المحاضرة الثانية

مكتبة المستقبل Future Library هاتف 2119292

مقدمة:

سنتكلم في هذه المحاضرة عن مجموعة من المفاهيم، منها ما هو جديد ومنها ما هو مألوف بل بديهي بالنسبة إلينا وذلك بسبب دراستنا للغة ++C، لذا لن نعيد شرح البديهيات وإنما سنشير إليها إشارةً..
لدينا ٣ نقط في المحاضرة السابقة يجب أن نعقب عليها:

١. ذكرنا أن الحقول والتوابع ضمن الصف لها عدة أنواع: (public, private, protected, default)، والنقطة الهامة هي أن default ليست كلمة محجوزة مثل باقي الكلمات، وإنما يكون الحقل أو التابع default عندما لا يسبق بأي كلمة من الكلمات المتبقية: (public, private, protected).

٢. قلنا أن جميع الصفوف في Java مشتقة من الصف Object، وهذا ما يسمى بـ (The singly rooted hierarchy)، والفكرة أن هذا الصف يحوي مجموعة من التوابع التي يمكن استخدامها مباشرة في أي class أو نستطيع عمل (Overriding) لها، وبالتالي فإن جميع صفوف Java تتشارك في هذه التوابع، والمفروض من المبرمج أن يعيد كتابتها (Overriding) لكل class يبرمجه، ولكن مبرمجينا يتكاسلون عن هذا.. من هذه التوابع:
تابع نسخ object (clone)، وتابع اختبار المساواة (equals) وغيرها من التوابع الهامة..
إذا لم يتطرق لها الدكتور فسأشرحها في أحد المحاضرات القادمة.

٣. تكلمنا أن للـ JVM عدة أنواع بحسب الآلة التي تشغل برنامج الـ Java، إذ أن الآلات تختلف عن بعضها في الموارد *وضوحاً: الموبايل يختلف في موارده عن السيرفر*، لذا نجد أن لدينا الأنواع التالية:
J2ME: ويستخدم على الأجهزة الصغيرة (Micro) ومثالها أجهزة الموبايل.
J2SE: (Standard Edition) ويستخدم في تطبيقات Java التي تشغل على الانترنت.
J2EE: (Enterprise Edition) مثل سابقتها ولكن مع أدوات أخرى، ومثالها (applet, Jsp).

بالنسبة لـ J2ME لدينا طبقتين بين الـ JVM والتطبيق البرمجي وهما (CLDC, Profile) وهما طبقتان تتعلقان بخصوصية أجهزة الموبايل..

:Everything is an Object

- ذكرنا في المحاضرة السابقة أن كل مكونات Java تقريباً هي classes حيث توفر لك اللغة مجموعة كبيرة من الصفوف الجاهزة، كما يمكنك تعريف صفوفك الخاصة..

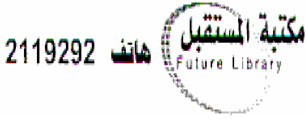
لكي نستطيع أن نتعامل مع الـ class يجب أن نعرف object منه، ولا يمكن التعامل مع الـ objects إلا بطريقة الحجز الديناميكي، وبالتالي فنحن مضطرون للتعامل مع هذه الـ object عن طريق المؤشرات .(references)

- كيف يتم الحجز الديناميكي؟ يتم عن طريق تعليمة (new) على الشكل التالي:

```
String s = new String("Ammar");
```

يوضح لنا المثال طريقة الحجز، حيث أن العبارة (String s) قد عرفت reference من النوع String، ثم قامت تعليمة (new) بإنشاء object في الذاكرة وجعل s يؤشر عليه عن طريق عملية الإسناد، أما ما تبقى ("Ammar") فهو استدعاء للـ constructor الذي يأخذ متحولاً من نوع .String

- ليس علينا أن نهتم بهدم الـ objects بعد إنشائها إذ أن الـ gc (garbage collector) تتحسس أن الـ object لم يعد مستخدماً فتقوم بهدمه تلقائياً، وسنعرف من خلال المحاضرة متى يصبح الـ object غير مستخدم.



حالة خاصة:

ذكرنا مراراً وتكراراً أن جميع عناصر Java هي objects تحتاج لحجز ديناميكي، ولكن ألا يسبب هذا حرجاً في بعض الأحيان؟؟

تصور مثلاً أنني كلما أردت أن أعرف متحولاً من نوع (integer) أنا بحاجة لكتابة العبارة التالية بأكملها:

```
Integer i = new Integer(4);
```

لحل هذه المشكلة أنشأت Java ما يدعى بـ (primitive types) وهي عبارة عن أنماط عادية *أي ليست classes* نتعامل معها كما نتعامل مع المتحولات في باقي لغات البرمجة، أي أنها ليست objects ولا يوجد مؤشر عليها، كما أنها تمرر للتوابع (by value) بعكس الـ objects التي تمرر (by reference).

التمرير (by value): تعني أننا ننسخ نسخة جديدة عن المتحول المدخل للتابع، وبالتالي التغيير على المتحول داخل التابع لا تؤثر على المتحول الخارجي.

أما التمرير (by reference): هنا يكون المتحول داخل التابع هو نفسه المدخل من الخارج وبالتالي التعديل عليه داخل التابع سيؤثر على المتحول الخارجي.

لم تمنع Java المبرمج من التعامل مع هذه الأنماط كـ objects فأتاحت ما يعرف بـ **(Wrapper type)** وهي صفوف مقابلة للـ primitive types.

الجدول التالي يبين لنا أنواع الـ primitive types:

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	Boolean
char	16-bit	Unicode 0	Unicode 2^{16}- 1	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15}	$+2^{15}-1$	Short
int	32-bit	-2^{31}	$+2^{31}-1$	Integer
long	64-bit	-2^{63}	$+2^{63}-1$	Long
float	32-bit	IEEE754	IEEE754	Float
double	64-bit	IEEE754	IEEE754	Double
void	—	—	—	Void

مثال :

```
char c1 = 'a';                             // primitive types
Character c2 = new Character('b');         // Wrapper type
```

نلاحظ الفرق بين تعريف متحول char في الطريقتين.

مكتبة المستقبل Future Library هاتف 2119292

ملاحظات:

١. primitive types تخزن في الـ stack في الذاكرة (RAM) *وهو صغير الحجم سريع الوصــول*، أما الـ objects فتخزن بما يعرف بـ Heap *وهو كبير الحجم بطيء الوصول*، والذي يخزن علــى في الـ stack هو الـ reference الذي يشير إلى الـ object.

٢. من الميزات الهامة التي تقدمها Java أن حجم المتحولات من النوع primitive types ثابت على جميع الأجهزة والأنظمة التي تشغل Java، وليس متغيراً تبعاً للآلة كما في C++، وهذا يعني أن حجم المتحول من النوع integer على سبيل المثال يساوي ٤ بايت أينما كان.

دورة حياة المتحولات في Java:

اعتدنا في C++ على أن دورة حياة المتحولات تنتهــي بنهاية الـ scope التي تعرف داخلها وهذا صحيح فــي Java من أجل المتحــولات مــن النــوع primitive types، والمثال المجاور يوضح الفكرة..

```
{
    int x = 12;
    // Only x available
    {
        int q = 96;
        // Both x & q available
    }
    // Only x available
    // q "out of scope"
}
```

ولكن ماذا عن الـ objects؟

```
{
   String s = new String("Ammar");
}
```

في المثال المجاور لدينا object مـن النـوع String مؤشر عليه بـ s.

عندما نصبح خارج الـ scope فإن المؤشر s يمـوت وبالتالي يبقى الـ object في الذاكرة بدون أي مؤشر عليه، هنا تأتي مهمة الـ JVM حيث يضع علامة على هذا الـ object، وعندما يمتلئ حيز معين من الذاكرة يعمل الـ gc (garbage collector) ويقوم بحـذف هذا الـ object وأمثاله.

نتيجة: يصبح الـ objects غير مستخدم وبالتالي جاهز للحذف بالنسبة للـ gc عندما لا يبقى أي مؤشر يشير عليه في البرنامج، علماً أن هدم المؤشرات (references) يتم بشكل طبيعي مع نهاية الـ scope.

كتابة برنامج بلغة Java:

سنبدأ بدراسة البنية الأساسية لبرنامج Java وهي: الـ class.

```
(class modifier) class  (class name)
{
      instance fields
      instance methods
      class fields
      class methods
      constructors
}
```

يتكون الـ class مما يلي:

١- نوعه (class modifier): وهو ويأخذ أحد الشكلين التاليين:

- **public**: وتعني أن الصف عام، أي أنه يمكن رؤيته وإنشاء objects منه خارج الـ package.
- **unpublic**: أي أننا لا نكتب أي شيء قبل كلمة class، وهـذا يعنـي أن الـصف خـاص بالـ package التي تحويه، أي لا يمكن رؤيته وإنشاء objects منه إلا للصفوف التي تقع معه في نفـس الـ package.

٢- الكلمة المحجوزة **class**.

٣- اسم الـ class.

٤- constructors: وهي التوابع التي يستدعى أحدها عند تعريف object من الصف..

٥- وهو ما يعرّف ضمن الـ class من توابع (methods) أو حقول (fields) ولها نوعان:

a. خاصة بالـ object وتدعى (instance):

إن المتحولات التي تعرف على أنها instance تكون مستقلة في كل object، أي أننا عندما ننـشئ object من صف ما ، فإننا نكون قد أنشأنا نسخ جديدة من هذه المتحولات، وحجزنا لها مساحة فـي

الذاكرة، وبالتالي فإن قيمتها تختلف من object لآخر.. باختصار : لكل object نسخته الخاصة من هذه المتحولات.

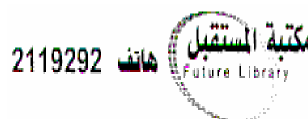
كذلك التوابع الــ instance فهي خاصة بالــ object، وبالتالي فإنها لا تستطيع التعامل إلا مع المتحولات الــ instance الخاصة بالــ object الذي يحتويها.

b. تتعلق بالــ class كنمط وليست خاصة بــ object ما:

وهي ما يعرف بالمتحولات الــ static والتوابع الــ static، وسنفصل فيها بعد قليل..

لنأخذ مثالاً يوضح لنا المفاهيم السابقة:

```
public class Student
{
      private String name;
      private int age;
      static int number;

      public Student (String name) {            // first constructor
            this.name = name;
            Student.number++;
      }

      public Student (String name, int age) {   // second constructor
            this.name = name;
            this.age = age;
            Student.number++;
      }

      public void setName(String name) {
            this.name = name;
      }

      public void setAge(int age) {
            if (age > 5)
                  this.age = age;
      }

      public String getName() {
            return this.name;
      }

      public int getAge() {
            return this.age;
      }

      public static int getNumber() {
            return Student.number;
      }
}
```



المثال السابق مليء بالمفاهيم الجديدة والملاحظات الهامة، فلنركز فيه جيداً ولنشرحه خطوة خطوة:

- نلاحظ أن نوع الصف public، وهذا يعني أننا نستطيع تعريف objects منه في packages أخرى غير الـ package التي تحتويه.
- الصف Student يحوي حقول من النوع instance وهما:(name - age)، وهذا يعني أننا كلما عرفنا object من هذا الصف سيملك نسخة خاصة من هذين المتحولين تختلف عن نسخة أي object آخر.

<u>**مثال:**</u> قد يكون لدينا two objects في برنامجنا أحدهما اسمه "أحمد" وعمره ٢٠، والآخر اسمه "علي" وعمره ٣٠، وبالتالي فهما مختلفان في الحقلين (name - age).

- كما يحوي الصف عدة توابع (methods) من النوع instance وهي (setName, setAge, getName, getAge)، وهذه التوابع لها وصول إلى الحقلين (name - age) الخاصين بالـ object الذي يحوي هذه التوابع.
- نلاحظ وجود بانيين (constructors)، ونتذكر أن الـ constructor هو تابع لا يرد أي قيمة، ويكون اسمه مطابق لاسم الـ class، ويستدعى عند إنشاء object، ويمكن أن يحوي الـ class عدة بواني يستدعي المبرمج أحدها عند إنشاء الـ object وهذا ما يعرف اصطلاحاً بـ (overload).

<u>**ملاحظات:**</u>

١. مما اشتهر بين مبرمجي Java وأصبح عرفاً أن يبدأ اسم الصف بحرف كبير، ولكن بدءه بحرف صغير ليس خطأً، كما اشتهر على تبدأ أسماء الـ method وأسماء الحقول بحرف صغير.
٢. كما ذكرنا في المحاضرة السابقة: ليكون الصف محققاً لمفهوم الـ OOP يجب أن تكون الحقول private أي لا يمكن تغيير قيمتها من خارج الصف مباشرة، وإنما يتم ذلك عن طريق التابعين الشهيرين المعرفين (set - get) لكل حقل ضمن الصف وهما من النوع public، مما يساعد مبرمج الصف على وضع قيود على القيم التي ستوضع في الحقول، ومثال ذلك تابع setAge حيث وضعنا قيداً على العمر بحيث يكون أكبر من ٥.
٣. نلاحظ أننا استخدمنا الكلمة المحجوزة this والتي تمثل reference على الـ object الذي نعمل ضمنه.
٤. يجب أن يكون الـ constructor من النوع public، لأنه يستدعى دائماً من خارج الـ class .
٥. هناك سؤال هام: لماذا لم نعرف هادماً (destructor) للـ class؟؟

الجواب: لأن (garbage collector) قد تحمل عنا عبء هدم الـ object وكل ما يحويه من حجز ديناميكي، وقد تكلمنا عن هذا في المحاضرة السابقة.

٦. يمكن تعريف reference على object ما ضمن حقول الـ class *التجميع Composition*.
٧. لم أجد من الضروري أن أشرح بنية الـ method لأنها مطابقة لبنية التابع في C++، ولكن يجب الانتباه إلى أن المتحولات ضمن الـ method لا تأخذ قيماً ابتدائية بشكل تلقائي وإنما بشكل يدوي.

٨. كان بالإمكان إعطاء قيمة ابتدائية للحقول الـ instance عند تعريفها مباشرة، ولكن الأصح أن يكـون هـذا عـن طريق الباني.

٩. إن Java تعطي قيمـاً ابتدائيـة للحقـول ذات الأنـواع (primitive types)، وهذه القـيم يوضـحها الجـدول التالي:

Primitive type	Default
boolean	false
char	'\u0000' (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

مفهوم الـ static:

وهو مفهوم مهم جداً وخصوصاً في Java، ويختلف قليلاً عن مفهوم الـ static في C++:

ملاحظة: الكثير من الأسئلة التي تمثل (مطبات) في الفحص تركز على مفهوم static.

لنفرض أنني أريد أن أعرف حقلاً ما في class، بحيث تكون قيمته نفسها عند جميع الـ object ، أي أن هذا الحقل ليس خاصاً بـ object معين بل هو مشترك بين الجميع، ومثل هذه الحقول عادة تكون متعلقة باسم الـ class.

أعتقد أننا بحاجة لمثال حتى يتضح القصد..

لنعد إلى مثال الصف Student حيث عرفنا فيه الحقل number على أنه static.

يمثل الحقل number عداداً نزيده بمقدار ١ كلما عرفنا object جديد ، وبالتالي فهو يحوي عدد الطلاب في البرنامج حتى الآن.

نلاحظ أن هذا الحقل ليس خاصاً بـ object معين وإنما يعتبر من خواص الـصف Student، وبالتـالي لا يجوز أن يحوي كل object نسخة منه لأن هذا يمثل هدراً في الذاكرة وإنما يكون الحقل نفسه مـشتركاً بـين جميع الـ objects من النوع Student، ونلاحظ أننا نصل إليه عن طريق اسم الـ class :
(Student.number) ، كما يمكن الوصول إليه عن طريق الـ object.

مثال:

لنكتب برنامجاً صغيراً يوضح لنا مفهوم الـ static:

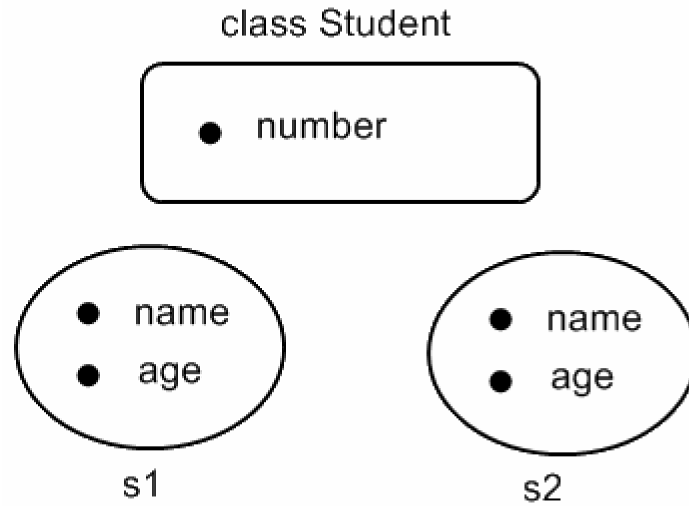
```
System.out.println("student number = " + Student.number);
Student s1 = new Student("Ammar", 20);
System.out.println("student number = " + s1.number);
Student s2 = new Student("Amjad", 25);
System.out.println("student number = " + s2.number);
```

إن خرج البرنامج السابق سيكون كالتالي:

```
student number = 0
student number = 1
student number = 2
```

نلاحظ أنه قبل إنشاء أي نسخة من الصف Student كانت قيمة الحقل number تساوي الصفر، وكلما أنشأنا object جديد زادت القيمة بمقدار ١ *وذلك لأنني أزيدها في الـ constructor*.
ملاحظة: التعليمة (System.out.println()) هي تعليمة الطباعة في الخرج النظامي في Java.

الشكل التالي يوضح مفهوم الحقول في الصف بأنواعها، حيث نلاحظ كما أسلفنا أن لكل object نسخته الخاصة من الحقول ذات النوع instance، بينما يكون الحقل ذو النوع static مشتركاً بين الجميع..



ماذا عن التوابع من النوع static؟؟
كما أن الحقل من النوع static ليس حكراً على object معين، كذلك الـ static method عبارة عن تابع يستدعى من اسم الصف ولا يخص object معين، وبالتالي لا يجوز له أن يصل إلى أي حقل instance ولا أن يستدعي أي تابع instance، لأنهما خاصان بالـ object الذي يقبعان فيه، وبالتالي فإن الـ static method لا يستطيع التحكم إلا بالحقول من النوع static.

<u>مثال:</u>
التابع getNumber في الصف Student حيث يتعامل مع الحقل number والذي هو بالضرورة من النوع static.

هناك ميزة هامة للتوابع الـ static، وهي أنها يمكن أن تستدعى من اسم الـ class وبالتالي لست مضطراً لتعريف object من الـ class لاستدعائها، وهذا يشبه كتابة تابع حر (ليس ضمن class) في C++.

<u>مثال:</u>
في Java لدينا class اسمه Math يحوي جميع التوابع الرياضية مثل (.. sin, cos) وجميع هذه التوابع معرفة ضمن على أنها static، وبالتالي يمكن استدعاؤها دون تعريف object من الصف Math كالتالي:

```
double d = Math.sin(5);
```

نلاحظ أن دور الصف Math إذاً هو دور تجميعي (تجميع التوابع الرياضية ضمنه) لا أكثر.

ملاحظات:

١. كما أسلفنا: يمكن الوصول للحقول والتوابع الـ static من اسم الصف ومن اسم أي object من هذا الصف، ولكن العرف السائد أن يتم الوصول إليها من اسم الـ class حصراً.

٢. لا يمكن تعريف متحولات من النوع static ضمن تابع ما، والمكان الوحيد المسموح فيه استخدام الـ static هو بين حقول الـ class.

٣. يمكن إعطاء قيمة ابتدائية للحقل الـ static عند تعريفه مباشرة، ولكن هذه القيمة تعطى له عند ترجمة الـ class وليس عند إنشاء object، وبالتالي فهو يأخذها مرة واحدة فقط.

بنية برنامج Java:

يقسم برنامج Java إلى عدة packages تتواصل فيما بينها، تتكون هذه الـ packages من ملف واحد أو عدة ملفات، هذه الملفات تحوي class أو أكثر..

سنوضح هذه البنية من الأسفل إلى الأعلى:

ذكرنا مسبقاً أن برنامج Java يتكون من مجموعة من الـ classes، وقد فرغنا للتو من شرح بنية الـ class.

تكتب هذه الـ classes في ملفات نصية عادية ولكنها ذات اللاحقة (java.) بحيث يحوي كل ملف class واحد أو أكثر، ولكن بشرط أن يكون أحدها فقط من النوع (public).

العرف السائد بين مبرمجي Java يقتضي ألا يحوي الملف على أكثر من class واحد، ولكن قد يضطر المبرمج إلى استخدام صفوف صغيرة مساعدة لعمل الصف الأساسي في الملف *بالطبع هو فقط من النوع public وما تبقى من الصفوف فجميعها من النوع unpublic* عندها يمكن وضعها معه في نفس الملف.

ملاحظات هامة:

- إن ترجمة الملفات ذات اللاحقة java. عن طريق الـ compiler ينتج عنها ملفات ذات اللاحقة class. تحوي byte code يفهمه الـ JVM، وبالتالي ينتج لدي برنامج تنفيذي، ولكن الـ compiler يفصل بين الصفوف ولا يتركها في نفس الملف وإنما يفرد ملف (class.) لكل صف على حدة.
- يجب أن يكون اسم الملف مطابقاً تماماً لاسم الـ class الأساسي الـ (public) فيه.

يشكل كل ملف أو مجموعة ملفات ما يشبه المكتبة وتدعى package، وهي تمثل على الحاسب بمجلد يحوي مجموعة من الملفات ذات اللاحقة (java.)، وعادة ما يسبق اسم الـ package بسابقة تدل على مصدر هذه

المكتبات وذلك تمييزاً بينها عند تشابه أسمائها وعادة ما تكون هذه السابقة هي نفس (domain) الموقع الـذي يحوي هذه المكتبة.. مثال: borland.student , sun.student

نستنتج مما سبق أن Java source code يكتب في ملف (.java)، لذا لنبدأ بدراسة بنية هذا الملف عن طريق مثال، وليكن نفس المثال السابق (الصف Student):

```
package myPackage;
import package2.book;
import package3.*;

public class Student {
      private String name;
      private int age;
      .
      .
      .
}

class ss {
      .
      .
      .
}
```

نلاحظ أن أول سطر في الملف نكتب فيه اسم الـ package التي ينتمي لها هذا الملف.

بعد ذلك نضمن صفوفاً من packages أخرى *طبعاً لابد أن تكون public* عن طريق تعليمة import وذلك لكي نستطيع أن نعرف منها objects، والفرق بين السطرين الثاني والثالث في المثال أعلاه هو أننا في السطر الثاني ضمنّا class محدد من جميع الـ classes التي تحويها package2، أما في السطر الثالث فإننا ضمنّا جميع الصفوف الـ public في package3.

بعدها يمكننا أن نكتب صفوفنا بشرط أن يكون أحدها فقط public وأن يكون اسمه مطابقاً لاسم الملف، أي أن اسم ملفنا الذي كتبنا فيه ما سبق هو: Student.java

هناك ملاحظة أخيرة: لن نستطيع استخدام هذا الملف في أي برنامج ما لم نضعه في مجلد اسمه (myPackage) وذلك لأن الملف ينتمي للـ package التي اسمها myPackage، وذكرنا سابقاً أن الـ package تمثل على الحاسب بمجلد..

بعد أن تعلمنا كيف ننشئ ملفاً يحوي كود Java وأصبحنا نستطيع أن نقسم برنامجنا إلى عدد من الملفات موزعة في عدة packages تتواصل فيما بينها، آن لنا أن نتساءل: من أين سيبدأ البرنامج تنفيذه؟

في C++ كنا نكتب تابع اسمه main يبدأ منه تنفيذ البرنامج، ولكننا لا نستطيع أن نكتب أي تابع خارج class في Java فما الحل؟؟

الحل يكمن في الاستفادة من مفهوم الـ static حيث أن بإمكاننا تعريف تابع من النوع static ضمن أي class واستدعاء هذا التابع بدون تعريف object من هذا الـ class..

فإذاً: التابع main في Java عبارة عن method في أي class، وله الشكل التالي:

```
public static void main(String[] args){

}
```

فهو public لكي نستطيع استخدامه مباشرة، و static كي يستدعى بدون تعريف object من الـــ class، ولا يرد أي قيمة فخرجه من النوع void واسمه main ويأخذ مصفوفة String كمتحول دخل، هذه المصفوفة تحوي ما يعرف بـــ (command line)، وهو يُدخل عند تشغيل الـــ البرنامج كما سنرى لاحقاً.
لا يمكن أن نجد تابعين main في نفس الملف لأنه يجب أن يوضع ضمن class من النوع public، ونحـــن نعلم أن كل ملف يحوي public class واحد فقط، وبالتالي فإننا نستطيع لو شئنا أن نضع تابع main في كل ملف من ملفات البرنامج، ولكن التابع الذي سيبدأ منه التنفيذ يجب أن يكون واحد فقط، وهو التابع الموجود فـــي الملف الذي سيبدأ التنفيذ منه كما سنرى بعد قليل..

بعض أساسيات Java:

سنتكلم باختصار عن بعض الأساسيات المهمة في كتابة البرنامج والتي لا مجال للتفصيل فيها وذلك لأن معظمها قد شرح بوفرة في منهاج الـــ C++ في العام الفائت، ولمن يحب مراجعة معلوماته، فعليه بقراءة البحث الثالث من المرجع.

- عملية الإسناد: لدينا نوعان للإسناد في Java:

١. نسخ القيمة: ويكون بين المتحولات من النوع (primitive types)، وفيه يتم نسخ القيمة من الطـــرف اليميني ووضعها في المتحول في الطرف اليساري، وبالتالي يبقى كل متحول مستقل وقائم بذاته.
٢. إسناد مؤشرات: ويكون بين مؤشرين، بحيث يصبح المؤشر في الطرف الأيسر يشير إلى object الذي كان يشير إليه المؤشر في الطرف الأيمن أي أن المؤشرين أصبحا يشيران على نفس الـــ object وإذا كان المؤشر في الطرف الأيسر يشير مسبقاً إلى object فهذا يعني أنني فقـــدت هـــذا الـــــ object وسيقوم الـــ gc بهدمه.

- العمليات الحسابية (/,*,-,+): ولا يمكن استخدامها إلا مع المتحولات من النـــوع (primitive types)، والصف الوحيد الذي يستطيع استخدام بعضها هو الصف String، وهو صف له معاملة خاصة إذ يمكـــن استخدامه وكأنه primitive type حيث يمكن تعريف object منه بدون تعليمة new كما يمكن استخدام عملية الجمع معه:

```
String s1 = "Java";
String s2 = new String("class");
s1 = "Object" + s1 + s2;
```

- Casting: سنتكلم في هذه الفقرة عن المتحولات (primitive types):
تقوم Java بالتحويل تلقائياً من النمط الصغير إلى النمط الكبير. مثال: عند إجراء عملية جمع بين مجموعة من المتحولات من الأنواع (int, byte, short) فإن الناتج حتماً سيكون من النمط (int)، وبنفس الطريقة سيكون ناتج عملية جمع بين مجموعة من المتحولات من الأنواع (float, double) من النوع double.

- العمليات المنطقية:
  - المقارنة: إما بين قيم (primitive types)، أو بين مؤشرات (Objects) وبالتالي فإن مقارنة مؤشرين لا تقارن بين الـ objects وإنما بين المؤشرات، ولتقارن بين objects يجب أن تستخدم تابع (equals) بعد عمل (Overriding) له.

مثال:

```
String s1 = "Ammar", s2 = "Ammar";
if (s1 == s2)
      System.out.println("Equal");
else
      System.out.println("not Equal");
```

إن خرج البرنامج السابق: not equal لأن المقارنة تمت بين المؤشرين وبما أن كل منهما يشير إلى object مختلف فالمقارنة أعطت false.

  - يجب التفريق بين العمليات المنطقية والشروط المنطقية:
الشروط المنطقية تكون بين شروط بوليانية وقد تعاملنا معها كثيراً وهي: (==, =!, ||, &&) وهذه الشروط يطبق عليها ما يعرف بـ (short circuit) أي إذا كنا نختبر الشرط المنطقي التالي:

```
if ((bool1==true) && (bool2==true))
```

لنفترض أن قيمة bool1 كانت تساوي false فإن القسم الثاني من الشرط لن يختبر أبداً وسنعتبر أن نتيجة الشرط ككل = false.

أما العمليات المنطقية فهي عمليات تنفذ على الأنماط الطبيعية (integral types) بشكلها الثنائي أي أنها تعامل كبايتات، وهذه العمليات هي: (&, |, ^, ~).

هناك عمليات ثنائية مهمة جداً تدعى Shift operators تقوم بإزاحة البتات كالتالي:

<<: left shift وتقوم بإزاحة بتات المتحول إلى اليسار n بتاً، وتضع أصفاراً في البتات التي تفرغ من اليمين.

>>: signed right shift وتقوم بإزاحة بتات المتحول إلى اليمين n بتاً، وتضع في البتات التي تفرغ من اليسار أصفاراً إذا كان المتحول موجباً و واحدات إذا كان المتحول سالباً.

>>>: unsigned right shift وتقوم بإزاحة بتات المتحول إلى اليمين n بتاً، وتضع في البتات التي تفرغ من اليسار أصفاراً دوماً.

ملاحظات:

١. كل انزياح نحو اليسار يكافئ ضرب العدد بـــ ٢، وكل انزياح نحو اليمين يكافئ القسمة على ٢.
٢. إن Shift operators ترقي المتحولات (byte, short ..) إلى int.
٣. بالنسبة للـــ int فإن أكبر رقم يمكن أن أزيح وفقه هو (31) وذلك لأن عدد البتات التـــي يمثـــل عليها الـــ int هو ٣٢.

مثال:

```
int i = 16, x, y;
x = i << 1;                    // Now  x = 32
y = i >> 2;                    // Now  y = 4
```

- الشروط والحلقات في Java مطابقة تماماً للـــ C++ ولكن الفرق أن الشرط المنطقي الذي نختبره يجب أن يكون من النوع Boolean حصراً في Java بينما كنا نستطيع وضع int في C++.

مثال:

```
if (1)                         // legal in C++ not in Java
if (x == 5)                    // legal in C++ and Java
```

أعتذر عن التفصيل أكثر من هذا للأسباب التي ذكرتها سابقاً، لذا أنصح بالعودة للبحث الثالث من المرجع.

JDK:

سنتكلم في هذه الفقرة عن كيفية كتابة برنامج Java وتشغيله عن طريق الـــ JDK، وسنبدأ ببرنـــامج بـــسيط يحوي class واحد فقط ونعرف ضمنه main method.

١. افتح المفكرة (Notepad) واكتب الـــ class التالي:

```
public class MyClass {
      public static void main(String[] args) {
            System.out.println("This is My first program in Java..");
      }
}
```

٢. احفظ الملف باسم myclass.java وليكن ذلك على المسار التالي: c:\first.
٣. هذا الملف بحاجة لترجمة (compile) عن طريق برنامج جاهز في JDK يدعى javac.exe.
٤. ومن ثم يجب أن ينفذ (executable) عن طريق برنامج جاهز في JDK يدعى java.exe.
٥. سنتعامل مع البرنامجين السابقين عن طريق محرر DOS، علماً أن تشغيل أي برنامج تنفيذي فـــي بيئـــة DOS يقتضي أن نكتب مساره كاملاً، ولكن هناك طريقة لتشغيل البرنامج بكتابة اسمه فقط وذلـــك عنـــدما نضع مسار المجلد الذي يحويه في الـــ path.

٦. ما هو الـــ path؟ هو متحول يحوي عدة مسارات لبرامج يمكن تشغيلها في بيئة DOS بكتابة اسمها فقط دون كتابة مسارها، ويمكن الإضافة عليه يدوياً عن طريق الـــ windows، لذا وللتعامل مع البرنـــامجين السابقين الموجودين في مجلد الـــ bin داخل الـــ JDK يجب وضع مسار هذا المجلد في الـــ path.

٧. افتح محرر DOS وانتقل إلى المسار الذي وضعنا فيه الملف السابق وهو: (c:\first)

٨. شغل البرنامج javac ومرر له اسم الملف المطلوب ترجمته كما يلي:

```
c:\first> javac myclass.java
```

نلاحظ أن البرنامج أنشأ ملفاً جديداً في نفس مكان الملف myclass.java يحوي ناتج الترجمة واسمه:
Myclass.class

٩. يتم تنفيذ هذا الملف بتشغيل برنامج java وتمرير اسم الملف (myclass) ولكن مع الانتباه إلى عدم وضع اللاحقة (.class) كما يلي:

```
c:\first> java myclass
```

١٠. إذا كان لدينا أخطاء في الكود فستظهر، وإلا فسيظهر على الشاشة الخرج المتوقع من البرنامج وهو في حالة برنامجنا:

This is My first program in Java..

انتهت المحاضرة ..

lectures_team@hotmail.com

FIGURE 4.3 Widening conversions