Traitement des erreurs et utilisation des exceptions

Programmation objets
et Java

(B. Sonntag)

Définition

- Une exception est un événement qui est lié à l'apparition d'une erreur et qui interrompt le déroulement normal des instructions d'un programme
- Lorsqu'une erreur survient lors de l'éxecution d'une méthode, un objet de la classe *Exception* est créé et transmis au système qui se charge de traiter l'événement

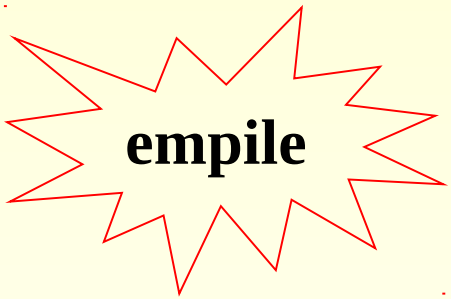
Notion de pile d'exécution

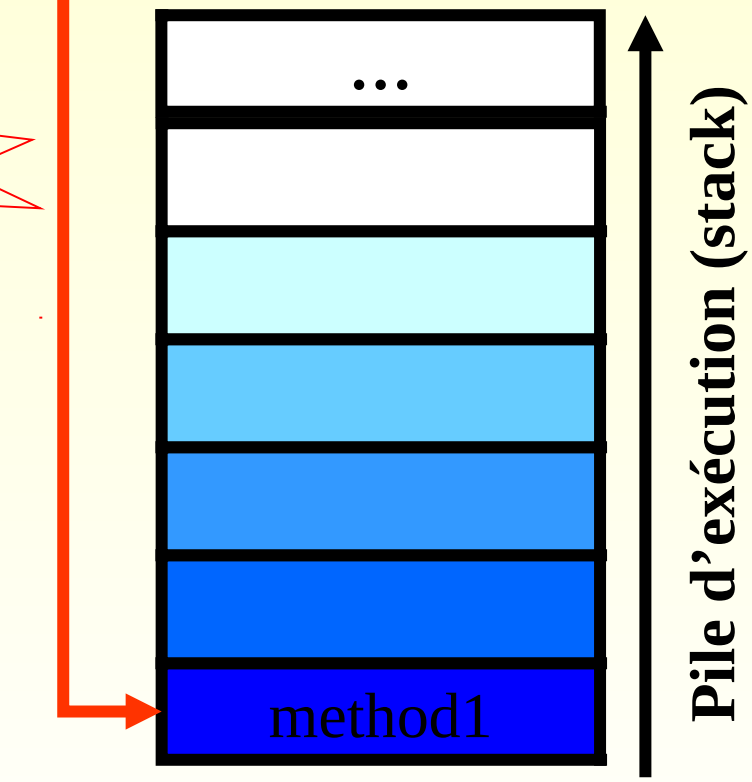
```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```





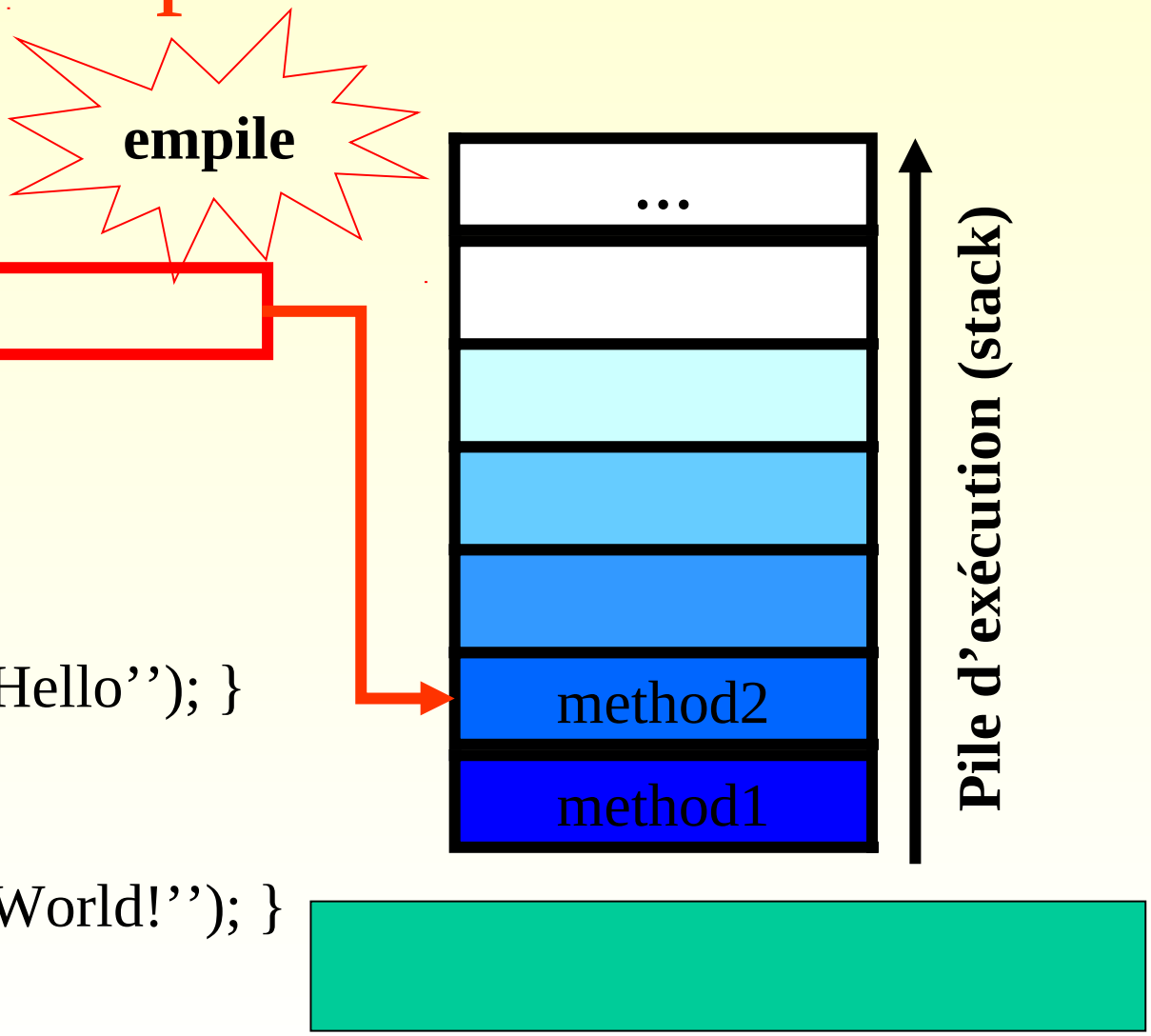
Notion de pile d'exécution

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```



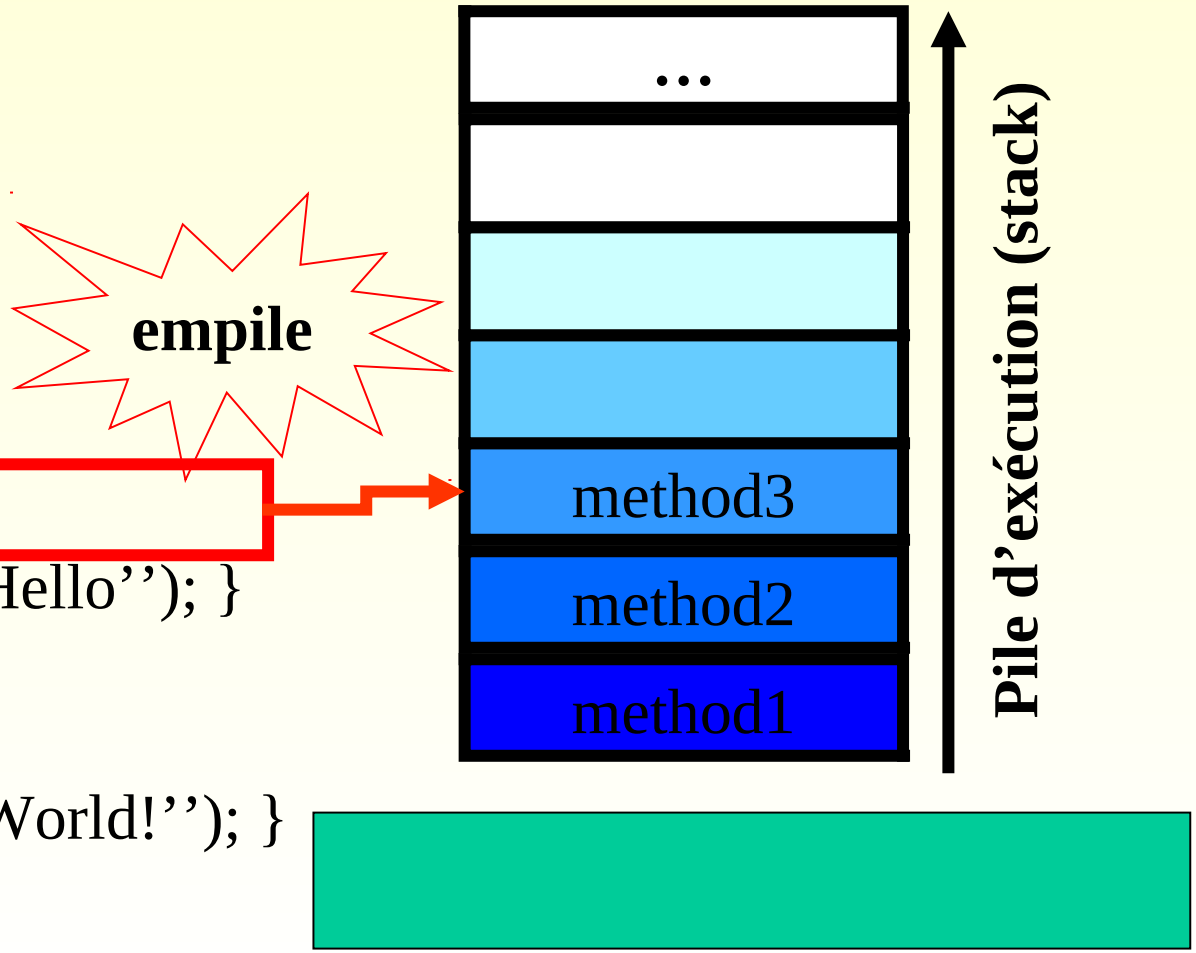
Notion de pile d'exécution

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```



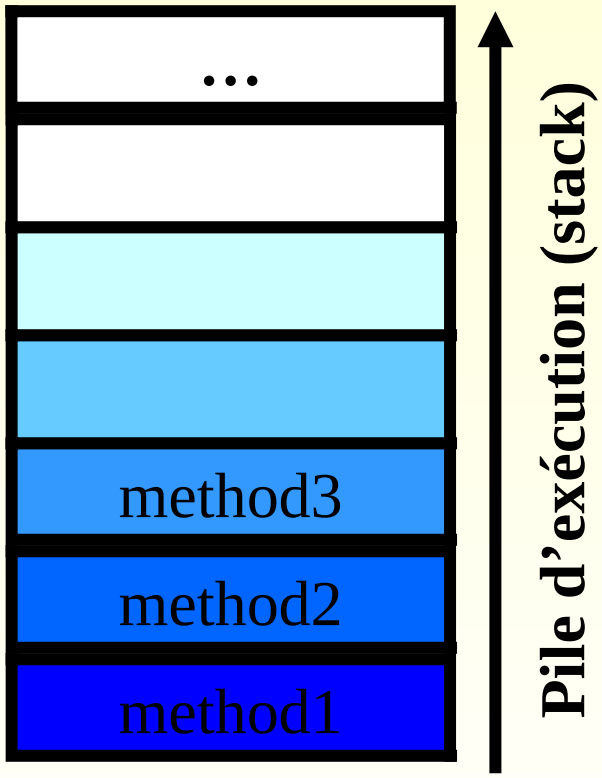
Notion de pile d'exécution

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```




```
Hello
```

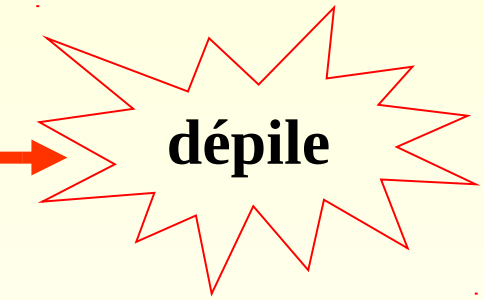
Notion de pile d'exécution

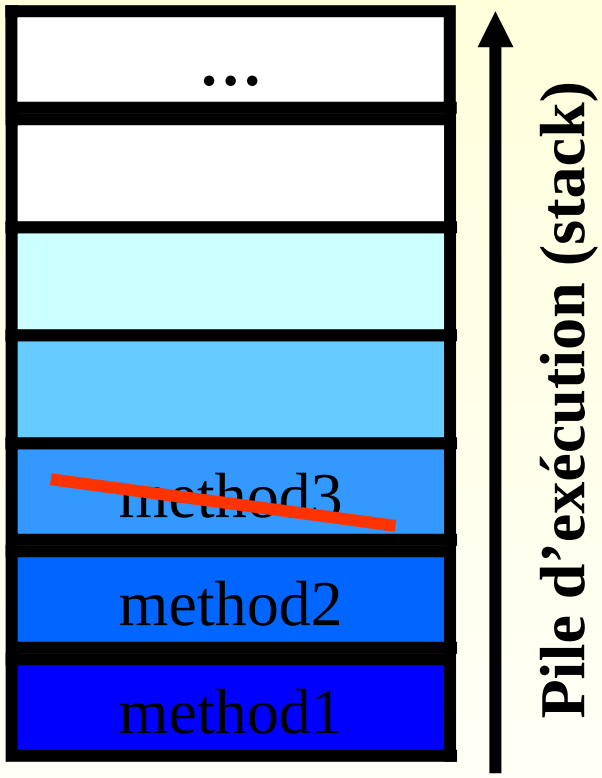
```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```






```
Hello
```

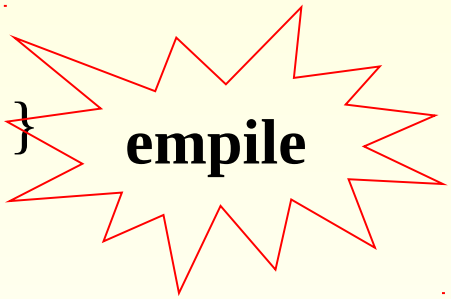
Notion de pile d'exécution

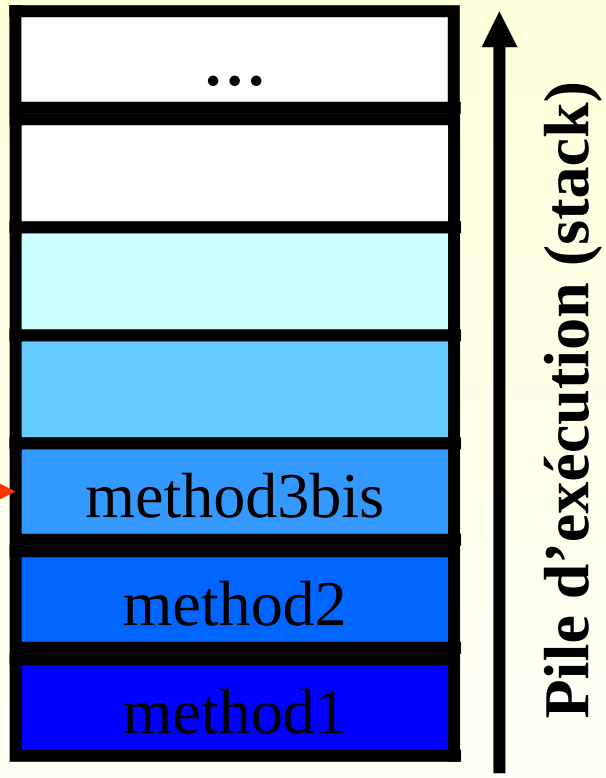
```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.out.print(''Hello''); }

void method3bis()
{  System.out.print(''World!''); }
```






```
Hello
```

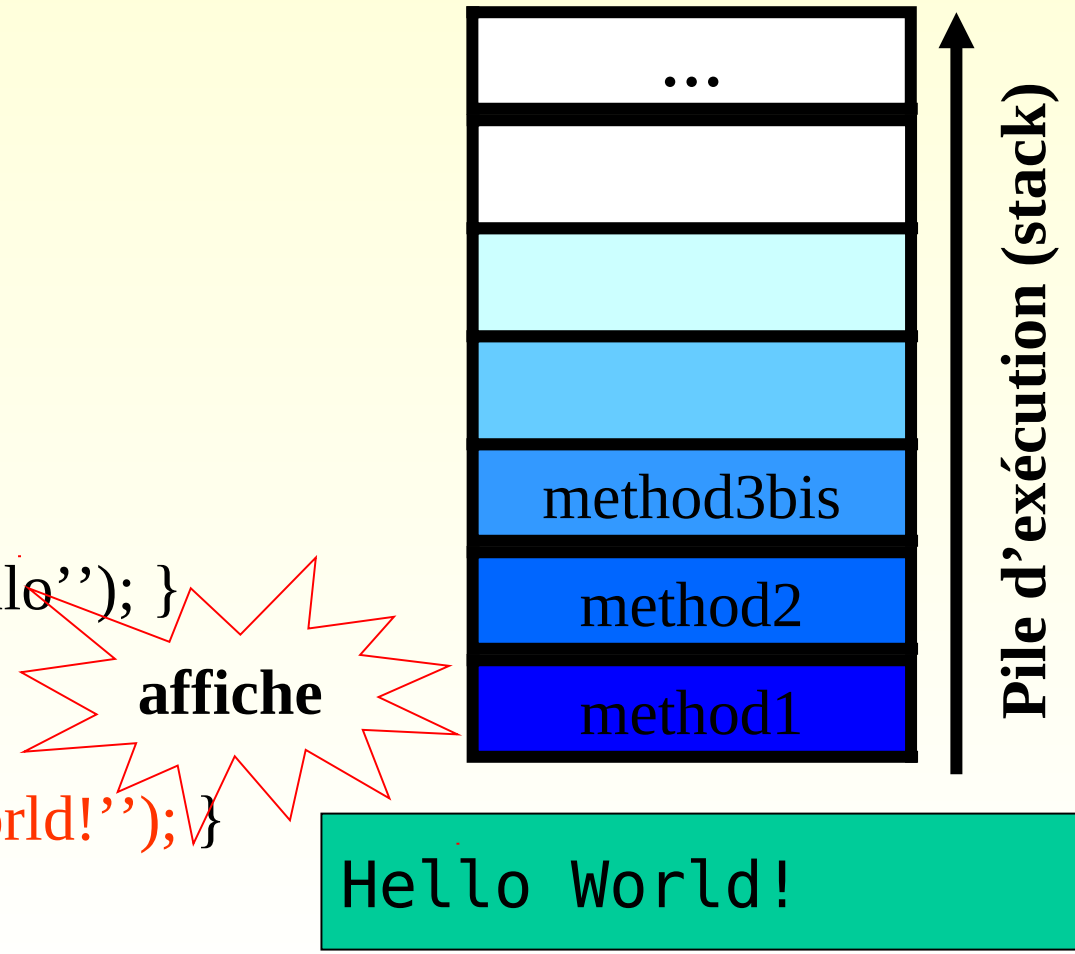
Notion de pile d'exécution

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.io.out(''Hello''); }

void method3bis()
{  System.io.out(''World!''); }
```



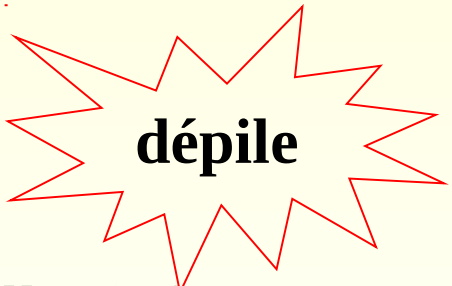
Notion de pile d'exécution

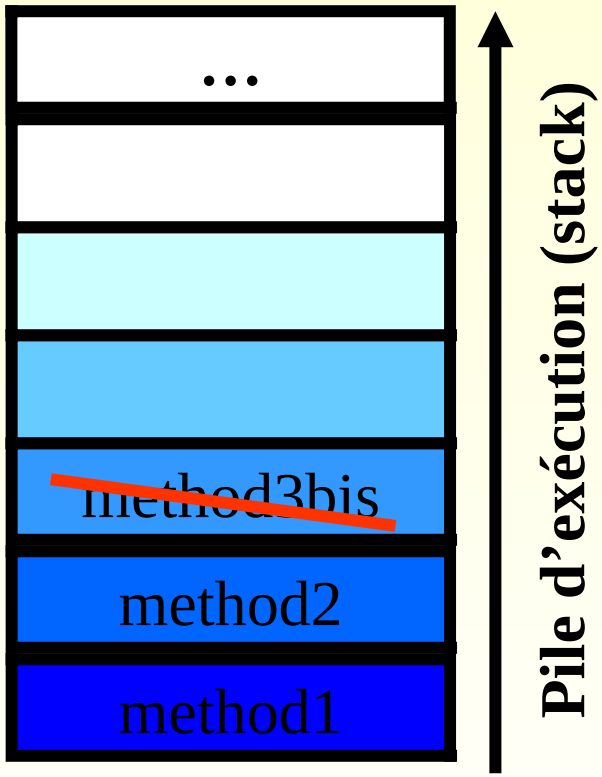
```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.io.out(''Hello''); }

void method3bis()
{  System.io.out(''World!''); }
```






```
Hello World!
```

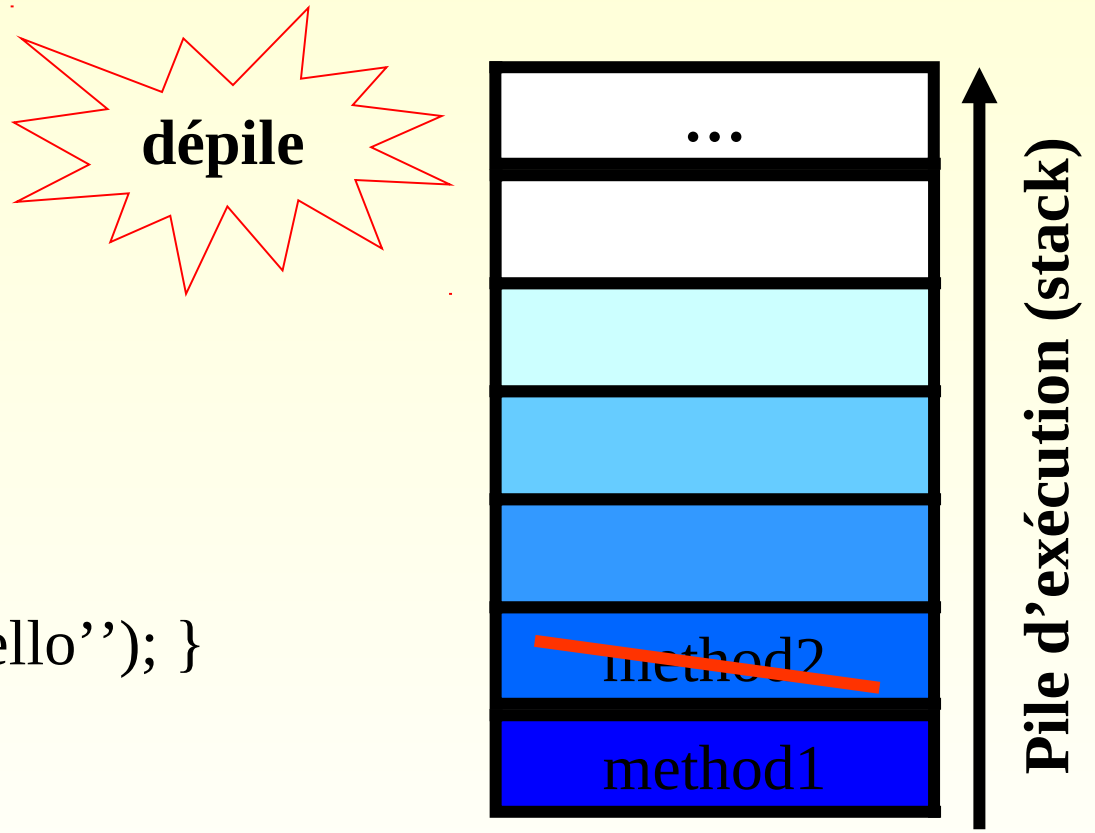
Notion de pile d'exécution

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.io.out(''Hello''); }

void method3bis()
{  System.io.out(''World!''); }
```




```
Hello World!
```

Notion de pile d'exécution

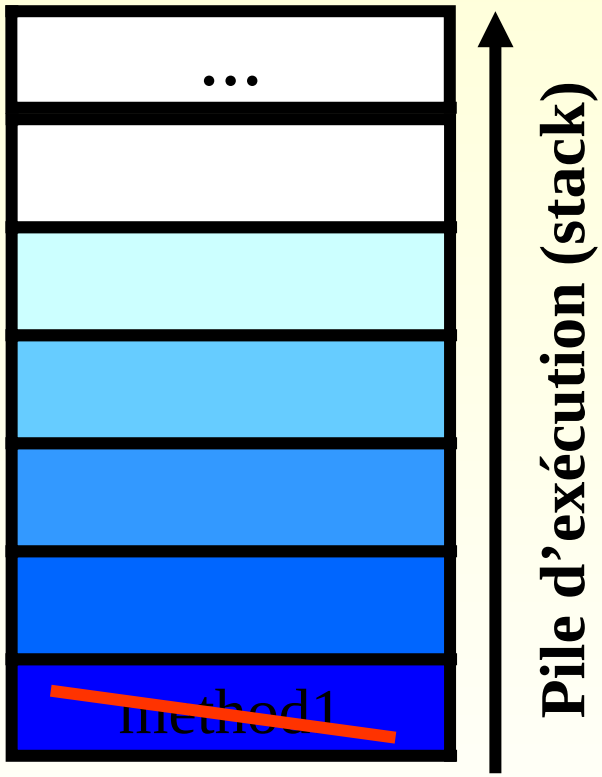
dépile

```
void method1()
{  method2(); }

void method2()
{  method3();
   method3bis(); }

void method3()
{  System.io.out(''Hello''); }

void method3bis()
{  System.io.out(''World!''); }
```




```
Hello World!
```

Exemple simple

```
void method1()
{
    method2(0);
}

int method2(int val)
{
    return (10 / val);
}
```

Exemple simple

```
void method1()
{
    method2(0);
}

int method2(int val)
{
    return (10 / val);
}
```

Exemple simple

```
void method1()
{
    method2(0);
}

int method2(int val)
{ int result;
   try {
     result = 10 / val;
   } catch (ArithmeticException e) {
     System.out.println(''Attention division par 0 dans method2!'');
     result = 0;
   }
   return (result);
}
```

Pose du *handler* d'exception directement sur le code sensible.

Exemple simple

```
void method1()
{
   try {
     method2(0);
   } catch (ArithmeticException e) {
     System.out.println(''Attention division par 0 dans method2!'');
   }
}

int method2(int val)
{  return (10 / val);
}
```

OU Pose du *handler* d'exception
sur l'appelant, ou l'appelant de l'appelant, ...

Exemple 2

```
public class Exemple {
  private int[] t;
  private int capacity;
  private int length;

  public Exemple() {
    t = new int[10];
    capacity = 10;
    length = 0;
   }
…}//fin classe
```

Exemple 2

```
public int moyenne(){
   int somme = 0;
   for (int i = 0; i < t.length; i++) {
      somme += t[i];
   }
   return somme/length;
}
```

Exemple 2

```
public static void main(String[] args) {
   Exemple e = new Exemple();
   System.out.println("Moyenne = "
                          +e.moyenne());
}
```

Exemple 2

```
java.lang.ArithmeticException: / by zero
at testException.Exemple.moyenne(Exemple.java:45)
at testException.Exemple.main(Exemple.java:53)
Exception in thread "main"
```

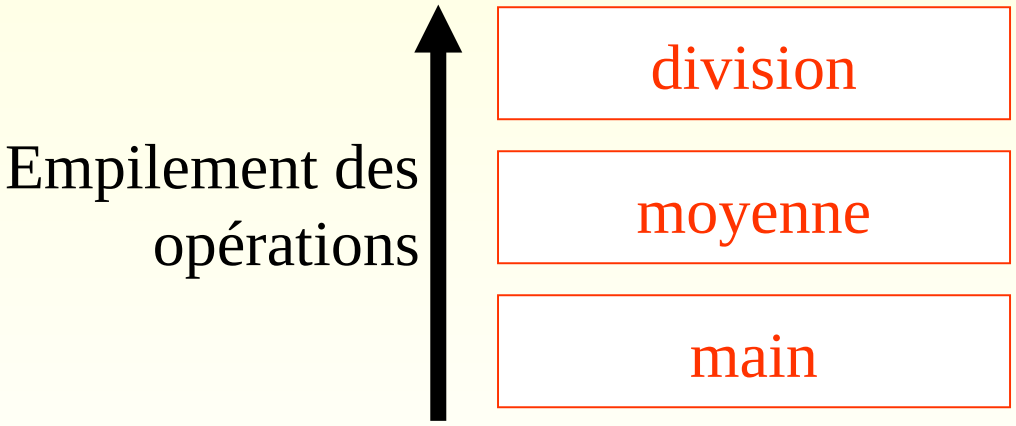
Exemple 2 : Pose de l'exception

```
public int moyenne(){
  int somme = 0;
  for (int i = 0; i < t.length; i++) {
    somme += t[i];
  }
  int res;
  try {
    res = somme/length;
  } catch (ArithmeticException e) {
    res = 0;
  }
  return res;
}
```

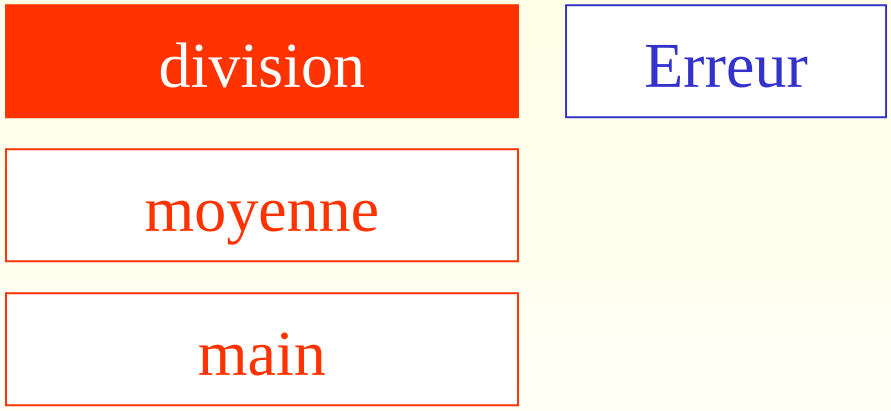
Exemple 2

```
Moyenne = 0
```

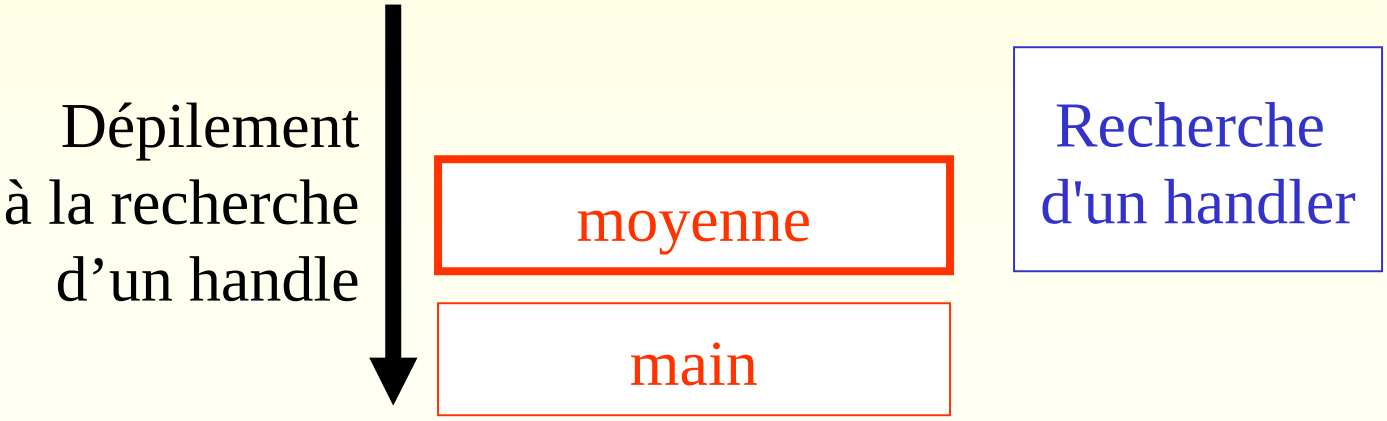
Principe (Exemple2 sans *handle*)



Principe (Exemple2 sans *handle*)



Principe (Exemple2 sans *handle*)



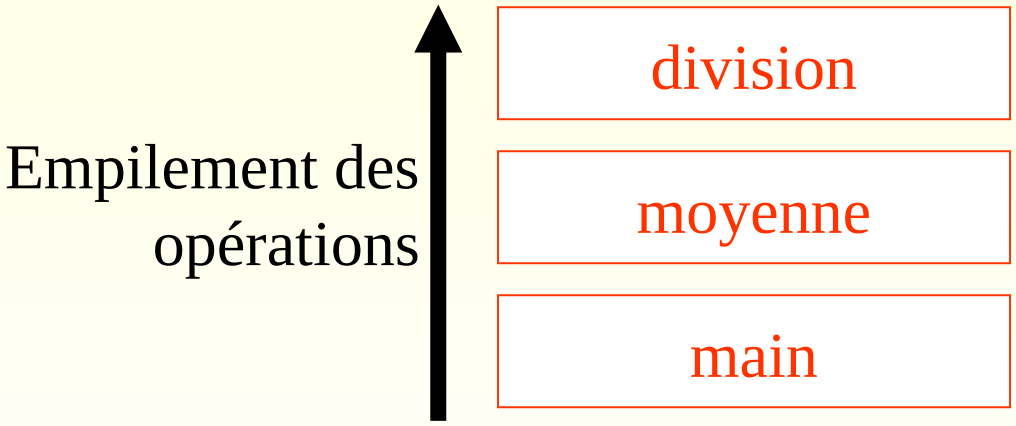
Principe (Exemple2 sans *handler*)

Principe (Exemple2 sans *handle*)

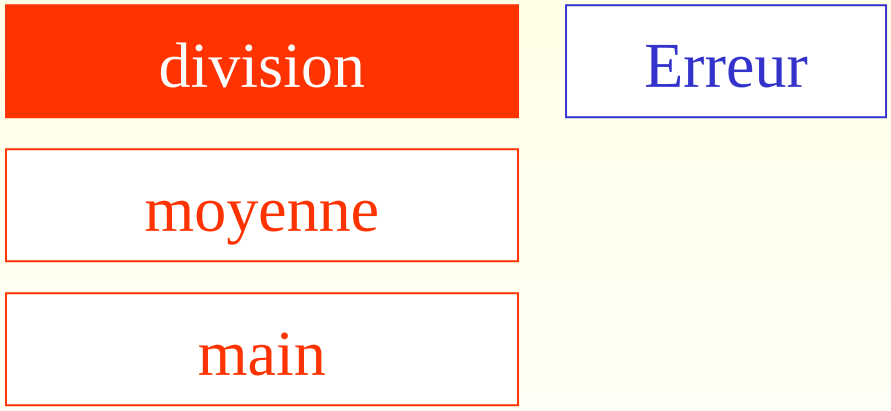
```
java.lang.ArithmeticException: / by zero
at testException.Exemple.moyenne(Exemple.java:45)
at testException.Exemple.main(Exemple.java:53)
Exception in thread "main"
```

Retour
système
(par défaut:
printStackTrace)

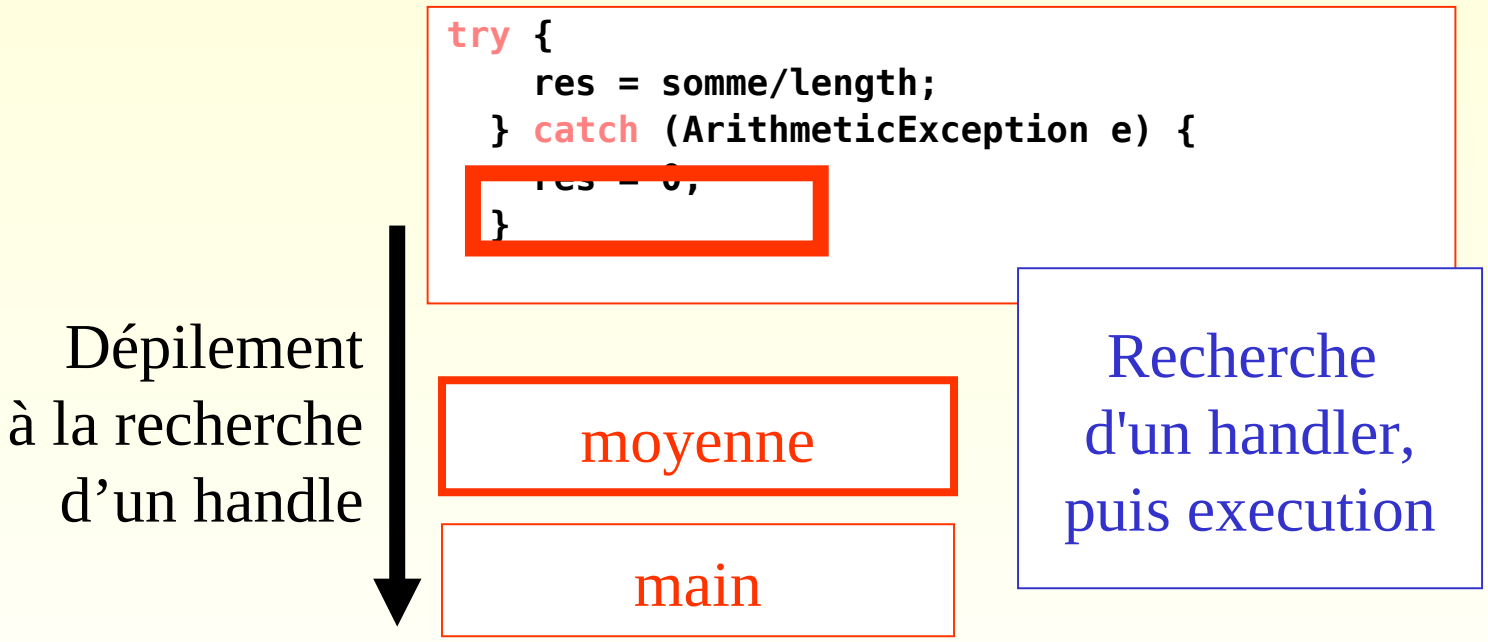
Principe (Exemple2 avec *handle*)



Principe (Exemple2 avec *handle*)



Principe (Exemple2 avec *handle*)



Création d'une exception

```
public static void main(String[] args)
{ int val;
   val = e.get(3);
}
```

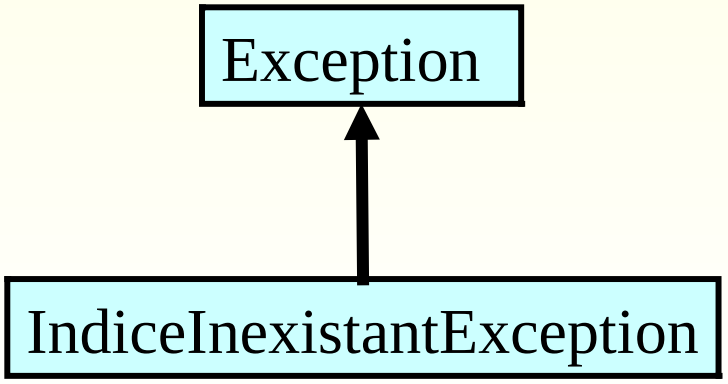
```
public int get(int index)
{ int result;
   result =
tableau[index];
   return result;
}
```

Si l'index est invalide, nous voudrions déclencher une exception !

Création d'une class *exception*

```
public class IndiceInexistantException
                          extends Exception{

  public IndiceInexistantException()
  {
    super("Indice inexistant");
  }
}
```



Production de l'exception

```
/**
 * Accéder à un élement
 */
public int get(int index)
        throws IndiceInexistantException{
 if (index >= length) {
  throw(new IndiceInexistantException());
 }
 int result = tableau[index];
 return result;
}
```

Récupération de l'exception

```
public static void main(String[] args) {
 int val;
 try {
 val = e.get(3);//erreur
 } catch (IndiceInexistantException ex) {
   ex.printStackTrace();
 }
}
```

Exemple 3

```
testException.IndiceInexistantException:
  Indice inexistant
at testException.Exemple.get(Exemple.java:42)
at testException.Exemple.main(Exemple.java:74)
```

Bloc *try*

```
try{
   <instructions>…
}
```

Bloc *catch*

```
try{
   <instructions normales>…
}catch(<Type d'exception> <var>){
   <instructions d'erreur>…
}catch(<Type d'exception> <var>){
   <instructions d'erreur>…
}…
```

Bloc *finally*

```
try{
   <instructions normales>…
}catch(<Type d'exception> <var>){
   <instructions d'erreur>…
}finally
   <instructions de nettoyage>…
}
```

Instruction *throw*

```
throw (<objet Throwable>);
```

Avantages

- Séparation du code normal du code d'erreur
- Possibilité de traiter l'erreur au niveau adéquat (grâce à la propagation sur la pile d'appel)
- Classification et identification des erreurs