بسم الله الرحمن الرحيم

﴿ إن الله وملائكته يصلون على النبي يا أيها الذين آمنوا صلوا عليه وسلموا تسليماً ﴾

صدق الله العظيم

# إحترف جافا

إعداد

المبرمجة : إسراء فيصل / جمهورية العراق

مراجعة الكتاب و النشر المهندس : أحمد سعيد / جمهورية مصر

# مقدمة

هذا ملخص لدراسة وإحتراف لغة جافا (J2SE) . . . وأنا أعتبرها إنطلاقة وبداية سريعة وموفقة في نفس الوقت . . . لإن وضحت فيها كل المواضيع التي تتركز عليها اللغة . . . تم طرح وشرح المواضيع بأسلوب مُبسط أولي . . .لأني راعيت أن يكون المتلقي أنه يتعلم لغة جافا لأول مرة ولا يمتلك أي خلفية عن اللغة . . . وتم ذكر الأمثلة . . .ووضحت في كل مثال . . . كل مصطلح من مصطلحات اللغة . . . مفهومه، أين أستخدمه ، متى أستخدمه، الفائدة من إستخدامه، تأثيره في حالة عدم إستخدامه . . . وجميع هذه الأمثلة تم عمل لها تنفيذ (run) في بيئة (jdeveloper) إن شاء الله لا توجد أخطاء . .

إن شاء الله . . . الكل يستفيد من هذا الكتاب . .والتوفيق للجميع . .وأعتذر عن أي تقصير . .

نسألكم الدعاء بالرحمة لي و والداي . . .

قال رسول الله صلى الله عليه وآله وصحبه وسلم

((إذا مات إبن آدم إنقطع عمله إلا من ثلاث :صدقة جارية . . . أو علم ينتفع به . . . أو ولد صالح يدعو له))

إسراء

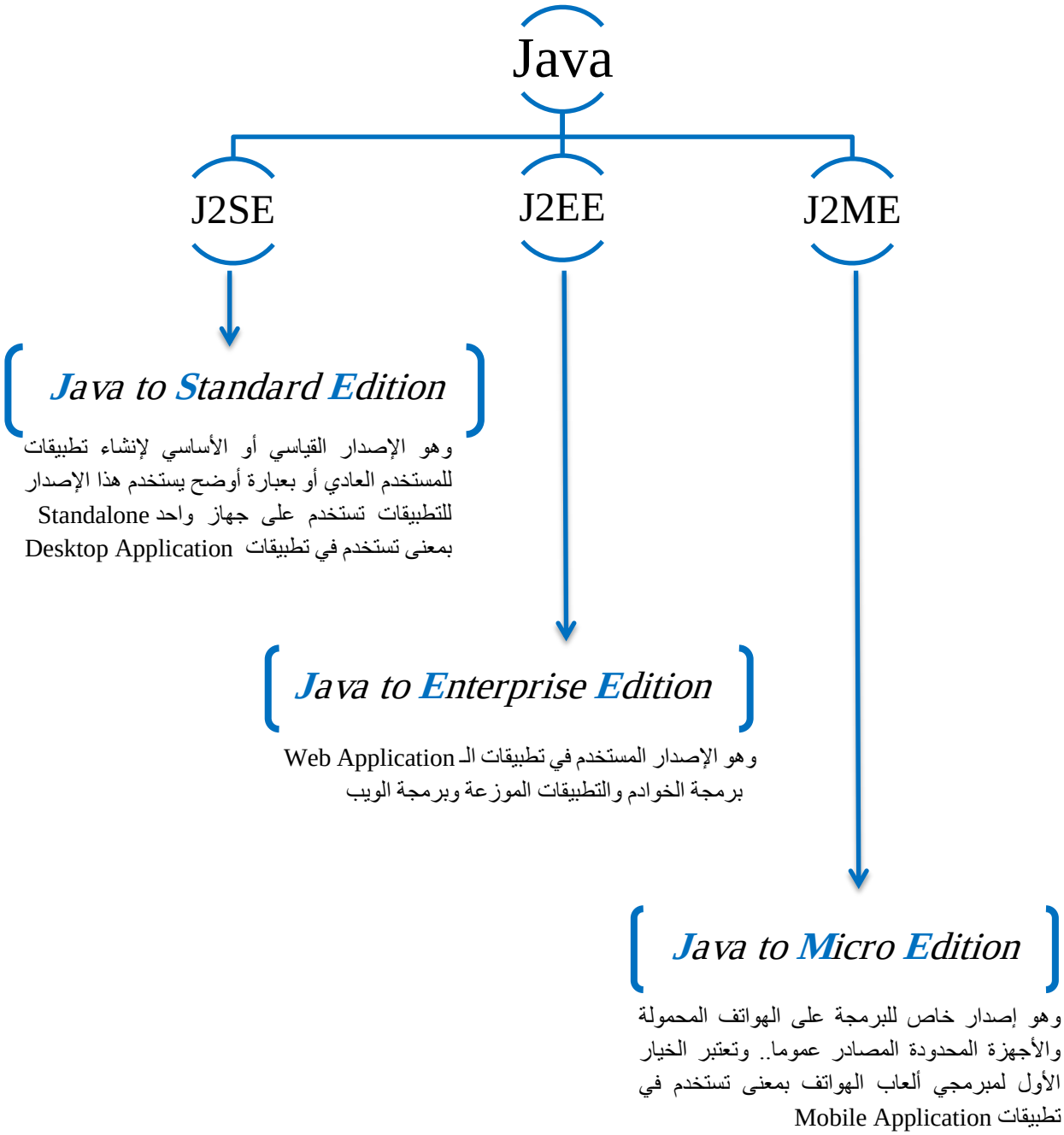
# المحتويات

إسم الموضوع | رقم الصفحة

١. مقدمة عن لغة جافا ..... (٤)
٢. المتغيرات (Variables) ..... (٧)
٣. أنواع البيانات (Data Types) ..... (٩)
٤. مفهوم الـ (Classes and Objects) ..... (١٣)
٥. مفهوم الـ (Methods) ..... (١٤)
٦. جمل التحكم (Control Statements) ..... (١٨)
٧. المصفوفات (Arrays) ..... (٢٤)
٨. مفهوم الـ (Constructor) ..... (٢٧)
٩. التوارث (Inheritance) ..... (٣٠)
١٠. مفهوم الـ (Polymorphism) شرح مفهوم الـ (Overload) و (Override) ..... (٣٢)
١١. مفهوم الـ (Abstract) و (Final) ..... (٣٦)
١٢. مفهوم الـ (Modifiers) ، (Static Modifiers) ..... (٤١)
١٣. مفهوم الـ (Encapsulation) ..... (٤٨)
١٤. مفهوم الـ (Interfaces) ..... (٥٠)
١٥. مفهوم الـ (Casting) ..... (٦٢)
١٦. مفهوم الـ (Exception)، (Try...Catch) ..... (٦٨)
١٧. Java Basic Operators ..... (٧٩)

# مقدمة عن لغة جافا . . .

إن لغة جافا غالباً ما يقال عنها إنها على ثلاثة أنواع أو أقسام .. وهذا الأمر فيه بعض من عدم الصحة فهي لاتنقسم لإنها ليست أنواع بل إصدارات ... حسناً إذن ...هنالك ثلاث إصدارات وهي كما موضحة في الشكل أدناه وتم بيان كل إصدار أين يتم إستخدامه:



## ... ملاحظة هامة جداً

كل لغات البرمجة عند عمل ( Compiler) لبرامجها... فإنه يتم تحويل الـ (code) المكتوب إلى مايعرف بـ (Binary code) الإ لغة جافا أتبعت إسلوب الـ (Byte Code) بسبب تزويد لغة جافا بما يعرف بتقنية (JVM) أي (**J**ava **V**irtual **M**achine) وهي بيئة عمل وهمية لتطبيقات الجافا وهو إسلوب راقي متبع أعطى الإمكانيات لتطبيقات الجافا أن تعمل تحت أي بيئة عمل سواء كانت هذه البيئة هي Desktop Application أو Web Application أو Mobile Application وتحت أي نظام تشغيل سواء كان Windows أو Linux أو....

**وهذه أهم وأقوى ميزة من المميزات التي تمتلكها لغة جافا وهي إستقلالية لغة جافا وعدم إعتمادها على أنظمة التشغيل...**

بالإضافة الى ذلك تتميز لغة الجافا بمميزات خاصة مما يجعلها أكثر لغات البرمجة قوة حيث تمكننا من الآتي:

١. إضافة الحركة والصوت إلى صفحات الويب.
٢. برمجة الألعاب والبرامج المساعدة .
٣. إنشاء برامج ذات واجهة مستخدم رسومية.
٤. تصميم برمجيات تستفيد من كل مميزات الأنترنت.
٥. توفر لغة الجافا بيئة تفاعلية عبر الشبكة العنكبوتية وبالتالي تستخدم لكتابة برامج تعليمية للإنترنت عبر برمجيات المحاكاة الحاسوبية للتجارب العلمية وبرمجيات الفصول الإفتراضية للتعليم الإلكتروني عن بعد.
٦. كل الإمكانيات التي ذكرت في أعلاه كانت نتيجة لإمتلاك لغة جافا مكتبات قوية والمتمثلة بـ (API) (**A**pplication **P**rogramming **I**nterface) والتي تمثل لنا مجموعة من الحزم البرمجية المكتوبة المتمثلة بـ (classes) و (interfaces) والتي تمكننا كما ذكرنا من بناء الواجهات الجميلة بشكل يسهل التفاعل مع المستخدم وجهاز الحاسوب، بالإضافة الى إمتلاكها معظم أو كل الفصائل المطلوبة مثل التعامل مع الملفات وقواعد البيانات والشبكات و الرسومات المجسمة والحركة وكذلك التعامل مع الإنترنت .

وبهذه المميزات أكسبت لغة جافا القوة مما جعلها تناسب مع التطبيقات الحديثة فهي تناسب تطبيقات الإنترنت حيث أصبحت هي قلب برمجة الإنترنت بما توفره من إمكانيات ... وكذلك مدى تناسبها في تطبيقات الهواتف المحمولة وبرمجتها.

## ... كيفية كتابة البرنامج في لغة جافا

البرنامج بلغة جافا يكتب بشكل (classes) وبداخل هذه الـ (classes) تكتب الـ (methods) ويتم أيضاً كتابة الـ (**main class**) لإستدعاء وتنفيذ هذه الـ (methods) التي تم كتابتها داخل (classes).

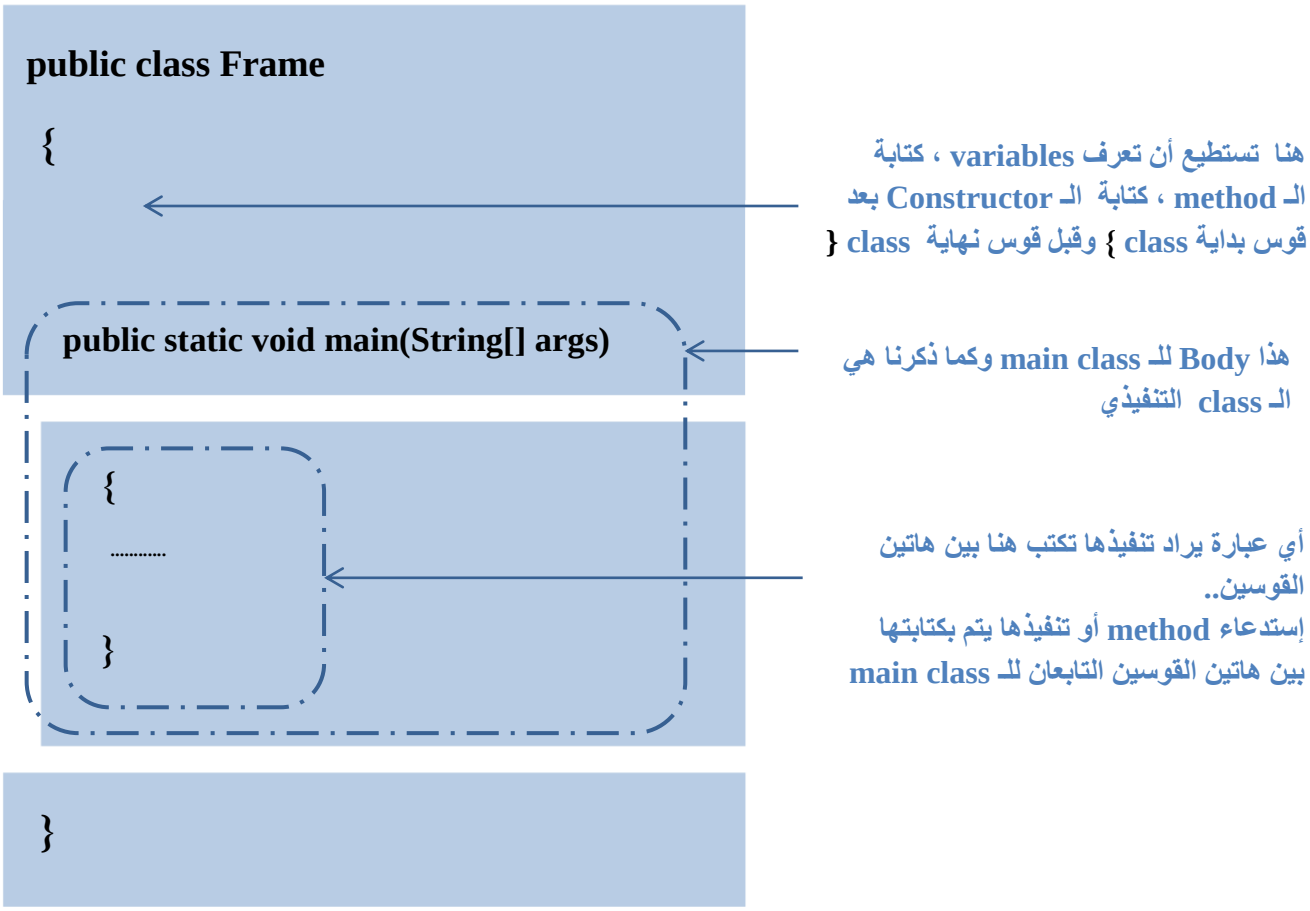
الـ (**main class**) هي الـ (class) الرئيسي أو التنفيذي لانها تحتوي على صيغة:

**public static void main(String[] args)**

وأي برنامج بلغة جافا حتى وأن لم يحتوي على أي (method) لكن **يجب** أن يحتوي على الأقل على (class) واحدة وبداخلها الـ (**main class**)..التي تذكر لمرة واحدة فقط لغرض تنفيذ البرنامج.

## ... الشكل التالي يوضح هيكلة البرنامج في لغة جافا

هنا إسم الـ class هو Frame وبداخله الـ (**main class**) وكما ذكرنا أنه الـ (class) التنفيذي لانه يحتوي على الصيغة **public static void main(String[] args)** وهي تمثل الـ **main class**




## ... الموضوع سيكون بمنتهى السهولة مع التقدم بالشرح و عند كتابة البرنامج بأيدينا نتعرف أكثر على هيلكة البرنامج في لغة الجافا

# Variables in Java

**المتغيرات في لغة جافا** ... في البداية لنتعرف على معنى المتغيرات في لغات البرمجة عامة ... المتغيرات هي (Storage Location) مكان خزن البيانات... بمعنى يتم حجز مكان في الذاكرة ونخصصه لنضع فيه بيانات التي سيتم إستخدامها في البرنامج ...أو ربما نخصص مكان حتى نضع فيه بيانات مسترجعة من تنفيذ دالة أو إجراء تمت كاتبته داخل البرنامج ...حسناً مفهوم المتغيرات (variables) وضح إن شاء الله هي مكان تم حجزه في الذكرة لخزن فيه بيانات أستخدمت في البرنامج.

البيانات التي تم وضعها في هذا المتغير تسمى ... قيمة المتغير (value of variable)... جيد.

هنالك بعض القواعد أو الشروط التي يتوجب على المستخدم عندما يحجز هذا المكان في الذاكرة والذي إتفقنا على أن مكان الخزن هذا يسمى متغير (variable)..أن يأخذ بنظر الإعتبار هذه الشروط ويتبعها وهي كالآتي:

١- أن يضع إسم لهذا الـ (variable) أي بمعنى إسم للمخزن وأن هذا الإسم لايتكرر مع إسم (variable) آخر داخل البرنامج.

٢- أن لا يكون هذا الإسم أحد الكلمات المحجوزة في لغة البرمجة بمعنى أن لا يكون من الكلمات (keywords) في لغة البرمجة ...تم ذكر جدول في نهاية الكتاب بهذه الكلمات المحجوزة (keywords).

٣- أن يوضح نوع البيانات التي سوف تخزن أو توضع داخل هذا المخزن (variables) .. أي بمعنى هذا المخزن الذي تم حجزه .. ماذا يضع فيه..؟ هل مخصص لقيمة رقمية صحيحة.. ؟ هل مخصص لخزن قيمة رقمية عشرية..؟ هل مخصص لقيمة نصية حرفية.. ؟ وهكذا ... أي يجب تحديد نوع البيانات المخزنة داخل هذا المتغير ... أنواع البيانات تسمى بلغات البرمجة بـ (data type)...

٤- أن يحدد كيفية الوصول إلى هذا المتغير، أي بمعنى يتم تحديد نوع الـ (Access Modifier) والـ (Access Modifier) المتمثلة بـ (public, private, static, default, final ...etc) وقد تم توضيح شامل لإنواع الـ (Access Modifier) ضمن فصل في هذا الكتاب.

... بصورة عامة يجب إتباع هذه الشروط أعلاه عند تعريف المتغيرات.. إذن عند تعريف المتغير يتطلب تحديد الـ access modifier يليه data type يليه variable name ثم ; semicolon وسيتضح أكثر مع المثال التالي:

...الصيغة العامة لتعريف المتغيرات في لغة الجافا تكون كالتالي...

access modifire data type variable name ; ← **نكتب نوع الـ access modifire ثم الـ data type يليه إسم المتغير ثم semicolon ;**

static int num1 ;

**الـ (access modifire) ذكره إختياري بمعنى نستطيع أن لا نكتبه ..وغالباً ما يكتب لتغيير أو لتحديد سلوك المتغير داخل البرنامج**

**نستطيع إسناد قيمة للمتغير بنفس وقت تعريفه كما موضح** ←

```
int num1 = 24;
```

**نستطيع تعريف أكثر من متغير بنفس السطر وإسناد قيمة للمتغير بنفس وقت التعريف على أن يفصل بين متغير والآخر بفارزة (فاصلة) كما موضح..** ←

```
int num1 =24, num2=23;
```

**شرط على أن يكون كلا المتغيرين لهما نفس Data type بمعنى أنه كلاهما int أو كلاهما char وهكذا**

... تم توضيح كيفية تعريف المتغيرات مع كل نوع من أنواع البيانات (data types) بشكل تفصيلي أكثر مع مثال:

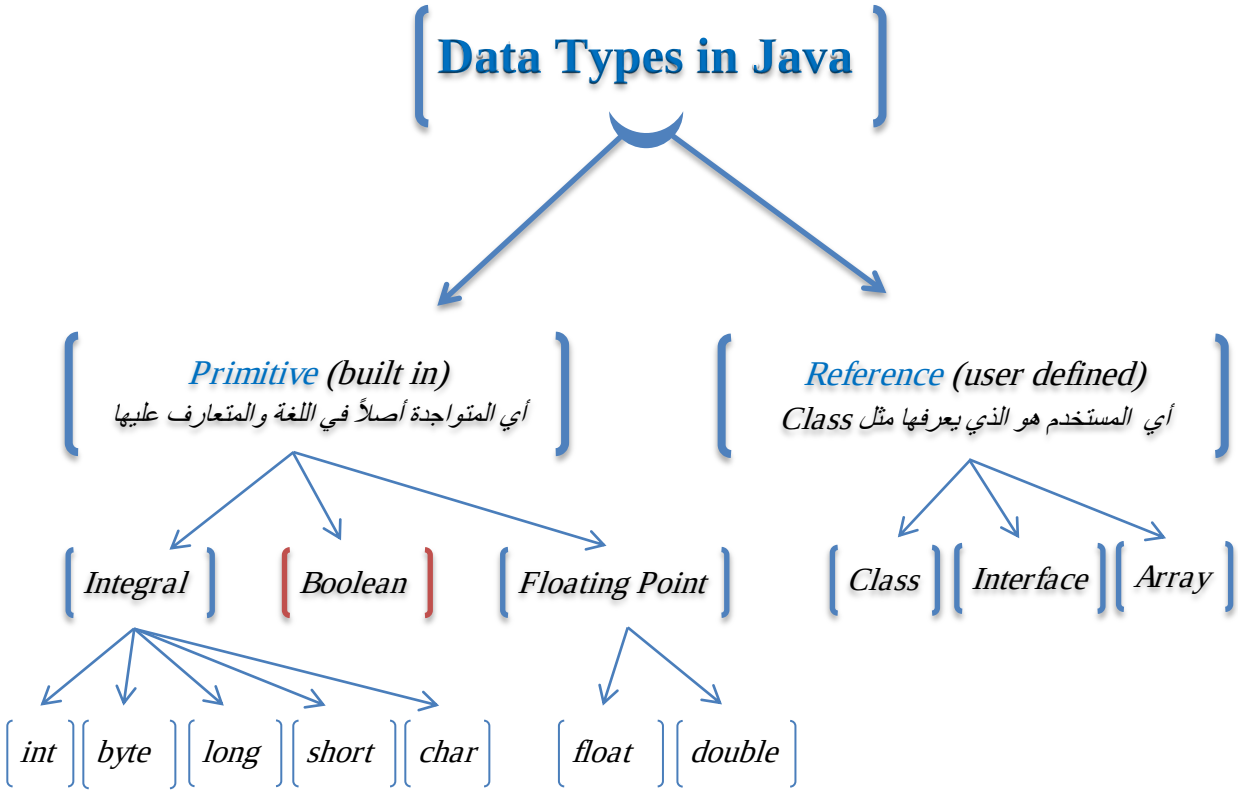
# Data Types in Java

البيانات المستخدمة في لغة جافا تنقسم إلى نوعين ... وهي:

١. (***Primitive***) أو ما يعرف بـ (***built in***).

٢. (***Reference***) أو ما يعرف بـ (***user defined***).

... وأدناه توضيح كل نوع منها بإسلوب مُبسط كما في الشكل:



... نوضح الأن primitive data type كل نوع منها بشيء من التفصيل:

## ١- Integral

### ١.١ byte

يأخذ (1 byte) أي يساوي (8 bits) ... المدى المخصص له هو : (128 ← 128-) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (byte) هي (127) وأقل قيمة يمكن إسنادها هي (128-) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد. إذن عند تعريف متغير وأُسندت له قيمة (129-) أو أي قيمة ما بعد (128-) أو أُسندت له قيمة أكبر من (127) على الفور سوف يظهر Compiler Error لأنه تم تعيين قيمة خارج نطاق المدى المحدد.

**علماً إن مراعاة الحد الأدنى والحد الأعلى عند إسناد القيم لمتغير ينطبق على كافة الأنواع بالتأكيد...**

### ١.٢ short

يأخذ (2 byte) أي يساوي (16 bits) ... المدى المخصص له هو: (32,767 ← 32,768-) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (short) هي (32,767) وأقل قيمة يمكن إسنادها هي (32,768-) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد.

### ١.٣ int

يأخذ (4 byte) أي يساوي (32 bits) ... المدى المخصص له هو:
(2,147,483,647 ← 2,147,483,648-) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (int) هي (2,147,483,647) وأقل قيمة يمكن إسنادها هي (2,147,483,648-) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد.

### ١.٤ long

يأخذ (8 byte) أي يساوي (64 bits) ... المدى المخصص له هو:
(9,223,372,036,854,775,807 ← 9,223,372,036,854,775,808-) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (long) هي (9,223,372,036,854,775,807) وأقل قيمة يمكن إسنادها هي (9,223,372,036,854,775,808-) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد.

### ١.٥ char

يأخذ (2 byte) أي يساوي (16 bits) أعلى قيمة يمكن إسنادها لمتغير من نوع (char) هي (65,535) وأقل قيمة يمكن إسنادها هي (0) بمعنى أستطيع أن أسند (65,535) حرف لمتغير.

(char) في لغة الجافا يمثل بـ (unicode)

## ٢- Floating point

### ٢.١ double

يأخذ (8 byte) أي يساوي (64 bits)... المدى المخصص له هو:

(±4.94065645841246544E-324 → ±1.79769313486231570E+308) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (double) هي (±1.79769313486231570E+308) وأقل قيمة يمكن إسنادها هي (±4.94065645841246544E-324) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد.

### ٢.٢ float

يأخذ (4 byte) أي يساوي (32 bits) المدى المخصص له هو:

(±1. 40129846432481707E-45 → ±3.40282346638528860E+38) ... ويقصد بذلك أعلى قيمة يمكن إسنادها لمتغير من نوع (float) هي (±3.40282346638528860E+38) وأقل قيمة يمكن إسنادها هي (±1. 40129846432481707E-45) بمعنى عند تعريف متغير يتوجب علينا إسناد قيمة لهذا المتغير تتراوح بين هذا المدى المحدد.

## ٣- boolean

يأخذ (1 bit) ويتم إسناد قيمة له أما (true) أو (false).

... الخلاصة كما في الجدول التالي:

| Category | Types | Size (bits) | Minimum Value | Maximum Value | Example |
|---|---|---|---|---|---|
| Integer | byte | 8 | -128 | 127 | byte b = 65; |
| | char | 16 | 0 | $2^{16}$-1 (65,535) | char c = 'A'; char c = 65; |
| | short | 16 | $-2^{15}$ (-32,768) | $2^{15}$-1 (32,767) | short s = 65; |
| | int | 32 | $-2^{31}$ (-2,147,483,648) | $2^{31}$-1 (2,147,483,647) | int i = 65; |
| | long | 64 | $-2^{63}$ (-9,223,372,036,854,775,808) | $2^{63}$-1 (9,223,372,036,854,775,807) | long l = 65L; |
| Floating-point | float | 32 | $2^{-149}$ (±1. 40129846432481707E-45) | $(2-2^{-23})\cdot 2^{127}$ ±3.40282346638528860E+38 | float f = 65f; |
| | double | 64 | $2^{-1074}$ (±4.94065645841246544E-324) | $(2-2^{-52})\cdot 2^{1023}$ (±1.79769313486231570E+308) | double d = 65.55; |
| Other | boolean | 1 | -- | -- | boolean b = true; |

... المثال التالي يوضح الكيفية.. لتعريف المتغيرات في لغة الجافا ...

```
public class Class1
 {
  public static void main(String[] args)
   {
    byte age;
    age = 33;

  System.out.println(age);

    double d = 2.7;

    float f1 = 1.5;

    float f1 = 1.5f;
    float f2 = (float)1.5;

   System.out.println((int)(99.9999));
     // Returns 99

    int n1 = 10;
    int n2 = 5;
    int n3 = 56, n4 = 24;

    double div =n1/(double) n2;
    System.out.println(div);

    boolean b1 = true; //or put false

    char c1 = 'a';

    char c2 = 105;

    System.out.println(c1);
    System.out.println(c2);

   int w = 'z';
   System.out.println(w);
  }
 }
```

هيكلية البرنامج في لغة الجافا

main class

تعريف متغير إسمه age من نوع byte

إسناد قيمة للمتغير age 33

تعريف متغير إسمه d من نوع double في الوقت نفسه نستطيع إسناد قيمة هي 2.7

تعريف متغير أسمه f1 من نوع float وفي الحقيقة أي متغير يكون بشكل كسرعشري في الجافا لايعرف على أنه نوع **float** بل يعرف على أنه **double** على الرغم من اننا ذكرنا أثناء التعريف على أنه float **إذن كيف لنا أن نعرف متغير نوع float ..؟**

**ليتم تعريف متغير نوع float يكون بطريقتين هما:**

١- أما بكتابة حرف f جنب الرقم العشري

٢- أو بكتابة بين قوسين كلمة (float) جنب الرقم العشري حتى يفهم على أن تعريفه من نوع float ولا يعرف على أنه double والطريقة الثانية هي الأصح وهذا مايعرف بـ **casting** وهو تغيير data type لمتغير

**Casting تغيير من data type الى آخر هنا سوف يطبع 99**

تعريف أكثر من متغير على نفس السطر وإسناد لهم قيم على أن يفصل بين متغير وأخر بفاصلة ويجب أن يكونا نفس النوع بمعنى نفس data type نقصد كلاهما نوع byte أو كلاهما نوع int وهكذا...

تعريف متغير إسمه b1 من نوع boolean وإسناد قيمة true وهو أما يأخذ true أو false

تعريف متغير إسمه c1 من نوع char وإسناد قيمة ' a ' عند إسناد قيمة من نوع char يجب أن توضع بين ' ' single quotation دلالة على أنه قيمة حرفية

تم تعريف متغير إسمه c2 من نوع char وإسناد قيمة رقمية 105 كيف هذا..؟ نعم يمكن هذا فهو سوف يرجع القيمة الحرفية لهذا الرقم والتي تمثل i لانه عرف من نوع char إذن يرجع قيمة **char فهو يعتمد على الـ data type التي عُرف بها**

تعريف متغير إسمه w من نوع int وإسناد قيمة حرفية 'z' كيف هذا..؟ نعم يمكن هذا فهو سوف يرجع القيمة الرقمية لهذا الحرف والتي تمثل 122 ..لانه عرف من نوع int إذن يرجع قيمة **int فهو يعتمد على الـ data type التي عُرف بها**

# Classes and objects

إن مبدأ عمل لغة جافا هو الـ (Class) ، (object) أو مايعرف بـ
(Object Oriented Programming O.O.P)

إن فكرة الـ (Object Oriented Programming) هي لوصف كائن ... أي كائن في حياتنا اليومية نستطيع وصفه ... هذا الكائن يعرف أو يسمى بـ (class) ... لحد الآن جيد ..إذن الـ (Class) هي وصف لكائن حسناً ... كل كائن أكيد يمتلك صفات أو خصائص ويمتلك وظائف أيضاً ... جيد.

إذن لنوضح الكلام أعلاه برمجياً كيف يفسر ..؟ ماهو الـ (Class) ، وماهو الـ (object)... برمجياً..؟

الـ **(Class)** : هي **وصف لكائن** والكائن يمتلك صفات أو خصائص (attributes)..

بالبرمجة **الصفات أو الخصائص** تسمى بـ (variables) بمعنى المتغيرات ... والكائن هذا يمتلك وظائف ...

بالبرمجة **الوظائف** تعرف بـ (methods) بمعنى الدوال أو الإجراءات.

مما ذكر أعلاه نستلخص التالي:

الـ **(Class)** : عبارة عن حقيبة أكواد برمجية تضم بداخلها المتغيرات (variables) ، الدوال (method)، والمشيدات (constructors) والتي سوف نوضحها لاحقاً ... ولنأخذ المثال البسيط التالي حتى يتضح مفهوم الـ (Class) أكثر...

لنأخذ وصف لكائن من حياتنا اليومية ويمثل لنا (Class) وهو الإنسان (Human)...

الأنسان : يمتلك مجموعة من الصفات على سبيل المثال نذكر منها:

(الطول، الإسم، تاريخ الميلاد، الوظيفة، الوزن...الخ) هذه الصفات كما إتفقنا تسمى متغيرات (variables).

وفي نفس الوقت هذا الإنسان يمتلك وظائف يقوم بها... نذكر منها:

(المشي، التكلم، الإكل، التفكير، العمل....إلخ) وهذه الوظائف تعرف بـ (methods) أو الدوال.

حسناً ... تبقى أن نفهم ماهو الـ (object)...

الـ **(object)**: هو متغير من نوع (Class) ...

كيف هذا..؟ لم أفهم..!! حسناً .. ببساطة وعلى سبيل المثال عندما نعرف متغير (n) من نوع ( int) إذن هذا المتغير (n) سوف يكون نوعه (int) نفس الطريقة والإسلوب المتبع عندما نعرف متغير من نوع (Class) فهذا هنا لايسمى متغير بل يسمى بـ (object) وكأنما أخذنا نسخة من الـ (Class) لنتمكن من التعامل مع الـ (methods) داخل البرنامج من خلال هذا الـ (object) الذي عرفناه من نوع (Class) ... كما في مثالنا الذي ذكرناه أعلاه والذي أعتبرنا أن (Human) هو (class) إذن (سارة) سوف تمثل لي (object) بمعنى نوع من الـ (Class) الذي أسمه (Human) ... وكإننا نقول (سارة) هي أحد أنواع البشر وسوف يتضح هذا كما ذكرت لاحقاً ... **إذن (object) .... هو متغير من نوع (Class) ...** إن شاء الله الفكرة وضحت ... فيما يخص الـ (Class) والـ (object).

# Method

**Method:** هي ببساطة دالة (Function) أو إجراء (Procedure) في لغات البرمجة الأخرى بينما **في لغة جافا تعرف بإسم (Method)**..

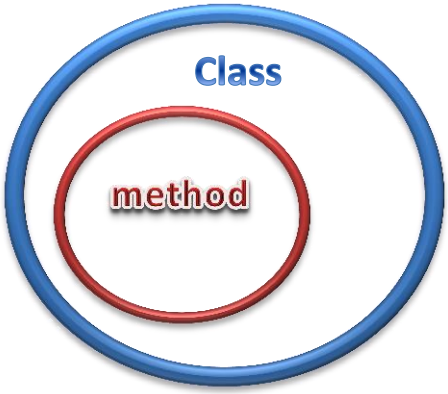
إذن مبدأ العمل تقريباً نفسه .. لكن هنالك إختلاف بالمسميات أو بالأحرى بالتسميات...

حسناً... الدالة في لغة الجافا هي إسمها (method) وذكرنا إنها نفس مبدأ العمل لكن تختلف تقريباً... أين تختلف..؟

تختلف أنه لاتكتب الـ (method) إينما كان في البرنامج ولكنها تكتب داخل الـ (Class) ..لان هو مبدأ عمل الجافا الـ (Class , object) إذن فالـ (method) تكتب داخل هذا الـ (Class).

إذن الإختلاف الثاني ماهو..؟ حسناً .. بما إنها تكتب داخل (Class) إذن طريقه إستدعائها أيضاً تختلف... بمعنى لانستطيع إستدعائها بشكل مباشر لإنها لم تكتب بشكل مباشر هي كتبت داخل (Class) إذن حتى يتم إستدعائها يجب أن أدخل داخل هذه الـ (Class) المكتوبة فيها الـ (method) ثم أستدعيها ... فقط نفهم هذه الآلية سوف يكون كل شيء واضح بإذن الله تعالى ...

بمعنى آخر أنا لدي حيز وفي داخله صندوق إذن حتى أحصل على الصندوق يجب أن أدخل أولاً إلى هذا الحيز حتى أخذ الصندوق ..إن شاء الله الفكرة تكون وضحت وكما في الشكل التالي : التوضيح هنا بشكل عام وسوف توضح التفاصيل لاحقاً المهم نفهم الفكرة كيف يكون الإستدعاء...




كما في الشكل التالي لدي Class ومكتوب بداخلها method
فحتى أستدعي هذه الـ method يتوجب أولاً أن أدخل داخل هذه Class
ثم أستدعي الـ method المكتوبة داخلها ..كيف أفعل هذا برمجياً ..؟
الجواب : في البرمجة حتى أدخل الـ Class يجب أن أعرف متغير من نوع Class ثم أنشأ نسخة منه وهذا ما تكلمنا عنه إسمه object ... كالتالي:

| | |
|---|---|
| Class cl1 ; ← | عرفنا متغير cl1 من نوع Class مثل أي متغير آخر وكإننا نقول int x1; |
| cl1=new Class(); ← | أنشأنا نسخة من Class للمتغير الذي عرفناه للتو |
| cl1.meth1(); ← | نستدعي الـ method الآن ..لأن وصلنا لها |

... هنا الـ cl1 يمثل لنا object ... وهو الذي سوف نتعامل مع (methods) المكتوبة داخل الـ (class) من خلاله في البرنامج إن شاء الله وضحت الفكرة ...

## ... الصيغة العامة لتعريف الـ (method) في لغة الجافا ...

**الصيغة العامة لتعريف الـ (method ) تتألف من جزئين وهما : method header وmethod body**

**الـ (method header) يتالف من:**

- **Access Modifier:** والمتمثلة بـأحد أنواع للـ (Modifier) وهي:

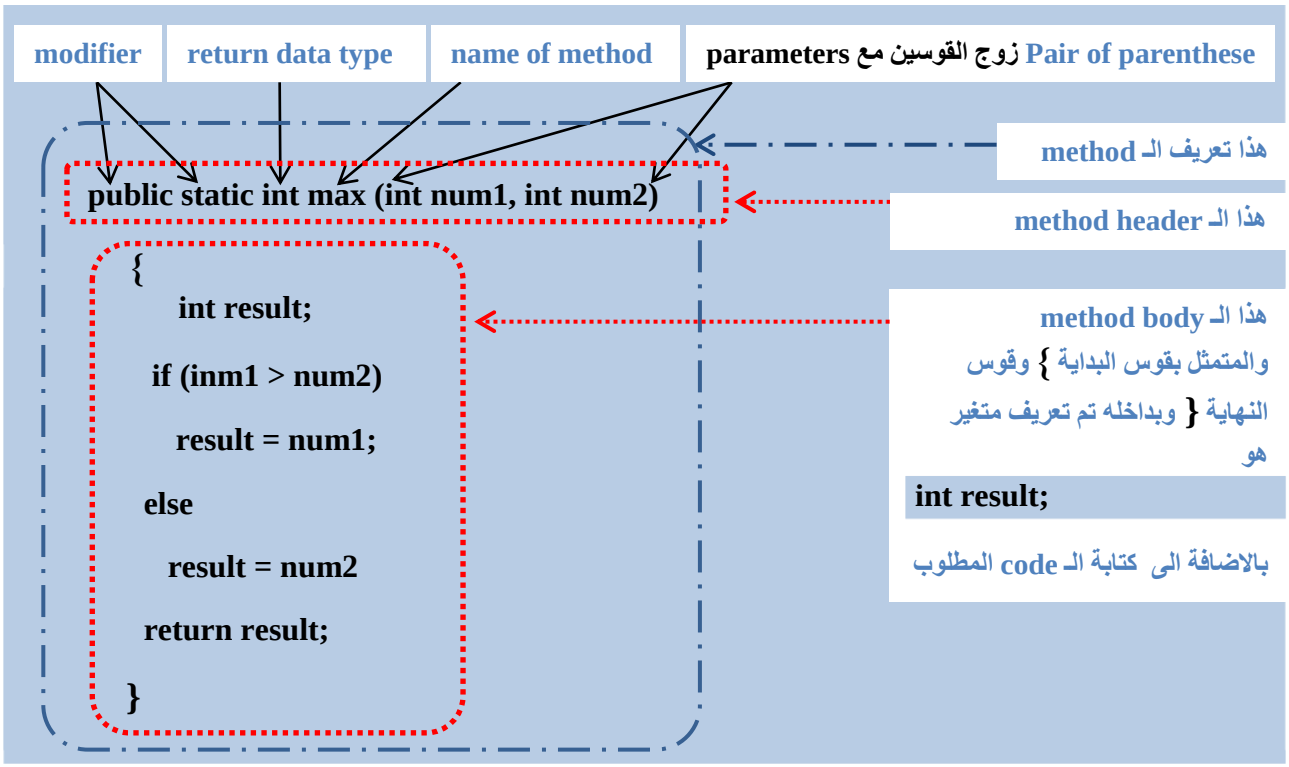
(public, private, local, default…etc.) وتم توضيحها لاحقاً بشكل أكثر تفصيلاً ... وهي إختارية بمعنى نحن مخيرين بذكرها... وغالباً ما تذكر لتغير أو تحديد سلوك الـ (method) داخل البرنامج بمعنى أن الـ (Access Modifier) تصف لنا كيفية الوصول للـ (method).

- **Return Type :** نوع البيانات المسترجعة من هذه الـ (method)... وهي أما **(void)** أو أحد أنواع الـ **(data type)** وقد سبق وتم توضيح أنواع البيانات (data types).
- **Method Name :** إسم الـ (method) ويفضل أن نختار إسم ذات معنى ودلالة تتعلق بعمل هذه الـ (method).
- **Pair of parenthese ():** زوج القوسين والتي غالباً ما يذكر بينهما الـ (Parameters) أو لا يذكر بينهما أي (Parameters) لإن بعض من الـ (method) لاتحتوي على أي (parameter) وهذا يعتمد حسب الوظيفة المطلوب أن تؤديها الـ (method).

**أما :**

**الـ (method body)** يتالف من قوس البداية **{** وقوس النهاية **}** وبداخلهما يتم تعريف المتغيرات الخاصة بالـ (method) (إن لزم الامر) ويتم كتابة (code) الإيعازات البرمجية **(الوظيفة التي ستؤديها الـ method).**

... لنوضح ماتم ذكره في أعلاه مع مثال لصيغة تعريف الـ (**method**) ...




... الـ (method) تأخذ أحد أنواع الـ (Access modifier) لنتعرف عليها وعلى مدى تأثيرها على سلوك الـ (method) داخل البرنامج (تم توضيح الـ Access modifier لاحقاً من الكتاب)

**Method**

**public**

الـ Method المكتوبة داخل هذه الـ Class نستطيع إستدعائها وإستخدامها (الوصول لها) في الـ classes الأخرى

**private**

الـ Method المكتوبة داخل هذه الـ Class لا نستطيع إستدعائها وإستخدامها (لا يتم الوصول) إليها في الـ classes الأخرى

**local**

**default**

الـ Method لايكتب في بدايتها لا كلمة public ولا كلمة private وتفهم على إنها public..وغالباً تستخدم default لتعريف المتغيرات

... توضيح أكثر ... الـ Method إذا كانت (public, private, local, default) فهي تأتي بصيغتين أما (void) ... أو تأخذ أحد أنواع الـ (data type) ... كما بينا في أعلاه وهنا نوضح الفرق بينهما :

**تاخذ أحد أنواع الـ data type كإن يكون (int,double,String,....)**

الـ Method هنا تشبه الـ function... في لغات البرمجة الأخرى أي أنه يسترجع قيمة ... حسب الـ Data type المعرفة به ... أو بعبارة أخرى **يحتوي** على كلمة return

```
int meth1(int num1,int num2)
{
return(num1+num2);
}
```

Data type

**void**

الـ Method هنا تشبه الـ procedure .. في لغات البرمجة الاخرى أي أنه لا يسترجع قيمة...أو بعبارة أخرى **لا** يحتوي على كلمة return

```
void meth1()
{
System.out.println("Hello world");
}
```

كلمة void

إن شاء الله تتضح أكثر عند تطبيقها وإستدعائها داخل البرنامج كما في المثال التالي

```
public class Class1
{
  public  int meth1 ()
   {
     return(num1+num2);
   }
public static void main(String[] args)
      {
      Class cls ;
      cls = new Class ();
      int result ;
      result = cls.meth1();
      }
}
```

هذه هي الـ Method

نعرف object من الـ (Class) التي كتبت بداخلها الـ (method)

أنشأنا نسخة من الـ Class

نعرف متغيرمن نفس نوع الـ data type للـ method لوضع القيمة المسترجعة لنا من الـ method .. هنا بحاجة لتعريف هذا المتغير لإن method التي لدينا من نوع الذي يسترجع قيمة (return type)

ثم نستدعي الـ method كالاتي :

```
result = cls.meth1();
```

إن شاء الله تتضح أكثر عند تطبيقها وإستدعائها داخل البرنامج كما في المثال التالي

```
public class Class1
{
  public void meth1 ()
   {
     System.out.print("Hello world ");
   }
public static void main(String[] args)
  {
    Class cls;
    cls = new Class ();
    cls.meth1();
  }
}
```

هذه هي الـ Method

نعرف object من الـ (Class) التي كتبت بداخلها الـ (method)

أنشأنا نسخة من الـ Class

ثم نستدعي الـ method كالاتي :

```
cls.meth1();
```

# Control Statements

هي الجمل المتعارف عليها بكل لغات البرمجة جمل التحكم في البرنامج ... بمعنى الجمل التي من خلالها نستطيع التحكم والسيطرة على سير البرنامج ونحدد سلوكه... و تشمل :-

1- if or if else statement
2- for loop
3- while
4- do…while
5- continue
6- break
7- switch…case

... الآن يتم توضيح الصيغة العامة (syntax) لكل منها مع الأمثلة بشكل تفصيلي:

١- الصيغة العامة للـ if or if else statement

```
if(boolean-expression)
 {
  if-code;
 }
  else
  {
   else-code;
  }

rest-of-code
```

If ثم نكتب بين قوسين الشرط المنطقي الذي نريد التحقق منه أو العبارة المطلوب السؤال عنها

إذا كان الشرط صحيح ستنفذ الجملة التي تكون مكتوبة بين قوسين البداية والنهاية بمعنى بين { و}

أما إذا لم يتحقق الشرط فسوف تنفذ جملة else ولا ننسى مكتوبة بين قوسين البداية والنهاية أيضاً بمعنى بين { و}

**Example:**

```
public class Class1
 {
 public static void main(String[] args)
  {
    int degree = 100;

      if (degree<= 50)
       {
        System.out.println("this is bad degree");
       }
        else
         {
          System.out.println("this is good degree");
         }
    }
 }
```

المثال التالي يوضح أكثر ...حيث أخذنا متغير من نوع int degree ويتم التحقق منه... إذا كان أقل من 50 سوف يطبع لنا إنها درجة سيئة وإذا لا ... إذن سوف ينتقل الى جملة else ليطبع إنها درجة جيدة

## ٢- الصيغة العامة للـ for loop statement

for (البداية ; الشرط المنطقي الذي نتوقف عنده الدورة ; الزيادة أو النقصان حسب المطلوب )

```
for (initialization ; condition ; increment or decrement)
  {
    <statement>;
      ...;
      ...;
  }
```

الجملة التي ستنفذ تكون مكتوبة بين قوسين البداية والنهاية بمعنى بين { و}

**Example:**

```
public class Class1
 {
  public static void main(String[] args)
   {
    for (int i = 3; i< 20; i+= 3)
     {
     System.out.println("this is i Value" +" \t" +i);
     }

   }
 }
```

for (البداية ; الشرط المنطقي الذي نتوقف عنده الدورة ; الزيادة أو النقصان )

```
for (int i = 3;   i< 20;     i+= 3)
```

هنا البداية تبدأ من 3 والشرط الذي سنتوقف عنده هو عندما العداد يكون أقل من 20 ومقدار الزيادة كل مرة يزيد بمقدار 3 **الزيادة تكتب بشكل i+=3 وهي تكافئ i= +3** بمعنى أي منهما تكتب فهي صحيح

## ٣- الصيغة العامة للـ while statement

while ( الشرط المنطقي الذي نتوقف عنده)

```
while(Boolean_Expression)
  {
    <statement>;
  }
```

الجملة أو الجمل التي ستنفذ إذا تحقق الشرط تكون مكتوبة بين قوسين البداية والنهاية أي بمعنى بين { و}

**Example:**

```
public static void main(String args[])
  {
   int count = 10;

   while( count < 20 )
    {
    System.out.println(" value of count : " + count );

    count++;
    }
  }
```

عرفنا متغير إسمه count من نوع int count وأعطايناه قيمة أولية = 10 وداخل الـ (while) يتحقق كل مرة من قيمة هذه المتغير هل مازالت أقل من 20 إذن ينفذ الجملة التي داخل الـ (while) والتي هي جملة الطباعة

الجملة التي ستنفذ إذا تحقق الشرط تكون مكتوبة بين قوسين البداية والنهاية أي بمعنى بين { و }

الزيادة أو النقصان للعداد دائماً يذكر قبل قوس النهاية } الخاص بـ (while)

## ٤- الصيغة العامة للـ do.... while Statement

```
do
  {
    <statement>;
  }
while (expression);
```

do ثم كتابة الجملة أو الجمل المراد تنفيذها تكون مكتوبة بين قوسين البداية والنهاية أي بمعنى بين { و}

ثم يتم التحقق من الشرط

**Example:**

```
public static void main(String[] args)
 {
     int count = 1;
     do
      {
    System.out.println("Count is: " + count);
    count++;

      }

  while (count < 11);
 }
```

عرفنا متغير إسمه count من نوع int count وأعطايناه قيمة أولية = 1 وداخل الـ (do) ينفذ الجملة التي بداخله هي جملة الطباعة ... ثم يتحقق كل مرة من قيمة هذه المتغير هل مازالت أقل من 11 ويستمر أو يتوقف عند تحقق الشرط

الجملة التي ستنفذ إذا تحقق الشرط أو إذا لم يتحقق الشرط سوف تنفذ على الأقل لمرة واحدة ... تكون مكتوبة بين قوسين البداية والنهاية أي بمعنى بين { و}

الزيادة أو النقصان للعداد دائماً يذكر قبل قوس النهاية } العائد للـ (do)

ثم يتم التحقق من الشرط وهذا بعد قوس النهاية } العائدة للـ (do)

## ٥- الصيغة العامة للـ continue

هي من الكلمات المحجوزة في لغة جافا ... تستخدم مع جمل (control statement) لتؤدي غرض معين عند إستخدامها وسوف يتضح هذا في المثال التالي ... وغالباً ما تستخدم مع حلقات التكرار الـ ( for loop) و الـ (while)

```
continue;
```

الصيغة العامة وهي أن تكتب ثم تتبعها ; semicolon وكما موضح

**Example**

```
public static void main(String[] args)
  {
     for (int i = 1; i< 10; i++)
      {
         if(i%2 == 0)

        {
         continue;
        }
     System.out.println("this is Odd number" +"\t"+ i);
      }
  }
```

لدينا هنا حلقة تكرار (for loop) وهي من 1 الى 10 وتتخللها جملة (if) لتتحقق هل العدد زوجي .. إذا كان نعم إذن يتعدى ويذهب على الذي بعده وهكذا ... وبهذا نحن إستبعدنا الأعداد الزوجية ،، وإستخرجنا الأعداد الفردية وهو المطلوب

```
this is Odd number    1
this is Odd number    3
this is Odd number    5
this is Odd number    7
this is Odd number    9
```

The output

## ٦- الصيغة العامة للـ break

هي من الكلمات المحجوزة في لغة جافا ... تستخدم مع جمل (control statement) لتؤدي غرض معين عند إستخدامها ... وسوف يتضح هذا في المثال التالي ... وغالباً ما تستخدم مع حلقات التكرار الـ ( for loop) و الـ (while).

```
break;
```

الصيغة العامة وهي أن تكتب ثم تتبعها ; semicolon وكما موضح

**Example**

```
public static void main(String[] args)
  {
     for (int i = 1; i< 10; i++)
     {
        if(i%2 == 0)
        {
         break;
        }
   System.out.println("this is Odd number" +"\t"+ i);
     }
  }
```

لدينا هنا حلقة تكرار (for loop) وهي من 1 الى 10 وتتخللها جملة (if) لتتحقق هل العدد زوجي .. أذا كان نعم إذن يتوقف ويخرج من الحلقة التكرارية فوراً وهكذا... ونلاحظ الفرق واضح بالـ (output)

```
this is Odd number    1
```

The output

### ملحوظة هامة جداً ...

عند استخدام عبارة (continue) أو عبارة (break) فإن عملها مع الـ ( for loop) يختلف بالنتائج المخرجة فيما لو إستخدمت مع (while) أو (do..while)..... مهلاً سوف نلاحظ ونفهم هذا الأمر مع بعض... بالتوضيح التالي:

```
for ( i = 0 ; i < num ; i++ )
 {
  if ( i > num / 2 )
  continue ; // Jumps to update, i++
  System.out.println("Here I am!" ) ;
 }
```

```
while ( i != 3 )  // continue jumps here
  {
   continue ; // Jumps to condition, i != 3

   System.out.println( "Here I am!" ) ;
  }
```

... بعبارة أوضح يجب تجنب إستخدام الـ (continue) مع (while) أو (do..while) لإنها تدخل البرنامج في (infinity loop) في هذه الحالة ...

## ٧- الصيغة العامة للـ switch .... case

```
switch(expression)
 {
 case value-1:
     statement.......;
     break;
 case value-2:
     statement.......;
     break;
     .
     .
     .
 default:

    default- statement.......;
    break;
 }

rest-of-code;
```

switch ثم بين قوسين (كتابة التعبير المراد التحقق من حالته ) ثم قوس البداية { و } مكتوب فيما بينهما الحالات أو الإحتمالات لهذ التعبير

يجب كتابة default لإنها جزء من صيغة الـ (switch) في حال إذا لم يتحقق الشرط لأي من هذه الحالات سوف تنفذ على الجملة التالية .... وتكون مكتوبة ضمن قوسين البداية والنهاية التابعة لصيغة الـ (switch) أي بمعنى بين { و}

**Example:**

```
public static void main(String[] args)
 {
   int day = 5;
   String dayString;
            switch (day)
            {
            case 1:  dayString = "Friday";
                 break;
            case 2:  dayString = "Saturday";
                 break;
            case 3:  dayString = "Sunday";
                 break;
            case 4:  dayString = "Monday";
                 break;
            case 5:  dayString = "Tuesday";
                 break;
            case 6:  dayString = "Wednesday";
                 break;
            case 7:  dayString = "Thursday";
                 break;
              default: dayString = "Invalid day";
                          break;
            }
            System.out.println(dayString);
  }
```

عرفنا متغير من نوع int day وإسناد له قيمة أولية = 5 ومتغير أخر عرفناه هو من نوع String dayString وداخل الـ (switch) أخذنا الحالات التي نأخذها ..فمثلاً إذا كان 1 إذن يطبع لنا Friday وإذا كان 2 يطبع لنا Saturday وهكذا .. وفي حال كان غير أي من هذه القيم إذن يطبع القيمة الإفتراضية وهي Invalid day

كل عبارة من عبارات الـ ( case:) هي بمثابة (if) أي بمعنى نحن عندما نقول ← case 1: فنحن بمثابة نقول ← if(day == 1)

بعبارة أخرى إن (switch case) هي تكافئ عدة (if)

كلمة (**break**) وغالباً تستخدم مع صيغة (switch) وتذكر هنا للخروج نهائياً من كل الـ (switch case) في حال تحقق الشرط المطلوب

# ... خلاصة القول

١- الفرق بين (while) و(do… while) هو أنه في الـ (while) أن يتم التحقق من الشرط ثم ينفذ لذلك في حال أنه لم يتحقق الشرط لا ينفذ أي جملة ..أما الـ (do… while) فأنه يتم التنفيذ ثم يتحقق من الشرط لذلك هنالك دائماً على الأقل تنفيذ مرة واحدة عند إستخدام الـ (do… while).

٢- عمل أو سلوك **(continue)** بصورة عامة هو كالآتي ... نأخذ مثال بسيط من الواقع حتى تقترب الصورة والفكرة تكون أوضح ... كإنما شخص واقف على مدخل الطريق ويتحقق من الأشخاص المارة ... فيسأل كل من يريد الدخول ...هل أنت خريج هندسة حاسبات..؟ على سبيل المثال فإذا كان نعم ... يتعداه وينتقل على الذي يليه من الاشخاص ... ولا يتوقف من السؤال حتى ينتهي العداد في الحلقة التكرارية ... إن شاء الله الفكرة وضحت ...

أما عمل أو سلوك **(break)** بصورة عامة فهو ... نأخذ نفس المثال حتى تقترب الصورة وحتى نفهم وندرك الفرق ... فالشخص الذي يقف على مدخل الطريق ويسأل ... هل أنت خريج هندسة حاسبات..؟ فإذا كان الجوب نعم ... إذن يغلق الطريق ويخرج فور حصوله على أول شخص حقق الشرط ولا يكمل السؤال مع البقية حتى وإن كان العداد لم ينتهي من الحلقة التكرارية ... إن شاء الله الآن الفكرة وضحت..

٣- كلمة (continue) أو (break) تستخدم مع الحلقات التكرارية والتي هي (for loop) و (while) و(do… while) ونلفت الإنتباه إن الـ (continue) يتجنب أستخدامها مع (while) و(do… while) لانها تدخل البرنامج في (infinity loop).

٤- كلمة (continue) لاتستخدم مع جملة (switch … case) لإنها لاجدوى من إستخدامها ... بينما تستخدم كلمة ( break) مع جملة (switch … case) للخروج من جملة (switch … case) في حال تحقق الشرط الذي يسبقها...

# Arrays

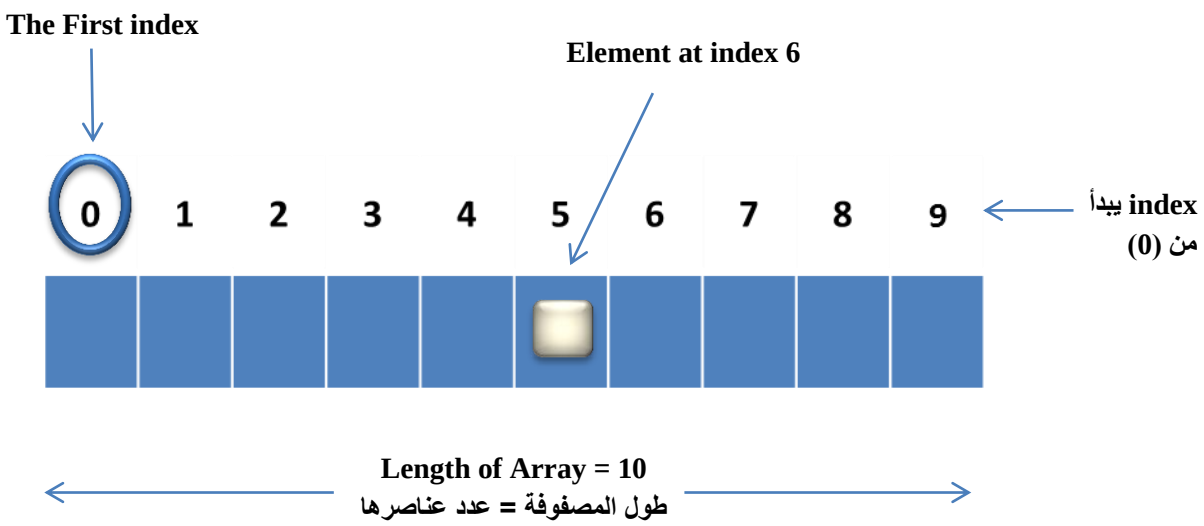
ماهي المصفوفات في لغة جافا..؟ ماهي الصيغة العامة (syntax) لتعريف المصفوفة داخل برنامج جافا..؟ ماهو سلوك المصفوفة داخل الـ (method)..؟ ماهو سلوك المصفوفة كـ (parameter) في الـ (method) ؟ كل هذه الاسئلة سـأجيب عنها ونوضحها مع الأمثلة .. إن شاء الله...

مفهوم المصفوفة بكل لغات البرمجة هو مفهوم واحد: هي الحاوية التي تحمل أو تضم في داخلها عدد محدد من العناصر بشرط أن تكون كل هذه العناصر من نوع واحد .. كإن تكون جميع عناصر أو قيم المصفوفة جميعها من نوع (int) أو تكون جميعها من نوع (double) أو جميعها من نوع (character) أو جميعها من نوع (String) وهكذا .. ولا يجوز أن تكون البعض من عناصرها من نوع (int) والبعض منها نوع (double) على الرغم من أن كلاهما نوع رقمي .

...كل مصفوفة في أي لغة من لغات البرمجة تحتوي على:

١. طول المصفوفة أو يطلق عليه (حجم المصفوفة ) وهي تساوي (عدد العناصر التي تحتويها المصفوفة).
٢. مؤشر (index) وهو الذي نستطيع من خلاله أن نصل إلى أي عنصر من عناصر المصفوفة.

المؤشر (index) في لغة جافا يبدأ من الصفر ..وفي الشكل التالي توضيح لكل ما ذكر في أعلاه...



## الصيغة العامة (syntax) لـتعريف المصفوفة

```
Data type array-name [ ];
int   myList  [ ];
```

**Or**

```
Data type [ ] array-name;
byte  [ ]  maybyteArray;
```

```
array-name = new  data type [ size ] ;
intArray = new  int [ 10 ] ;
```

أما الصيغة العامة لتعريف المصفوفة مع تحديد حجم المصفوفة بمعنى نحدد عدد العناصر التي تحويها هذه المصفوفة فيتم بشكل التالي

أما الصيغة العامة لتعريف المصفوفة مع تحديد حجم المصفوفة بمعنى نحدد عدد العناصر التي تحويها هذه المصفوفة وقيم لهذه العناصر في الوقت ذاته فيتم بشكل التالي

```
Data type  array-name [ ] = { value1, value2, value3 .... .... , valueN }
Int    intArray  [ ]  = { 1,2,3,4,5,6,7,8,9,10} ;
```

```
array-name [ index ] = value ;
```

أما الصيغة العامة لإعطاء قيمة معينة لأحد العناصر المصفوفة فيتم بشكل التالي

```
intArray [4] = 45;
```

قيمة العنصر في الـ (index) الخامس ... قيمته = 45
intArray [4] = 45 المكان الخامس لماذا ونحن ذكرنا
وذلك لإن البداية للـ index هى من 0

## Example:

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

    String[] names = { "Dad", "Mam", "Sister" };

     for (int i = 0; i <= names.length; i++)
       {
        System.out.println(names[i]);
      }
  }
```

عرفنا مصفوفة من نوع string وفي الوقت ذاته أعطيناها قيم والمتمثلة بـ { "Mam", "Dad", "Sister" } ثم طبعنا قيمة كل عنصر من خلال إستخدام الـ for loop

إستفدنا من حجم أو طول المصفوفة كشرط لتوقف عداد التكرار حيث names.length هو يمثل لنا طول المصفوفة بمعنى أنه **دائماً Arrayname.lenght** يرجع لنا طول المصفوفة وهو يمثل لنا عدد عناصرها وهنا = 3

name of array.length ← يرجع لنا عدد عناصر المصفوفة

## ... إستخدام المصفوفة كـ ( Method)

```
public class Class1
  {
    public  void printArray(int[] array)
      {
        for (int i = 0; i < array.length; i++)
          {
            System.out.print(array[i] + " ");
          }
      }

public static void main(String[] args)
      {
        Class1 cl1 = new Class1();

       cl1. printArray(new int[]{3, 1, 2, 6, 4, 2});

      }
  }
```

هنا إستخدمنا method من نوع **(void)** تأخذ الـ parameter من نوع مصفوفة
هذه الـ method وظيفتها لطباعة عناصر المصفوفة

هنا في class main تم إستدعاء الـ method التي تأخذ الـ parameter من نوع مصفوفة وكما تعلمنا في البداية نعرف الـ (Class) المكتوبة فيها الـ (method) ثم يتم إستدعائها

نعرف الـ (Class) المكتوبة فيها الـ (method) وبنفس الوقت أنشأنا منها object cl1

ثم يتم إستدعاء (method)

```
public class Class1
{
      public  int[] reverse(int[] list)
        {
        int[] result = new int[list.length];

for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
          {
          result[j] = list[i];
          }
        return result;
      }

public static void main(String[] args)
    {
      Class1 cl1 = new Class1();

int[] rev = cl1.reverse( new int[] {33, 11, 22, 66, 2});
  for (int i = 0; i<= rev.length; i++)
        {
         System.out.print (rev[i]+ " ");
        }
    }
}
```

هنا إستخدمنا method من نوع **(return value)** بمعنى أنه ترجع مصفوفة من نوع int
وفي نفس الوقت الـ parameter لهذه الـ method من نوع مصفوفة
هذه الـ method وظيفتها هي ترجع عكس عناصر مصفوفة المدخلة لها
إذن بعبارة أخرى ترجع مصفوفة
والـ parameter الذي تاخذه أيضاً مصفوفة

هنا في main class تم إستدعاء الـ method التي تأخذ الـ parameter من نوع مصفوفة وكما تعلمنا في البداية نعرف الـ (Class) المكتوبة فيها الـ (method)

حيث هنا نعرف object من الـ (Class) المكتوبة فيها الـ (method)

وهنا يتم استدعاء (method) وهنا يتم تعريف متغير توضع فيه القيمة المرجعة من الـ method ثم يتم استدعائها وهو هذا السطر
int[] rev = cl1.reverse(new int[]{33, 11, 22, 66, 2});

```
2  66 22 11 33
```

The output

...المهم معرفة كيف يتم إستدعاء method... راجع الشرح في بداية الكتاب... حتى نفهم الفكرة أكثر...

# Constructor

المشيد أو المشيدات ... هو إنشاء (method) على أن تكون هذه الـ (method) التي تم إنشاءها تأخذ نفس إسم ونفس نوع الـ (class) الذي كتبت بداخله... بمعنى إذا كان الـ (class) من نوع (public) فيجب أن تكون من نوع (public) كذلك ... وهذا مايطلق عليه إسم (Constructor) أو مشيد ... بالإضافة الى ذلك فإن هذه الـ (method) تكون لها ميزة إنها ... تنفذ تلقائياً بمجرد أخذ (instance) منها ... بمعنى لها أسبقية التنفيذ عن غيرها من الـ (method) بمجرد أخذ (object) منها ... سوف نوضح هذا بالتفصيل مع الأمثلة...

... في البداية لنتعرف على الصيغة العامة للـ (Constructor) ... كيفية إنشائه ...

```
public class ClassName
{
    ..............
    ...........
public ClassName ()
    {
    ..............
    ...........
    }

public static void main(String[] args)
    {
      ClassName cls1 = new ClassName();

    }
}
```

هنا لدينا class ذات إسم معين

إنشأنا منه constructor حيث نلاحظ هذا الـ constructor أخذ نفس إسم الـ class ونوعه بمعنى نفس إسمه (ClassName)، ونفس نوع الـ modifier (public)

هذا هو constructor ببساطة هو method أخذت نفس إسم ونوع الـ class الذي عرفت بداخله

تم إستدعاء constructor في الـ (main class) بمجرد أخذ instance فقط

الفكرة تتضح أكثر من خلال المثال التالي الذي يمثل class يحتوي على: (variables, method, constructor) تم أخذ أكثر من (Constructor) وإستخدمنا هنا (constructor with parameter) وتم توضيح كيفية الإستدعاء داخل (main Class)

```
package apptest;
public class employee
  {
    String empno;
    String Fname;
    String Lname;
    String Job;

  public employee (String Fname,String Lname)
  {
    this.Fname = Fname;
    this.Lname = Lname;
  }

  puplic employee (String empno,String Fname,String Lname)
  {
    this(Fname,Lname);
    this.empno = empno;
  //  this.Fname = Fname;
  //  this.Lname = Lname;
  }

  public  void print()
   {
    System.out.println("The number of emp is " + empno);
    System.out.println("The Fname of emp is " + Fname);
    System.out.println("The Lname of emp is " + Lname);
   }

    public static void main(String[] args)
     {
   employee emp = new employee("Esraa","Faisal");
   // emp.Fname = "Esraa";
   // emp.Lname = "faisal";
    emp.empno = "Software";
    emp.print();
      }
  }
```

هنا لدينا (class) أسمها employee تحتوي على المتغيرات من نوع string كما موضح ويحتوي على constructor و method

هنا إنشأنا constructor من (employee class) وفيه two parameter وهي (Fname,Lname) حيث نلاحظ هذا الـ constructor أخذ نفس إسم الـ class ونوعه بمعنى نفس إسمه (employee)، ونفس نوعه public

هنا أستخدمت كلمة (**this**) متى تستخدم وما الفائدة من إستخدامها ؟
تستخدم كلمة (this) مع الـ (constructor) في حال كان الـ (parameter) المستخدم مع الـ (constructor) نفس الاسم والنوع (variable) الموجود في الـ (class) ... أما الفائدة من إستخدامها ... فهي حتى نميز بإن هذا المتغير تابع الى الـ (class) وغير تابع الى (constructor)

هنا إنشأنا constructor آخر من employee class وفيه three parameter وهي (empno,Fname,Lname)

هنا تم حجز كلمة (this) ؟
لان لو لاحظنا الـ (constructor) الاول لوجدنا فيه نفس هذه الجملة موجودة وهي
this.Fname = Fname;
this.Lname = Lname;
إذن لاداعي لكتابتها مرة أخرة ويكتب بدلها
this (Fname,Lname);
ويجب ان تكتب بالسطر الاول كما موضح فى أعلاه ....

وهنا لدينا (method) عادية تقوم بعملية طباعة

وهنا (main class)

تم إستدعاء constructor الذي يحتوي على two prametres

## ...خلاصة القول

١- الـ (Constructor) ببساطة شديدة هو (method) تأخذ نفس إسم ونوع الـ (class) المعرفة بداخله وتستخدم متغيرات الـ (class) المعرفة بداخله كـ (parameter) لها .. فتسمى عندئذ بـ (Constructor). و تكون لها ميزة إنها ... تنفذ مباشرة بمجرد أخذ (instance) منها ... بمعنى لها أسبقية التنفيذ عن غيرها من الـ (method) الأخرى بمجرد أخذ (object) منها.

٢- نستطيع إنشاء عدد لاحصر له من الـ (Constructor) من الـ (class) الواحدة ... لكن بشرط كل (Constructor) يجب أن تختلف عن الـ (Constructor) الأخرى بالـ (parameters) أو الـ (data type) للـ (parameters) وهذا مايسمى بـ (overload) وسوف نوضحه لاحقاً بشرح أكثر تفصيلي.

٣- تستخدم كلمة (this) مع الـ (Constructor) في حال كان الـ (parameters) المستخدمة مع الـ (constructor) نفس إسم ونوع الـ (variables) الموجودة في الـ (class) ...أما الفائدة من إستخدامها فهي حتى نميز بإن هذا (variables) تابع إلى الـ (class) وغير تابع إلى (constructor).

٤- يتم إنشاء الـ (Constructor) من الـ (method) التي تكون من نوع (void) فقط ... و لايمكن إنشاء (Constructor) من (method) ذات نوع (return value).

# Inheritance

الوراثة أو التوارث وهو من أهم مبادئ لغات البرمجة الموجهة الـ (Object Oriented Programming) وهو أهم مايميزها عن باقي اللغات التقليدية هو مبدأ **الوراثة**...

حتى نطبق مبدأ الوراثة يجب **على الأقل** يكون لدينا (two classes) واحدة من الـ (class) تمثل لنا الأصل أو (superclass) بمعنى أخر (الأب) التي نرث منها ... والـ (class) الأُخرى تمثل لنا الفرع (subclass) أو بمعنى آخر (الإبن) الوارث..

**ماهي الفائدة من الوراثة**..؟ حسناً ... عندما نجعل (subclass) (الإبن) ترث من (superclass) (الأب) بمعنى أنه كل ما تمتلكه الـ (superclass) من (methods, variables) يكون ملك وتحت تصرف الـ (subclass) بالإضافة الى ماتمتلكه الـ (subclass) نفسها من (methods, variables)... لكن العكس غير صحيح ... بمعنى أنه ليس كل ماتمتلكه الـ (subclass) يكون من ملك الـ (superclass)... هذه هي آلية الوراثة في لغات (o.o.p) **(Object Oriented Programming)**

... الصيغة العامة للـ (inheritance) ....هي كالاتي:

class ثم إسم الـ class الذي سوف يرث (الإبن) ثم **extends** ثم إسم الـ class الموروث (الأب)

```
class subclass extends superclass
```

الصيغة العامة للـ (inheritance)

```
public class Sun extends Father
{
  ................
  .........
  ......

}
```

هنا بمجرد ما ذكرنا الصيغة العامة للـ (inheritance) أصبح الـ (class sun) وارث كل الـ (method,variables) الموجودة في class father لكن ليس كل مايملك class sun يصبح ملك الـ class father ...كما ذكرنا ليس العكس صحيح

... الفكرة تتضح أكثر إن شاء الله من خلال المثال التالي الذي يمثل two classes يحتوي أحدهما على: (variables, method, constructor) والـ class الأخر لايحتوي على أي سطر الـ code فقط عبارة التوارث من الـ class الأول أي عبارة **public class Person1 extends Human;** فقط

لدينا (two classes) واحدة هي (class Human) والتي سوف تمثل لنا الأب والأخرى وهي (person1class) وسوف تمثل لنا الإبن وسوف نشرح المثال بالتفصيل ... بالتأكيد كل (class) نقوم بإنشائها على حدا حتى تتضح لنا الفكرة أكثر:

```
package apptest;

public class Human
  {
    String name;
    int age;
    String gender;

    Human(String name,int  age,String gender)
    {
    System.out.println("I in Human Constructor");
    }

    void eat()
     {
        System.out.println(" I eat Apple");
     }

    void drink()
     {
        System.out.println(" I drink Water ");
     }

    public static void main(String[] args)
     {
      Person1 per = new Person1();
      per.eat();
      per.drink();
     }
  }
```

متغيرات (variables)

مشيد (constructor) مع parameter

Method عادية

Method أخرى

main class

في main class لو نلاحظ أنه تم عمل object من الـ class Person1 وتم إستدعاء method المكتوبة داخل class Human ... وهذا هو فائدة التوارث أنه class Person1 ورث كل الـ method المكتوبة داخل الـ class Human مع أنه لو نلاحظ لم نكتب أي method ولا حتى سطر (code) داخل class Person1

```
package apptest;

public class Person1 extends Human;
  {

  }
```

هنا لم نكتب أي سطر (code) فقط كتبنا الصيغة العامة للوراثة أي جعلنا class Person1 يرث من الـ class Human
وبهذا أصبح كل ما تمتلكه الـ class Human تحت تصرف وملك الـ class Person1

The output

```
I eat Apple
I drink Water
```

# Polymorphism

كلمة polymorphism تتألف من مقطعين هما كالآتي :

poly ← Many تعني تعدد ... والمقطع الآخر morphisms ← phases وتعني أشكال

بمعنى أن المقصود بـ polymorphism هو تعدد الأشكال ...إذن ما المقصود بتعدد الأشكال ...؟

مفهوم تعدد الأشكال هو التعامل مع أي حالة من الحالات الموجودة في حياتنا بأوجه أو أشكال مختلفة أو معالجة الأمور بطرق مختلفة للحصول على المطلب الذي نريده بعبارة أخرى تسخير أو تكيف الشيء ليتم الحصول على ما نريد ...

بينما في لغة جافا هو قدرة وإمكانية التعامل مع الـ (method) بأشكال مختلفة وأوجه مختلفة حتى نحقق ويتم تلبية المطلوب بأقل كتابة (code) ..كيف هذا..؟

لنفرض لدينا (method) تقوم طباعة كلمة (Hello) حسناً .. وطُلب مني أن أعمل (method) لطباعة كلمة (Welcome) ومرة أخرى طباعة (I' m Here) وهكذا فإيهما أفضل إنشاء كل مرة (method) جديدة لغرض طباعة المطلوب أم تغيير وتحوير أو تسخير بما لدي من (method) موجودة وأجعلها تطبع الكلمة المطلوبة ..؟ أكيد الخيار الثاني هو المرجح .. وهو إستغلال وتغيير (method) موجودة لدي أصلاً ...

إذن أنا استغليت أو وظفت بما لدي من إمكانيات و (method) وحصلت على المطلوب وأستغنيت عن كتابة (method) جديدة بمعنى أنه تعاملت مع (method) نفسها لكن غيرت بها بشكل آخر حتى أحقق المطلوب .. وهذا بإختصار شديد وبساطة أشد مفهوم الـ (Polymorphism).

إذن مفهوم الـ (Polymorphism) هو تغيير أو تحوير (تلاعب) في (methods)...لنستخدمها بشكل آخر

وفي لغة جافا مفهوم الـ (Polymorphism) يتجسد بـ (override) و (overload) وخاصة مع الوارثة

**إذن ماهو (overload)؟** هو عند الحفاظ على إسم الـ (method) والـ (code) المكتوب بداخلها و (return type) لها ... والتغيير يكون بعدد الـ (parameter) أو بـ (data type) للـ (parameter) .. يسمى عندئذ بـ **(overload)** وإن شاء الله تتضح الفكرة مع المثال..

*If name of the method remains common but the number and type of parameters are different, then it is called method overloading in Java.*

**إذن ماهو (override)؟** هو الحفاظ على إسم الـ (method) وعدد ونوع الـ (data type) للـ (parameter) الذي معها والـ ( return type) لها .. والتغيير يكون بالـ (code) المكتوب بداخلها ... يسمى عندئذ بـ **(override)** ونذكر أن شرط الأساسي لتحقق الـ (override) هو أن يكون لدينا (superclass) و (subclass)...إذن شرط وواجب وجود الوراثة مع (override) ... وإن شاء الله تتضح الفكرة مع المثال..

# Overload

هو تجسيد أو تمثيل لمبدأ أو مفهوم تعدد الأشكال الـ (Polymorphism).. لنأخذ المثال التالي لتتضح لنا فكرة (Overload)

```
package polymer;
public class Polymor
  {
  public double calclateSla(double salary)
      {
       return salary*0.2;
      }
  double calclateSla(double salary, double comm)
      {
       return  salary * comm;
       }
  double calclateSla(double salary, int comm)
      {
       return  salary * comm;
       }
  int calclateSla(double salary, int comm)
      {
       return  salary * comm;
       }

  public static void main(String []args)
    {
   Polymor pol =new Polymor();
   System.out.println (pol.calclateSla(2000,9));
    }
  }
```

لدينا class إسمها class Polymor تمتلك method إسمها calclateSla مع parameter واحد عمل هذه الـ method لحساب المرتب

method إسمها calclateSla مع parameter واحد وعمل هذه الـ method لحساب المرتب

هنا تم عمل overload نفس إسم method مع إختلاف في عدد parameter

هنا تم عمل overload نفس إسم method مع إختلاف في data type لأحد parameter

هنا لايمثل لنا overload لإنه تم التعامل مع الـ **return data type** للـ method... **وهذا ليس من شروط ال overload التغيير مع الـ return data type للـ method**

هنا main class وتم فيها إستدعاء method التي تقوم بوظيفة حساب المرتب واضح من عدد الـ parameter عند الإستدعاء

# Override

هو تجسيد أو تمثيل لمبدأ أو مفهوم تعدد الأشكال الـ (Polymorphism)، حتى نحقق مبدأ الـ (override) يجب توفر لدينا superclass و subclass ... إذن يجب أن يتوفر لدينا وراثة ... تتضح الفكرة إن شاء الله مع المثال التالي: والذي يمثل two class أحدهما تمثل superclass وهي class Employee والأخرى subclass وهي class Manager

```
package Employee;

public class Employee
  {
```

method إسمها calclateSla مع parameter واحد وعمل هذه الـ method لحساب المرتب

```
public double calclateSla(double salary)
  {
    system.out.println("Salary of Employee");
    return salary*0.2;
  }
```

method عادية للطباعة

```
public void Display()
    {
  system.out.println("This Super class");
    }

public static void main(String []args)
    {
      Manager mang=new Manager();
      double sal = mang.calclateSla(2000);
       System.out.println (sal);
       mang.Display();
    }
  }
```

هنا class main وتم عمل object للـ class Manager وتم إستدعاء method التي تقوم بوظيفة حساب المرتب **سوف يأخذ التي عمل عليها override** كما نرى output

```
package Employee;
```

Sub class Manager ورثت من superclass Employee حسب الصيغة العامة للوراثة

```
public class Manager extends Employee
  {
```

هنا تجسد مبدأ override لان تحققت جميع شروطها وهي: method موجودة superclass وذكرت في subclass بنفس الاسم ونفس عدد ونوع الـ parameters ونفس return data type

```
public double calclateSla(double salary)
    {
      system.out.println("Salary of Manager");
      return salary*0.2;
```

The output

```
Salary of Manager
 400
This is super class
```

# Override and Overload with Inheritance

ماهو مفهوم الـ (override) والـ (overload) في الوارثة ..؟ تتضح الفكرة أكثر مع المثال..إن شاء الله..

```
package apptest;

public class Human
  {
   String name;
   int age;
   String gender;
```

متغيرات (variables)

```
   Human(String name,int  age,String gender)
   {
   System.out.println("I in Human Constructor");
   }
```

مشيد (constructor) مع parameter

```
   void eat()
    {
        System.out.println(" I eat Apple");
    }
```

Method عادية

```
   void drink()
    {
        System.out.println(" I drink Water ");
    }
```

Method اخرى

```
    public static void main(String[] args)
     {
      Person1 per = new Person1();
      per.eat();
      per.drink();
     }
 }
```

main class

```
package apptest;

public class Person1 extends Human;
 {
```

هنا تتجسد مبدأ الـ override وهي المحافظة على إسم الـ method والـ parameters المستخدمة معه إن وجد .. التغيير فقط بالـ (code) المكتوب داخل الـ method ... يسمى هذا بـ **override**

```
   void eat()
    {
        System.out.println(" I eat Fish ");
        System.out.println(" I in Override ");

    }
```

هنا يتجسد مبدأ الـ overload وهي المحافظة على الـ (code) المكتوب داخل الـ method ... التغيير فقط الـ parameters المستخدمة مع الـ method أن وجدت ... بمعنى أنه أما نضيف parameters أو نحذف parameters ... وهذا ما يسمى بـ **overload**

```
   Human (int age,String gender)
    {
   System.out.println("I in Human Constructor");
    }

 }
```

إذا كان هنالك class أخرى وعملنا لها وراثة من الـ class Person1 وبتالي هذه الـ class سوف ترث كل ماتحتويه class peson1 و class Humane لإن class person1 هي ورثت من class human لذلك فإن الـ class الجديدة سوف ترث من الأثنين لكن إن كان هنالك method في class الإبن وعمل لها overload, override فإن الـ class الجديدة تأخذ الـ method التي تم عمل لها override أو overload وتترك الـ method الموجودة class الأب

# Abstract and Final with Inheritance

**Abstract** هي أحد أنواع الـ (Access modifier)... أي هي (Keyword) من الكلمات المحجوزة تستخدم بكتابتها في بداية تعريف الـ (class) أو الـ (method) ...أما الفائدة منها فهي لحماية الـ (class) من أن يأخذ منه (instance) لكن نستطيع أن نجعله يورّث أي بعبارة أخرى ... إذا كتبت (abstract) في بداية تعريف (class) فهذا الـ (class) لايمكن أخذ منه (object) ليتم التعامل معه داخل البرنامج ... لكن يمكن لهذا الـ (class) أن نورّث منه الى class آخر..

أما إذا كتبت **(abstract)** في بداية الـ (method) فإن هذه الـ (method) لايكتب داخلها أي (code) ..لذلك يعمل عليها (override) لكتابة (code) بداخلها.

**final** هي أحد أنواع الـ (Access modifier)... أي هي (Keyword) من الكلمات المحجوزة تستخدم بكتابتها في بداية تعريف الـ (class) أو الـ (method) ...أما الفائدة منها فهي لحماية الـ (class) من أن يورّث الى (class) آخر لكن نستطيع أن نأخذ منه (instance) أي بعبارة أخرى... إذا كتبت (final) بداية تعريف (class) .. فهذا الـ (class) لا يمكن أن نرث منه ..

أما إذا كتبت **(final)** بداية تعريف الـ (method) فهذه الـ (method) لايمكن عمل عليها (override).

حتى نعرف أكثر عن (abstract) و (final) ومدى تأثيرها وسلوكها داخل البرنامج مع الوراثة ... نتعرف أولاً الصيغة العامة لكتابة كلاً من (abstract , final) ... و نأخذ نفس المثال حتى نفهم الفكرة أكثر..ويتضح لدينا الفرق

## الصيغة العامة للـ (abstract) ... هي كالآتي:

وهي تكتب في بداية تعريف الـ class أو الـ method المطلوب حمايتها

```
abstract public class ();
```

الصيغة العامة للـ (abstract)

## الصيغة العامة للـ (final) ... هي كالآتي:

وهي تكتب في بداية تعريف الـ class أو الـ method المطلوب حمايتها

```
final public class ();
```

الصيغة العامة للـ (final)

... المثال التالي نوضح فيه سلوك وعمل (abstract) ... في البرنامج:

```
package apptest;

abstract public class Human
  {
   String name;
   int age;
   String gender;


 Human(String name,int  age,String gender)
   {
    System.out.println("I in Human Constructor");
   }

  void eat()
   {
    System.out.println(" I eat Apple");
   }


  void drink()
   {
    System.out.println(" I drink Water ");
   }

  public static void main(String[] args)
   {
    Human hum= new human();
    Human hum = new Person1 ();
    hum.eat();
    hum.drink();
   }

 public class Person1 extend Human;
   {
   }
```

هنا class Human أصبحت abstract

لا نستطيع أن ناخذ منه instance.. أي لانستطيع عمل منه (object)

لكن نستطيع أن نأخذ له instance من class وراث منه

إستطعنا أن نرث منها

... المثال التالي نوضح فيه سلوك وعمل (final) ... في البرنامج:

```
package apptest;

final public class Human
{
 String name;
 int age;
 String gender;


Human(String name,int  age,String gender)
 {
   System.out.println("I in Human Constructor");
 }

void eat()
 {
   System.out.println(" I eat Apple");
 }


void drink()
 {
   System.out.println(" I drink Water ");
 }

public static void main(String[] args)
 {
   Human hum= new human();
   Human hum = new Person1 ();
   hum.eat();
   hum.drink();
 }

public class Person1 extend Human;
 {
 }
```

هنا class Human أصبحت final

نستطيع أن نأخذ له instance

لا نستطيع أن نأخذ له instance من class وراث منه..لان final تمنع الوراثة أصلا .. فكيف بنا أن نأخذ instance من class على أساس أنه وارث منها...

لانستطيع أن نرث منه الى class آخر

# Behavior of Constructor with Inheritance

نأخذ مثال مختلف قليلاً لتتضح سلوك المشيدات مع الوراثة ..في هذا المثال لدينا (two classes) واحدة إسمها (Employee) والتي هنا تمثل لنا الـ (superclass) الأصل (الأب) الذي سوف نرث منه والـ (class) الأخرى (Manager) والتي تمثل لنا الـ (subclass) الفرع ... بمعنى (الإبن) الذي سوف يرث وكل من هذه الـ (classes) تحتوي على (variables) (method) وعلى أكثر من (constructor)...

```
package apptest;

public class Employee
{
  String empnm;
  String Fname;
  String Lname;
  String Job;
```

متغيرات (variables)

مشيد (constructor)

```
  employee()
   {
    System.out.println("I in Empty employee");
   }
```

مشيد (constructor) معه (parameters)

```
  employee(String empnm,String Fname, String Lname)
   {
  System.out.println("I in Employee Constructore");
  this.empnm = empnm;
  this.Fname = Fname;
  this.Lname = Lname;
   }
```

(method) عادية تقوم بالطباعة

```
  void print()
   {
   System.out.println("The number of emp is " + empnm);
   System.out.println("The Fname of emp is " + Fname);
   System.out.println("The Lname of emp is " + Lname);
  }
```

(main class)

```
    public static void main(String[] args)
      {
        Manager man;
        man = new Manager("esraa","faisal","aa");
        man.Fname = "Esraa";
        man.Lname = "faisal";
        man.dept = "Software";
        man.print();
      }
  }
```

Output

```
I in employee Constructore
I in Manager Constructore
The number of emp is esraa
The Fname of emp is Esraa
The Lname of emp is faisal
```

```
package apptest;

public class Manager extends Employee
  {
 String dept;
```

مشيد (constructor)

```
  Manager ()
    {
   // super(null,null,null);
    System.out.println("I in Empty Manager");
    }
```

مشيد (constructor) معه (parameters)

```
  Manager(String empnm,String Fname,String Lname)
    {
     super(empnm,Fname,Lname);
     System.out.println("I in Manager Constructore");
     //this.empnm = empnm;
     //this.Fname = Fname;
     //this.Lname = Lname;
    }
 }
```

إستخدمت كلمة (super) لماذا؟ متى تستخدم؟ ومافائدة إستخدامها؟
كلمة (super) تستخدم **مع (constructor) أثناء الوراثة فقط** وعملها مثل عمل الـ this التي ذكرناها سابقاً وتستخدم super عند ظهور constructor في الـ class الإبن وفي نفس الوقت ظهوره في class الأب مع نفس الـ (parameters) لكلاهما... إذن نستخدم كلمة super حتى نميز ان هذه الـ constructor تعود الى الـ class الأب وليس للإبن لذلك نكتبها ولا داعي لكتابه الـ constructor مع متغيراته مرة أخرى لذلك تم حجز كلمة this لان وجود كلمة super يلغي فائدة this مع متغيراتها ..إذا كانت المتغيرات التي مع this هي نفسها الـparameter المستخدمة مع الـ constructor .. والفائدة الثانية هي حتى عند التنفيذ لايذهب الى أي constructor الخالي من الـ parameter المكتوب داخل class الاب ويجب أن تكتب عبارة super في أول سطر

```
  Manager(String empnm,String Fname,String dept)
    {
     super.empnm = empnm;
     System.out.println("I in Manager Constructore");
     this.dept = dept;
    }
```

نلاحظ إختلاف استخدام this و super هنا نظراً لإختلاف parameter في الـ constructor

# ...خلاصة القول

١- الـ (Inheritance) أو مبدأ الوراثة يطبق إذا توفر على الأقل (two classes) حتى تمثل إحدهما لنا الـ (superclass) الأصل أو الأب الذي سوف نرث منه ..و الآخر (subclass) الـذي يمثل لنا الفرع الإبن الذي سوف يرث .

٢- الـ (superclass) الأب يورث الى الـ (subclass) الإبن كل شيء من نوع (public) سواء كان (class) أو كان (method) أو (variables) ... إذن شرط الوراثة أن تكون الـ (class) أو (method) أو (variables) من نوع ( public modifier) حتى يرثها الـ (subclass) الإبن.

٣- تستخدم الـكلمة (super) وهي من الكلمات المحجوزة ((keyword)) مع الـ (Constructor) أو مع ((method)) عمل لها (override) في الـ (subclass) الابن) وهذا في حال كان هنالك **وراثة** في البرنامج .. والفائدة من إستخدامها هي حتى تمنع تنفيذ الـ ( constructor) الخالي ... بمعنى الـ (constructor) الذي لايحمل (parameters) والمكتوب داخل الـ (superclass) الأب **لاينفذ**...

٤- غالباً ماتستخدم الـ (modifier) مثل (Abstract) أو (Final) مع الـ (class) أو (method) ...
حيث تستخدم **abstract** بكتابتها في بداية تعريف الـ (class) أو الـ (method) ...أما الفائدة منها فهي لحماية الـ (class) من أن ياخذ منه (instance) لكن نستطيع ان نجعله يورّث... أي بعبارة أخرى ... إذا كتبت (abstract) في بداية تعريف (class) ..فهذا الـ (class) لايمكن أُخذ منه **(object)** ليتم التعامل معه داخل البرنامج ... لكن يمكن أن نورّث منه الى class آخر...
أما إذا كتبت (abstract) في بداية تعريف الـ (method) فإن هذه الـ (method) لايكتب داخلها أي (code) ..لذلك يعمل عليها (override) لكتابة (code) بداخلها.
أما **final** فهي تستخدم بكتابتها في بداية تعريف الـ (class) أو الـ (method)...والفائدة منها فهي لحماية الـ (class) من أن يورّث الى class آخر لكن نستطيع ان نأخذ منه (instance) أي بعبارة أخرى ... إذا كتبت (final) في بداية تعريف (class) ..فهذه الـ (class) لانستطيع أن نرث منها ... أما إذا كتبت (final) في بداية تعريف الـ (method) فهذه الـ (method) لايمكن عمل عليها (override).

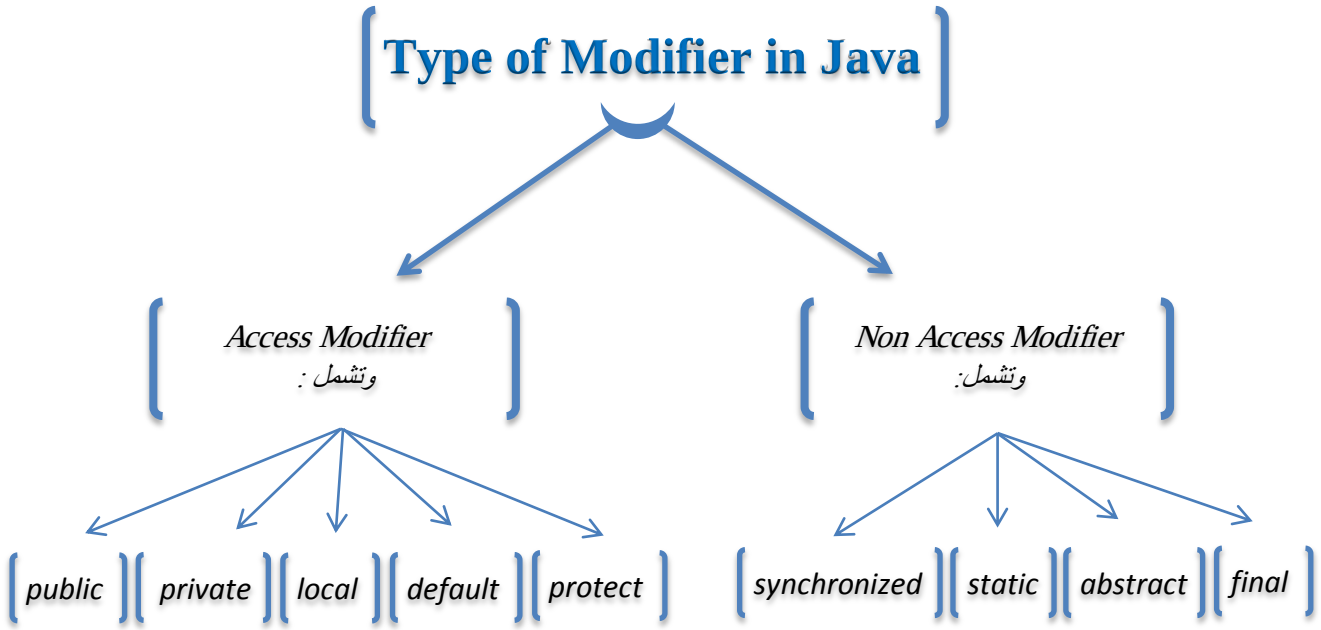
... إن شاء الله أحطت بأهم مايتعلق من المصطلحات التي تتعامل مع مفهوم الوراثة (inheritance) وسلوكها كيف يختلف داخل البرنامج ...

# Modifier

ماهو المقصود بـ (**Modifier**) ..؟ هي المُبدلات .. المُغيرات ..المعدلات.. بمعنى الذي يغير لنا من وضع إلى آخر .. أو تعدّل أو تحدد من حالة الى أخرى ..كيف هذا ..؟

حسناً هو إستخدام بعض الكلمات المحجوزة في لغة جافا الـ (keywords) وتضمينها أي جعلها ضمن تعريف ( class أو method أو variable) وبهذا سوف تغير من سير أو تحدد سلوك و تعريف كلاً من ( class أو method أو variable) داخل البرنامج ...وقد وضحنا في بداية الكتاب صيغة تعريف كل من ( class أو method أو variable) وبينا أنه الـ (**Modifier**) تكتب في بداية التعريف لكلاً منهما.

... لغة جافا تمتلك نوعين من الـ Modifier وكما موضح في أدناه:




وقد تطرقنا في بداية الكتاب هذا حول كيفية إستخدام كلاً من (abstract ,final) وذلك من خلال جعلها ضمن تعريف ( class أو method أو variable) وايضاً وضحنا مدى تأثير والفائدة منها

وقد تطرقنا في بداية الكتاب هذا حول كيفية إستخدام كلاً منها وذلك من خلال جعلها ضمن تعريف ( class أو method أو variable) وايضاً وضحنا مدى تأثير والفائدة منها

حسناً لم يتبقى لدينا من أنواع الـ (Modifier) سوى الـ (static) وسوف نوضحها بالتفصيل مع الأمثلة... إن شاء الله

# Static Modifier

تستخدم (static) مع :

- تعريف المتغيرات (Variable)
- الدوال (Method)
- (Class nested within another Class)
- (Initialization Block)

لا تستخدم (static) مع :

- Class (Not Nested)
- Constructor
- Interfaces
- Method Local Inner Class (Difference then nested class)
- Inner Class methods
- Instance Variables
- Local Variables

حسناً.. تعرفنا أين تستخدم (static) وأين لا تستخدم.. تبقى أن نتعرف ماالفائدة أو الغرض من إستخدمها...

تستخدم كلمة (static) للمشاركة ... كيف هذا..؟ بمعنى إذا استخدمت كلمة (static) مع تعريف المتغيرات جعلت من هذه المتغيرات إمكانية إستخدامها ومشاركتها من قبل كل أجزاء البرنامج وفي أي مكان بالبرنامج ... وكإنما جعلت هذه المتغيرات (global variable) ونفس الحال ينطبق إذا إستخدمت مع الـ ( class أو method) ...

... سوف تتضح الفكرة أكثر مع الأمثلة و مع كل حالة من حالات إستخدام (static)...

## *Static with Variables*

...الفائدة من إستخدام (static) مع (variables) المتغيرات هي لمشاركة المتغيرات وكما موضح في المثال التالي:

```
public  class Employee

  {
    int idemp;
    static int empno;

 public static void main(String[] args)
   {
     Employee emp1 = new Employee();
     emp1.idemp++;
     emp1.empno++;
     System.out.println(emp1.idemp);

     System.out.println(emp1.empno);


     Employee emp2 = new Employee();
     emp2.idemp++;
     emp2.empno++;
     System.out.println(emp2.idemp);

     System.out.println(emp2.empno);


     Employee emp3 = new Employee();
     emp3.idemp++;
     emp3.empno++;
     System.out.println(emp3.idemp);

     System.out.println(emp3.empno);

   }
  }
```

لدينا class Employee تمتلك متغيرين هما:.. idemp وهو نوع int وليس static ... ومتغير الثاني empno وهو نوع static int ... بمعنى أنه من نوع static لإن كتبنا في بداية تعريف المتغير كلمة static

في class main عملنا object من class Employee

قمنا بزيادة العداد لكل من idemp, empno

هنا سوف يطبع لنا قيمة idemp هي 1

وهنا سوف يطبع لنا قيمة empno هي 1

أنشأنا object جديد من class Employee مرة أخرى

قمنا بزيادة العداد لكل من idemp, empno

وهنا سوف يطبع لنا قيمة empno هي 1

وهنا سوف يطبع لنا قيمة empno هي 2

أنشأنا object جديد من class Employee مرة أخرى

قمنا بزيادة العداد لكل من idemp, empno

وهنا سوف يطبع لنا قيمة empno هي 1

وهنا سوف يطبع لنا قيمة empno هي 3

**The output is:**

```
1
1
1
2
1
3
```

نلاحظ output نجد في كل مرة نعمل object من class Employee سوف يبدأ المتغير idemp من قيمة 0 وكأنه يراه متغير جديد ... بينما قيمة المتغير empno بقيت تزيد في كل مرة نعمل فيها initialize للـ class Employee لإنه تم تعريفه أنه static ... وهذه هي الفائدة من تعريف المتغير static وهي **مشاركة المتغيرات...** من قبل الكل... بمعنى إنه كل الـ object المعرفة من class Employee وهي : (emp1,emp2,emp3) تعرفت على empno لإنه تم تعريفة من نوع static ولم تتعامل معه على أنه متغير جديد كما هو الحال مع المتغير idemp الذي عرف على أنه ليس static... يرجى تتبع المخرجات حتى توضح الفكرة أكثر

## Static with Methods

الفائدة من إستخدام static مع Methods : هو عند إستدعاء الـ method يكون بشكل مباشر بذكر إسم الـ class يليه إسم الـ method مباشرة دون الحاجة الى تعريف object كما موضح في المثال التالي:

```
public class Employee
  {
    int idemp = 24;
    static int empno = 28;
```

لدينا class Employee تمتلك متغيرين وهما .. idemp وهو نوع int ليس static ومتغير الثاني empno وهو نوع static int.. بمعنى أنه من نوع static لإن كتبنا في بداية تعريف المتغير كلمة static وأعطينا لكل واحد منهما قيمة

```
public static void print1()
  {
   System.out.println(idemp);
   System.out.println(empno);
   }
```

لدينا method من نوع **static** إسمها print1 تقوم بعملية **طباعة المتغير idemp وهو متغير ليس static طباعة المتغير empno وهو متغير static**

```
public  void print2()
  {
   System.out.println(idemp);
   System.out.println(empno);
  }
```

لدينا method من نوع **ليس static** إسمها print2 تقوم بعملية **طباعة المتغير idemp وهو متغير ليس static وطباعة المتغير empno وهو متغير static**

```
public static void printHello()
  {
   System.out.println("I At static Method");
  }
```

Method من نوع **static** تقوم بطباعة جملة عادية

```
public void printHello2()
  {
   System.out.println("I At Not static Method");
  }
```

Method من نوع **ليس static** تقوم بطباعة جملة عادية

```
public static void main(String arg[])
 {
   Employee.print1();

   Employee  emp= new Employee ();
   emp.print2();
   Employee.printHello();

   Employee.printHello2();
 }

  }
```

Class main

عملت (run) للبرنامج هذا وظهرت لي خطئين هما:

- Error(9,30): non-static variable idemp cannot be referenced from a static context
- Error(31,21): cannot find method printHello2()

**الخطأ الاول** يخص أن method من نوع **static** يجب أن تتعامل مع متغيرات من نوع **static** أيضاً والمتغير idemp هو **Non static** وإستخدم مع method من نوع **static** لذلك تم الاعتراض عليه
**الخطأ الثاني** أن printHello2 () هي method من نوع **Non static** لذلك لايمكن استدعائها بذكر اسم الـ class مباشرة وإنما يتم إستدعائها عن طريق object المأخوذ من class وهو emp بمعنى أنها تستدعى هكذا **emp.printHello2();**

```
I'm At static Method
28
24
28
I'm At  Non static Method
```

بعد أن نرفع جملة الطباعة الخاصة بالمتغير idemp من print2() ونستدعي printHello2() بشكل التالي `emp.printHello2();` فإن out put يكون هكذا ..يرجى تجربته بأنفسكم لتتضح الفكرة أكثر

## *Static Block*

ماهي static block ..؟ وأين تكتب ..؟ وماهي الصيغة العامة لكتابتها..؟ ومتى تنفذ ..؟ كل هذه الأسئلة سأجيب عنها بوضوح إن شاء الله ويتم توضيحها أكثر مع كل مثال:

... إليك التعريف التالي الذي يوضح ماهي *static block*

***Static block:*** *is a normal block of code enclosed in braces, { }, and preceded by the static keyword.*

إذن هي مجموعة من الأسطر البرمجية أو الأوامر البرمجية محصورة بين قوسين البداية والنهاية { } تسبقها كلمة static

أما الصيغة العامة لها فهي: كما ذكرنا في التعريف مجموعة من الإيعازات المراد تنفيذها تم حصرها بين قوس بداية ونهاية { } تسبقها كلمة static

Static Block in java Syntax:

```
static
  {
.....
.....
  }
```

قوس البداية والنهاية مسبوق بكلمة static وبداخلهما تكتب أي إيعازات برمجية مراد تنفيذها سواء كانت عبارة عن method، constructor، إعطاء قيمة لمتغيرات ، جمل طباعة ،..... الخ

... تعرفنا على static block وتعرفنا على الصيغة العامة لها ... تبقى أن نتعرف أين تكتب..؟ ومتى تنفذ..؟

تكتب في أي مكان في البرنامج حتى لو بعد قوس النهاية الخاص بـ (main class) ..أما متى تنفذ ... فهي تنفذ أول ما يتحمل البرنامج في بيئة الـ ((JVM) أي (Java Virtual Machine)) بمعنى إنها تنفذ قبل الإيعازات الأ خرى المكتوبة داخل الـ (main class) أي تنفذ قبل الكل ... سنلاحظ هذا الأمر أكثر وضوحاً مع الأمثلة ...

... في المثال التالي أخذنا (class) تحتوي على أكثر من (static block) و أكثر من (methods) و (variables) و(constructor) الغرض أو الغاية من هذا كله حتى نفهم تسلسل التنفيذ مع وجود كل هذه الأمور .. أيها ينفذ أولاً ... وأيها ينفذ فيما بعد وهكذا... لنرى المثال ونرى (output) له ... لتتضح الفكرة أكثر...

```
package blk;
public class BlockStatic
{
static
 {
  System.out.println("This is first static block");
 }

 public BlockStatic()
 {
   System.out.println("This is constructor");
 }

 public static String StaticString = "Static Variable";

 static
  {
   System.out.println("This is second static block and " + StaticString);
  }

 public static void main(String[] args)
  {
   BlockStatic statEx = new BlockStatic();
   BlockStatic.staticMethod2();
  }

 static
  {
   staticMethod();
   System.out.println("This is third static block");
  }

 public static void staticMethod()
  {
   System.out.println("This is static method");
  }

 public static void staticMethod2()
  {
   System.out.println("This is static method2");
  }
}
```

لدينا class إسمها BlockStatic تحتوي على static block و على method و constructor كما موضح في أدناه كلاً منها

لدينا **static block** متمثلة بأمر طباعة فقط

لدينا **constructor** تقوم بعملية طباعة فقط

تعريف متغير من نوع **static String** وإعطاؤه قيمة

لدينا **static block** متمثلة بأمر طباعة جملة معينة مع قيمة المتغير الذي عرفناه الآن وهو **StaticString**

class main وفيها تم إستدعاء للـ method التي إسمها **staticMethod2()**

لدينا **static block** تتكون من استدعاء للـ method التي إسمها **staticMethod();** وكذلك فيها إيعاز طباعة جملة معينة **ونلاحظ إنها كتبت بعد الـ class main**

لدينا method من نوع **static void** إسمها **staticMethod()** تقوم بعملية **طباعة**

لدينا method من نوع **static void** إسمها **staticMethod2()** تقوم بعملية **طباعة**

**The output :**

```
This is first static block
This is second static block and Static Variable
This is static method
This is third static block
This is constructor
This is static method2
```

عند متابعة (output) نعرف تسلسل التنفيذ فنجد كيف أول ما تم تنفيذه هو static block الاولى ومن ثم الثانية وهكذا وهذا الغرض من البرنامج لنرى ونعرف تسلسل التنفيذ

## ...خلاصة القول

١- الـ (Modifiers) المُبدلات أو المغيرت ..هي (Keywords) كلمات محجوزة في لغة جافا... والتي عند إستخدامها في بداية تعريف الـ (variables) أو (methods) أو (class) ستؤدي الى تغيير سير عمل أو سلوك هذه الـ (variables) أو الـ (methods) أو (class) نتيجة إستخدامنا لهذه الكلمات معها ..

٢- الـ (Modifiers) تكون على نوعين:

أما (Access Modifier) وتشمل ( public, private, protect, ) .

أو (Non Access Modifier) وتشمل ( abstract, final, static ,synchronize)

وقد تم توضيح كلاً منهما في بداية الكتاب وتعرفنا على مدى تأثيرها على (variables) و (methods) في حال إستخدمت معهم...

٣- الـ (static modifier) تستخدم مع الـ (variables) والـ (methods) والـ (class) و(block).

٤- الفائدة من إستخدام (static) مع (variable) فهي لمشاركة هذه المتغيرات مع الكل أي أصبح المتغير الذي تم تعريفه من نوع (static) وكإنه (global variables) متعرف عليه من قبل كل إجزاء البرنامج.

٥- الفائدة من إستخدام (static) مع (method) أيضاً للمشاركة هذه الـ (method).. أو بعبارة أخرى لا حاجة لتعريف (object) لإستدعاء هذه الـ (method) وإنما نذكر class name . method name()

٦- (static block) مجموعة أو حزمة من الإيعازات البرميجية تم كتلبتها بين قوسي بداية ونهاية تسبقها كلمة static ... تسمى هذه بـ (static block) ..تنفذ هذه الحزمة البرمجية قبل أي جزء من البرنامج بمعنى إنها تنفذ قبل الجمل البرمجية المكتوبة بالـ (main class).

# Encapsulation

***Encapsulation***: ترجمتها تعني تغليف أو تضمين..وماذا بعد..؟ مازال الأمر مُبهم وغير واضح .. حسناً الأمر ببساطة شديدة في بعض الأحيان يتم تعريف متغيرات من نوع (private modifier) أي من النوع الخاص فإذا تم تعريف المتغيرات بطريقة خاصة داخل (class) فإنه لايمكننا الوصول إليها بطريقة مباشرة من خارج هذه الـ (class) التي عُرفت بداخلها .. يتطلب الوصول إليها بشكل خاص أيضاً.

وبالتالي تم إخفاء هذه المتغيرات داخل هذا الـ (class) وهذا مايشار اليه بـ (encapsulation) أي تغليف أو إخفاء البيانات وعدم الوصول اليها بطريقة مباشرة أو بطريقة عشوائية وإنما يتم الوصول اليها بطريقة خاصة آمنة...

إذن يمكن تعريف (encapsulation) هي تقنية تعريف متغيرات من نوع (private) داخل الـ (class) وتوفير إمكانية الوصول إلى هذه المتغيرات عبر الأساليب العامة.

كما يمكن وصف (encapsulation) كحاجز وقائي يمنع الأوامر أو الإيعازات البرمجية والبيانات التي يتم الوصول إليها بشكل عشوائي من قبل الايعازات البرمجية الأخرى المعرفة خارج (class).

والفائدة الرئيسية من (encapsulation) هو القدرة على تعديل الإيعازات البرمجية لدينا وتنفيذها دون كسر الطريقة التي كتبت أو عُرفت بها... مع ميزة (encapsulation) يعطي الصيانة، والمرونة والتوسع في برمجة الإيعازات المكتوبة سابقاً...

... وفرت لنا لغة الجافا أو لغات (O.O.P) الـ ( Object Oriented Programming) دالتين جاهزتين (methods) والتي من خلالها أعطتنا إمكانية الوصول والتغيير في المتغيرات المعرفة بشكل (private) وهما :

دالة (set) ودالة (get) إن شاء الله سوف يتضح المفهوم مع الأمثلة...

دالة (set) : لتعيين أو تحديد أو إسناد قيمة للمتغير الذي تم تعريفة بداخل الـ (class) من نوع private.

دالة (get) : لجلب أو إسترجاع تلك القيمة التي تم إسنادها بدالة (set).

```
package client;

public class Encap
  {
    private String Myname;
    private  int IDemp ;

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

  public void setIDemp(int IDemp)
     {
      this.IDemp=IDemp ;
     }

  public String getName()
     {
      return Myname;
     }

  public int getIDemp()
     {
      return IDemp;
     }

  public static void main(String arg([]
    {
      Encap name = new Encap ();
      name.setName("Esraa Faisal");
      name.setIDemp(14);
      System.out.println( name.getName());
      System.out.println( name.getIDemp());
    }
}
```

لدينا class إسمها Encap بداخلها عُرف متغيران من نوع private

إستخدام set method ليتم تعيين قيمة للمتغير **Myname لانه من نوع** private

إستخدام set method ليتم تعيين قيمة للمتغير **IDemp لانه من نوع** private

إستخدام get method ليتم جلب قيمة المتغير **Myname**

إستخدام get method ليتم جلب قيمة المتغير **IDemp**

لو نلاحظ إستخدمت **كلمة this** وقد ذكرنا سابقاً عندما method تستخدم نفس إسم ونوع المتغير الموجود في class فإننا نستخدم this حتى نميز إن هذا المتغير تابع للـ class وليس للـ method

في class main تم إعطاء القيم وطباعتها

تمكنا من خلال الدوال الجاهزة الموفرة لدينا من قبل لغات (O.O.P)
**وهي دالة set ودالة get من تعيين قيم وإسترجاع هذه القيم لمتغيرات من نوع private من غير ما نكسر التعريف الاصلي لها وهو private**

```
Esraa Faisal
14
```

The output

# Interfaces

هي أحد مفاهيم ومبادئ الـ (object oriented programming) وبما إن (object oriented programming) هي وصف لكل كائن بحياتنا اليومية إذن حتى نفهم معنى الـ (interface) فلنأخذ مثال قريب من حياتنا اليومية ..

(interface) هي كإنما مدير يملي على الموظف الشروط والواجبات التي يقوم بها لإنجاز العمل ولايهمه الكيفية التي يجري بها العمل المهم هو أعطى الخطوط العريضة لإتمام عمل ما... والموظف هنا يمثل لنا (class) الذي ماعليه سوى الطاعة لتنفيذ **كل** الأوامر فقط ... إذا إحدى هذه الشروط لايتم تنفيذها من قبل الـ (class) يكون هناك (compiler error).

إذن :

***Interfaces** define a set of commands that a class will obey.*

إذن تعرف على إنها مجموعة من الأوامر المتمثلة بـ (methods) و (variables) والتي سوف تستخدم داخل الـ (class) والـ (class) عليه التنفيذ .. وكإنما هي القالب الذي يشير أو يصف لنا ماذا سينفذ من varaiable, method داخل البرنامج

إن (interface) شبيه الى حدٍ ما بـ (class) من ناحية الهيكلة البرمجية وتختلف معها في بعض النقاط ...حتى نفهم الفكرة أكثر سنقوم بشرح وتوضيح كل مايتعلق بـ (interface) ...وهي:

- أوجه الشبه والإختلاف بين الـ (interface) والـ (class).
- الصيغة العامة لتعريف وبناء الـ (interface).
- كيفية إستخدام الـ (interface) في الـ (class). Implements of interface
- ذكر أكثر من مثال لتوضيح أكثر من حالة من حالات الـ (interface).

...أهم أوجه الشبه والإختلاف بين الـ (interface) والـ (class) نوضحه بسبعة نقاط... ماهي:

... تم ذكر كل نقطة من النقاط باللغة الإنجليزية وتوضيحها باللغة العربية لتكون الفكرة أوضح ...

***1-*** *The syntax (Similar to public class definition rules).Use the Keyword **interface** to define an interface.*

١- الصيغة العامة متشابهة تقريباً ... فعند تعريف الـ (class) كانت بالشكل التالي: كلمة class تتبعها إسم الـ class ثم قوسي البداية والنهاية وبداخلهما الإيعازات البرمجية (cods).

```
public class classname
{
}
```

أما في (interface) الصيغة نفسها فقط نضع بدل كلمة class نستبدلها بكلمة interface تتبعها إسم الـ interface ثم قوسي البداية والنهاية وبداخلهما الإيعازات البرمجية (cods).

```
public interface interfacename
{
}
```

***2-*** *Interfaces don't have constructors as they can't be initiated*

٢- لا وجود للـ (constructors) إطلاقاً بـ (interface).

الإيعازات البرمجية التي تكتب داخل الـ (class) تتمثل بـ (variables, methods, constructors)

إما الايعازات البرمجية التي تكتب داخل الـ (interface) تتمثل بـ (variables, methods) فقط لاغير ...

إذن لا وجود للـ (constructors) إطلاقاً بـ (interface).

***3-*** *The interface definition the names of the methods and their return types and argument signatures. There is no executable body for any method that is left to each class that implements the interface.*

٣- في الـ (interface) تكتب الـ (method) إسمها فقط بدون ذكر التفاصيل بمعنى أنه (no body of method) بعبارة أخرى (method without code)... أي أنه إشارة فقط لوجود (method)..هكذا ← **methodname();**

حيث تترك مهمة تنفيذ الـ (method) على الـ (class) التي سوف تطبق وتنفذ الـ (interface).

أما داخل الـ (class) فتكتب الـ (method) بكافة تفاصيلها أي بمعنى بما تحتويه من (parameters) و(code) المطلوب تنفيذه والـ (return type) لها...

*4- No objects can be created from an interface (you can't do this)you can that at one state only when is implemented*

٤- في الـ (interface) **لا يمكن** إنشاء منها (instance) أي لايمكن إنشاء منها نسخة... بمعنى لايمكن تعريف (object) منها... لإن تنفيذها يتم من خلال (implemented by class)..لكن يمكن في حالة واحدة فقط.. أخذ (object) عندما تنفذ داخل الـ (class) فقط....وسنرى ذلك مع الأمثلة موضح أكثر..

أما في الـ (class) فعند تنفيذها ... نستطيع أن ناخذ منها (instance) بمعنى نستطيع تعريف منها (object) لكي نستطيع من خلاله التعامل مع هذه الـ (class) وإستدعاء وتنفيذ الـ (method) المكتوبة بداخلها في (classmain) من خلال هذه الصيغة: ← **classname objname = new classname();**

*5- An Interface can extends one or more interfaces.*

٥- يمكن تطبيق مبدأ الوراثة مع (interface). حيث يمكن للـ (interface) أن ترث لواحدة أو أكثر من (interface) أخرى.

*6- interface functions should be public and abstract. Interface fields should be public and final.*

٦- الـ (method) المستخدمة مع الـ (interface) تكون نوع (abstract public) فقط لاغير ... والـ (variables) المستخدمة تكون نوع (final) سواء تم الإعلان عنها صراحة أم لا... فهى تأخذ من هذا النوع فقط .

أما الـ (method) المعرفة داخل الـ (class) فيمكن إستخدام أي نوع من أنواع الـ (modifier) المتاحة لدينا كإن تكون (public, private ,protected) مع ( abstract أو final)...

*7- A class implementing an interface should use the keyword implements.*

٧- الـ (class) التي سوف تنفذ الـ (interface) يجب أن نستخدم كلمة implements لإن الـ (interface) لاتنفذ في الـ (main class) بل تنفذ من قبل إستدعائها بـ (class) أي بمعنى (implemented by class).

... أوجه الشبه والإختلاف بين الـ (**class**) والـ (**interface**) حتى تكون الفكرة أوضح ... نراها مع الأمثلة وتوضيح كل نقطة من النقاط التي ذكرت في أعلاه ...

هذا المثال نوضح أوجه الشبه والإختلاف وعمدت على عدم ربطها برمجياً بل توضيح **syntax** لكلاً منهما فقط...

الصيغة العامة أو syntax هو نفسه....
لكن هنا نكتب class وإسمها

```
public class Person1
{

  Person1 ()
  {
  System.out.println("I in Constructor");
   }

 void Eat()
  {
   System.out.println(" I eat Fish ");
  }

public static void main(String [] args)
 {
    Person1 per1= new person1();
    per1. Eat(); =
  }
}
```

الـ code الذي داخل class تمثل لنا
method,constructor مع الـ code الخاص بها

تم أخذ instance من الـ class كما ذكرنا

```
Person1 per1= new Person1();
```

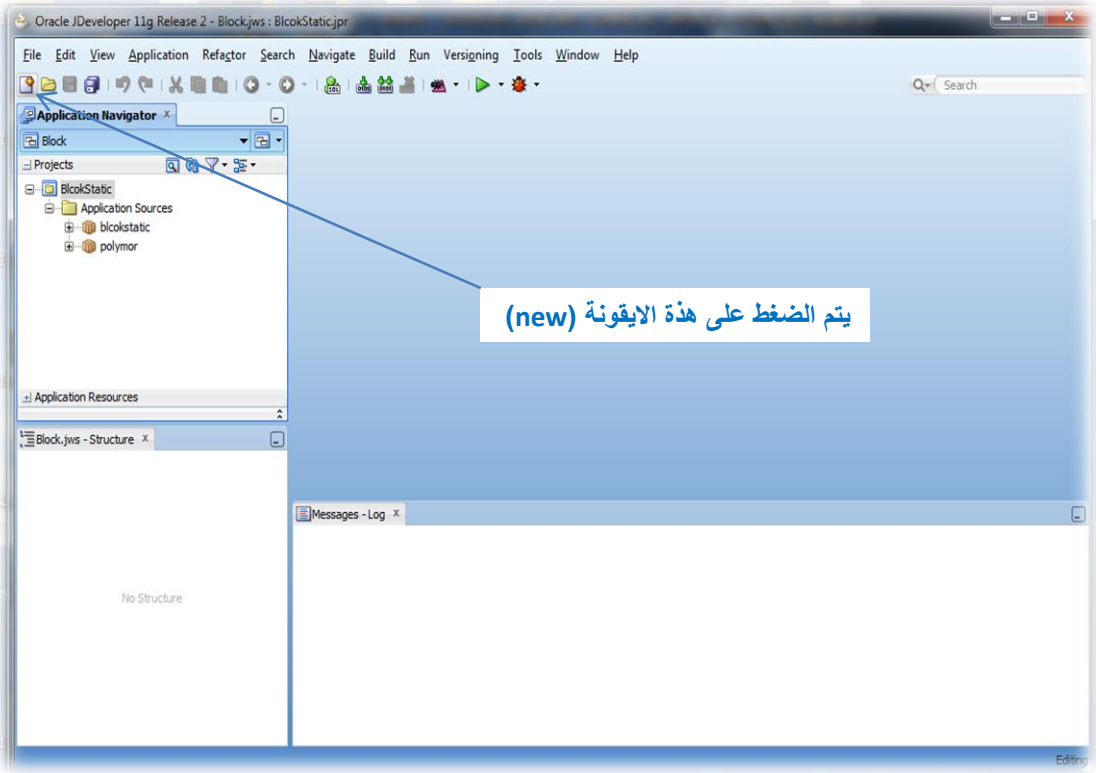
الصيغة العامة أو syntax هو نفسه..
لكن هنا نكتب interface وإسمها

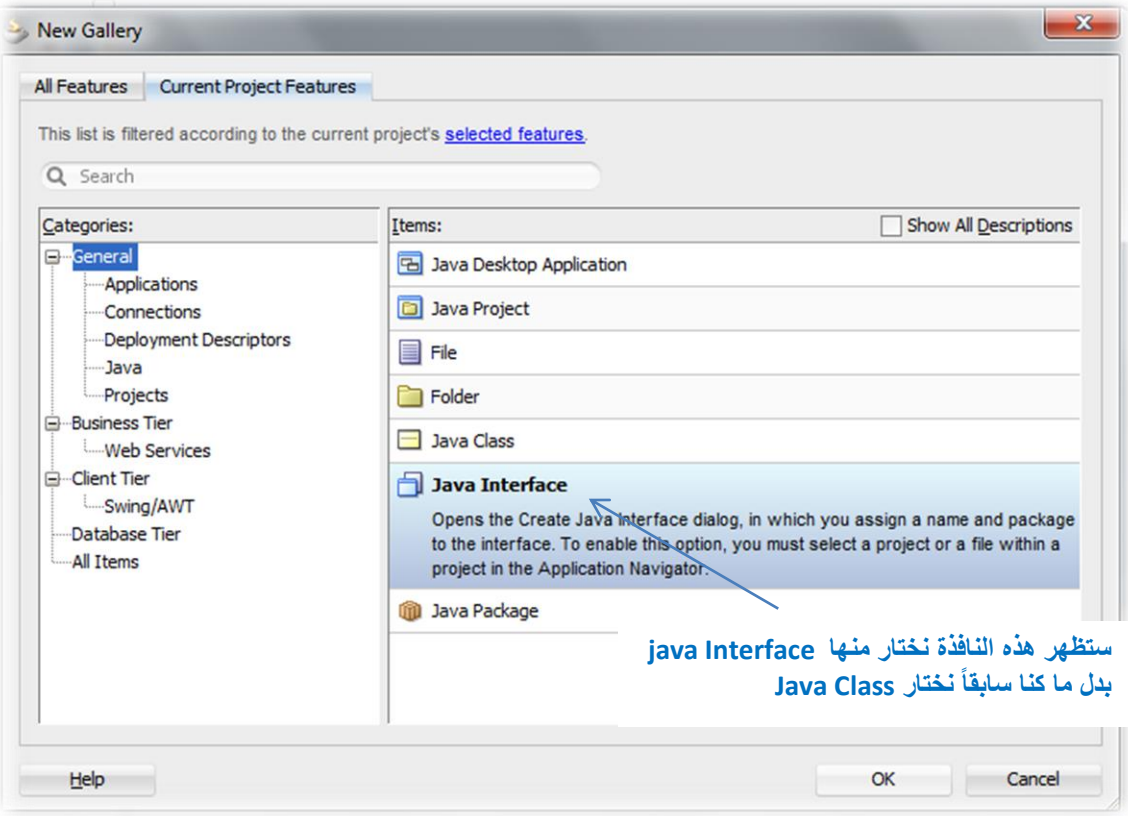
```
public interface Human
  {
      ........
      .............

    void eat();
    String str(String name, int id);
    void drink();


  }
```

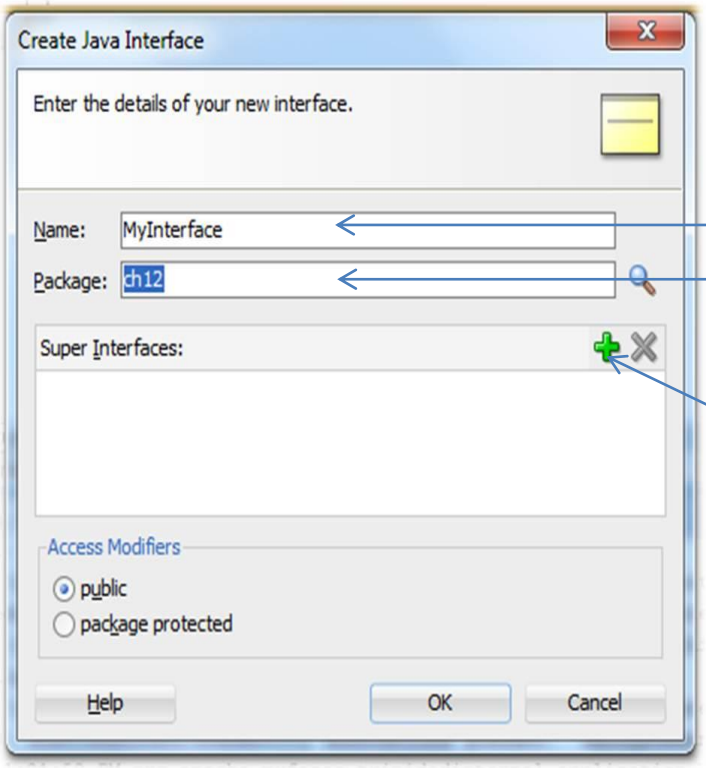
الـ code الذي داخل interface يمثل الأوامر والشروط والتي تمثل لنا method بدون (code) فقط نذكر أسماء الـ method وعدد الـ (method) غير محدود

## ...تعريف إنشاء الـ (interface):

لإنشاء وتعريف الـ (interface) في البرنامج نتبع التالي من الخطوات كما في الصور التالية:



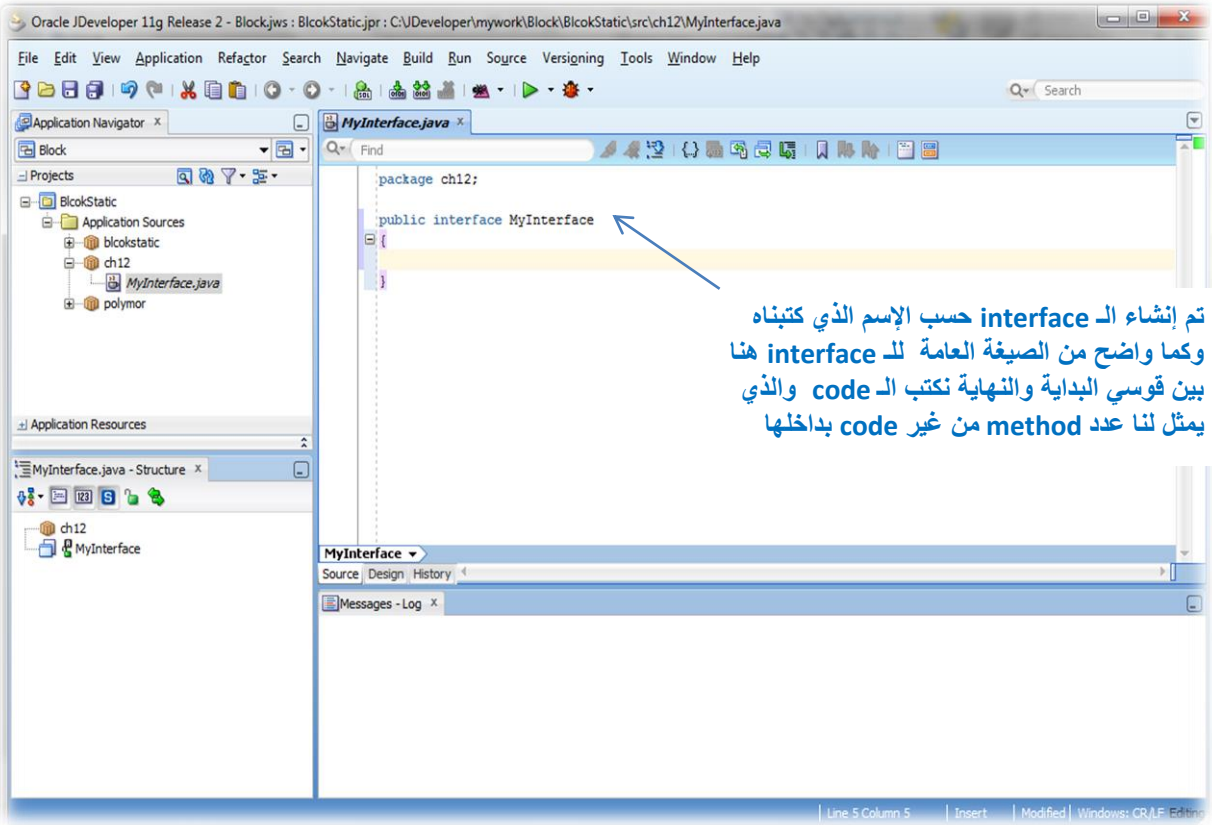



هنا نكتب إسم الـ interface الذي نرغب به

هنا نكتب إسم الـ package

من هنا نختار إسم الـ interface الذي نريد
ان نرث منها في حال طبقنا مبدأ الوراثة

بعد إعداد الخطوات ... نضغط على زر (ok) لتظهر النافذة التالية:




تم إنشاء الـ interface حسب الإسم الذي كتبناه
وكما واضح من الصيغة العامة للـ interface هنا
بين قوسي البداية والنهاية نكتب الـ code والذي
يمثل لنا عدد method من غير code بداخلها

## ...كيفية إستخدام الـ (interface) في الـ (class): *Implements of interface*

حسناً..أنشأنا الـ (interface) وكتبنا فيها الذي نريده بقي الآن معرفة كيفية تطبيق هذه الـ (interface) في الـ (class)

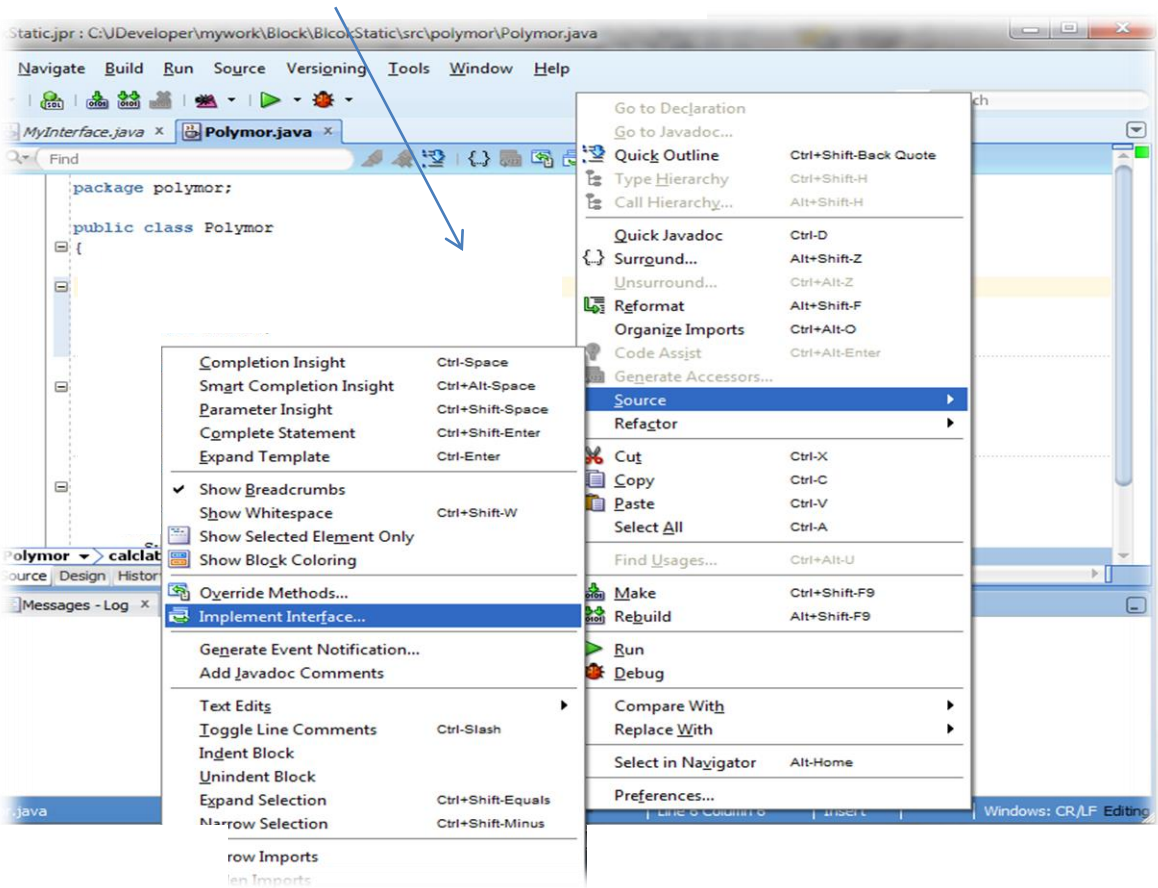
يتم تنفيذ أو تطبيق الـ (interface) داخل الـ (class) حسب الصيغة العامة التالية :

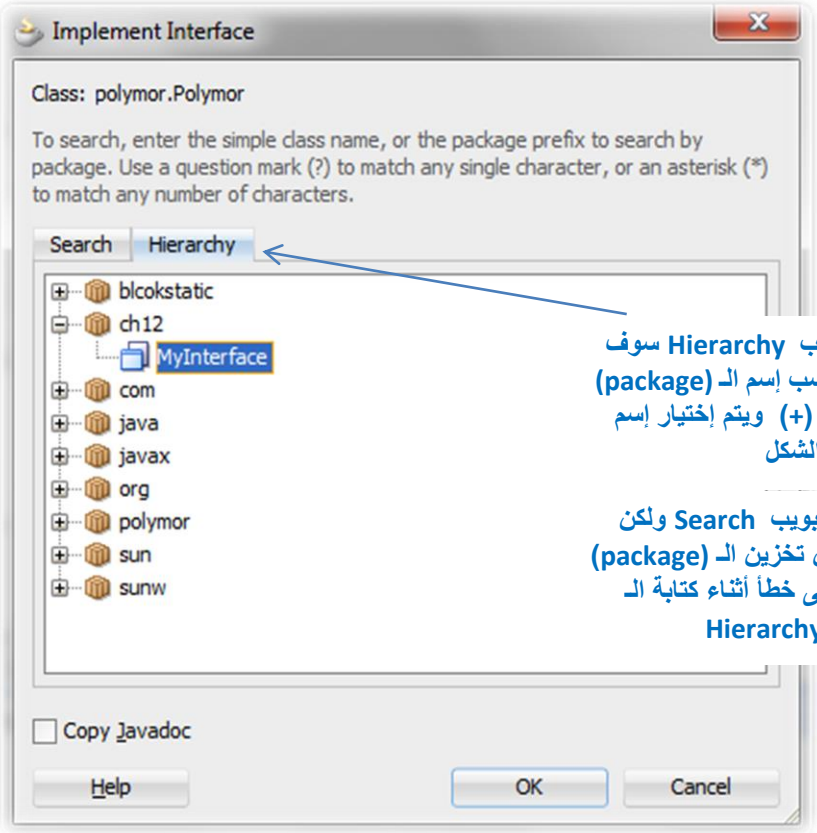
**الصيغة العامة لتطبيق interface داخل class يتم كالآتي:**
كلمة class يليه إسم الـ class ثم كلمة **implements** ثم إسم interface

```
public class classname implements interfacename
{
}
```

وكما موضحة بالخطوات التالية:

**الذهاب الى class المراد تنفيذ الـ (interface) فيها ثم نقف على المساحة الفارغة من نافذة التي نكتب بها الـ code و right click سوف تظهر قائمة نختار منه source ثم Implements Interface كما موضح في الشكل**

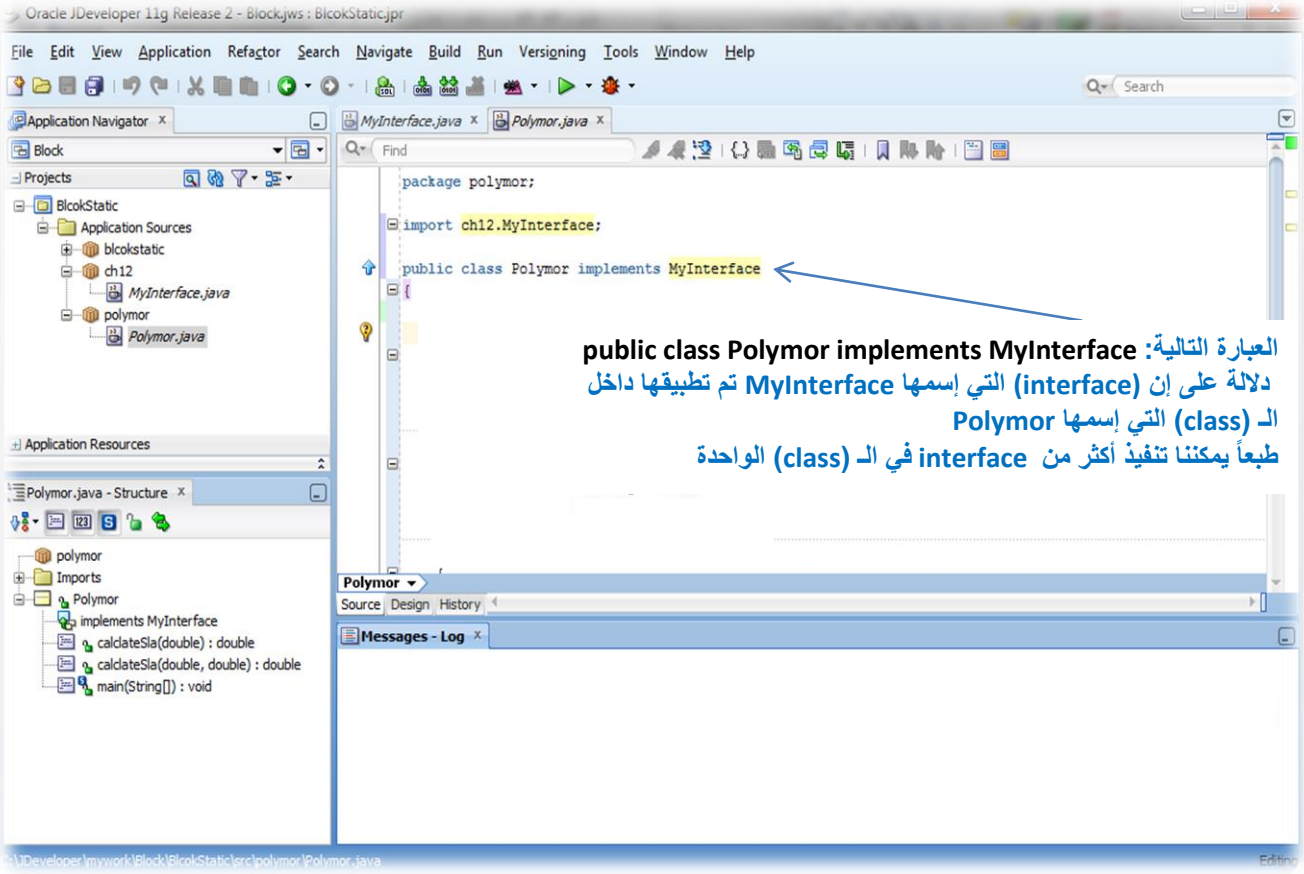





سوف تظهر النافذة التالية وبالضغط على تبويب Hierarchy سوف تظهر عدد من الـ (package) التي عندنا وحسب إسم الـ (package) التي فيها عملنا نتم فتحها بضغط على علامة (+) ويتم إختيار إسم الـ (interface) التي نريدها كما موضح في الشكل

طبعا يمكننا إختيار interface بالضغط على تبويب Search ولكن هنا يتطلب أن نكتب الـ (path) كامل عن مكان تخزين الـ (package) التي فيها الـ (interface) وهذا ربما يؤدي الى خطأ أثناء كتابة الـ (path) لذلك تجنباً لهذا كلها نلجأ الى تبويب Hierarchy

بعدما تم إعداد كل هذه الخطوات ثم نضغط على زر (ok) لتظهر النافذة التالية:




العبارة التالية: public class Polymor implements MyInterface
دلالة على إن (interface) التي إسمها MyInterface تم تطبيقها داخل الـ (class) التي إسمها Polymor
طبعاً يمكننا تنفيذ أكثر من interface في الـ (class) الواحدة

# ... الأمثلة لحالات مختلفة عن (interface)

## 1. One Interface One Class

المثال الأول نأخذ حالة (interface) واحدة و (class) واحدة

الـ interface وكإنما تمثل لنا القالب الذي يصف الخطوط العريضة لما يحتويه البرنامج من أسماء الـ method والمتغيرات الذي سوف يطبق في داخل الـ (class)

```
public interface Shape
  {
   public   String baseclass="shape";
   public void Draw();
  }
```

تم أخذ instance من الـ class كما ذكرنا

```
Circle cir1 = new Circle ();
```

كما يمكن أخذ instance من الـ interface لانه تم عمل لها implements by the class كما ذكرنا

```
Shape  cir1 = new Circle ();
```

```
package interfacetest;
```

الصيغة العامة لتطبيق interface داخل class

```
public class Circle implements Shape
  {
```

تم إستدعاء الـ (method) المشار لها في interface وعمل عليها (override) يجب لان كتبنا الـ code الذي داخلها وهذا يعرف بـ (override)

```
public void Draw()
  {
   System.out.println("Drawing Circle "+ baseclass );
  }
```

في class main تم إستدعاء الـ (method) بالشكل المعتاد عليه

```
public static void main(String[] args)
   {
  Circle cir1 = new Circle ();
  cir1.Draw();
   }

}
```

The out put

```
Drawing circle shape
```

## 2. Interface Inheritance an Interface

المثال الثاني نأخذ حالة (interface) ترث من (interface) أخرى

لدينا two interface واحدة تمثل لنا superinterface والاخرى subinterface لترث منها حسب مبدأ الوراثة

الـ interface الأولى

```
public interface Shape
  {
   public  String baseclass="shape";
   public void Draw();
  }
```

عملنا interface أخرى تحتوي على متغير وإسم method وجعلنها ترث من الـ (interface) الأولى حسب مبدأ الوراثة extends

```
public interface Shape2 extends Shape
  {
   public  String baseclass="shape1";
   public void Draw2();
  }
```

تم أخذ instance من الـ class كما ذكرنا

```
Circle cir1 =new Circle ();
```

كما **يمكن** أخذ instance من الـ **Shape2 interface** لإنه تم تطبيقها بالـ (class) أي implements by the class

**ولا يمكن** أخذ instance من **Shape interface** لإنه لم يتم تطبيقها بالـ (class) بمعنى إننا لم نعمل عليها implements بالـ (class)

```
Shape2 cir1 =new Circle ();
```

ولن يؤثر على output ويمكن تجربة ذلك

الـ object الذي يأخذ من interface يرى method لهذه الـ interface هذه فقط ... بمعنى التالي

```
Shap2 cir1 =new Circle ();
 cir1.Draw();
 cir1.Draw2();
```

```
Shape1 cir1 =new Circle ();
  cir1.Draw();
  cir1.Draw2();..........  method Draw2() لايرى
```

```
package interfacetest;
```

تم عمل تطبيق للـ **interface الوراثة** داخل class

```
public class Circle implements Shape2
 {
     public  String baseclass="My shape";
```

تم إستدعاء الـ (method) للـ superinterface (الأب)

```
public void Draw()
  {
   System.out.println("First Circle "+baseclass);
  }
```

تم إستدعاء الـ (method) للـ subinterface (الابن)

```
public void Draw2()
  {
  System.out.println("Second Circle:"+baseclass);
  }
```

في class main تم إستدعاء الـ (method) بالشكل المعتاد عليه

```
public static void main(String[] args)
  {
     Circle cir1 =new Circle ();
     cir1.Draw();
     cir1.Draw2();
  }
}
```

The output

```
First Circle: My shape
Second Circle: My shape
```

تم أخذ قيمة (baseclass) المعرفة بالـ (class) وليس المعرفة بالـ (interface)

# 3. Many Interface one Class

المثال الثالث نأخذ حالة (class) واحدة تطبق أكثر من (interface) أي (*implements more interface by one class*)

لدينا three interface (Shape, Shape2,Shape3)
وفيها (Shape2) extend من (Shape)

الـ interface الاولى

```
public interface Shape
  {
   public  String baseclass="shape";
   public void Draw();
  }
```

الـ (Shape2) extends من (Shape)

```
public interface Shape2 extends Shape
  {
   public  String baseclass="shape2";
   public void Draw2();
  }
```

الـ interface الثالثة

```
public interface Shape3
  {
   public  String baseclass="shape3";
   public void Draw3();
  }
```

تم أخذ instance من الـ class كما ذكرنا

```
Circle cir1 =new Circle ();
```

كما **يمكن** أخذ instance من كل الـ **interface** لإنه تم تطبيقها جميعاً بالـ (class) أي جميعها
implements by the class

```
package interfacetest;
```

تطبيق أكثر من **interface** داخل class يكون بهذه الصيغة

```
public class Circle implements Shape, Shape2,Shape3
 {
    public  String baseclass="shape2";
```

تم إستدعاء الـ (method) للـ superinterface (الأب)

```
public void Draw()
  {
System.out.println("First Circle:"+Shape1.baseclass);
  }
```

تم إستدعاء الـ (method) للـ subinterface (الإبن)

```
public void Draw2()
  {
System.out.println("Second Circle:"+Shape2.baseclass);
  }
```

تم إستدعاء الـ (method) للـ interface (آخرى)

```
public void Draw3()
  {
System.out.println("Third Circle:"+Shape3.baseclass);
  }
```

في class main تم إستدعاء الـ (method) كالمعتاد

```
   public static void main(String[] args)
    {
     Circle cir1 =new Circle ();
        cir1.Draw();
        cir1.Draw2();
        cir1.Draw3();
    }
 }
```

The out put
تم أخذ قيمة baseclass المعرفة بكل (interface) منهما

```
First Circle :shape1
Second Circle :shape2
Third Circle :shape3
```

## ...خلاصة القول

١- الـ (interface) ماهي الإ قالب يصف لنا (variables), (methods) التي سوف تنفذ من قبل الـ (class) وكإنه يعطينا الخطوط العريضة لما يحدث داخل البرنامج من أوامر وشروط التي تقع على عاتق الـ (class) تنفيذها.

٢- الـ (interface) تشبه الى حد ما الـ (class) من ناحية الـ (Syntax).

٣- في داخل الـ (interface) تكون الـ (variables) نوع (final) أي ثوابت .. والـ (methods) نوع (public abstract) حتى لو لم يتم الإعلان عنها بصريح العبارة ،، ونذكر دائماً على إن الـ (interface) لاتحتوي على (constructor) إطلاقاً..

٤- الـ (methods) التي في داخل (interface) تكون بدون (code) ..أي فقط إشارة لوجود (method) بمعنى تذكر بهكذا صيغة ← methodname();

٥- الـ (interface) لايمكن إنشاء منها (object) الإ إذا تم تطبيقها داخل الـ (class) وكل (object) يرى (methods, variables) الـ (interface) التي عُرّف منها.

٦- الـ (interface) تنفذ داخل (class) أي (implemented by class) لذلك يجب أن تحتوي الـ (class) على كلمة implements + interfacename ويجب على الـ (class) أيضاً أن تنفذ جميع الـ (methods) المشار إليها في (interface) مهما كان عدد هذه الـ (methods).. فإذا لم تنفذ ولو (method) واحدة سوف يظهر (compiler error).

٧- بإمكان الـ (class) تنفيذ أكثر من (interface).

٨- يمكن أن ترث (interface) أكثر من (interface) أخرى.

# Casting Data Type in Java

ما هو مفهوم أو معنى (casting) في لغة الجافا ..؟ ماهي الفائدة الأساسية منها ولماذا تستخدم ..؟

ما هي شروط إستخدامها..؟ سأوضح الإجابة على كل هذه الأسئلة بالتفصيل ...

حسناً... (casting) هي بمعنى (convert) التحويل.. وتستخدم في لغة جافا لتحويل أو لتغيير من نوع بيانات الى نوع أخر ... بمعنى التحويل أو التغيير في الـ (data type) لمتغيرٍ عُرف داخل البرنامج إلى (data type) آخر حسب الضرورة.

ويتم تطبيق مفهوم الـ ( casting) مع (data type) بنوعيها ونقصد نوعيها أي بمعنى مع الـ (primitive type) و (reference Type)...وتم إيضاح كلا النوعين في بداية الكتاب...

## ... ماهي الفائدة أساساً من الـ (casting) ..؟

في بعض الأحيان يتم تعريف متغير على إنه (int) ومع كتابة البرنامج وإجراء بعض العمليات الحسابية والرياضية على هذا المتغير نجد أنه من الإفضل أن يتغير نوعه إلى (float) مثلاً ... فهل في هذه الحالة يتوجب علي أن أبحث عن هذا المتغير أين في البرنامج ثم يتم تغيير نوع البيانات له وهكذا... وربما الأمر يتسع ليشمل أكثر من متغير فهل علي أن أبحث في كل مكان عن تعريف كل هذه المتغيرات وأبدأ بتغيير في نوع البيانات وربما يؤدي الأمر الى ظهور أخطاء ..

إذن لغة الجافا وفرت خاصية الـ (casting) ... التغيير لنوع البيانات في المكان الذي نرغب فيه داخل البرنامج كيف هذا.. ؟ حسناً ... لنوضح الصيغة العامة للـ (casting) ونأخذ بعض الحالات لذلك ...

## ماهي الصيغة العامة للـ (casting) ..؟

لتغيير نوع بيانات متغير إلى نوع آخر... كما موضح في أدناه ...

```
int y ;
……..
float s = y;
…..

(byte) w;

//Example
byte  w;

System.out.println( (int) w);

System.out.println((int)(99.9999));
        // Returns 99
```

**تم تعريف متغير إسمه y من نوع int**

**من خلال Assignment (الإحلال)**
**حتى نعمل لمتغير (casting) يتم تعريف متغير آخر من النوع الذي نريد التغير له ثم عمل مساواة لهما**

**يتم تحويله الى نوع آخر وهو byte وذلك من خلال وضع الـ data type الجديد بين قوسين قبل إسم المتغير** هذه الصيغة تستخدم داخل تنفيذ جملة معينة مثل جملة طباعة ...أو ناتج عملية رياضية

**casting** داخل جملة طباعة

**casting** ناتج عملية رياضية أو تغيير للتخلص من الكسور العشرية

## ... أنواع الـ (casting)

(casting) مع البيانات نوع (Primitive type) تنقسم إلى قسمين وهما:

### ١- (Implicit) *widening primitive conversions*

ماذا نقصد بهذا النوع من التحويل ...هو إتساع الحجم المحدد للمتغير ... حسناً مازال الأمر غامض ومُبهم ... حسناً إذا فرضنا لدينا متغير من نوع (int) ونحتاج إلى تغييره إلى نوع (long) بمعنى نريد أن نعمل له (casting) فهذا نوع يعرف بـ (widening primitive conversion) كيف هذا..؟ حسناً لأن الـ (int) هو (4 byte) ويراد تحويله إلى (long) وهو (8 byte) فهذا إتساع فأنا أعطيته مجال أكثر و وسعت الحجم المحدد له بالذاكرة ..إن شاء الله وضحت الفكرة ...

هذه كل الأنواع الممكنة من (**widening primitive conversions**).

- byte to → short, int, long, float, or double
- short to → int, long, float, or double
- char to → int, long, float, or double
- int to → long, float, or double
- long to → float or double
- float to → double

### ٢- (Explicit) *Narrowing Primitive Conversion*

ماذا نقصد بهذا النوع من التحويل ...هو تضييق الحجم المحدد للمتغير ...حسناً كما في أعلاه نوضحه لو فرضنا لدينا متغير من نوع (long) ونحتاج إلى تغييره إلى نوع (int) بمعنى نريد أن نعمل له (casting) فهذا نوع يعرف (narrowing primitive conversion) كيف هذا..؟ حسناً لأن الـ (long) هو (8 byte) ويراد تحويله إلى (int) وهو (4 byte) فهذا تضييق فأنا أعطيته مجال أقل و ضيّقت الحجم المحدد له بالذاكرة ..إن شاء الله وضحت الفكرة ...

**ويستبعد إستخدام هذا النوع من التحويل (casting) لإنه يُفقد المعلومات دقتها...**

هذه كل الأنواع الممكنة من (**narrowing primitive conversions**).

- short to → byte or char
- char to → byte or short
- int to → byte, short, or char
- long to → byte, short, char, or int
- float to → byte, short, char, int, or long
- double to → byte, short, char, int, long, or float

## ... حالات مختلفة لـمفهوم الـ (casting) في لغة الجافا ...

```
public class Class1
 {
   public static void main(String[] args)
    {
    int   x = 120;
```

تعريف متغير إسمه x من نوع int وإسناد له قيمة 120

```
    float y = x;
```

**التحويل من int الى float**
تعريف متغير أخر (y) وجعله يساوي المتغير القديم (x)
وهذا هو casting بطريق الإحلال assignment

```
    float f1 = 1.5f;
    float f2 = (float)1.5;
```

**ليتم تعريف متغير نوع float يكون بطريقتين هما:**
١- أما بكتابة حرف f جنب الرقم العشري
٢- أو بكتابة بين قوسين كلمة (float) جنب الرقم العشري حتى يفهم على أن تعريفه من نوع float ولا يعرف على أنه double والطريقة الثانية هي الأصح وهذا مايعرف بـ **casting** وهو تغيير data type لمتغير

```
    int n1 = 10;
    int n2 = 5;
    int n3 = 56, n4 = 24;
    double div =n1/(double) n2;
    System.out.println(div);
```

**Casting تغيير من data type الى آخر**

```
 }
```

### ... الـ (casting) مع البيانات نوع (Reference Data type)....

هنا نوضح الـ (casting) مع الـ ( Reference data type)، وكما تعرفنا أنه (Reference Data type) يمثل لنا بـ (class, array, interface) ... حتى نطبق هذا النوع من الـ (casting) يعتمد على توفر علاقة بين الـ (classes) نفسها أي وجود علاقة وراثة (inheritance) بمعنى يجب توفر (superclass) و (subclass) حتى نستطيع عمل (casting)... أو يجب ان تكون الـ (interface) تم تطبيقها داخل الـ (class) في حال يتطلب عمل (casting) بين الـ (interface) و (class) ...

وهنا الـ (casting) يكون على نوعين أيضاً (Upcasting) و(Downcasting):

### ١- (implicitly) *Upcasting or*

عندما نجعل (object) الـ (subclass) يكون أو يصبح (object) للـ (superclass) فهذا يسمى (upcasting) .... (upwards in the hierarchy from subclass to super class) أي (من الأسفل إلى الأعلى)

### ٢- (explicit) *Downcasting or*

عندما نجعل (object) الـ (superclass) يكون أو يصبح (object) للـ (subclass) فهذا يسمى (downcasting) .... (superclass to subclass) أي (من الأعلى الى الأسفل) وهذا غير مسموح بلغة الجافا لذلك يتطلب خطوة توضيح ولهذا يسمى بـ (**explicit**)

### ...الفكرة سوف تتضح مع هذا المثال:

في هذا المثال لدينا (Two classes) أحدهما يمثل لنا (super class) وهي (class Flower) والأخرى تمثل لنا (subclass) وهي (class Rose)..لإن شرط الـ (casting) مع (reference type) أن يكون هنالك علاقة بين الـ (classes) ...

```
public class Flower
 {
   public void smell()
     {
       System.out.println("All flowers give smell");
     }
 }
```

(class Flower) وتمثل لنا ( super class) هنا تحتوي على (method) هي (smell) تقوم بعملية طباعة

```
public class Rose extends Flower
 {
   public void smell2()
     {
       System.out.println("Rose gives rosy smell");
     }
```

(class Rose) وتمثل لنا (subclass) هنا تحتوي على (method) هي (smell2) تقوم بعملية طباعة

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

      Flower f = new Flower();

      Rose r = new Rose();

      f.smell();
      r.smell2();


      f = r;     // subclass to super class, it is valid
      f.smell();

      r = f;     // super class to subclass, not valid
      r = (Rose) f;     // explicit casting
      f.smell2();
   }

}
```

(class main)

تم تعريف object من (class Flower) وبنفس الوقت يشير لنفس الـ (class)

تم تعريف object من (class Rose) وبنفس الوقت يشير لنفس الـ (class)

تم إستدعاء كل (method) بكل (class) بإستخدام الـ (object) الخاص لكل (class)

أن نجعل (object) الـ (subclass) يشير الى مايشير اليه Object الـ (super class) فهذا ممكن .. ونستدعي من خلاله (method) الـ (super class)

أن نجعل (object) الـ (super class) يشير الى ما يشير إليه Object الـ (subclass) فهذا غير ممكن .. فيجب أن يسبقه توضيح حتى استطيع من خلاله أن أستدعي (method) الـ (subclass) من خلال (object) الـ (super class)

هذا هو casting

... حقيقة الأمر أن موضوع الـ (casting) كبير ومتشعب ولكن أنا ذكرت ما يواجه المبرمج أثناء العمل والأمور الضرورية ذكرتها ...

... الشروط أو القواعد التي يطبق عندها الـ (casting) ...

- *A value of any data type can be cast to its own type.*
  - يمكن تحويل أي نوع بيانات الى نوع بيانات أخر تابع له .

- *A value of any arithmetic data type can be cast to any other arithmetic data type. Casting a floating-point value to an integer data type rounds toward zero.*
  - يمكن تحويل نوع بيانات لناتج أي عملية رياضية الى نوع بيانات آخر كما عند تحويل من الـ (flaot) الى (int) للتقريب أو للتخلص من الكسور العشرية والاصفار.

- *A value of the boolean data type cannot be cast to any other data type, nor can a value of any other data type be cast to boolean.*
  - لايمكن تحويل نوع البيانات (boolaen) الى أي نوع بيانات آخر...أي البيانات التي من (boolean) لايمكن عمل لها (cast).

- *A value of any primitive data type cannot be cast to a reference data type, nor can a reference be cast to any primitive data type.*
  - أي قيمة من أي نوع من أنواع (primitive data type) لايمكن عمل لها (casting) وتحويلها إلى نوع (reference data type) والعكس صحيح ..أي لايمكن أيضاً تحويل نوع (reference data type) وتحويلها الى نوع (primitive data type).

- *A reference to a class type can be cast to the type of the superclass of that class.*
  - يمكن عمل (cast) الى (object) خاص بـ (super class).

- *A reference to a class type can be cast to an interface type if the reference* actually refers to an object of a class that implements the specified interface.
  - يمكن عمل (cast) بين (object) للـ (interface) والـ (class) إذا كانت هذه الـ (interfac) طبقت داخل هذه الـ (calss).

- *you cannot use String as a cast type for a primitive type*
  *String s = (String)x is invalid*
  *you can use → String s = new Byte(x).toString();*

- لايمكن عمل (cast) والتغيير إلى (String)... أي لايمكن تغيير أي نوع من أنوع الـ (primitive type) إلى String لايمكن هذا ... ويتم ذلك بإتباع الصيغة هذه ...كما موضحة الصيغة في أعلاه.

# Exception

ماهو الـ (Exception) ..؟ ماهي أنواع الـ (Exception) ..؟ وكيف يتم معالجتها ..؟ سأوضح الإجابة على كل هذه الأسئلة ... لكن لنأخذ هذه المقدمة حتى تتضح الأمور لدينا أكثر ..

عند عمل برنامج (تطبيق) لابد من المرور بثلاث مراحل .. بأي لغة برمجة كانت وأي برنامج كان لابد من المرور بهذه المراحل..حسناً ..ماهي هذه المراحل الثلاث:

١- Edit Coding كتابة الإيعازات البرمجية.
٢- Compiler عمل compiler لهذه الإيعازات.
٣- Execution عمل التنفيذ (run).

لا يَخلى أي برنامج ... وبأي لغة كان المكتوب بها من عمل هذه الخطوات التي تم ذكرها ... حسناً قد نواجه بعض الأخطاء في أحد هذه المراحل .. وغالباً ما نكتشف الأخطاء عند عمل (Compiler) للبرنامج فتكون هذه الاخطاء قواعدياً (Syntax error) نتيجة عدم إتباع كتابة الإيعاز قواعدياً .. بمعنى لم نتبع القاعدة والوصف الصحيح أثناء كتابة الـ (code) الإيعاز ..أكيد كل هذه الأخطاء مهما كانت طفيفة يتم التنويه عنها والإستدلال عليها برسائل تخبرنا عن وجود خطأ ويتم تصحيحها ليتم تنفيذ البرنامج ...

هنا تم التنويه عنها لإنها حدثت أثناء عمل Compilation للبرنامج بمعنى حدثت عند الـ ( Compiler Time) لكن هنالك أخطاء تحدث أثناء تنفيذ البرنامج (Run Time) مما يتسبب في إيقاف البرنامج عن التنفيذ وبالتالي التوقف عن العمل نتيجة لحدوث خطأ غير متوقع ..وهذا هو محور حديثنا هنا ..**الأخطاء الغير متوقعة الحدوث والتي تحدث إثناء الـ (Run Time) وهو مايعرف بـ (Exception) (إستثناء)**

كيف يتم معالجة أو السيطرة على هذه الاخطاء الغير متوقعة والتي تحدث أثناء تنفيذ البرنامج (Run Time) والتي تعتبر أخطاء خطيرة لانها تتسبب في إيقاف العمل للتطبيق الذي نعمل عليه ..وهذا مايعرف بـ (Exception)

إذن نستخلص مما تم ذكره في أعلاه ..

أن الـ **(Exception)** هو حدوث أو وقوع أخطاء غير متوقعة وقت التنفيذ (Run Time) إثناء تنفيذ البرنامج أو بمعنى أخر أثناء عمل التطبيق ..مما يتسبب في وقوف التطبيق عن العمل.

... ما هي أنواع الـ (Exception) .. بمعنى آخر ماهي أنواع هذه الأخطاء الغير متوقعة الحدوث

هنالك نوعين من الإستثناءات:

١- (Unchecked Exceptions) التي لانستدل عليها عند عمل compiler.

٢- (Checked Exceptions) التي نستدل عليها عند عمل compiler.

... وفي أدناه جدول لإنواع كل منها بشكل أكثر تفصيلاً:

- **Unchecked Exceptions**

| Exception | Description |
|---|---|
| ArithmeticException | Arithmetic error, such as divide-by-zero.<br>خطأ في العمليت الرياضية مثل عملية القسمة على صفر . |
| ArrayIndexOutOfBoundsException | Array index is out-of-bounds.<br>الـ (index) يشير إلى خارج النطاق المحدد للمصفوفة (أكبر أو أقل من طول المصفوفة). |
| ArrayStoreException | Assignment to an array element of an incompatible type.<br>إسناد قيمة لإحد عناصر المصفوفة بنوع (Data type) مغاير أو مختلف عن نوع الـ (Data type) للمصفوفة. |
| ClassCastException | Invalid cast.<br>عملية الـ (casting) غير مطابقة للشروط . |
| IllegalArgumentException | Illegal argument used to invoke a method.<br>يحدث هذا الإستثناء من إستدعاء الـ ( method) والخطأ الحاصل بتمرير (parameters) لها. |
| IllegalMonitorStateException | Illegal monitor operation, such as waiting on an unlocked thread. |
| IllegalStateException | Environment or application is in incorrect state.<br>حدوث خطأ في بيئة التطبيق . |
| IllegalThreadStateException | Requested operation not compatible with current thread state.<br>العملية المطلوب إجرائها غير متوافقة مع الحالة الحالية . |
| IndexOutOfBoundsException | Some type of index is out-of-bounds.<br>الإشارة إلى بعض من الـ (index) التي تكون خارج النطاق المحدد للمصفوفة. |
| NegativeArraySizeException | Array created with a negative size.<br>خلق مصفوفة مع تعين نطاق الحجم لها قيمة بالسالب وهذا لايمكن ..أن يكون طول المصوفة أو عدد عناصر المصفوفة بالسالب. |
| NullPointerException | Invalid use of a null reference.<br>يحدث هذا الإستثناء عند الإشارة الى قيمة فارغة (null) خاصة مع linked list. |
| NumberFormatException | Invalid numeric format.<br>يحدث هذا الإستثناء من خلال خطأ بالصيغة المعطاة للرقم مثلاً 13rr هنا خطأ الـ (rr) هنا لا تمثل لنا صيغة رقمية. |
| SecurityException | Attempt to violate security.<br>محاولة لإنتهاك أو إختراق الأمن. |
| StringIndexOutOfBounds | Attempt to index outside the bounds of a string.<br>الإشارة الى الـ (index) خارج النطاق المحدد لسلسلة نصية (string ). |
| UnsupportedOperationException | An unsupported operation was encountered. |
| Exception (others) | The most general type of exceptions of them all عام يشمل كل الإستثناءات الغير متوقعة الحدوث. |

## Checked Exceptions

| Exception | Description |
|---|---|
| ClassNotFoundException | Class not found. Class غير موجود. |
| CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneable interface. محاولة لعمل نسخة ثانية من object لم يتم ذكره أو تعريفه بالـ interface |
| IllegalAccessException | Access to a class is denied. لايمكن الوصول إلى Class . |
| InstantiationException | Attempt to create an object of an abstract class or interface. محاولة إنشاء object من interface أو إنشاء object من abstract class وهذا خطأ. |
| InterruptedException | One thread has been interrupted by another thread. |
| NoSuchFieldException | A requested field does not exist. طلب حقل غير موجود أصلاً |
| NoSuchMethodException | A requested method does not exist. طلب method غير موجودة أصلاً |

حسناً ..تعرفنا على الإستثناءات (Exceptions) وماهي أنواعها تبقى أن نتعرف كيف يتم معالجتها إثناء حدوثها داخل البرنامج حتى يتم السيطرة على الخطأ أثناء حدوثه في وقت تنفيذ البرنامج.

قبل معرفة كيفية معالجة هذه الـ (Exception) يتوجب علينا معرفة أمرين مهمين جداً حتى تتم معالجة الـ (Exceptions) بصورة صحيحة... حسناً ما هذين الأمرين؟ ....هي الآتي:

١- السطر أو (الأسطر) الذي سبب المشكلة أو الذي تسبب بظهور الـ (Exceptions).
٢- السطر أو (الأسطر) الذي نريد أن يتم تنفيذه .. وبسبب حدوث الـ (Exceptions) لم ينفذ.

أما كيفية معالجة هذه الإستثناءات فيتم من خلال (**try….catch**) لنتعرف على الصيغة العامة للـ try….catch

## ... الصيغة العامة لـ (try....catch)

هذه الصيغة الكاملة والعامة لكتابة ومعالجة الإستثناءات والمتمثلة بالـ try... catch

```
try
  {
    statements that can throw exceptions
  }
catch (exception-type identifier)
  {
    statements executed when exception is thrown
  }
finally
  {
    statements that are executed whether or not
    exceptions occur
  }
```

نكتب كلمة try بعدها قوس البداية { وقوس النهاية } وبداخلهما نكتب السطر أو الاسطر التي تسبب الخطأ

نكتب catch (نوع الـ exception الذي سبب الخطأ) يليها قوس البداية { وقوس النهاية } وبداخلها الجملة أو الجمل التي نريده أن تنفذ حتى لو حدث الـ exception

finally يليها قوس البداية { وقوس النهاية } وبداخلها الجملة أو الجمل التي نريده أن تنفذ في حال حدث أو لم يحدث الـ exception

... الفكرة سوف تتضح أكثر مع الأمثلة التالية ..

... المثال التالي نوضح فيه النوع الأول من الإستثناءات هو... (ArithmeticException)

```
1  package ch01;
2
3  public class Class1
4
5  {
6      public static void main(String[] args)
7      {
8          int b = 0;
9          int a = 10;
10
11         int c = a / b;
12
13         System.out.println("Can't do that Error!");
14     }
15 }
```

Class1 ▸ main(String[])

Source Design History

Running: Lang.jpr - Log

```
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\J
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ch01.Class1.main(Class1.java:10)
Process exited with exit code 1.
```

Messages | Compiler | Running: Lang.jpr

Class java عادي تم تعريف فيه متغيرين هما b و a و إسناد قيمة لكل منهما حيث قيمة b=0 و قيمة a=10

السطر الذي يسبب المشكلة عند التنفيذ لانه لايمكن القسمة على صفر

السطر الذي لا يظهر عند التنفيذ بسبب حدوث المشكلة

الرسالة التي ظهرت إثناء التنفيذ ..وأنظر جيداً كيف وضحت نوع الـ Exception وهو ArithmaticException

... البرنامج بعد معالجة الـ Exception بالـ try..catch لنلاحظ الخطوات كيف تمت..

```
1  public class Class1
2  {
3      public static void main(String[] args)
4      {
5          int b = 0;
6          int a = 10;
7
8          try
9          {
10             int c = a / b;
11         }
12
13         catch (ArithmeticException e)
14         {
15             System.out.println("Can't do that Error!");
16         }
17
18     }
19 }
```

نكتب الـ try ..ثم نطوق السطر أو الاسطر التي سبب المشكلة بقوس البداية وقوس النهاية كما موضح

(نذكر نوع خطأ الذي ظهر لنا ) Catch وهو الخطأ ArithmaticException ثم نطوق السطر أو الاسطر التي نريدها تظهر إثناء التنفيذ كما موضح

```
Class1
Source  Design  History
Running: Lang.jpr - Log
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\JDeveloper\mywo
Can't do that Error!
Process exited with exit code 0.
Messages  Compiler  Running: Lang.jpr
```

نافذة التنفيذ ونلاحظ تنفيذ جملة الطباعة

...المثال التالي نوضح فيه نوع آخر من الإستثناءات هو (ArrayIndexOutOfBoundsException)

```
1  package ch01;
2
3  public class Class1
4  {
5
6      public static void main(String[] args)
7      {
8        System.out.println("Welcom We Will Print Array Value");
9        int arr[] = {'0','1','2'};
10
11       System.out.print("the value of array is "+ " "+ arr[4]);
12       System.out.println("Printing Array Value is done");
13     }
14 }
15
```

Class1 ▾
Source Design History

Running: Lang.jpr - Log

```
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\JDeveloper\mywork\App4\.adf;C:
Welcom We Will Print Array Value
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at ch01.Class1.main(Class1.java:11)
Process exited with exit code 1.
```

Messages Compiler Running: Lang.jpr

Class java عادي تم تعريف فيه مصفوفة arr وإسناد قيم .. وكما واضح إن المصفوفة تتكون من ثلاثة عناصر بمعنى أنه حجم المصفوفة هو ٣

السطر الذي يسبب المشكلة عند التنفيذ لانه طباعة قيمة خارج نطاق حجم المصفوفة المحدد لها بالبرنامج

السطر الذي لا يظهر عند التنفيذ بسبب حدوث المشكلة

الرسالة التي ظهرت إثناء التنفيذ ..وأنظر جيداً كيف وضحت نوع الـ Exception وهو ArrayIndexOutOfBoundsException

... البرنامج بعد معالجة الـ Exception بالـ try..catch لنلاحظ الخطوات كيف تمت..

```
3  public class Class1
4  {
5      public static void main(String[] args)
6      {
7          System.out.println("Welcom We Will Print Array Value");
8          int arr[] = {'0','1','2'};
9
10         try
11         {
12             System.out.print("the value of array is " + " " + arr[4]);
13         }
14
15         catch (ArrayIndexOutOfBoundsException e)
16         {
17             System.out.println("Printing Array Value is done");
18         }
19     }
20 }
```

Class1 ▸ main(String[])
Source Design History

Running: Lang.jpr - Log

```
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\JD
Welcom We Will Print Array Value
Printing Array Value is done
Process exited with exit code 0.
```

Messages | Compiler | Running: Lang.jpr

نكتب الـ try ..ثم نطوق السطر أو الاسطر التي سبب المشكلة بقوس البداية وقوس النهاية كما موضح

نكتب (نذكر نوع خطأ الذي ظهر لنا ) Catch وهو ArrayIndexOutOfBoundsException الخطأ ثم نطوق السطر أو الاسطر التي نريدها تظهر إثناء التنفيذ كما موضح

نافذة التنفيذ ونلاحظ تنفيذ جملة الطباعة

إن شاء الله تكون فكرة الـ (Exception) توضحت مع الأمثلة .. مع كيفية معالجتها داخل البرنامج ...حتى نتجنب توقف البرنامج إثناء التنفيذ.

الآن لو شاهدنا الحل وطريقة المعالجة نلاحظ هنالك سؤالين لابد من الإجابة عنهما وهما ..

**السؤال الأول:** هل نكتب كل أنواع الإستثناءات (Exception) التي تم ذكرها حتى أتجنب توقف البرنامج إثناء التنفيذ ؟؟

**السؤال الثاني:** أنا فقدت المعلومات التي تظهر تفاصيل الخطأ مثل بأي سطر حدث الخطأ وماسبب الخطأ .. فكيف إظهرها؟؟

الجواب على السؤال الأول... كلا بالتأكيد .. هنالك **إستثناء عام** يشمل **كافة الإستثناءات** نستخدمه لتغطية كل الاستثناءات الأخرى نذكر مثلاً لا للحصر (إنقطاع الاتصال بالشبكة إثناء العمل ، أو عند الكتابة على drive CD أو أي خطأ غير متوقع ) لذلك نستخدم (Exception) وهو **عام** وقد تم ذكره مع أنو اع الإستثناءات.

الجواب على السؤال الثاني ...نحن لم نفقد شيء نستطيع إظهار رسالة الخطأ التي كانت تظهر من خلال إستخدام الدالة printStackTrace() ونستطيع أن نظهر مسبب المشكلة من خلال الدالة getMessage()
وسوف نلاحظ هذا في المثال التالي :

... المثال التالي يوضح إستخدام الـ Exception عام و الـ try..catch لنلاحظ الخطوات كيف تمت

```
3   public class Class1
4   {
5       public static void main(String[] args)
6       {
7         System.out.println("Welcom We Will Print Array Value");
8         int arr[] = {'0','1','2'};
9
10        try
11          {
12            System.out.print("the value of array is " + " " + arr[4]);
13          }
14
15        catch (Exception e)
16          {
17           System.out.println("Printing Array Value is done");
18              System.out.println(e.getMessage());
19              e.printStackTrace();
20          }
21      }
22  }
23
```

Source Design History

نكتب الـ try .. ثم نطوق السطر أو الاسطر التي سبب المشكلة بقوس البداية وقوس النهاية كما موضح

نكتب Catch (Exception e) وهو الاستثناء العام Exception

لطباعة مسبب الخطأ

لطباعة تفاصيل الخطأ

Running: Lang.jpr - Log

```
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\JDeveloper
Welcom We Will Print Array Value
Printing Array Value is done
4
java.lang.ArrayIndexOutOfBoundsException: 4
        at ch01.Class1.main(Class1.java:12)
Process exited with exit code 0.
```

Messages Compiler Running: Lang.jpr

تم التنفيذ وطباعة الجملتين

ونلاحظ ظهور تفاصيل الخطأ إسمه ونوعه

...لقد تم ذكر الصيغة العامة لـ (try....catch) والتي كان من ضمن الصيغة الـ (finally) إذن ماهي هذه الـ (finally) وما الفائدة من إستخدامها..؟

سبق وإن ذكرنا الصيغة أو الكيفية التي تكتب بها الـ (finally) فهي دائماً تستخدم مع الـ (try..catch) أما الفائدة من إستخدامها .. فهي ذات أهمية كبيرة لإن الإيعازات المكتوبة ضمنها تنفذ دائماً سواء حدث (Exception) أم لم يحدث .. بالإضافة على أن الايعازات المكتوبة ضمنها تنفذ مباشرة بعد تنفيذ الإيعازات المكتوبة ضمن الـ (try..catch)، إن للـ (finally) أهمية كبيرة كما نوهنا عن ذلك فهي تعمل على التأكد أنه لم يتبقى أي إيعاز معلق لم يتم تنفيذه وكإنها تعمل على تنظيف أو إخلاء لكل الإيعازات وتضمن أنه لم يبقى أي متعلق لأي أيعاز.

لنأخذ المثال التالي ..حتى تتضح لنا الفكرة أكثر ... إن شاء الله

```
1  package ch10;
2  public class MyfinallyClass
3  {
4      public static void main(String[] arg)
5      {
6          try
7          {
8              int i = 10 / 0;
9          }
10         catch (Exception ex)
11         {
12             System.out.println("Inside 1st catch Block");
13         }
14         finally
15         {
16             System.out.println("Inside 1st finally block");
17         }
18
19         try
20         {
21             int i = 10 / 10;
22         }
23         catch (Exception ex)
24         {
25             System.out.println("Inside 2nd catch Block");
26         }
27         finally
28         {
29             System.out.println("Inside 2nd finally block");
30         }
31     }
32 }
```

نكتب الـ try ..ثم نطوق السطر أو الاسطر التي سبب المشكلة بقوس البداية وقوس النهاية كما موضح هنا السطر هو القسمة على صفر الذي سبب المشكلة

نكتب Catch (Exception ex) وهو الاستثناء العام Exception

Finally ومن بعدها طباعة جملة التي نريدها أن تنفذ سواء حدث exception أم لم يحدث

تم ذكر try..catch مرة أخرى وطباعة جملة اخرى حتى نميز نحن نقف عند ال try ...catch

finally تابعة للـ try..catch الثانية ومن بعدها طباعة جملة التي نريدها أن تنفذ سواء حدث exception أم لم يحدث

```
MyfinallyClass > main(String[]) > catch (Exception ex)
Source  Design  History
Running: Lang.jpr - Log
F:\Oracle\Middleware\jdk160_24\bin\javaw.exe -client -classpath C:\JDeveloper\mywork\App4\.ac
Inside 1st catch Block
Inside 1st finally block
Inside 2nd finally block
Process exited with exit code 0.
Messages  Compiler  Running: Lang.jpr
```

لنلاحظ الـ output بشكل دقيق نجد أنه تم تنفيذ جملة try..catch الاولى وبالتأكيد تم تنفيذ الجملة التابعة للـ finally ومن ثم القفز لتنفيذ جملة الـ finally الثانية

## ...خلاصة القول

١- الإستثناءات أو بما يعرف بـ (Exceptions) هو حدوث أخطاء غير متوقعة إثناء وقت التنفيذ.. أي عند الـ (Run Time) مما يتسبب في توقف البرنامج عن العمل ...ولمعالجة هذه الإستثناءات يتم من خلال الـ (try..cath..finally).

٢- للـ (finally) أهمية كبيرة لضمان تنفيذ إيعازات ضرورية سواء حدثت إستثناءات (Exceptions) في البرنامج أم لم تحدث على سبيل المثال هنالك إيعاز غلق الإتصال مع قاعدة البيانات فمثل هذا الإيعاز يتوجب ضرورة ملحة لتنفيذه ..لإن من الواجب أن تغلق قاعدة البيانات بعد فتحها لذلك دائماً ما يكتب إيعاز غلق الإتصال بقاعدة البيانات ضمن الـ (finally) حتى نضمن تنفيذه ..لإن كما ذكرنا الإيعازات المكتوبة ضمن الـ (finally) من مؤكد تنفيذها وهذه هي الفائدة من الـ (finally block).

٣- نستطيع أن نكتب أكثر من (try..catch) داخل البرنامج بما يتطلبه العمل لدينا.

# Java Basic Operators

لغة الجافا توفر لنا مجموعة كبيرة من المعاملات للتعامل مع المتغيرات ..وهذه المعاملات يمكن تقسيمها إلى ستة مجاميع وهي كالآتي:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

... سنوضح كل نوع من هذه الأنواع . .

- Arithmetic Operation : تستخدم في العمليات الحسابية

والجدول التالي يوضح كل نوع من أنوع (Arithmetic Operation) مع أخذ على سبيل المثال متغيرين أحدهما A= 10 والمتغير B=40 ولنلاحظ التعابير الرياضية مع هذين المتغيرين :

| Operator | Description | | Example |
|---|---|---|---|
| + | Addition | عملية الجمع بين المتغيرات | A + B will give 50 |
| - | Subtraction | عملية الطرح بين المتغيرات | A - B will give -30 |
| * | Multiplication | عملية الضرب بين المتغيرات | A * B will give 400 |
| / | Division | عملية القسمة بين المتغيرات | B / A will give 4 |
| % | Modulus | ناتج باقي القسمة | B % A will give 0 |
| ++ | Increment | مقدار زيادة قيمة المتغير بواحد (١) | B++ gives 41 |
| -- | Decrement | مقدار نقصان قيمة المتغير بواحد (١) | B-- gives 39 |

- Relational Operation : تستخدم لمعرفة العلاقة بين المتغيرين ..كيف هذا..؟ بمعنى معرفة علاقة المتغير A مع المتغير B هل هو أكبر من أو أقل من أو مساواة ..بمعنى آخر عمليات المقارنة بين المتغيرات

والجدول التالي يوضح كل نوع من أنوع (Relational Operation) مع أخذ على سبيل المثال متغيرين أحدهما A= 10 والمتغير B=20 ولنلاحظ التعابير العلائقية مع هذين المتغيرين :

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. التحقق من قيم المتغيرين هل هما متساويين | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. التحقق من قيم المتغيرين بعدم المساواة | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. التحقق من قيمة المتغير بالطرف الايسرلـ (operand) أكبر من قيمة المتغير الذي يقع يمين الـ (operand) | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. التحقق من قيمة المتغير بالطرف الإيسرلـ (operand) أصغر من قيمة المتغير الذي يقع يمين الـ (operand) | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. التحقق من قيمة المتغير بالطرف الإيسرلـ (operand) أكبر أو يساوي من قيمة المتغير الذي يقع يمين الـ (operand) | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. التحقق من قيمة المتغير بالطرف الإيسرلـ (operand) أصغر أو يساوي من قيمة المتغير الذي يقع يمين الـ (operand) | (A <= B) is true. |

- Logical Operation : تستخدم لمعرفة العلاقة المنطقية بين المتغيرين بشرط أن يكون نوع البيانات لكلا المتغيرين هو boolean..

والجدول التالي يوضح كل نوع من أنوع (Logical Operation) مع أخذ على سبيل المثال متغيرين أحدهما A= true والمتغير B=false ولنلاحظ التعابير االمنطقية مع هذين المتغيرين :

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. أذا كلا المتغيرين ليس قيمتهما (صفر) إذن النتيجة تعطيني صح (true) | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. أذا أحد المتغيرين ليس قيمته (صفر) إذن النتيجة تعطيني صح (true) | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

لنلاحظ المثال التالي لتتوضح الفكرة أكثر

```
public class Test

{

  public static void main(String args[])
  {
     boolean a = true;
     boolean b = false;

     System.out.println("a && b = " + (a&&b));

     System.out.println("a || b = " + (a||b) );

     System.out.println("!(a && b) = " + !(a && b));
  }
}
```

The out put is :

```
a && b = false
a || b = true
!(a && b) = true
```

- Bitwise Operation : توفر لنا لغة جافا مجموعة من المعاملات التي يتم تطبيقها على نوع بيانات (integral type) وهذا نوع من البيانات يشتمل كما بينا (long, int, short, char, and byte) على أن يكون قيمة المتغير بصيغة الـ (bits) لذلك سميت هذه العملية بالـ (Bitwise Operation)

والجدول التالي يوضح كل نوع من أنوع (Bitwisel Operation) مع أخذ على سبيل المثال متغيرين أحدهما A= 60 والمتغير B=13 وكما ذكرنا يجب أن تكون قيمة هذين المتغيرين بصيغة الـ (bits) إذن ستكون قيمهما كالاتي:

A = 0011 1100

B = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61 which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 1111 |
| >>> | Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. | A >>>2 will give 15 which is 0000 1111 |

- Assignment Operation : الإحلال ..إعطاء أو منح قيم الموجودة في جهة اليسار للـ (operand) تحل محل أو تمنح لجهة اليمين للـ (operand)

والجدول التالي يوضح كل نوع من أنوع (Assignment Operation) مع أخذ على سبيل المثال متغيرين A, B ولنلاحظ التعابير الإحلال للمتغير C مع هذين المتغيرين :

| perator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

- Misc Operation : وهو قليل الإستخدام ولكن ليس هنالك ضير من التطرق له ..وهو تعبير يستخدم بهذه الصيغة **Conditional Operator ( ? : )**

حيث أن العملية تتالف من ثلاث معاملات وهي: التعبير أو الشرط المنطقي الذي نتحقق منه ، القيمة المسترجعة أذا تحقق الشرط ، القيمة المسترجعة في حالة لم يتحقق الشرط ..بالضبط كإنها (if…else) وهذه صيغة كتابتها بالشكل التالي:

variable x = **(expression) ? value if true : value if false**

لنتابع المثال التالي:

```
public class Test
{
   public static void main(String args[])
  {
      int a , b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

**The output is:**

```
Value of b is : 30
Value of b is : 20
```

## ... جدول (keywords) الكلمات المحجوزة في لغة جافا

| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | Switch |
| assert*** | default | goto* | package | Synchronized |
| boolean | do | if | private | This |
| break | double | implements | protected | Throw |
| byte | else | import | public | Throws |
| case | enum**** | instanceof | return | Transient |
| catch | extends | int | short | Try |
| char | final | interface | static | Void |
| class | finally | long | strictfp** | Volatile |
| const* | float | native | super | While |

# المصادر

هذه مجموعة من المواقع الممتازة حقاً ... التي تعلمت منها عن جد موقع أفضل من الأخر..

http://appsstuff.appshosting.com/apex/f?p=606:14:2290465779602::NO:::#ADF_RESULT

http://www.sd4it.com/training/java_videos/

http://www.javatpoint.com/java-what-where-and-why

http://www.studytonight.com/java/

http://www.tutorialspoint.com/java/index.htm

http://www.learn-java-tutorial.com/JavaIntroduction.cfm#.UjSV8NJgfIs

http://www.javatutorialhub.com/java-tutorial.html

http://r4r.co.in/java/corejava/faqs/corejavabasics/corejavafaqs1.shtml موقع الأسئلة والاجوبة

http://www.freejavaguide.com/java-interview-questions.html موقع الأسئلة والاجوبة

تم بحمد الله

الأحد ١٧ ربيع الأول ١٤٣٥ — إسراء — الموافق ١٩ كانون الثاني ٢٠١٤