection لتقوم ببث إشارتها، وبالتالى يمكن ربط دالة الإرسال بأى دالة إستقبال لعمل اللازم مع الجهاز الطالب للإتصال.

الدالة nextPendingConnection :

وهى تقوم بإستدعاء آخر عملية إتصال من القائمة الداخلية للفصيلة QTcpServer، كما يمكن إدارة عملية الإتصال بطريقة أخرى:

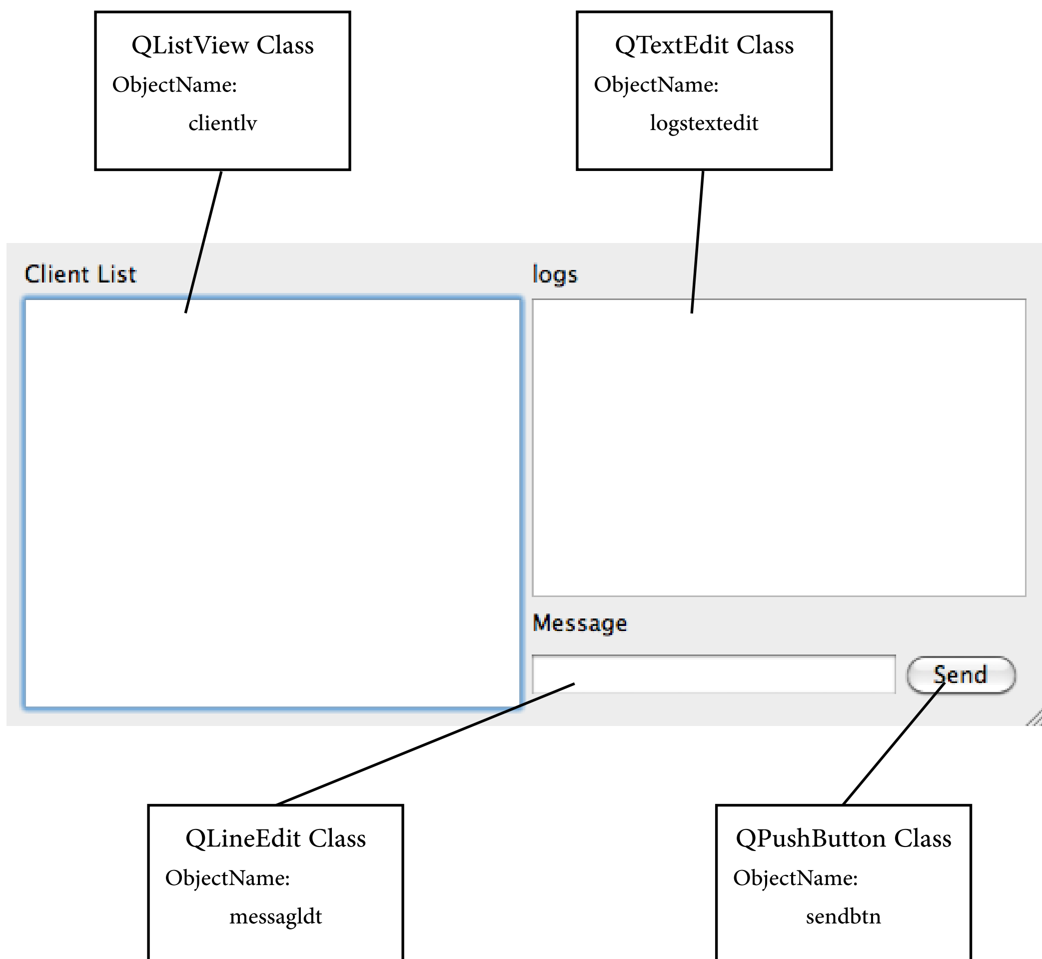
وهى إعادة كتابة الدالة incomingConnection، وفى هذه الحالة سنقوم بإلغاء الخطوة رقم 3 من الشرح السابق، ويكون للمبرمج حرية التعامل مع رقم إدارة عملية الإتصال، وتعتبر هذه هى الطريقة الأكثر إستخداماً.

سنقوم بعمل مثال بسيط وهو إنشاء تطبيق خادم Server يقوم ببث رسالة إلى كل عميل Client يقوم بالربط معه.

نقوم بفتح مشروع جديد بالأداة QtCreator من النوع widget واسم المشروع server_message .

وليتعامل المشروع مع خصائص الشبكات Network Module نضيف الكود ( QT+= network ) للملف server_message.pro.

ثم نصمم widget.ui كالتالى:



widget.h

EXAMPLE NO 35

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtNetwork>

namespace Ui {
    class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    QTcpServer *server;
    QMap<QTcpSocket*,QString> users;

public Q_SLOTS:
    void login();
    void logout();
    void on_sendbtn_clicked();
    void updatelist();

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H
```

شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
QTcpServer *server;
```

الإعلان عن المتغير **server** لإدارة فصيلة **QTcpServer**.

```
QMap<QTcpSocket*,QString> users;
```

الإعلان عن المتغير **users** لإدارة فصيلة حاوية من النوع **QMap**، ويحتوى كل عنصر فيها على كلاً من:
القيمة **QTcpSocket** كمفتاح ( **Key** ) ، والقيمة **QString** كقيمة ( **Value** )،
حيث يتم تسجيل اسم المستخدم **client** كقيمة **QString**،
وتسجيل مؤشر للوصلة ( **connection** ) كمفتاح **QTcpSocket**.

```
void login();
```

دالة إستقبال تقوم بأداء مهمتها
عند إستقبال الفصيلة لإشارة طلب دخول من قبل مستخدم **client**.

```
void logout();
```

دالة إستقبال تقوم بأداء مهمتها
عند إستقبال الفصيلة لإشارة خروج من قبل مستخدم **client**.

```
void on_sendbtn_clicked();
```

دالة إستقبال تقوم بأداء مهمتها
عند الضغط على المفتاح **sendbtn**.

```
void updatelist();
```

دالة تقوم بتحديث قائمة المستخدمين المتصلين مع الجهاز **server**.

..........................................................................

widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
#include <QtNetwork>
#include <QStringListModel>

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    server = new QTcpServer(this);
    server->listen(QHostAddress::Any , 4444);

    connect(server,SIGNAL(newConnection()) , this , SLOT(login()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::login()
{
    QTcpSocket *newsocket = (QTcpSocket*) server->nextPendingConnection();
    int socketnum = newsocket->socketDescriptor();
    QString socketstr;
    socketstr.setNum(socketnum);
    users.insert(newsocket , "user_"+socketstr);
    ui->logstextedit->append(users.value(newsocket)+"Logged In");
    connect(newsocket, SIGNAL(disconnected()), this, SLOT(logout()));
    updatelist();
}


void Widget::logout()
```

```
{
    QTcpSocket *closesocket = (QTcpSocket*)sender();
    ui->logstextedit->append(users.value(closesocket)+"Logged Out");
    users.remove(closesocket);
    updatelist();
}

void Widget::updatelist()
{
    QStringList userlist;
    foreach(QString username , users.values())
        userlist << username;
    QStringListModel *usermodel = new QStringListModel;
    usermodel->setStringList(userlist);
    ui->clientlv->setModel(usermodel);
}

void Widget::on_sendbtn_clicked()
{
    foreach(QTcpSocket *user, users.keys())
        user->write((ui->messageldt->text()+"\n").toUtf8());
}
```

### شرح الكود السابق

```
#include <QtNetwork>
#include <QStringListModel>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات،

إدراج  تعريف الفصيلة **QStringListModel**.

```
server = new QTcpServer(this);
```

حجز مكان للمتغير **server** داخل الذاكرة بمساحة الفصيلة **QTcpServer**،

ووضعه كإبن للفصيلة **widget** بواسطة المعامل **this**.

```
server->listen(QHostAddress::Any , 4444);
```
يقوم **server** بفتح بوابة بيانات **socket**، وإنتظار أى مستخدم يطلب الإتصال على العنوان الشبكى ( **IP : localhost** ) و من خلال المنفذ ( Port : 4444 ) .

```
connect(server,SIGNAL(newConnection()) , this , SLOT(login()));
```
ربط دالة الإرسال **newConnection** من **server**،
بدالة الإستقبال **login** من **this widget**،
وكنتيجة لهذا الربط كلما قام مستخدم **Client** بطلب الإتصال بـ **server**، تبث دالة الإرسال إشارتها لتستقبلها دالة الإستقبال **login** وتقوم بعمل اللازم.

.......................................................................................

شرح محتوى الدالة login

```
QTcpSocket *newsocket = (QTcpSocket*) server->nextPendingConnection();
```
إنشاء الوصلة **newsocket** لتعمل كمؤشر للإتصال ( **connection** ) القادم من المستخدم طالب الإتصال،
والدالة **nextPendingConnection** تقوم بإعطاء آخر وصلة إتصال وتمررها للوصلة **newsocket**.

```
int socketnum = newsocket->socketDescriptor();
```
الدالة **socketDescriptor** تقوم بإيجاد رقم إدارة عملية الإتصال القادم من **newsocket** وتمريره للمتغير **socketnum**.

```
QString socketstr;
socketstr.setNum(socketnum);
```
تقوم بتحويل رقم إدارة عملية الإتصال إلى متغير نصى.

```
users.insert(newsocket , “user_“+socketstr);
```
إدخال عنصر جديد للفصيلة الحاوية **users** يتكون من:
القيمة **newsocket** كمفتاح ( **key** ) والقيمة **”user_“+socketstr** كقيمة ( **vlaue** ).

```
ui->logstextedit->append(users.value(newsocket)+“Logged In“);
```
إضافة سطر نصى لـ **logstextedit** يفيد بدخول مستخدم جديد.

```
connect(newsocket, SIGNAL(disconnected()), this, SLOT(logout()));
```

ربط دالة الإرسال **disconnected** من **newsocket**،

بدالة الإستقبال **logout** من **this widget**،

وكنتيجة لهذا الربط كلما إنقطع الإتصال مع المستخدم **Client**، تبث دالة الإرسال إشارتها لتستقبلها دالة الإستقبال **logout** وتقوم بعمل اللازم.

```
updatelist();
```

تقوم بتحديث قائمة المستخدمين فى واجهة الإستخدام للتطبيق.

..............................................................................................

شرح محتوى الدالة logout

```
QTcpSocket *closesocket = (QTcpSocket*) sender();
```

تبدأ الدالة **logout** عملها عند تلقيها إشارة من دالة الإرسال **disconnected** القادمة من مستخدم **client** تفيد بإنقطاع الإتصال معه.

الدالة **sender()** تقوم بإيجاد مرسل الإشارة ويتم تمريره للوصلة **closesocket**.

```
ui->logstextedit->append(users.value(closesocket)+"Logged Out");
```

إضافة سطر نصى لـ **logstextedit** يفيد بخروج مستخدم.

```
users.remove(closesocket);
```

مسح المستخدم من القائمة **users**.

```
updatelist();
```

تقوم بتحديث قائمة المستخدمين فى واجهة الإستخدام للتطبيق.

..............................................................................................

شرح محتوى الدالة updatelist.

```
QStringList userlist;
```

الإعلان عن المتغير **userlist** لإدارة قائمة حاوية تحتوى على قائمة أسماء المستخدين المتصلين بـ **server**.

```
foreach(QString username , users.values())
   userlist << username;
```

تمرير أسماء المستخدمين من الفصيلة الحاوية **users** إلى الفصيلة الحاوية **userlist**.

```
QStringListModel *usermodel = new QStringListModel;
```

الإعلان عن المتغير **mymodel** لإدارة فصيلة **QStringListModel**،
وهى تقدم نموذج مصمم لإحتواء قائمة عناصر .

```
usermodel->setStringList(userlist);
```

تمرير قائمة أسماء المستخدمين لـ **usermodel**.

```
ui->clientlv->setModel(usermodel);
```

توجيه النموذج **usermodel** إلى كائن عرض النموذج **clientlv** ليقوم بعرضه.

.........................................................................................

شرح محتوى الدالة on_sendbtn_clicked.

```
foreach(QTcpSocket *user, users.keys())
    user->write((ui->messageldt->text()+"\n").toUtf8());
```

تقوم الدالة **write** بإرسال محتوى **messageldt** إلى جميع المستخدمين الموجودين فى القائمة **users**.
ونرى هنا أنه تم :

- إضافة الحرف"\n" إلى نهاية النص المراد إرساله.
- تحويل النص للصيغة utf8.

وبدون الخطوتين السابقتين لن يتم إرسال البيانات بصورة صحيحة.

.........................................................................................

# QTcpSocket Class

**نسب الفصيلة :**

ترث فصيلة QAbstractSocket

التى ترث QIODevice

التى ترث QObject

**تعريف الفصيلة :**

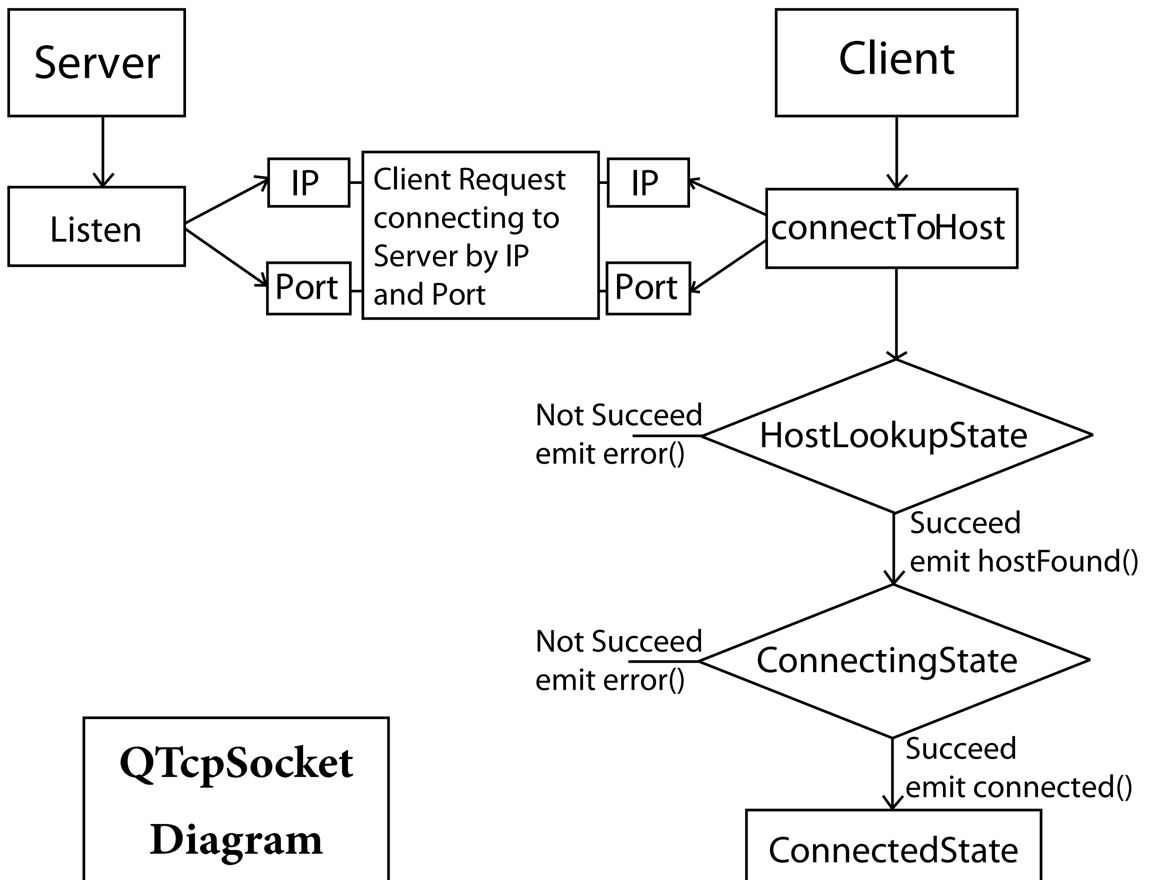
هى الفصيلة المسؤولة عن التعامل مع الربط بأجهزة الخوادم Servers ونقل البيانات عبر الشبكة بطريقة TCP/IP.

TCP - ( Transmission Contol Protocol ) هى طريقة لنقل البيانات عبر الشبكات،

ولهذه الطريقة عدة مميزات منها :

- أنها طريقة موثوقة لنقل البيانات - ففى حالة فقد أية بيانات يتم بثها تلقائياً مرة اخرى.

- يتم عن طريقها توجيه الإتصال و البيانات - يتم الإتصال و إرسال البيانات إلى جهاز محدد.



**كيف يحدث الإتصال بين المستخدم client و الخادم server فى طريقة TCP ؟**

فى هذه الطريقة يتم الإتصال بين الأجهزة بمعرفة متغيرين هامين هما :

- رقم العنوان الشبكى IP Address.
- رقم منفذ الربط Port Numbaer.

مراحل عملية الإتصال :

- عند إستخدام دالة الإتصال connectToHost() :

تقوم الفصيلة بالبحث عن الجهاز المضيف، وتسمى هذه المرحلة HostLookupState.

- عند إيجاد الجهاز المضيف :

يدخل الإتصال فى مرحلة جارى الإتصال ConnectingState،

وتبث دالة الإرسال hostFound() إشارتها.

- عند حدوث الإتصال :

يدخل الإتصال فى مرحلة متصل ConnectedState،

وتبث دالة الإرسال connected() إشارتها.

- عند حدوث أى خطأ فى أى مرحلة :

تبث دالة الإرسال error() ودالة الإرسال stateChanged() إشارتهما.

تقدم الفصيلة الدالة state() للتعرف على حالة الإتصال فى أى مرحلة من المراحل السابقة:

الوضع التلقائى للقيمة المرتدة من الدالة state() : غير متصل UnconnectedState.

يتم إرسال و إستقبال البيانات بين الجهاز المستخدم client و الجهاز الخادم server بواسطة الدالتين read() , write().

دالة الإرسال readyRead() هى أساس عملية إستقبال البيانات، حيث يتم إشعالها لتقوم ببث إشارتها عند توافر أى بيانات مرسلة لوصلة الإتصال.

............................................................................

سنقوم بعمل مثال بسيط وهو إنشاء تطبيق مستخدم Client يقوم بطلب إتصال إلى تطبيق server السابق وإستقبال الرسائل منه.

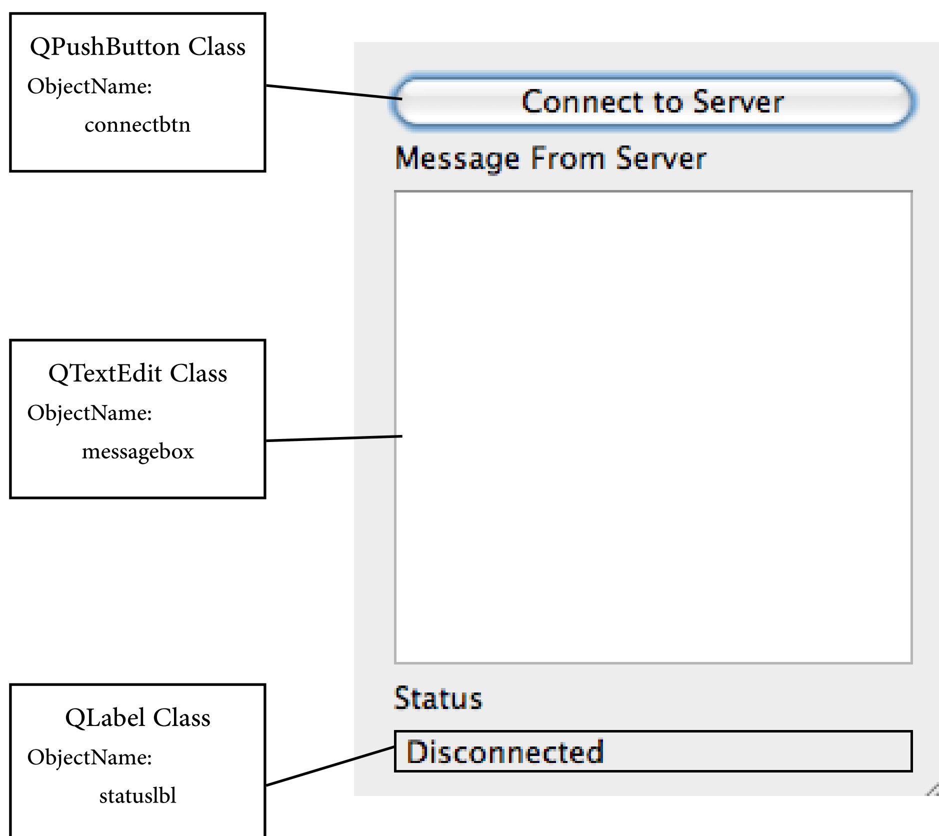
نقوم بفتح مشروع جديد بالاداة QtCreator من النوع widget

واسم المشروع client_message .

وليتعامل المشروع مع خصائص الشبكات Network Module

نضيف الكود ( QT+= network ) للملف client_message.pro.

ثم نصمم widget.ui كالتالى:



widget.h

EXAMPLE NO 36

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtNetwork>

namespace Ui {
    class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    QTcpSocket *client;

public Q_SLOTS:
    void on_connectbtn_clicked();
    void readmessage();
    void setconnected();

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H
```

## شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
QTcpSocket *client;
```

الإعلان عن المتغير **server** لإدارة الفصيلة **QTcpServer** .

```
void on_connectbtn_clicked();
```

دالة إستقبال تقوم بأداء مهمتها
عند الضغط على المفتاح **connectbtn**.

```
void readmessage();
```

دالة إستقبال  تقوم بأداء مهمتها
عند إستقبال الفصيلة لإشارة دالة الإرسال **readyRead** التى تبث عند وجود بيانات للقراءة.

```
void setconnected();
```

دالة إستقبال  تقوم بأداء مهمتها
عند إستقبال الفصيلة لإشارة الدالة **connected** التى تبث عند نجاح عملية الإتصال.

.............................................................................................

widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
#include <QtNetwork>

Widget::Widget(QWidget *parent) :QWidget(parent),ui(new Ui::Widget)
{
    ui->setupUi(this);
    client = new QTcpSocket(this);
    connect(client, SIGNAL(readyRead()), this, SLOT(readmessage()));
    connect(client, SIGNAL(connected()), this, SLOT(setconnected()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::on_connectbtn_clicked()
{
    client->connectToHost("localhost", 4444);
}

void Widget::readmessage()
{
    while(client->canReadLine())
    {
        QString line = QString::fromUtf8(client->readLine()).trimmed();
         ui->messagebox->append("Message :" + line);
    }
}

void Widget::setconnected()
{
    ui->statuslbl->setText("Connected");
}
```

## شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
client = new QTcpSocket(this);
```

حجز مكان للمتغير **client** داخل الذاكرة بمساحة الفصيلة **QTcpSocket**،
ووضعه كإبن للفصيلة **widget** بواسطة المعامل **this**.

```
connect(client, SIGNAL(readyRead()), this, SLOT(readmessage()));
```

ربط دالة الإرسال **readyRead** من **client**
بدالة الإستقبال **readmessage** من **this widget**،
وكنتيجة لهذا الربط كلما توفرت بيانات للقراءة قادمة من وصلة الإتصال **client**، تبث دالة الإرسال **readyRead** إشارتها لتستقبلها دالة الإستقبال **readmessage** وتقوم بعمل اللازم.

```
connect(client, SIGNAL(connected()), this, SLOT(setconnected()));
```

ربط دالة الإرسال **connected** من **client**
بدالة الإستقبال **setconnected** من **this widget**،
وكنتيجة لهذا الربط فعند نجاح عملية الإتصال تبث دالة الإرسال **connected** إشارتها لتستقبلها دالة الإستقبال **setconnected** وتقوم بعمل اللازم.

..........................................................................................

شرح محتوى الدالة on_connectbtn_clicked.

```
client->connectToHost("localhost", 4444);
```

تقوم وصلة الإتصال **client** بطلب الإتصال مع الجهاز **server** من خلال اسم النطاق **localhost** و من خلال المنفذ ( Port ) رقم **4444** .

..........................................................................................

شرح محتوى الدالة readmessage

```
while(client->canReadLine())
{
        QString line = QString::fromUtf8(client->readLine()).trimmed();
        ui->messagebox->append("Message :" + line);
}
```

تقوم الدالة **canReadLine** برد القيمة **true** فى حالة إستطاعتها قراءة سطر بيانات كامل،
وتقوم الدالة **readLine** بقراءة سطر البيانات القادم إلى الوصلة **client**،
ثم يتم عرض سطر البيانات فى الكائن **messagebox** بواسطة الدالة **append**.

ونلاحظ هنا أنه تم تحويل البيانات إلى الصيغة UTF8 بالعملية **QString::fromUtf8**، لكى تستطيع ***كيوت*** التعامل معها، وبالتالى فعند الإستقبال نقوم بعكس العملية،
ونرى ذلك فى المثال السابق **server** حيث قمنا بتحويل البيانات إلى الصيغة **Utf8** قبل إرسالها.

.................................................................................

شرح محتوى الدالة setconnected.
تنشط هذه الدالة عند تمام نجاح عملية الإتصال.

```
ui->statuslbl->setText("Connected");
```

تقوم الدالة **setText** بوضع النص **Connected** داخل الكائن **statuslbl** لإظهار تمام عملية الإتصال.

.................................................................................

# QUdpSocket Class

**نسب الفصيلة :**

ترث فصيلة QAbstractSocket

التى ترث QIODevice

التى ترث QObject

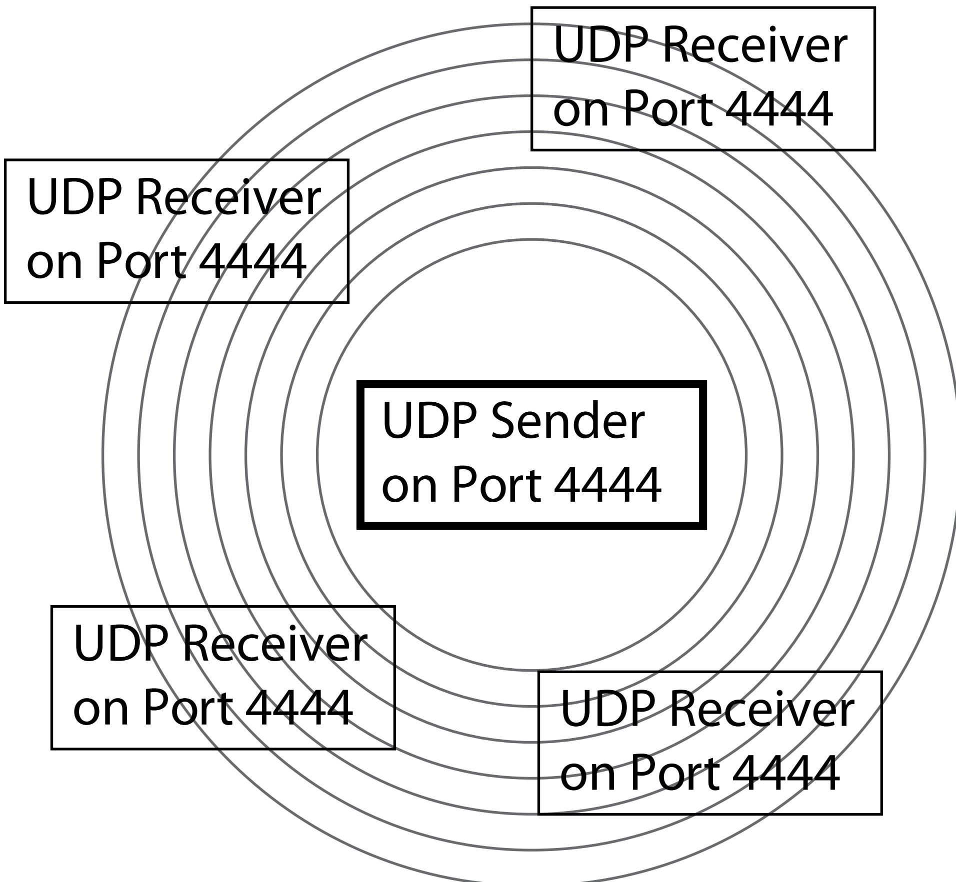
**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن التعامل مع نقل البيانات عبر الشبكة بطريقة UDP، وكما علمنا سابقاً فإن بث البيانات بطريقة UDP تستخدم للتطبيقات التى يمكننا فيها التجاوز عن فقد البيانات كتطبيقات المحادثة الصوتية أو الفيديو ( Audio ,Video Chat )، فالعامل الأساسى المطلوب هنا هو السرعة، وفقدنا بعض البيانات قد لايكون بالأمر الهام.

لاتستخدم UDP دوال لإقامة وصلة إتصال بين الجهاز المرسل و الجهاز المستقبل،

ولكن الجهاز المرسل يقوم بإرسال البيانات على رقم منفذ ( Port number ) معين،

ليقوم أى جهاز UDP Receiver فى نطاق الإستقبال UDP Sender بإستقبال البيانات المرسلة إلى هذا المنفذ.



سنقوم بعمل مثال بسيط وهو إنشاء تطبيقين هما :

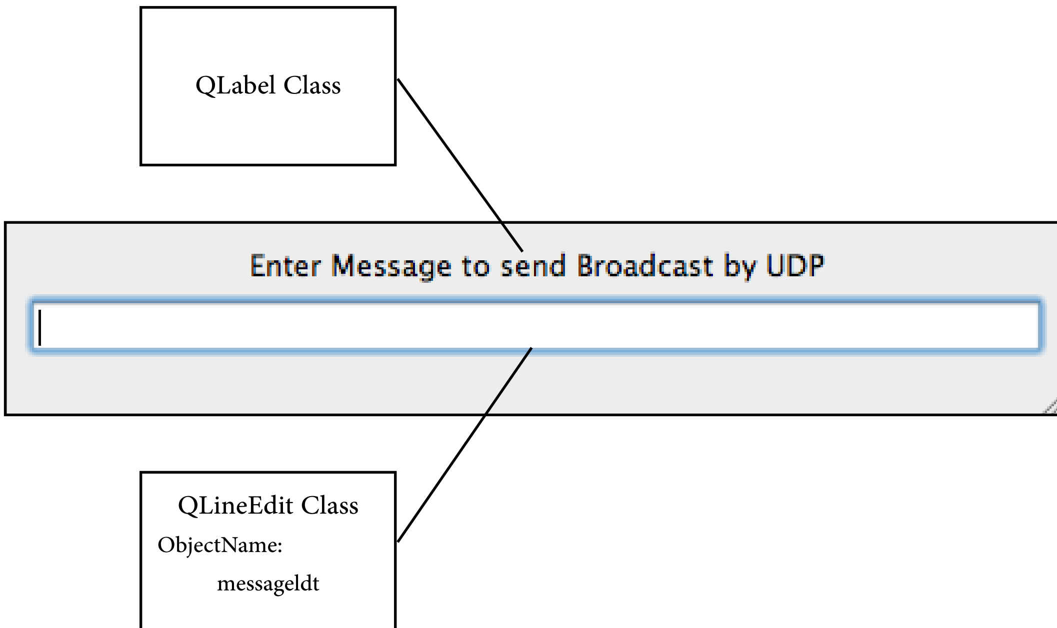
- مرسل بيانات UDP : وسيقوم ببث رسالة من المنفذ 4444.
- مستقبل بيانات UDP : وسيقوم بإستقبال رسالة من المنفذ 4444.

.........................................................................

تطبيق مرسل البيانات :

نقوم بفتح مشروع جديد بالأداة QtCreator من النوع widget
واسم المشروع udp_sender .
وليتعامل المشروع مع خصائص الشبكات Network Module
نضيف الكود ( QT+= network ) للملف udp_sender.pro.

ثم نصمم widget.ui كالتالى:



widget.h

EXAMPLE NO 37

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QtNetwork>

namespace Ui {
   class Widget;
}

class Widget : public QWidget
{
   Q_OBJECT
public:
   explicit Widget(QWidget *parent = 0);
   ~Widget();
public Q_SLOTS:
   void senddata();

private:
   Ui::Widget *ui;
   QUdpSocket *udpsender;
};
#endif // WIDGET_H
```

شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
QUdpSocket *udpsender;
```

الإعلان عن المتغير **udpsender** لإدارة فصيلة **QUdpSocket**.

```
void senddata();
```

دالة إستقبال تقوم بأداء مهمتها

عند إستقبال الفصيلة لإشارة دالة الإرسال **textChanged** التى تبث عند تغيير محتوى الرسالة.

widget.cpp

```
#include “widget.h“
#include “ui_widget.h“

#include <QtNetwork>

Widget::Widget(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::Widget)
{
   ui->setupUi(this);
   udpsender = new QUdpSocket(this);
   connect(ui->messageldt, SIGNAL(textChanged(QString)), this, SLOT(senddata()));
 }

Widget::~Widget()
{
   delete ui;
}

void Widget::senddata()
{
   QString str = ui->messageldt->text();
   QByteArray datagram;
   datagram.append(str);
   udpsender->writeDatagram(datagram.data(), datagram.size(), QHostAddress::Broadcast, 4444);
}
```

## شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
udpsender = new QUdpSocket(this);
```

حجز مكان للمتغير **udpsender** داخل الذاكرة بمساحة الفصيلة **QUdpSocket**،
ووضعه كإبن للفصيلة **widget** بواسطة المعامل **this**.

```
connect(ui->messageldt, SIGNAL(textChanged(QString)), this, SLOT(senddata()));
```

ربط دالة الإرسال **textChanged** من **messageldt**
بدالة الإستقبال **senddata** من **this widget**،
وكنتيجة لهذا الربط كلما تغير النص الموجود بالكائن **messageldt** تبث دالة الإرسال **textChanged** إشارتها لتستقبلها دالة الإستقبال **senddata** وتقوم بعمل اللازم.

.............................................................................................

شرح محتوى الدالة senddata.

```
QString str = ui->messageldt->text();
```

الإعلان عن المتغير **str** وإستقباله للمحتوى النصى الموجود بالكائن **messageldt**.

```
QByteArray datagram;
```

الإعلان عن المتغير **datagram** لإدارة مصفوفة من النوع بايت.

```
datagram.append(str);
```

إضافة المتغير **str** إلى المصفوفة **datagram**.

```
udpsender->writeDatagram(datagram.data(), datagram.size(),
                                    QHostAddress::Broadcast, 4444);
```

إرسال النص **datagram.data()** ذو المساحة **datagram.size()**
إلى العنوان **QHostAddress::Broadcast**
من المنفذ رقم **4444**.

تطبيق مستقبل البيانات :

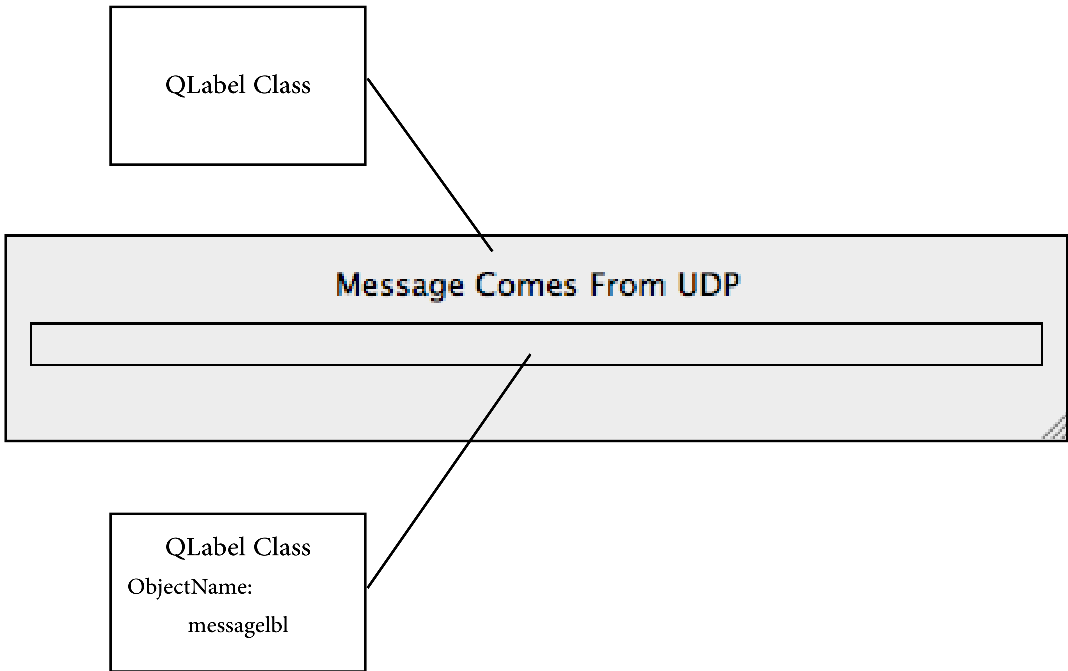
نقوم بفتح مشروع جديد بالأداة QtCreator من النوع widget

واسم المشروع udp_receiver .

وليتعامل المشروع مع خصائص الشبكات Network Module

نضيف الكود ( QT+= network ) للملف udp_receiver.pro.

ثم نصمم widget.ui كالتالى:



widget.h

EXAMPLE NO 38

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QtNetwork>

namespace Ui {
    class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
public Q_SLOTS:
    void readdata();

private:
    Ui::Widget *ui;
    QUdpSocket *udpreceiver;
};
#endif // WIDGET_H
```

شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
QUdpSocket *udpreceiver;
```

الإعلان عن المتغير **udpreceiver** لإدارة فصيلة **QUdpSocket** .

```
void readdata();
```

دالة إستقبال تقوم بأداء مهمتها
عند إستقبال الفصيلة لإشارة دالة الإرسال **readyRead** التى تبث عند وجود بيانات للقراءة.

widget.cpp

```
#include “widget.h“
#include “ui_widget.h“
#include <QtNetwork>

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    udpreceiver = new QUdpSocket(this);
    udpreceiver->bind(4444, QUdpSocket::ShareAddress);

    connect(udpreceiver, SIGNAL(readyRead()),this, SLOT(readdata()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::readdata()
{
    while (udpreceiver->hasPendingDatagrams()) {
        QByteArray datagram;
        datagram.resize(udpreceiver->pendingDatagramSize());
        udpreceiver->readDatagram(datagram.data(), datagram.size());
        ui->messagelbl->setText(datagram.data());
    }
}
```

شرح الكود السابق

```
#include <QtNetwork>
```

إدراج تعريف **QtNetwork** لتعريف كل الفصائل المتعلقة بالشبكات.

```
udpsender = new QUdpSocket(this);
```

حجز مكان للمتغير **udpreceiver** داخل الذاكرة بمساحة الفصيلة **QUdpSocket**،
ووضعه كإبن للفصيلة **widget** بواسطة المعامل **this**.

```
udpreceiver->bind(4444, QUdpSocket::ShareAddress);
```

تقوم bind بربط الوصلة **udpreceiver** على رقم المنفذ **4444**،
و **QUdpSocket::ShareAddress** تسمح لأى تطبيق آخر بالربط على نفس رقم المنفذ.

```
connect(udpreceiver, SIGNAL(readyRead()),this, SLOT(readdata()));
```

ربط دالة الإرسال **readyRead** من **udpreceiver**
بدالة الإستقبال **readdata** من **this widget**.
وكنتيجة لهذا الربط كلما توفرت بيانات للقراءة قادمة من المنفذ **4444** تبث دالة الإرسال **readyRead** إشارتها لتستقبلها دالة الإستقبال **readdata** وتقوم بعمل اللازم.

.................................................................................................

شرح محتوى الدالة readdata.

```
QByteArray datagram;
```

الإعلان عن المتغير **datagram** لإدارة مصفوفة من النوع بايت.

```
datagram.resize(udpreceiver->pendingDatagramSize());
```

تغيير حجم المصفوفة **datagram** ليساوى حجم البيانات القادمة.

```
udpreceiver->readDatagram(datagram.data(), datagram.size());
```

إستقبال البيانات و تخزينها فى **datagram.data()** بمساحة **datagram.size()**.

```
ui->messagelbl->setText(datagram.data());
```

وضع البيانات القادمة داخل الكائن **messagelbl** الذى يقوم بعرض البيانات على واجهة التطبيق.

.................................................................................................

Notes

# QtSql Module

# وحدة فصائل

# التعامل مع قواعد البيانات

## متطلبات هذه الوحدة

إدراج

```
#include <QtSql>
```

داخل ملفات الكود

إدراج

```
QT += sql
```

داخل ملف المشروع **project.pro** .

وحدة التعامل مع قواعد البيانات هى من الوحدات الهامة للمبرمج والتى تمكنه من التواصل مع قواعد البيانات المختلفة، لذلك تقدم ***كيوت*** مجموعة من الفصائل التى تتعامل مع مختلف قواعد البيانات، وسوف نقوم بعرض هذه الفصائل تبعاً للأهمية كالآتى:

## فصيلة QSqlDatabase :

هي الفصيلة المسؤولة عن الإتصال مع قاعدة البيانات من خلال SQL Database Drivers، وتتعامل QSqlDatabase مع تسعة أنواع من قواعد البيانات وهم الموضحين بالجدول:

| Qt Driver Name | DataBase Type |
|---|---|
| QDB2 | IBM DB2 |
| QIBASE | Borland InterBase |
| QMYSQL | MySQL |
| QOCI | Oracle Call Interface |
| QODBC | Open Database ( Microsoft ) |
| QPSQL | PostgreSQL |
| QSQLITE2 | SQLite v2.0 |
| QSQLITE | SQLite v3.0 |
| QTDS | Sybase Adaptive server |

## فصيلة QSqlQuery :

هى الفصيلة المسؤولة عن تمرير وتنفيذ عبارات لغة SQL
مثل ( SELECT , INSERT ) إلى قاعدة البيانات .

## فصيلة QSqlField :

هى الفصيلة المسؤولة عن التعامل مع حقول البيانات بقاعدة البيانات.

## فصيلة QSqlError :

هى الفصيلة المسؤولة عن إعطاء تقرير المعلومات حول أخطاء قاعدة البيانات.

.....................................................................

كيف تقوم **كيوت** بالتواصل مع قاعدة البيانات؟

تتواصل **كيوت** مع قاعدة البيانات بالربط معها عن طريق الفصيلة **QSqlDatabase** وبإستخدام الدوال الآتية :

- الدالة addDatabase : لتحديد نوع قاعدة البيانات التى سيتم التعامل معها.
- الدالة setHostName : لتحديد مكان خادم قاعدة البيانات ( Database Server ).
- الدالة setDatabaseName : لتحديد اسم قاعدة البيانات.
- الدالة setUserName : لإدخال اسم المستخدم.
- الدالة setPassword : لإدخال كلمة المرور الخاصة بالمستخدم.
- الدالة open : للقيام بفتح الإتصال مع قاعدة البيانات.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("dbhost");
db.setDatabaseName("dbname");
db.setUserName("myuser");
db.setPassword("mypass");
db.open();
```

ثم يبدأ التعامل مع قاعدة البيانات بتمرير العبارات الخاصة بلغة SQL عن طريق الفصيلة QSqlQuery، حيث يتم تمرير العبارات مباشرة بإستخدام الدالة exec .

```
QSqlQuery query;
query.exec("INSERT INTO userpoints VALUES( 'Mahmoud' , 2000 ) ");
```

أو تجهيزها قبل التمرير بواسطة الدالتين prepare ، bind، ومراحلها كالآتى :

- تستخدم الدالة prepare لتجهيز عبارة SQL .
- ثم الدالة bind لتمرير المتغيرات للعبارة .
- ثم الدالة exec لتنفيذ العبارة .

ويجب ملاحظة أن هناك أكثر من طريقة لتمرير المتغيرات لعبارة SQL.

**طرق تمرير المتغيرات لعبارة SQL :**

الطريقة الأولى : توضع المتغيرات فى الدالة prepare كأسماء مثل varname, :varpoints: ، ثم تمرر bind المتغيرات بأسمائها varname, :varpoints:.

```
query.prepare("INSERT INTO userpoints (name, points) "
              "VALUES (:varname, :varpoints)");
query.bindValue(":varname", "Ali");
query.bindValue(":varpoints", 500);
query.exec();
```

الطريقة الثانية : توضع المتغيرات فى الدالة prepare كأسماء مثل varname, :varpoints: ، ثم تمرر bind المتغيرات بترتيب الأرقام .

```
query.prepare("INSERT INTO userpoints (name, points) "
              "VALUES (:varname, :varpoints)");
query.bindValue( 0 , "Ali");
query.bindValue( 1 , 500);
query.exec();
```

الطريقة الثالثة : توضع المتغيرات فى الدالة prepare كعلامات إستفهام تدل على مكان وجود المتغيرات، ثم تمرر bind المتغيرات بترتيب الأرقام .

```
query.prepare("INSERT INTO userpoints (name, points) "
              "VALUES ( ? , ? )");
query.bindValue( 0 , "Ali");
query.bindValue( 1 , 500);
query.exec();
```

الطريقة الرابعة : توضع المتغيرات فى الدالة prepare كعلامات إستفهام تدل على مكان وجود المتغيرات، ثم تمرر bind المتغيرات بترتيب الكود .

```
query.prepare("INSERT INTO userpoints (name, points) "
              "VALUES ( ? , ? )");
query.addBindValue( "Ali" );
query.addBindValue( 500 );
query.exec();
```

..................................

## الإبحار فى سجلات قاعدة البيانات :

فى حالة تمرير عبارة SELECT من لغة SQL ( وهى العبارة الخاصة بالبحث فى قاعدة بيانات وإستخراج نتائج منها ) يتم التعامل معها بعدة دوال هى:

- الدالة next() : لتشير إلى السجل التالى.
- الدالة previous() : لتشير إلى السجل السابق.
- الدالة first() : لتشير إلى السجل الأول.
- الدالة last() : لتشير إلى السجل الأخير.
- الدالة seek(int num) : لتشير إلى السجل رقم num.

وكما نعلم فإن جداول قواعد البيانات تتكون من سجلات، ويتكون السجل ( Record ) من عدد من الحقول ( Fields )، ويمكننا الحصول على القيمة الموجودة داخل كل حقل بتمرير رقم الحقل المطلوب للدالة value( field num).

مثال على ذلك : نقوم بتمرير عبارة SQL لإستخراج جدول بيانات يتكون من حقلين اسم name و رقم تعريف id.

ويقوم الكود الآتى بإستخراج جميع محتويات جدول قاعدة البيانات users.

```
QSqlQuery query("SELECT id , name FROM users");
while ( query.next() ) {
        int id = query.value( 0 ).toInt();
        QString name = query.value( 1 ).toString();
  }
```

## عرض سجلات قاعدة البيانات :

يمكن للمبرمج إنشاء واجهة الإستخدام الخاصة به لعرض و تعديل سجلات قاعدة البيانات، أو إستخدام الفصيلتين QSqlTableModel ، QSqlQueryModel، والتى قدمتهما **كيوت** ككائنات جاهزة للتعامل مع السجلات وعرضها، وتتبع هذه الطريقة طريقة نموذج/عرض ( Model / View Framework ) والتى تم تعريفها فى وحدة مكونات واجهة المستخدم الرسومية QtGui Module.

..................................

## المثال :

نريد إنشاء قاعدة بيانات تتكون من رقم مسلسل و اسم وعدد نقاط، مع إمكانية إضافة سجل جديد أو مسحه أو التعديل فيه،
ونبدأ المثال بوضع الثلاثة سجلات التالية :

تتكون قاعدة البيانات من 3 سجلات و كل سجل من 3 حقول.

| NO | Name | Points |
|---|---|---|
| 1 | Mahmoud | 2000 |
| 2 | Ali | 500 |
| 3 | Mohsen | 1200 |

وسوف نقوم بإستخدام قاعدة بيانات من النوع QSQLITE، حيث أنها لاتحتاج إلى مضيف قاعدة بيانات Database host.

لإنشاء قاعدة البيانات فى المثال السابق بلغة SQL

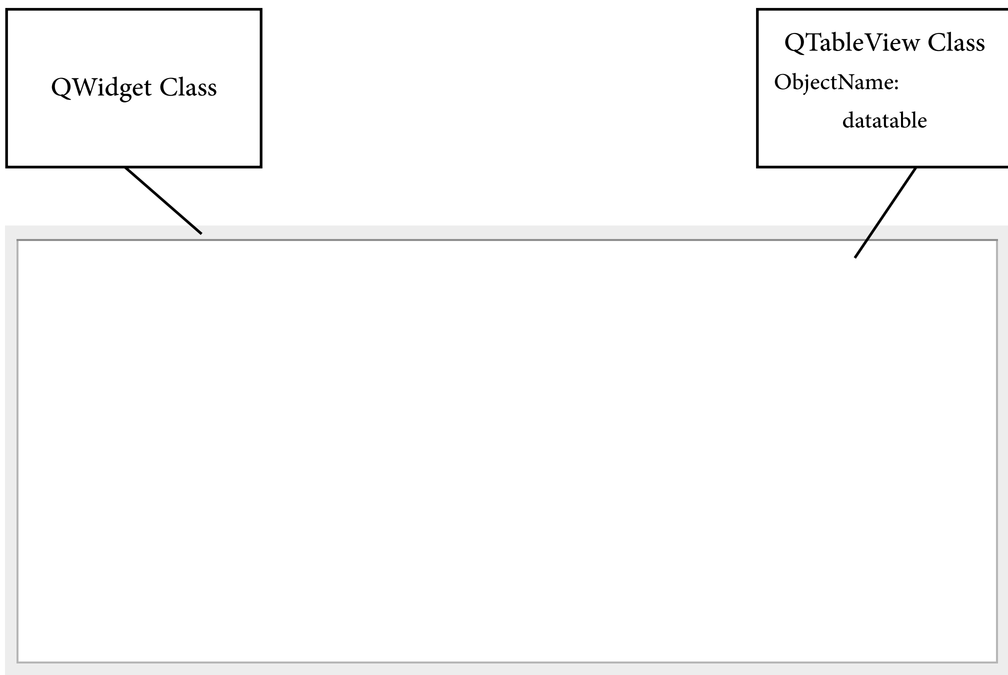
نتبع العبارات التالية:

```
CREATE  TABLE  userpoints(id int , name varchar(20) , points int)
INSERT INTO userpoints VALUES( 1 , “Mohamed“ , 2000)
INSERT INTO userpoints VALUES( 2 , “Ali“ , 500)
INSERT INTO userpoints VALUES( 3 , “Mohsen“ , 1200)
```

و سنرى الآن كيفية إنشاء قاعدة البيانات السابقة بواسطة وحدة التعامل مع قواعد البيانات من **كيوت** QtSQL Module.

نقوم بفتح مشروع جديد بالأداة QtCreator من النوع widget
واسم المشروع sql_module .
وليتعامل المشروع مع قواعد البيانات
نضيف الكود ( QT+= sql ) للملف sql_module.pro.

ثم نصمم widget.ui كالتالى:



widget.cpp

EXAMPLE NO 39

```
#include “widget.h“
#include “ui_widget.h“
#include <QtSql>
#include <QStandardItemModel>
#include <QTableView>

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    QSqlDatabase db = QSqlDatabase::addDatabase(“QSQLITE“);
    db.setDatabaseName(“:memory:“);
    db.open();

    QSqlQuery query;
    query.exec(“create table userpoints (id int primary key,“
            “name varchar(20), points int)“);
    query.exec(“INSERT INTO userpoints VALUES(1, ‹Mahmoud›, 2000)“);

    query.prepare(“INSERT INTO userpoints (id, name, points) “
              “VALUES (:id, :name, :points)“);
    query.bindValue(0, 2);
    query.bindValue(1, “Ali“);
    query.bindValue(2, 500);
    query.exec();

    query.prepare(“INSERT INTO userpoints (id, name, points) “
              “VALUES (:id, :name, :points)“);
    query.bindValue(“:id“, 3);
    query.bindValue(“:name“, “Mohsen“);
    query.bindValue(“:points“, 1200);
    query.exec();
```

```
    int rows = 0;
    QStandardItemModel *mymodel = new QStandardItemModel;
    mymodel->setColumnCount(3);
    query.exec("SELECT * FROM userpoints");
    while (query.next())
    {
        mymodel->setItem(rows , 0 ,new QStandardItem( query.value(0).toString()  ));
        mymodel->setItem(rows , 1 ,new QStandardItem( query.value(1).toString()  ));
        mymodel->setItem(rows , 2 ,new QStandardItem( query.value(2).toString()  ));
        rows++;
    }
    ui->datatable->setModel(mymodel);

}
```

شرح الكود السابق

```
#include <QtSql>
```

إدراج تعريف **QtSql** لتعريف كل الفصائل التى تتعامل مع قواعد البيانات .

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
```

الإعلان عن المتغير **db** لإدارة فصيلة **QSqlDatabase**،
وهى وصلة ربط مع قاعدة بيانات من النوع **QSQLite**.

```
db.setDatabaseName(":memory:");
```

تمرير **:memory:** كإسم لقاعدة البيانات الخاصة بوصلة الربط db،
و **:memory:** هو حالة خاصة بالنوع **QSQLite** حيث يقوم بالتعامل مع قاعدة بيانات يتم إنشائها والتعامل معها فى الذاكرة.

```
db.open();
```

فتح وصلة الربط **db** لإجراء العمليات على قاعدة البيانات من خلالها.

```
QSqlQuery query;
query.exec("create table userpoints (id int primary key,"
           "name varchar(20), points int)");
query.exec("INSERT INTO userpoints VALUES(1, 'Mahmoud', 2000)");
```

تمرير عبارتين **SQL** لإنشاء جدول بقاعدة البيانات، وإضافة سجل بالطريقة المباشرة بإستخدام **exec**.

```
query.prepare("INSERT INTO userpoints (id, name, points) "
              "VALUES (:id, :name, :points)");
```

تحضير عبارة **SQL** بإستخدام الدالة **prepare**، وإضافة متغيرات لها **:id, :name, :points** .

```
query.bindValue(0, 2);
query.bindValue(1, "Ali");
query.bindValue(2, 500);
query.exec();
```

تمرير المتغيرات بالترتيب :

رقم **0** للمتغير **:id**

رقم **1** للمتغير **:name**

رقم **2** للمتغير **:points**

```
query.prepare("INSERT INTO userpoints (id, name, points) "
              "VALUES (:id, :name, :points)");
query.bindValue(":id", 3);
query.bindValue(":name", "Mohsen");
query.bindValue(":points", 1200);
query.exec();
```

تحضير عبارة **SQL** بإستخدام الدالة **prepare** وإضافة متغيرات لها **:id, :name, :points**، وتمرير المتغيرات بواسطة الدالة **bind** بأسماء المتغيرات.

```
int rows = 0;
```
الإعلان عن المتغير **rows** من النوع **int** ليمثل عدد سطور الجدول.

```
QStandardItemModel *mymodel = new QStandardItemModel;
mymodel->setColumnCount(3);
```
تصميم نموذج الجدول لوضع البيانات بداخله.

```
query.exec("SELECT * FROM userpoints");
```
تنفيذ عبارة **SQL** مباشرة لإستخراج محتويات الجدول **userpoints**.

```
while (query.next())
  {
     mymodel->setItem(rows , 0 ,new QStandardItem( query.value(0).toString()  ));
     mymodel->setItem(rows , 1 ,new QStandardItem( query.value(1).toString()  ));
     mymodel->setItem(rows , 2 ,new QStandardItem( query.value(2).toString()  ));
     rows++;
  }
ui->datatable->setModel(mymodel);
```
عملية تكرارية مستمرة : فطالما وجدت سجلات **while query.next** يتم إضافة محتويات كل سجل من حقول داخل النموذج **mymodel**، ثم عرض النموذج بالجدول **datatable**.

نلاحظ هنا إستخدام **toString** للقيم الممثلة للحقول **query.value**، حيث أن القيمة المرتدة من الدالة **value** تكون من النوع **QVariant**، وقد قمنا بشرح هذه الفصيلة سابقاً، حيث يمكن تحويل القيمة **QVariant** إلى أى نوع بيانات مثل **toString , toInt , toDouble**.

فإذا أردنا فى المثال السابق إجراء عملية حسابية على حقل **points**، سنقوم بتحويل القيمة المرتدة من الدالة **value** إلى النوع **int**.

```
int pointsval = query.value(2).toInt();
```

..........................................................................

# البرمجــة الموازيــة

# Multithreaded Programmig

# (QThread Class)

البرمجة الموازية Multithreaded Programming هى من أهم مقومات البرمجة الحديثة التى لا يمكن لأى لغة برمجة الإستغناء عنها، وبالمفهوم العادى يمكننا تعريف البرمجة الموازية بأنها طريقة لتنفيذ أكثر من كود داخل نفس التطبيق وفى نفس الوقت.

طريقة البرمجة العادية تتم بتمرير عبارات الكود لتنفيذها بشكل تسلسلى، حيث يتم تنفيذ عبارة الكود الأولى ثم تتبعها الثانية فالثالثة وهكذا..

أما البرمجة الموازية فتأتى لتمكن المبرمج من تنفيذ مجموعتين من الأكواد فى نفس الوقت، مما يعطى للمبرمج إمكانية حل مشاكل برمجية كثيرة.

ويمكن أن نرى البرمجة الموازية فى كثير من التطبيقات مثل برامج المحادثات الصوتية والمرئية، حيث يتوجب علينا برمجة مجموعتين من الأكواد ليعملا معاً فى نفس الوقت داخل نفس التطبيق ( كود لإرسال البيانات و كود لإستقبال البيانات )، وهذا ما يمكن تحقيقه بواسطة البرمجة الموازية.

تقدم لنا ***كيوت*** البرمجة الموازية من خلال الفصيلة QThread، والتى سنكتفى بتعريفها وشرح مثال بسيط يوضح كيفية التعامل معها.

..............................................................................

# QThread Class

**نسب الفصيلة :**

ترث فصيلة QObject

**تعريف الفصيلة :**

هى الفصيلة المسؤولة عن تنفيذ مجموعة من أكواد البرمجة المتوازية زمنياً مع تنفيذ أكواد التطبيق.

**وظائف الفصيلة :**

- الدالة ()run : هى الدالة التى يوضع بداخلها الكود المـراد تنفيذه بالتـوازى زمنياً مع كـود التطبيق.
- الدالة ()start : تقوم ببدء تشغيل الفصيلة لتنفذ كود الدالة ()run.
- دالة الإرسال ()started : تقوم ببث إشارتها عند بدء تنفيذ كود الدالة ()run.
- دالة الإرسال ()finished : تقوم ببث إشارتها بعد إنتهاء تنفيذ كود الدالة ()run.
- دالة الإرسال ()terminated : تقوم ببث إشارتها عند إيقاف تنفيذ كود الدالة ()run.

ميكانيكية العمل :

ليتم إستخدام هذه الفصيلة بالطريقة الصحيحة نتبع الخطوات الآتية:

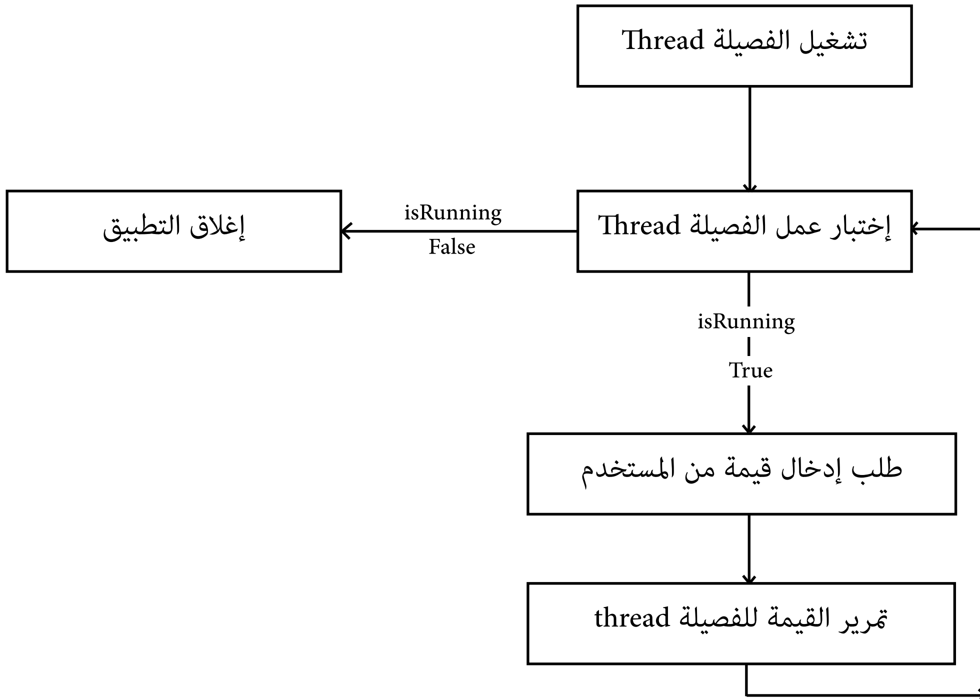
- توليد كود فصيلة جديدة ترث فصيلة QThread.
- تعريف الدالة ()run داخل النطاق العام (public) بالفصيلة الجديدة.
- كتابة الكود المطلوب تنفيذه بداخل الدالة ()run فى الملف التفصيلى للفصيلة الجديدة.
- بدأ التنفيذ عن طريق الدالة ()start.

وببدء تشغيل الدالة ()start يتم بث إشارة دالة الإرسال ()started، ويبدأ تنفيذ كود الدالة ()run، وعند الإنتهاء يتم بث إشارة دالة الإرسال ()finished، وفى حالة توقف التنفيذ لأى سبب مفاجئ يتم بث إشارة دالة الإرسال ()terminated.

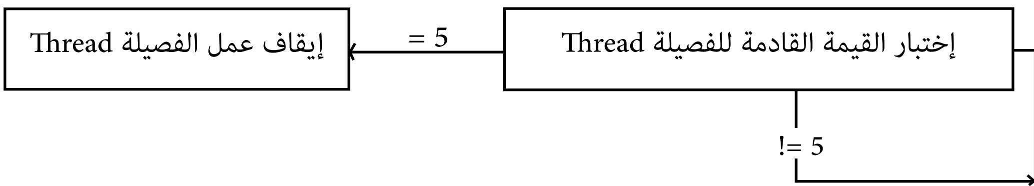
وسيقوم المثال التالى بتوضيح إمكانيات هذه الفصيلة بصورة مبسطة.

مثـــال : يقوم المثال التالى بمجموعتين منفصلتين من العمليات على التوازى.

المجموعة الأولى تقوم بها دالة التطبيق الاساسية main، وتتمثل فى الآتى :




المجموعة الثانية تقوم بها الدالة الموازية للفصيلة Thread، وتتمثل فى الآتى :




وبالتالى فعند تمرير القيمة 5 للفصيلة Thread سيتم عمل إيقاف الفصيلة،
و عند إختبار عمل الفصيلة Thread سوف نحصل على القيمة False،
مما يؤدى إلى إنهاء التطبيق.
ويتكون التطبيق من الملفات الآتية:
main.cpp : الملف الرئيسى و الذى يحتوى على دالة التطبيق الرئيسية main.
mythread.h: ملف تعريف الفصيلة الوارثة لفصيلة QThread.
mythread.cpp: ملف الكود التفصيلى للفصيلة الوارثة لفصيلة QThread.

mythread.h

EXAMPLE NO 40

```
#ifndef MYTHREAD_H
#define MYTHREAD_H
#include <QThread>
class mythread : public QThread
{
   Q_OBJECT
public:
   explicit mythread(QObject *parent = 0);
   void setTestVal(int);
   void run();
private:
   int testval;
};
#endif // MYTHREAD_H
```

mythread.cpp

```
#include «mythread.h»
#include <QDebug>

mythread::mythread(QObject *parent) :
   QThread(parent)
{
   testval = 0;
}
void mythread::setTestVal(int i)
{
   this->testval = i;
   msleep(10);
}

void mythread::run()
{
   while(testval !=5);
   qDebug() << «The thread finished\n»;
   this->testval = 0;
}
```

شرح الكود السابق

**الملف mythread.h**

```
class mythread : public QThread
```

تعريف الفصيلة **mythread** بأنها ترث الفصيلة **QThread**.

```
Q_OBJECT
```

إضافة الماكرو **Q_Object** ليكسب الفصيلة خواص الفصيلة **QObject**.

```
void setTestVal(int);
```

دالة لتقوم بإدخال قيمة المتغير **testval** الموجود بالنطاق **private**.

```
void run();
```

الدالة ()**run** هى الدالة الرئيسية لهذه الفصيلة.
حيث أننا الآن نتكلم عن دالة تقوم بتنفيذ مجموعة أكواد بالتوازى مع الدالة الرئيسية للتطبيق **main**، ومن هذا المنطلق يمكننا أن نعتبر أن الدالة ()**run** هى الدالة **main** للفصيلة **mythread**، وتبدأ بتنفيذ الأكواد الموجودة بداخلها عند إستدعاء الدالة ()**start** كما سنرى داخل الملف **main.cpp**.

........................................................................

**الملف mythread.cpp**

```
void mythread::run()
{
   while(testval !=5);
   qDebug() << «The thread finished\n»;
   this->testval = 0;
}
```

الكود التفصيلى للدالة ()**run**، حيث تقوم **while** بإختبار قيم المتغير **testval** فى عملية تكرارية غير منتهية، فقط تنتهى إذا مررت القيمة **5** إلى المتغير **testval**، وعندها يتم طباعة العبارة **The thread finished**، ثم تمرر القيمة **0** للمتغير **testval**، وينتهى عمل الدالة فتبث الفصيلة إشارة دالة الإرسال ()**finished**.

لابد أن نلاحظ الأتى:

نظراً لأن الكود ( **;( while( testval !=5** ) **هو** كود غير منتهى،
فإن وضعه داخل الدالة **main** فى الملف **main**.cpp سوف يتسبب فى تجميد شاشة التطبيق،
ولكن بوضعه فى دالة ()**run** بفصيلة **QThread** فإنه سيعمل كتطبيق منفصل ولايؤثر على الدالة **main**.

main.cpp

```
#include <QtCore/QCoreApplication>
#include <QTextStream>
#include <QDebug>
#include «mythread.h»
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    mythread tr;
    tr.start();
    QTextStream in(stdin);
    QString val;
    QObject::connect( &tr , SIGNAL(finished()) , &a , SLOT(quit()));
    while(tr.isRunning())
    {
      qDebug() << «The Thread Still Alive\n»;
      qDebug() << «Set the Value to close the thread :»;
      in >> val;
      tr.setTestVal(val.toInt());
    }
  return a.exec();
}
```

شرح الكود السابق

```
mythread tr;
```

الإعلان عن المتغير **tr** لإدارة فصيلة **mythread**.

```
tr.start();
```

الدالة ()**start** لتشغيل الدالة ()**run** بالفصيلة **mythread**.

```
QTextStream in(stdin);
```

الإعلان عن المتغير **in** لإدارة فصيلة **QTextStream**، والتعامل مع وحدة الإدخال **stdin**.

```
QString val;
```

الإعلان عن المتغير **val** لإدارة فصيلة **QString**.

```
QObject::connect( &tr , SIGNAL(finished()) , &a , SLOT(quit()));
```

كود للربط بين دالة الإرسال ()**finished** من الفصيلة **mythread**

بدالة الإستقبال ()**quit** من الفصيلة **QCoreApplication**،

وبالتالى فعندما تبث الفصيلة **mythread** دالة الإرسال ()**finished**،

ستقوم الفصيلة **QCoreApplication** بتشغيل دالة الإستقبال ()**quit** والتى تنهى التطبيق.

```
while(tr.isRunning())
 {
  qDebug() << «The Thread Still Alive\n»;
  qDebug() << «Set the Value to close the thread :»;
  in >> val;
  tr.setTestVal( val.toInt() );
 }
```

تقوم **while** بإختبار الدالة **isRunning**،

والتى ستعود دائما بالقيمة **True** طالما أن الدالة ()**run** فى الفصيلة **mythread** مازالت تعمل،

وبالتالى فلن تنتهى هذه العملية التكرارية حتى تعود الدالة **isRunning** بالقيمة **False**، وذلك عند إنتهاء الدالة ()**run** فى الفصيلة **mythread** من العمل،

وتقوم **in >> val** بطلب إدخال قيمة المتغير **val**.

ثم تقوم الدالة ()**setTestVal** بتمرير القيمة المدخلة إلى المتغير **testval** الموجود بالنطاق **private** فى الفصيلة **mythread**.

عند تشغيل التطبيق :

فى حالة إدخال أى رقم ستظل العملية التكرارية ولن يحدث شئ،

أما فى حالة إدخال الرقم **5** سيحدث الآتى :

- سيتم تمريره إلى المتغير **testval** بالفصيلة **mythread**.
- ونتيجة لإختباره فى داخل الدالة ()**run** ستنهى عملها، وتبث إشارة الدالة ()**finished**.
- وبالتالى سيعود إختبار الدالة **isRunning** بالقيمة **False** مما ينهى العملية التكرارية.
- ونتيجة لبث إشارة الدالة ()**finished** ستقوم الفصيلة **QCoreApplication** بتشغيل دالة الإستقبال ()**quit** والتى تنهى التطبيق.

## بعض الكتب الهامة باللغة الإنجليزية :

- Foundation of Qt Development
- C++  GUI Programming with Qt 4 (2nd Edition)
- C++ GUI Qt 4
- Qt 4 Professional programming with C++

## مواقع هامة :

- qt.nokia.com
- www.qtforum.org

## فى حالة وجود أى ملاحظات نرجو التواصل والإتصال على :


- ahbanna@gmail.com

# ملحقـــــات

# ملحـــق ( ١ )

## إدارة الذاكـــــرة

## Memory Managment

أهم جزئين من الذاكرة يتم التعامل معهما هما ( The Heap ) و ( The Stack ).

| The Heap Memory |
|---|
| الحجـــم : حجم كبير يكفى لتسجيل كم هائل من البيانات و التعامل معها. |
| الإستخدام : كل متغير يعرف بمؤشر( Pointer ) يتم وضعه فى الذاكرة heap. |
| int *p[100]; |
| المســـح : توضع المتغيرات فى الذاكرة Heap بواسطة new ويتم المسح بواسطة delete. |
| char *string = new char[100]; <----- Heap تم حجز المساحة فى الذاكرة |
| ... |
| delete string; <----- Heap تم المسح من الذاكرة |
| لابد من إستخدام delete لمسح المساحة المحجوزة من الذاكرة حتى لا يؤدى ذلك إلى إهدار مساحات بدون فائدة، ونظراً لكبر حجم الذاكرة Heap فينصح بوضع الفصائل و المصفوفات الكبيرة بها بدلاً من Stack. |

| The Stack Memory |
|---|
| الحجـــم : حجم صغير ويفضل وضع البيانات ذات الحجم الصغير بها. |
| الإستخدام : كل متغير لا يعرف بمؤشر( Pointer ) يتم وضعه فى الذاكرة stack. |
| int p[100]; |
| المســـح : يتم وضع المتغيرات فى الذاكرة stack بمجرد الدخول لنطاق الكود ( { ) ويتم مسحها منه بمجرد الخروج منه ( } ) |
| { |
| int p[100]; <----- Stack تم حجز المساحة فى الذاكرة |
| ... |
| } <----- Stack تم المسح من الذاكرة |

لذلك فان أفضل طريقة لإدارة الذاكرة داخل ***كيوت*** هى بوضع الفصيلة الأب العامة (parent) فى الذاكرة Stack، و وضع باقى الأبناء و التفرعات فى الذاكرة Heap، وبالتالى فبمجرد غلق البرنامج ( مسح الفصيلة الأب ) يتم مسح كل الأبناء بشكل تلقائى لإخلاء الذاكرة تماماً.

# ملحـــــق ( ٢ )

## هيكل بنية الفصائل فى سى ++
## C++ Classes Structure

نستعرض فى البداية مفاتيح الفصائل فى سى++ ( C++ Classes KeyWord ).

class - new - delete - public - private - protected - friend

وبعض الرموز :

( . , : , :: , -> )

الهيكل المعيارى للفصيلة ( Class Standard Structure ).

```
class myclass : inherites class
{
myclass()            <--- Constructor Function دالة البناء
~myclass()           <--- Destructor Function دالة الهدم

public:              <--- نطاق المتغيرات و الدوال العامة

private:             <--- نطاق المتغيرات و الدوال الخاصة

protected:           <--- نطاق المتغيرات و الدوال المحمية

};

myclass :: myclass()         <--- Constructor Function تفصيل دالة البناء
{
}
myclass :: ~myclass()        <--- Destructor Function تفصيل دالة الهدم
{
}
```

# ملحـــــق ( ٢ )

## هيكل بنية الفصائل فى سى ++
## C++ Classes Structure

عند الإعلان عن مؤشر Pointer يرمز لفصيلة.

```
myclass *ptr;
```

يتم إستخدام new لحجز مكان فى الذاكرة بحجم myclass.

```
ptr = new myclass;
```

تستخدم delete لإخلاء المكان فى الذاكرة.

```
delete ptr;
```

يستخدم الرمز ( <- ) للوصول إلى أعضاء الفصيلة
فى حالة تعريف الفصيلة فى الذاكرة Heap.

```
ptr->class_object;
```

يستخدم الرمز ( . ) للوصول إلى أعضاء الفصيلة
فى حالة تعريف الفصيلة فى الذاكرة Stack.

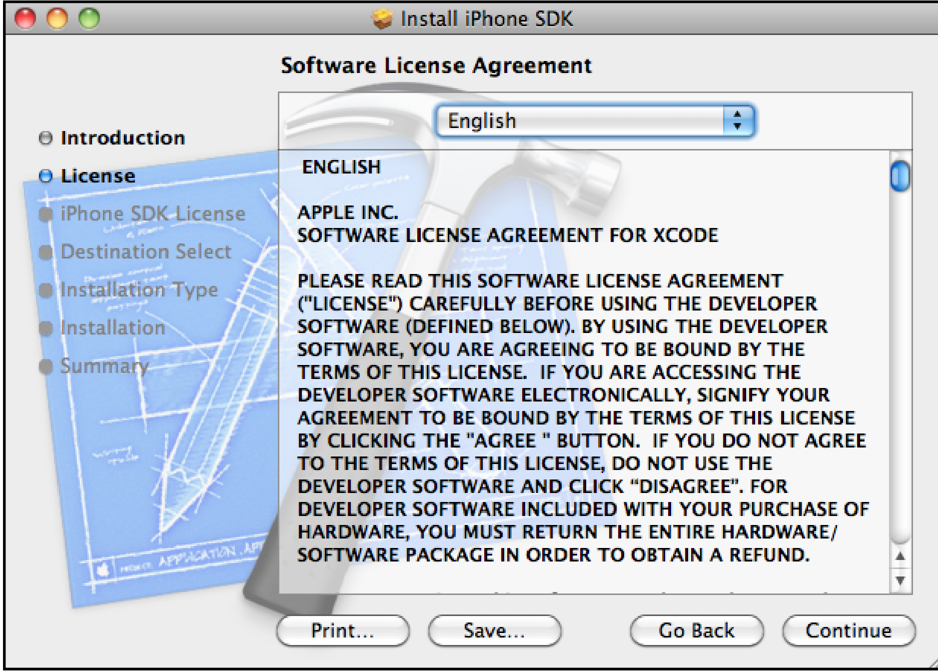
```
myclass ptr;
ptr.class_object;
```

..........................................................................

# ملحـــــق ( ٣ )

## تنصيب كيوت على نظام ماك Mac OS X

أولاً : يتم تنصيب xcode_iphone_sdk.dmg، ويمكنك تحميله مجاناً من موقع apple.com




ثانيا : يتم تنصيب Qt4.7.dmg، ويمكنك تحميله مجاناً من موقع qt.nokia.com

# ملحـــــق ( ٣ )

مكان وجود جميع أدوات **كيوت** :

/Developer/Application/Qt .

وهذه الأدوات :

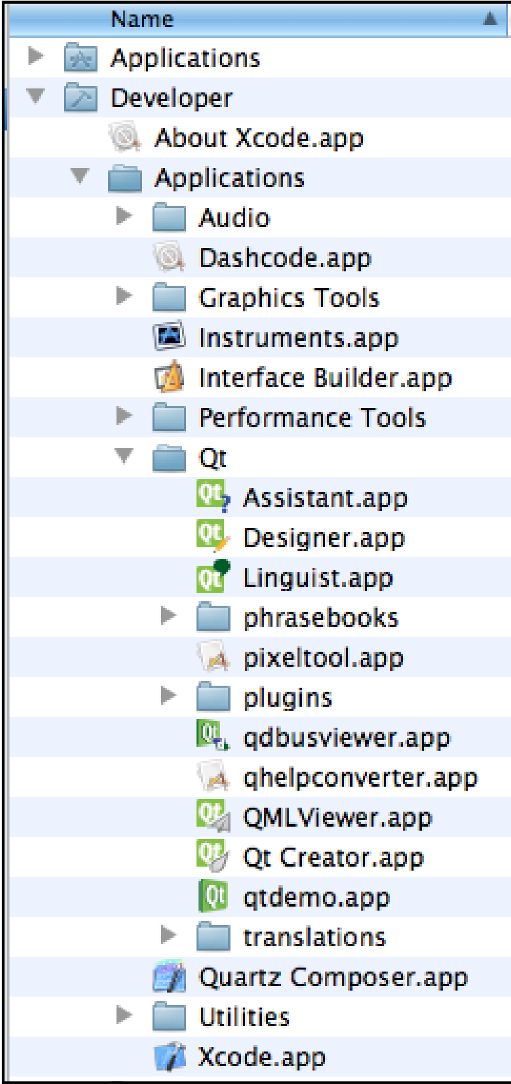
Assistant.

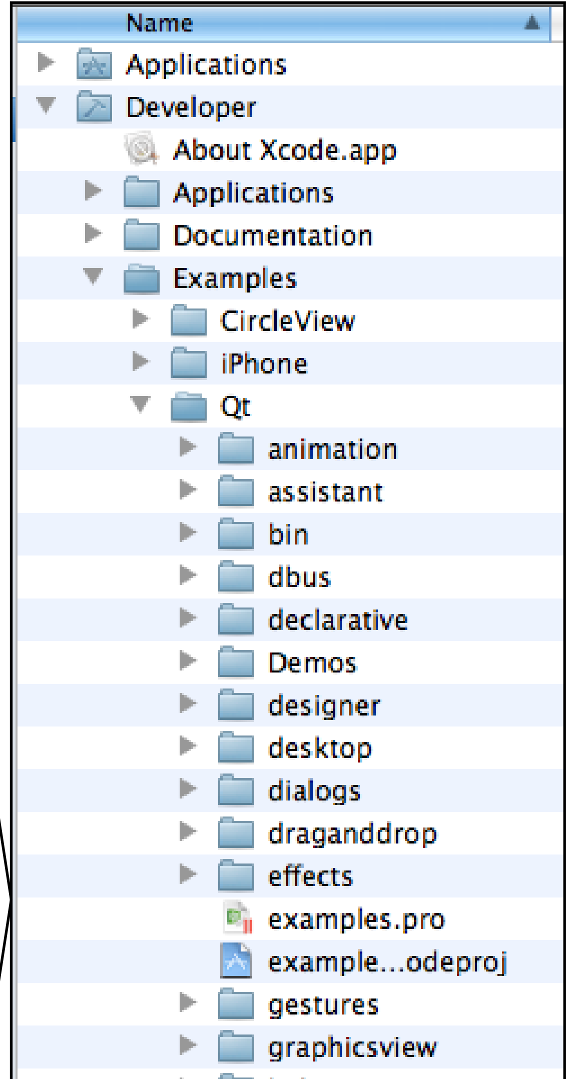
Designer.

Linguist.

وهذه الأدوات تجتمع داخل:

Qt Creator.




مكان وجود جميع أكواد أمثلة **كيوت** :

/Developer/Examples/Qt .

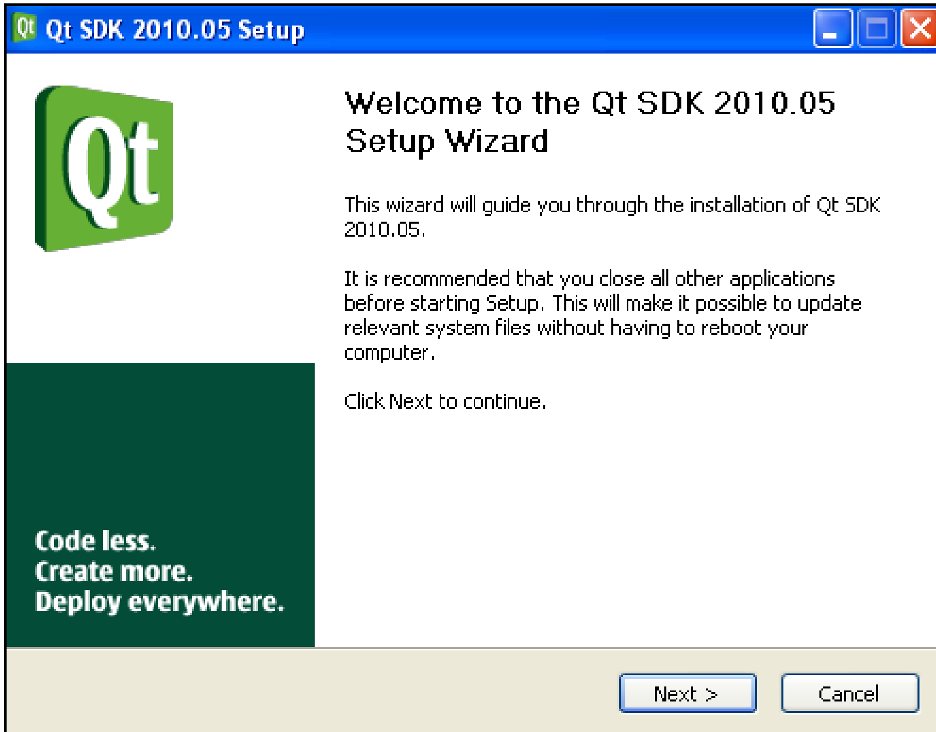
ويمكنك تشغيل هذه الأمثلة عن طريق التطبيق الموجود فى المجلد السابق :

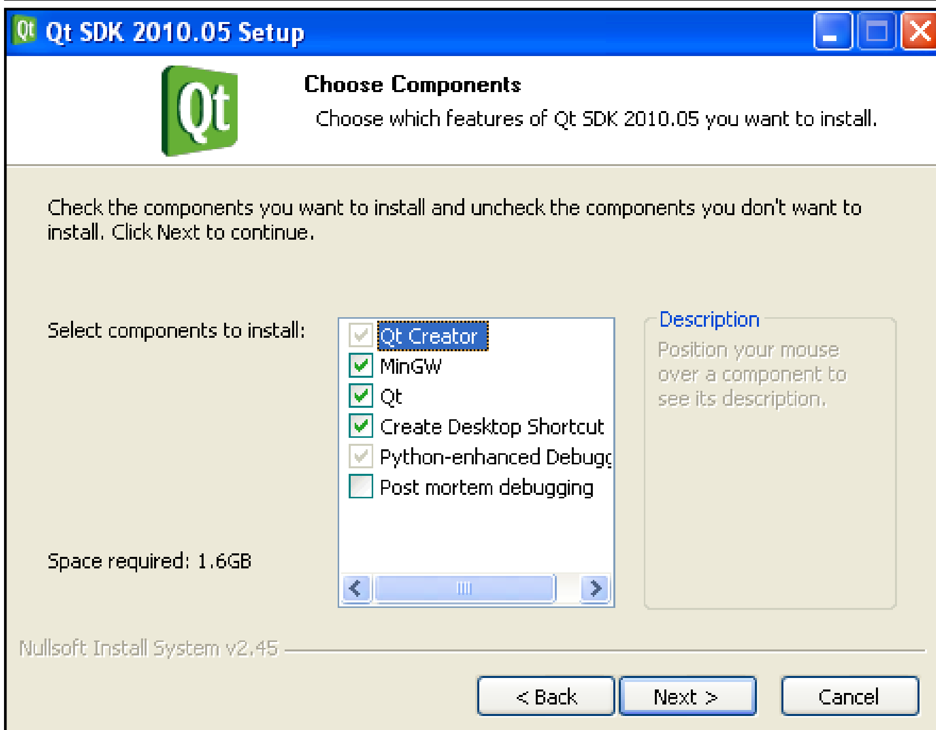
**/Developer/Application/Qt/qtdemo.**

# ملحـــــق ( ٣ )

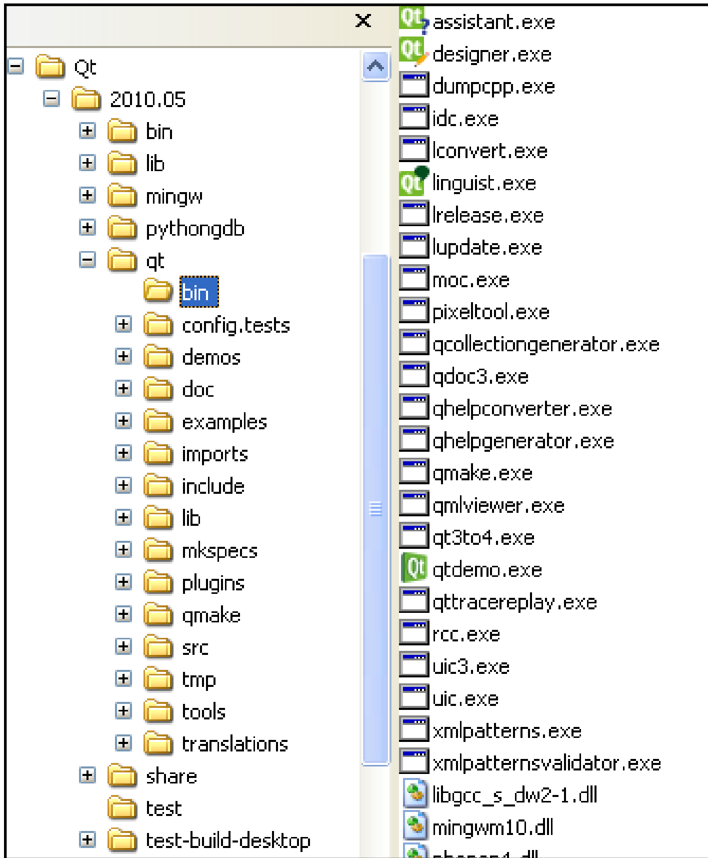
## تنصيب كيوت على نظام ويندوز Windows

يمكنك تحميله مجاناً من موقع qt.nokia.com ولا يلزمك أى أدوات أخرى.



# ملحـــق ( ٣ )




مكان وجود جميع أدوات **كيوت**:

C:\Qt\2010.05\qt\bin.

وهذه الأدوات :

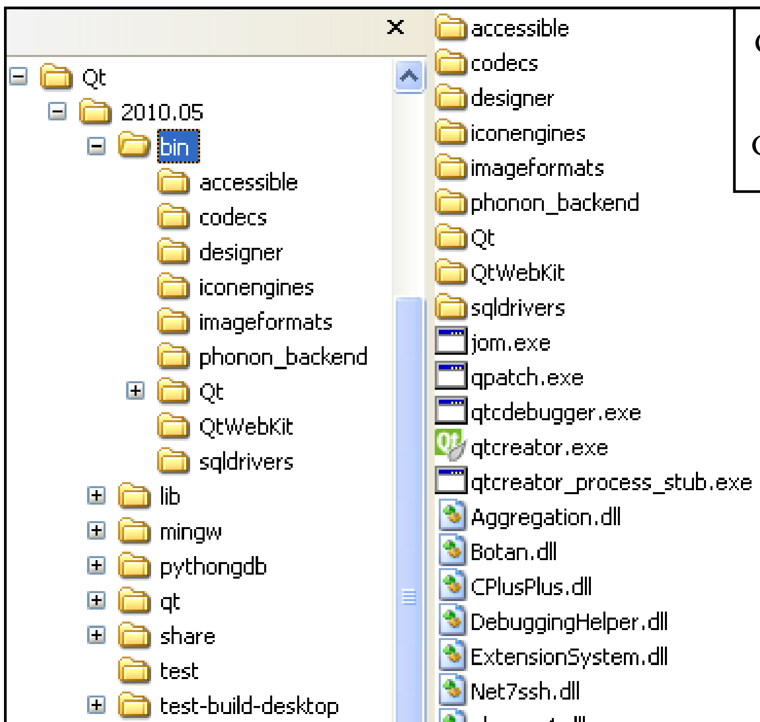
Assistant.

Designer.

Linguist.

وهذه الأدوات تجتمع داخل:

Qt Creator.




مكان وجود الأداة Qt Creeator

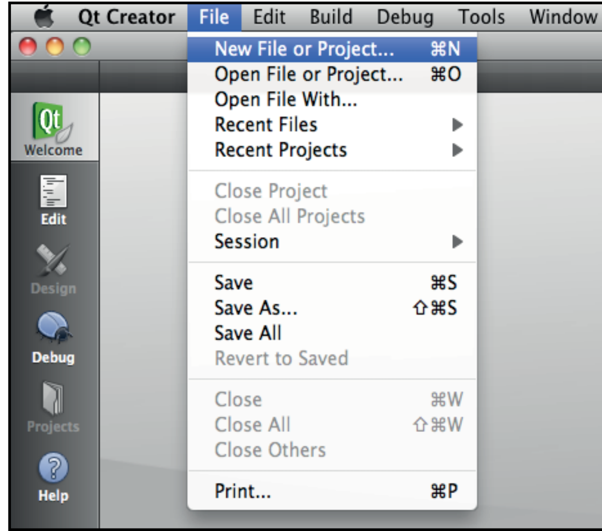
C:\Qt\2010.05\bin.

# ملحـــــق ( ٤ )

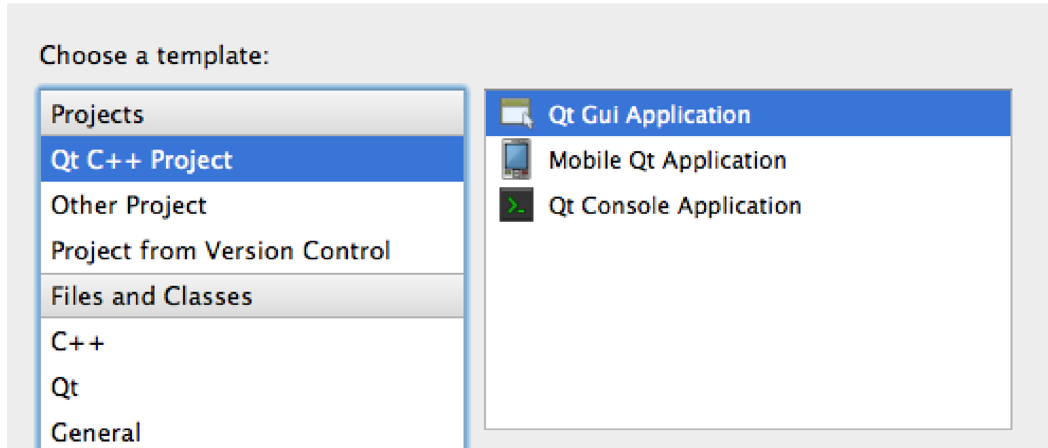
## كيفية بدء تطبيق بواسطة الأداة QtCreator

نبدأ بتشغيل Qt Creator

- ثم نختار New File or Project من القائمة File




-ثم تظهر شاشة إختيار نوع التطبيق ( رسومى أو غير رسومى )




للتطبيق الرسومى :

نختار Qt Gui Application

للتطبيق الغير رسومى :

نختار Qt Console Application

# ملحـــق ( ٥ )

| ClasName | | Inherits | |
|---|---|---|---|

**Macros , Constructors and Destructors**

| Level | Type | Function member |
|---|---|---|
| *NON* | *Macro* | *Q_OBJECT* |

**Variables**

| Level | Type | Name | Info |
|---|---|---|---|
| | | | |

**Setters and Getter Function members**

| Setters Functions | | | Getters Functions | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

**Slots and Signals**

| Q_SLOTS | | | Q_SIGNALS | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

# ملحـــــق ( ۵ )

| ClasName | | Inherits | |
|---|---|---|---|

**Macros , Constructors and Destructors**

| Level | Type | Function member |
|---|---|---|
| *NON* | *Macro* | *Q_OBJECT* |

**Variables**

| Level | Type | Name | Info |
|---|---|---|---|
| | | | |

**Setters and Getter Function members**

| Setters Functions | | | Getters Functions | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

**Slots and Signals**

| Q_SLOTS | | | Q_SIGNALS | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

# ملحـــــق ( ۵ )

<table>
<tr><td>ClasName</td><td colspan="2"></td><td>Inherits</td><td colspan="2"></td></tr>
<tr><td colspan="6">Macros , Constructors and Destructors</td></tr>
<tr><td>Level</td><td>Type</td><td colspan="4">Function member</td></tr>
<tr><td>NON</td><td>Macro</td><td colspan="4">Q_OBJECT</td></tr>
<tr><td colspan="6">Variables</td></tr>
<tr><td>Level</td><td>Type</td><td colspan="2">Name</td><td colspan="2">Info</td></tr>
<tr><td></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
</table>

<table>
<tr><td colspan="6">Setters and Getter Function members</td></tr>
<tr><td colspan="3">Setters Functions</td><td colspan="3">Getters Functions</td></tr>
<tr><td>Level</td><td>Type</td><td>Function</td><td>Level</td><td>Type</td><td>Function</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="6">Slots and Signals</td></tr>
<tr><td colspan="3">Q_SLOTS</td><td colspan="3">Q_SIGNALS</td></tr>
<tr><td>Level</td><td>Type</td><td>Function</td><td>Level</td><td>Type</td><td>Function</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

# ملحـــــق ( ٥ )

| ClasName | | Inherits | |
|---|---|---|---|

**Macros , Constructors and Destructors**

| Level | Type | Function member |
|---|---|---|
| *NON* | *Macro* | *Q_OBJECT* |

**Variables**

| Level | Type | Name | Info |
|---|---|---|---|
| | | | |

**Setters and Getter Function members**

| Setters Functions | | | Getters Functions | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

**Slots and Signals**

| Q_SLOTS | | | Q_SIGNALS | | |
|---|---|---|---|---|---|
| Level | Type | Function | Level | Type | Function |
| | | | | | |

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَقُلْ رَبِّ زِدْنِي عِلْما

سورة ( طه : ١١٤ )

* نظراً لإستحواذ شركة مايكروسوفت وهيمنة تطبيقاتها على معظم السوق العربية وتجنيد العقول لخدمة تطبيقاتها.

* ونظراً لرؤيتنا أن كيوت هى إحدى وسائل الخلاص من هذه الهيمنة وفتح الآفاق أمام العقول المبدعة.

* ونظراً لعدم وجود كتاب أو شرح وافى لهذه الأداة باللغة العربية.

قررنا إعداد هذا الكتاب

والذى يعتبر أول كتاب يتناول شرح أساسيات كيوت باللغة العربية.

كيف يمكنك تعلم كيوت؟

ما هى إمكانيات كيوت ؟

هل يكفيك أن تعلم من هم عملاء كيوت !!!

Designed for Google Earth

NOKIA Connecting People

هل يكفيك أن تعلم أنهم يستخدمون كيوت لبرمجة تطبيقاتهم !!!