



Programmation Orientée Objet

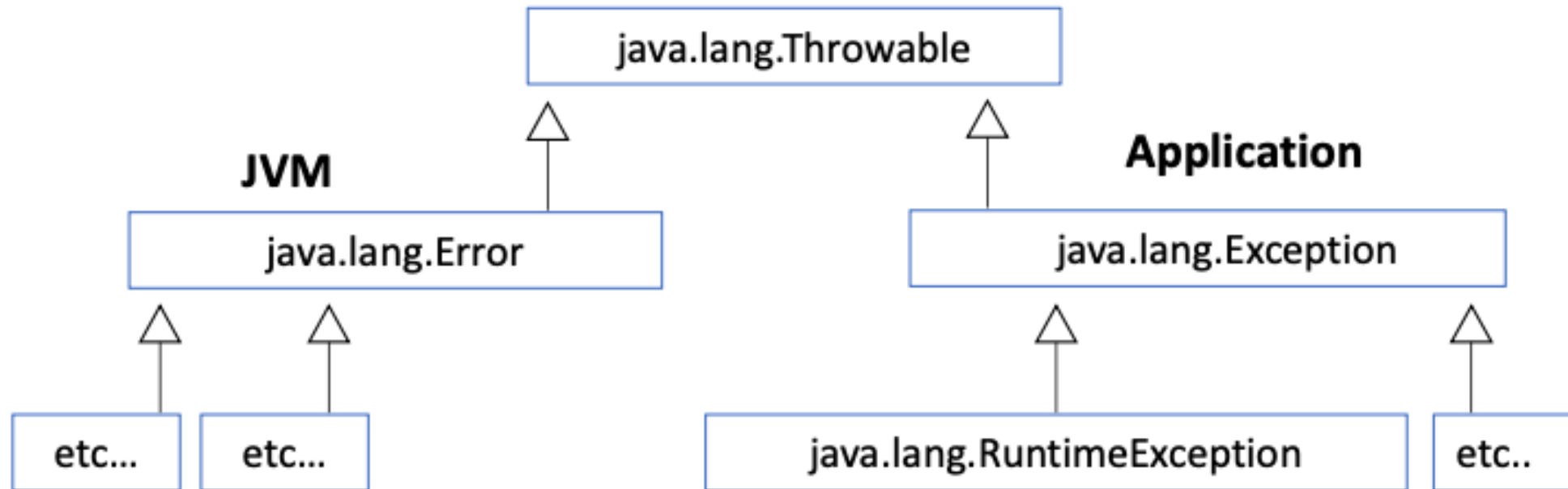
Gestion des Exceptions

Mme Hassna BENSAG
Email: h.bensag@gmail.com

Le principe général des exceptions

- Une exception est un événement indésirable de déroutement du programme c'est-à-dire qui perturbe le flux normal des instructions du programme.
- La différence entre une erreur et une exception en Java est :
 - Une Exception en java est un objet erreur généré par une application java qu'on peut attraper et traiter.
 - Une Erreur est un objet de type Error qui ne peut être ni traité ni prévu par le programmeur

La hiérarchie des exceptions

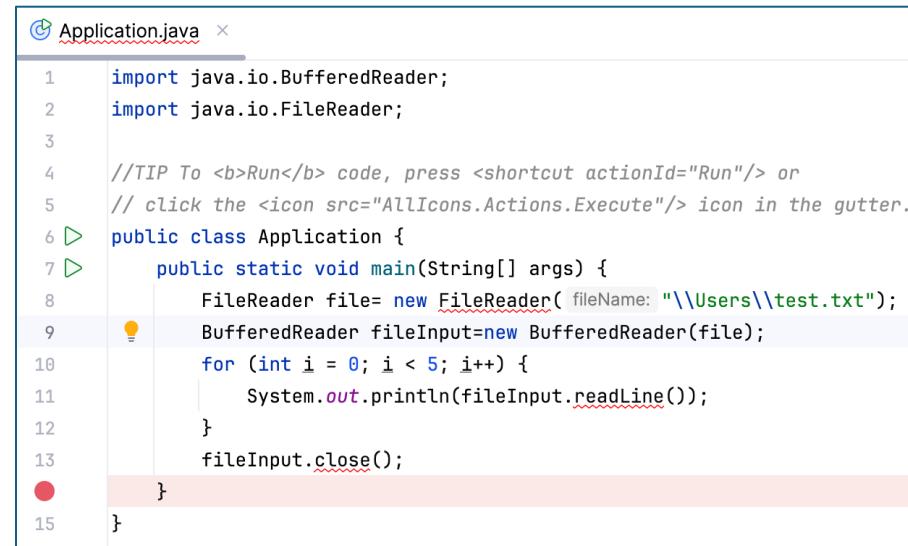


Les exceptions surveillées et non surveillées

- Les exceptions surveillées: Ce sont les exceptions qui sont vérifiées au moment de la compilation. Java oblige le programmeur à traiter les exceptions surveillées sinon nous allons voir une erreur de compilation. Les classes Throwable et Exception représentent des exceptions surveillées.
- Les exceptions non surveillées: Ce sont des exceptions qui ne sont pas vérifiées au moment de la compilation. Ces exceptions peuvent être traitées ou non par le programmeur. Les classes Error et RuntimeException représentent des exceptions non surveillées.

Les exceptions surveillées

- Le code suivant ne compile pas, car on utilise `FileReader()` et `BufferedReader()` qui vont lever une exception surveillée de type **`FileNotFoundException`**. On utilise également les méthodes `readLine()` et `close()`, et ces méthodes lèvent également une exception surveillée de type **`IOException`**.

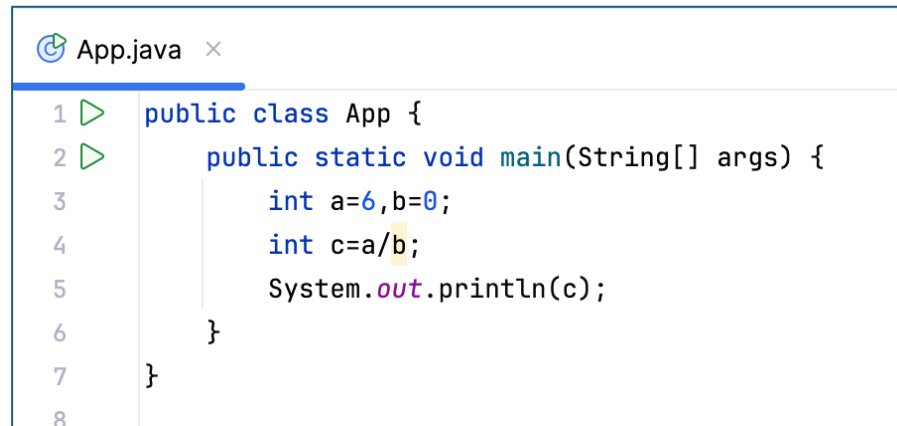


```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3
4  //TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
5  // click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
6  public class Application {
7      public static void main(String[] args) {
8          FileReader file= new FileReader( fileName: "\\Users\\test.txt");
9          BufferedReader fileInput=new BufferedReader(file);
10         for (int i = 0; i < 5; i++) {
11             System.out.println(fileInput.readLine());
12         }
13         fileInput.close();
14     }
15 }
```

- Afin de corriger le programme ci-dessus, nous devons soit spécifier une liste d'exceptions à l'aide de **`throws`** dans la signature de la méthode `main`, soit utiliser un bloc **`try-catch`** pour attraper ces exceptions.

Les exceptions non surveillées

- On considère le programme Java suivant qui illustre un exemple d'une exception non surveillée qui n'est pas détectée par le compilateur mais elle est générée au moment d'exécution :



```
App.java x
1  public class App {
2      public static void main(String[] args) {
3          int a=6,b=0;
4          int c=a/b;
5          System.out.println(c);
6      }
7  }
8
```

- Ce programme génère l'erreur suivante:

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
at App.main(App.java:4)
```

Les exceptions non surveillées

- Dans le programme précédent une exception non surveillée indiquant la division par zéro est générée.
- L'exception générée est de type **ArithmeticException** qui représente une exception non surveillée. Cette classe hérite de la classe **RuntimeException**.
- Lorsqu'une exception, non surveillée et qui n'est pas traitée par le programmeur, se produit, l'interpréteur la traite en affichant un message et en provoquant l'arrêt instantané du programme (Bug).

Attraper et traiter les exceptions

```
try {  
    // bloc de code pour surveiller les erreurs  
    // le code que peut lever une exception  
}  
catch (ExceptionType1 exOb) {  
    // bloc de code pour traiter une exception de type ExceptionType1  
}  
catch (ExceptionType2 exOb) {  
    // bloc de code pour traiter une exception de type ExceptionType2  
}  
finally {  
    // bloc de code à exécuter après la fin du bloc try  
}
```

- Le bloc **try**: Ce bloc contient les instructions où une exception peut se produire.
- Le bloc **catch**: Ce bloc est utilisé pour attraper et gérer l'exception qui se produit dans le bloc try associé.
- Le bloc **finally**: Ce bloc est facultatif. Il sera toujours exécuté, qu'une exception soit levée ou non.

Attraper et traiter les exceptions

- Exemple:

```
App.java ×  
1  public class App {  
2      public static void main(String[] args) {  
3  
4          try {  
5              int a=6,b=0;  
6              int c=a/b;  
7              System.out.println(c);  
8          } catch (ArithmeticException e) {  
9              System.out.println("Exception: Division par zero");  
10         }  
11     }  
12 }  
13 }
```

Attraper et traiter les exceptions

- Dans l'exemple précédent l'exécution se produit sans problème et indique que l'exception a été bien attrapée dans le bloc catch.
- Dans ce cas le programme empêche que l'exception soit transmise à la machine virtuelle java ce qui empêche l'arrêt du programme et donne la possibilité de traiter l'erreur.
- Le bloc **catch** qui permet d'attraper et de traiter l'exception est appelé «**handler d'exception**».

Attraper et traiter les exceptions

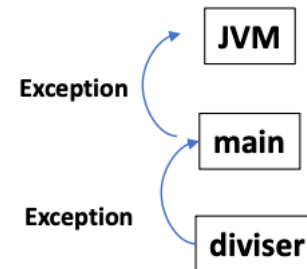
```
App.java x
1  public class App {
2      public static void main(String[] args) {
3
4          try {
5              int a=6,b=0;
6              int c=a/b;
7              System.out.println(c);
8          } catch (ArithmeticException e) {
9              System.out.println("Exception: Division par zero");
10             System.out.println(e.getMessage());
11             System.out.println(e.toString());
12             e.printStackTrace();
13         }
14     }
15
16 }
17 }
```

Exception: Division par zero
/ by zero
java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException Create breakpoint : / by zero
at App.main(App.java:6)

Attraper et traiter les exceptions

- Exemple:

```
Application.java x
1 public class Application {
2
3     public static int diviser(int a,int b){ 1 usage
4         return a/b;
5     }
6
7     public static void main(String[] args) {
8         try {
9             int c;
10            c=diviser(a: 5, b: 0);
11            System.out.println("Le résultat: "+c);
12        } catch (ArithmeticException e) {
13            System.out.println(e.getMessage());
14        }
15    }
16 }
```



- Dans ce programme, l'exception est lancée dans la méthode diviser. Celle-ci ne comporte pas de bloc catch pour attraper et traiter cette exception.
- L'exception lancée remonte alors dans le bloc de niveau supérieur, c'est-à-dire celui où a eu lieu l'appel de la méthode, dans cet exemple c'est la méthode main. Là, un bloc catch est trouvé et l'exception est traitée.
- Supposant que le bloc catch n'existe pas dans la méthode main, l'exception va remonter vers la JVM, qui va afficher un message d'erreur en provoquant l'arrêt instantané du programme.

Relancer ou jeter une exception

- Le mot-clé **throw** en Java est utilisé pour lancer explicitement une exception à partir d'une méthode ou de tout bloc de code.
- Nous pouvons lancer une exception surveillée ou non surveillée avec le mot clé **throw**.
- On considère le code suivant :

```
Application.java x
1  public class Application {
2
3      public static int diviser(int a,int b) throws ArithmeticException{
4          if (b==0) throw new ArithmeticException("Division par zéro");
5          return a/b;
6      }
7
8      public static void main(String[] args) {
9          try {
10             int c;
11             c=diviser( a: 5, b: 0);
12             System.out.println("Le résultat: "+c);
13         } catch (ArithmeticException e) {
14             System.out.println(e.getMessage());
15         }
16     }
17 }
```

Relancer ou jeter une exception

- Dans l'exemple précédent, la méthode diviser génère une exception dont le message est "Division par zéro", si la valeur de b est zéro.
- L'entête de la méthode diviser se termine par **throws** ArithmeticException pour indiquer que cette méthode est susceptible de générer une exception de type ArithmeticException.
- Dans l'application, à chaque fois que la méthode diviser est appelée, il faut prévoir le traitement de l'exception en utilisant:
 - Soit le bloc try catch.
 - Soit en ajoutant à la méthode qui fait appel à diviser(), l'indicateur **throws** ArithmeticException, pour indiquer que Si l'exception survient, elle ne sera pas attrapée, par la suite, elle va remonter vers le niveau supérieur.

Les exceptions personnalisées

- Java offre la possibilité de créer nos propres exceptions qui sont essentiellement des classes dérivées de la classe Exception.
- Par convention Il est recommandé d'inclure le mot « Exception » dans le nom de la nouvelle classe.
- Une exception personnalisée permet aux développeurs de séparer le code de gestion des erreurs du code normal.
- Dans l'exemple suivant on définit une nouvelle exception qu'on peut générer lors ce que le solde est insuffisant lors du retrait d'un montant d'un compte bancaire:

```
SoldeInsuffisantException.java ×  
1 public class SoldeInsuffisantException extends Exception{  
2     public SoldeInsuffisantException(String msg){  
3         super(msg);  
4     }  
5 }
```

Les exceptions personnalisées

- Dans l'exemple précédent nous avons défini une nouvelle exception appelée `SoldeInsuffisantException` qui est une sous-classe de la classe `Exception` :
- Il y a également un constructeur avec une chaîne comme paramètre qui va être passée vers le constructeur de la classe `Exception` en utilisant `super`. Ce message est obtenu en utilisant la méthode "`getMessage()`" sur l'objet créé.
- L'exception créée est levée dans la méthode `retirer` lors ce que le solde est insuffisant comme le montre le code suivant :

```
© CompteBancaire.java x
1  public class CompteBancaire { no usages
2      private int solde; 2 usages
3
4      public CompteBancaire(int solde) { no usages
5          this.solde = solde;
6      }
7
8      public void retirer(int montant) throws SoldeInsuffisantException{ no usages
9          if(solde < montant) throw new SoldeInsuffisantException("Solde insuffisant");
10     }
11
12 }
```


Les exceptions personnalisées

- On crée un compte bancaire avec un solde et on fait appel à la méthode retirer:

```
AppBancaire.java x
1  public class AppBancaire {
2      public static void main(String[] args) {
3          try {
4              CompteBancaire compte= new CompteBancaire( solde: 14500);
5              compte.retirer( montant: 25000);
6          } catch (SoldeInsuffisantException e) {
7              System.out.println(e.getMessage());
8          }
9      }
10 }
```

Résultat



Solde insuffisant