محتوى اليوم الثالث

Exceptions

الإستثناء (**Exceptions**) في البرمجة عبارة عن حدث يقوم بتعطيل السير الطبيعي للبرنامج. وهو كائن يتم إلقاؤه في وقت التشغيل.
يمكن أن يحدث استثناء لأسباب عديدة ومختلفة منها الخطأ النحوي و الخطأ وقت التشغيل والخطأ في المنطق.

Syntax error

الخطأ اللغوي (Syntax Error) هو الخطأ في كتابة الأوامر البرمجية سواء في الكتابة بالأحرف الكبيرة أو نسيان الأقواس أو عدم إستدعاء class وغيرها.

```
      system.out.print;
```

Runtime error

يحدث الخطأ في وقت التشغيل عندما يكون البرنامج صحيحًا من الناحية اللغوية ولكنه يحتوي على مشكلة يتم اكتشافها فقط أثناء تنفيذ البرنامج.

```
// Java program to demonstrate Runtime Error

class DivByZero {
    public static void main(String args[])
    {
        int var1 = 15;
        int var2 = 5;
        int var3 = 0;
        int ans1 = var1 / var2;

        // This statement causes a runtime error,
        // as 15 is getting divided by 0 here
        int ans2 = var1 / var3;
```

```
        System.out.println(
            "Division of va1"
            + " by var2 is: "
            + ans1);
        System.out.println(
            "Division of va1"
            + " by var3 is: "
            + ans2);
    }
}
```

Logical error

الخطأ المنطقي (Logical Error) هي الأخطاء التي يرتكبها المبرمجون. تعمل هذه البرامج التي بها هذه الأخطاء ولكنها لا تعطي النتائج المتوقعة.

```
class IncorrectMessage {
        public static void main(String args[])
        {
           System.out.println(sum(6,3));
        }
public static int sum(int num1,int num2){

    return num2-num1
    }
}
```

Solving Runtime Exceptions

try-catch

يتم استخدام try-catch في جافا لإحاطة التعليمات البرمجية التي قد تؤدي إلى إستثناء. يجب استخدامه ضمن الميثود.

إذا حدث استثناء في جملة معينة في try block ، فلن يتم تنفيذ باقي الكود. لذلك ، يوصى بعدم الاحتفاظ بالكود في try-block التي لن تؤدي إلى استثناء.

يجب أن يتبع Java try block إما catch أو block.

```
try {
  int[] myNumbers = {1, 2, 3};
  System.out.println(myNumbers[10]);
} catch (Exception e) {
  System.out.println("Something went wrong.");
}
```

throw & throws

تعريف throws

إذا قمت بتعريف دالة و أردت لهذه الدالة أن ترمي إستثناء إذا حدث شيء معين فعليك وضع الكلمة **throws**
بعد أقواس الباراميترات ثم تحديد نوع الإستثناء الذي قد ترميه الدالة, و إذا قمت مسبقاً بتعريف إستثناء يمكنك جعل الدالة تقوم برميه.

```
 public class Main {
  static void checkAge(int age) throws ArithmeticException {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must
be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enoug
h!");
    }
```

```
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

تعريف throw

يتم استخدام الكلمة throw لإنشاء خطأ مخصص.

يتم إستخدام تعليمة throw مع نوع الاستثناء.

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Types of Runtime Exceptions

Checked Exceptions

هذه هي الاستثناءات التي يتم التحقق منها في وقت الترجمة. إذا ألقى بعض الكود ضمن طريقة استثناءً محددًا ، فيجب على الطريقة إما معالجة الاستثناء أو تحديد الاستثناء باستخدام الكلمة الأساسية throws.

```
// Java Program to Illustrate Checked Exceptions
// Where FileNotFoundException does not occur

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

        // Main driver method
        public static void main(String[] args)
                throws IOException
        {

                // Creating a file and reading from local repo
sitory
                FileReader file = new FileReader("C:\\test\\a.
txt");

                // Reading content inside a file
                BufferedReader fileInput = new BufferedReader
(file);

                // Printing first 3 lines of file "C:\test\a.t
xt"
                for (int counter = 0; counter < 3; counter++)
```

```
                        System.out.println(fileInput.readLine
());

                // Closing all file connections
                // using close() method
                // Good practice to avoid any memory leakage
                fileInput.close();
        }
}
```

Unchecked Exceptions

هذه هي الاستثناءات التي لم يتم التحقق منها في وقت compile. في استثناءات Java ضمن فئات Error و RuntimeException هي استثناءات لم يتم تحديدها ، يتم التحقق من كل شيء آخر ضمن قابل للإلقاء.

```
// Java Program to Illustrate Un-checked Exceptions

// Main class
class GFG {

        // Main driver method
        public static void main(String args[])
        {
                // Here we are dividing by 0
                // which will not be caught at compile time
                // as there is no mistake but caught at runtim
e
                // because it is mathematically incorrect
                int x = 0;
                int y = 10;
```

```
                int z = y / x;
        }
}
```

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	
يمكن الوصول الى العناصر الخاصة بالكلاس من الكلاسات الموجوة بنفس المجلد	**الوضع الافتراضي**

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
```