4th Day

الدوال Functions

تعريف دالة Function

لتعريف دالة يمكننا استخدام كلمة `static` ثم نوع الدالة - مثل `void` و هي دالة لا تعيد لنا اي قيمة - متبوعةً باسم الدالة، وفي حالتنا هنا اسم الدالة `printlnMessage`. لتوضيح الفكرة، لاحظ المثال التالي:

```
static void printHelloWorld(){
    System.out.println("Hello World");
}
```

قمنا ببناء دالة باسم `printlnMessage` ، والتي تحتوي على أمر طباعة واحد حيث ان اي سطر برمجي يكتب داخل `{}` يكون تابع للدالة، وعند تنفيذ هذه الدالة، ستتم طباعة الرسالة الظاهرة بداخل الدالة.

استدعاء الدالة Function Call

لاستدعاء الدالة نقوم بكتابة اسم الدالة متبوعاً `()` ، على النحو التالي:

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

الدالة والمدخلات ومفهوم Parameters

لإنشاء الدالة مع `parameter` نقوم بتحديد نوع و اسم المدخل الذي سوف يستقبل القيمة بين الأقواس `(parameter)` ، و يصبح بإمكاننا استخدامه داخل الدالة. كما هو موضح بالشكل التالي مدخل من نوع `String` و اسمه `Name` :

```
static void printMessageWithParam(String message){
    System.out.println("Welcome to " + message);
}
```

الدالة والمدخلات ومفهوم Arguments

لاستدعاء الدالة السابقة نقوم بكتابة اسم الدالة ثم بين القوسين نكتب القيمة المطلوب إسنادها ، كما هو موضح بالشكل التالي :

```
public static void main(String[] args) {
    printMessageWithParam("منصة سطر");
}
```

ملاحظة: يجب ان يكون ترتيب قيم Arguments مبني على Parameters

الفرق بين *Parameter* و *Argument*

تسمى المدخلات التي يتم كتابتها عند تعريف الدالة parameter بينما تسمى القيم الممررة للدالة عند الاستدعاء argument

إنشاء الدالة مع Return

لإنشاء دالة تعود بقيمة نقوم بالبداية بتعريف الدالة بنوع القيمة المرجعة ثم نستخدم عبارة `return` متبوعةً بالقيمة التي سوف تعود بها الدالة .يوضح المثال التالي كيفية إنشاء دالة تعود بناتج جمع عددين:

```
static int sum(int one,int two ){
  return  one + two;
}
```

Programming Paradigms

Object-oriented programming

في البرمجة الشيئية ، يتم تمثيل المعلومات كفئة تصف مفاهيم مجال المشكلة ومنطق التطبيق. تحدد الفئات الطرق التي تحدد كيفية معالجة المعلومات.

```
Example
```

Procedural programming

بينما في البرمجة الشيئية ، يتم تشكيل هيكل البرنامج من خلال البيانات التي يعالجها ، في البرمجة الإجرائية ، يتم تشكيل هيكل البرنامج من خلال الوظيفة المطلوبة للبرنامج: يعمل البرنامج كدليل خطوة بخطوة للوظيفة المطلوب أداؤها.

Static keyword

الفرق بين Static vs. Non-Static

في الدروس السابقة قمنا بتعريف دوال بإستخدلم static كما في المثال التالي

```
static void printHelloWorld(){
    System.out.println("Hello World");
}
```

الفرق بين Static vs. Non-Static هو انه في الدوال من نوع static يمكننا استخدامها بدون انشاء object لهذا الكلاس بينما دوال Non-Static يجب علينا انشاء object لهذا الكلاس ثم استخدام هذه الدالة كما في المثال التالي:

```
 class Animal{
    private String name;
    private int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

     void hunt(){
        System.out.println("We Are Hunting Now ");
    }
    static void staticHunt(){
```

```
        System.out.println("StaticHunt: We Are Hunting Now ");
    }
}
Animal cat = new Animal("Cat",4);
cat.hunt();
Animal.staticHunt();
```

مفهوم Java Access Modifiers

قبل البدأ بشرح مفهوم Modifiers لاحظنا سابقا تكرار كلمة public كثيرا وهي احدى خيارات Modifiers في لغة Java

تنقسم Modifiers الى ثلاثة يمكن تعريفها في الجدول التالي

نوع Modifiers	التعريف
public	يمكن الوصول الى العناصر الخاصة بالكلاس من اي مكان
private	لا يمكن الوصول الى العناصر الخاصة بالكلاس الا من داخل الكلاس
protected	يمكن الوصول الى العناصر الخاصة بالكلاس من الكلاسات الموجوة بنفس المجلد والكلاسات التي ترث منه
الوضع الافتراضي	يمكن الوصول الى العناصر الخاصة بالكلاس من الكلاسات الموجوة بنفس المجلد

لشرح الفكرة بشكل مبسط سوف نقوم بالتجربة على object من نوع Animal في الوقن الحالي جميع العناصر الخاصة بهذا class لم يتم تعريف Modifiers خاصة بها اي انها الان من نوع **protected** و يتم الوصول لها من الكلاسات الموجودة بنفس المجلد.

```
class Animal{
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
```

```
        this.age = age;
    }
}

public class Main {
    public static void main(String[] args) {

        Animal cat = new Animal("Cat",4);
        System.out.println(cat.name);

    }

}
```

في المثال الأعلى يمكننا طباعة خاصية name الخاصة ب cat لكن لو قمنا بتغيير نوع Modifiers الخاصة بعناصر هذا الكلاس فسوف يصبح من غير الممكن طباعتها

```
class Animal{
    private String name;
    private int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

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

```
        Animal cat = new Animal("Cat",4);
        System.out.println(cat.name);


    }
}
```

و سوف نحصل على خطأ

```
java: name has private access in Animal
```