

# Module M15

La programmation orientée objet JAVA

# Chapitre 6

---

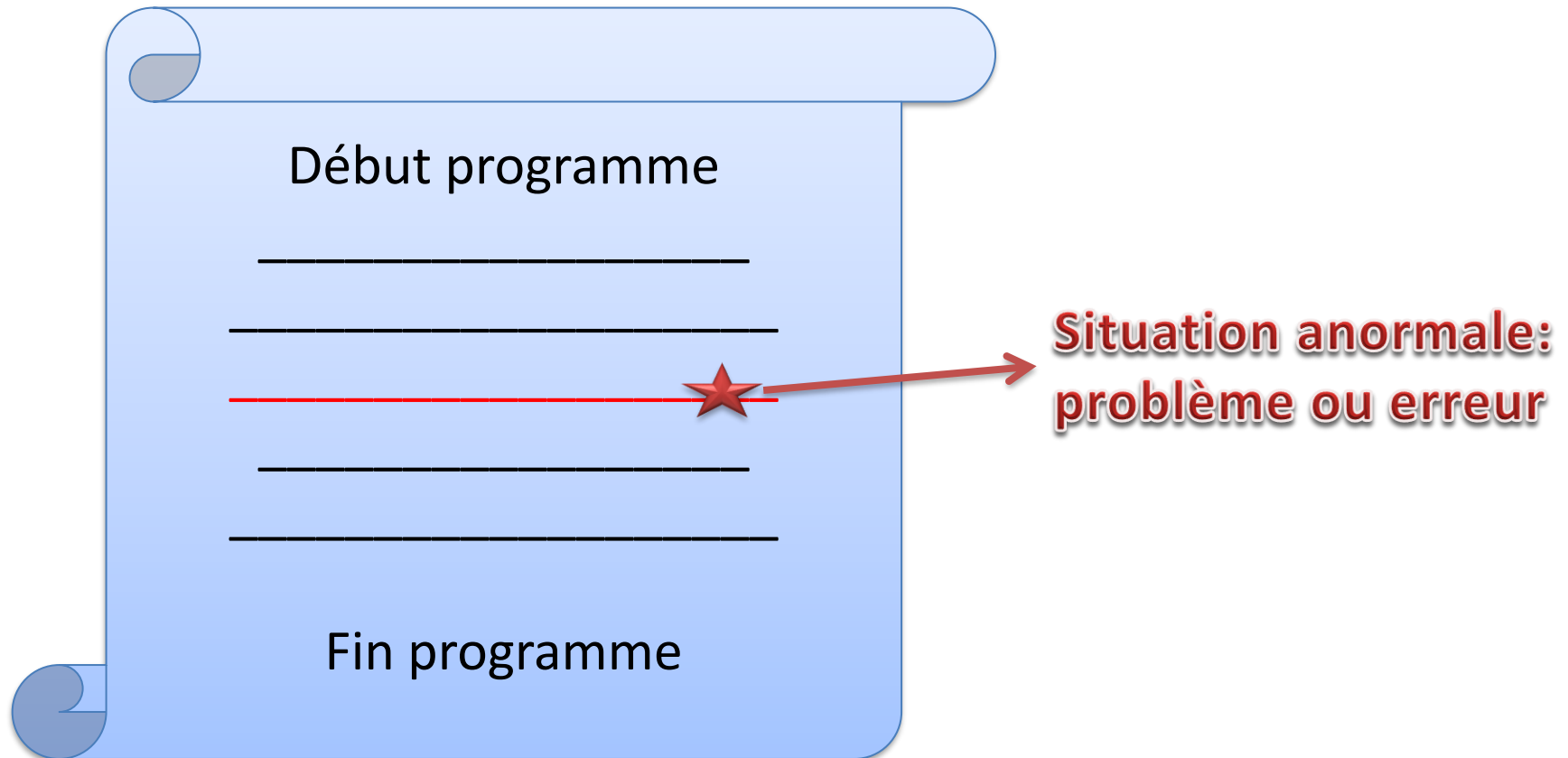
## **La gestion des exceptions**

### Plan du cours

---

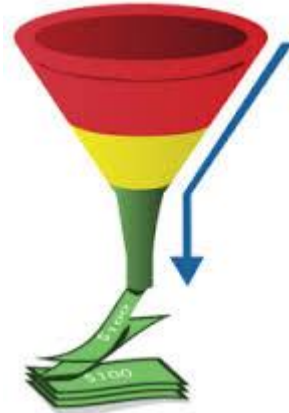
1. Les Exceptions
2. Nature d'exception
3. Conditions de génération des exceptions
4. Types d'Exception en Java
5. Exceptions vérifiées vs Exceptions non vérifiées

# Qu'est ce qu'une Exception



- Souvent, un programme doit traiter des situations exceptionnelles qui n'ont pas un rapport direct avec sa tâche principale.
- Ceci oblige le programmeur à réaliser de nombreux tests avant d'écrire les instructions utiles du programme

# Qu'est ce qu'une exception



`parseInt ("azerty")`



Débordement mémoire



Serveur non disponible

# Comment faire?

Début programme

If.... Else

iiiiiiif.... eeeeeeeelse

Fin programme



- Code difficile à lire et à maintenir
- Parfois, on ne peut pas traiter tous les cas exceptionnels

# Comment faire?

Début programme

If.... Else

iiiiiiif.... eeeeeeeelse

Fin programme



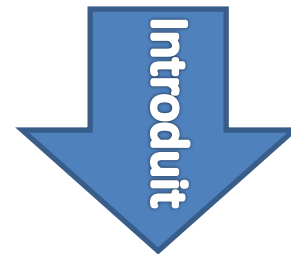
- Code difficile à lire et à maintenir
- Parfois, on ne peut pas traiter tous les cas exceptionnels



**Situation instable**

# Gestion des Exceptions

**Pour éviter ce problème**



**Concept de gestion des exceptions**

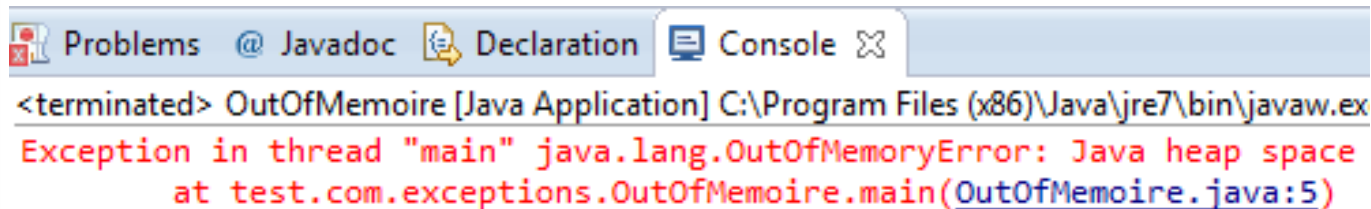


# Exemples des Exceptions

## Exemple 1

```
public class OutOfMemoire {  
    public static void main(String[] args) {  
        String t[]=new String [100000000]; // ligne 5  
        System.out.println(" la suite du programme");  
    }  
}
```

### Le résultat de l'exécution

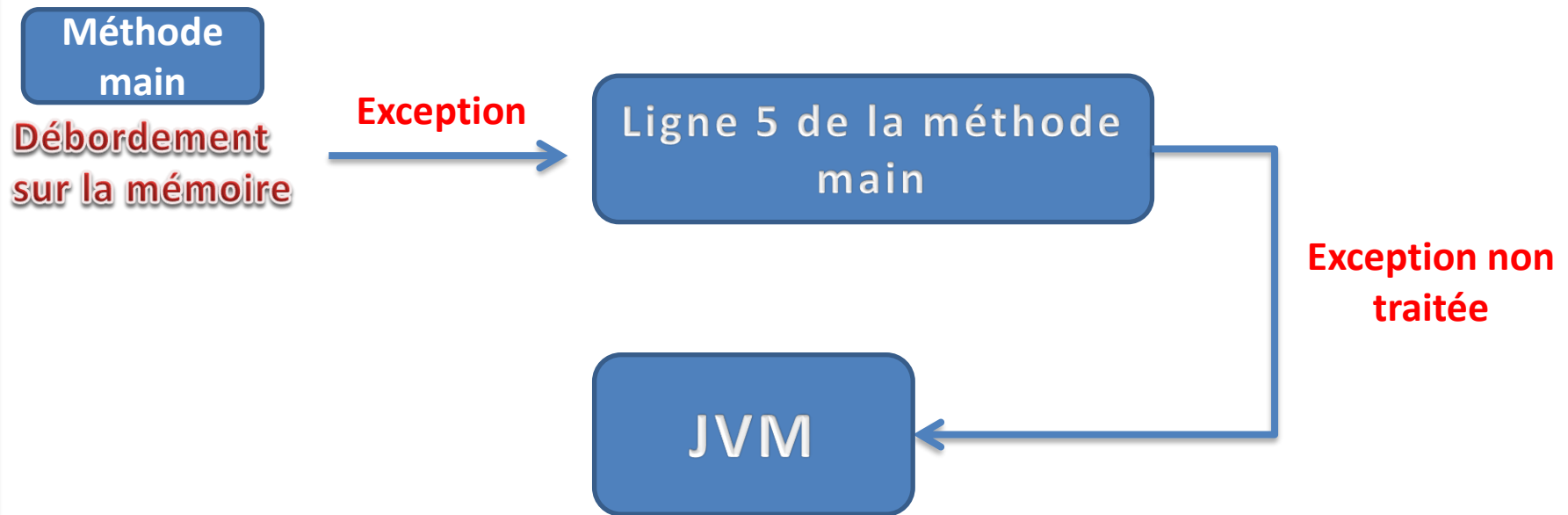


<terminated> OutOfMemoire [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe  
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  
at test.com.exceptions.OutOfMemoire.main(OutOfMemoire.java:5)

Erreur d'exécution à la ligne 5

# Exemples des Exceptions

## Exemple 1



- Génération d'un BUG
- Exécution arrêtée

*{" la suite du programme" }* est non affiché

# Exemples des Exceptions

## Exemple 2

```
public class DivPar0 {  
    public static float calcul(int a,int b){  
        float resultat;  
        resultat=a/b; //ligne 6  
        return resultat;  
    }  
    public static void main(String args[]){  
        int b=0;int a=4;float resultat=0;  
        resultat=calcul(a,b); //ligne 11  
        System.out.println("le résultat est "+resultat);  
        System.out.println("la suite du programme");  
    }  
}
```

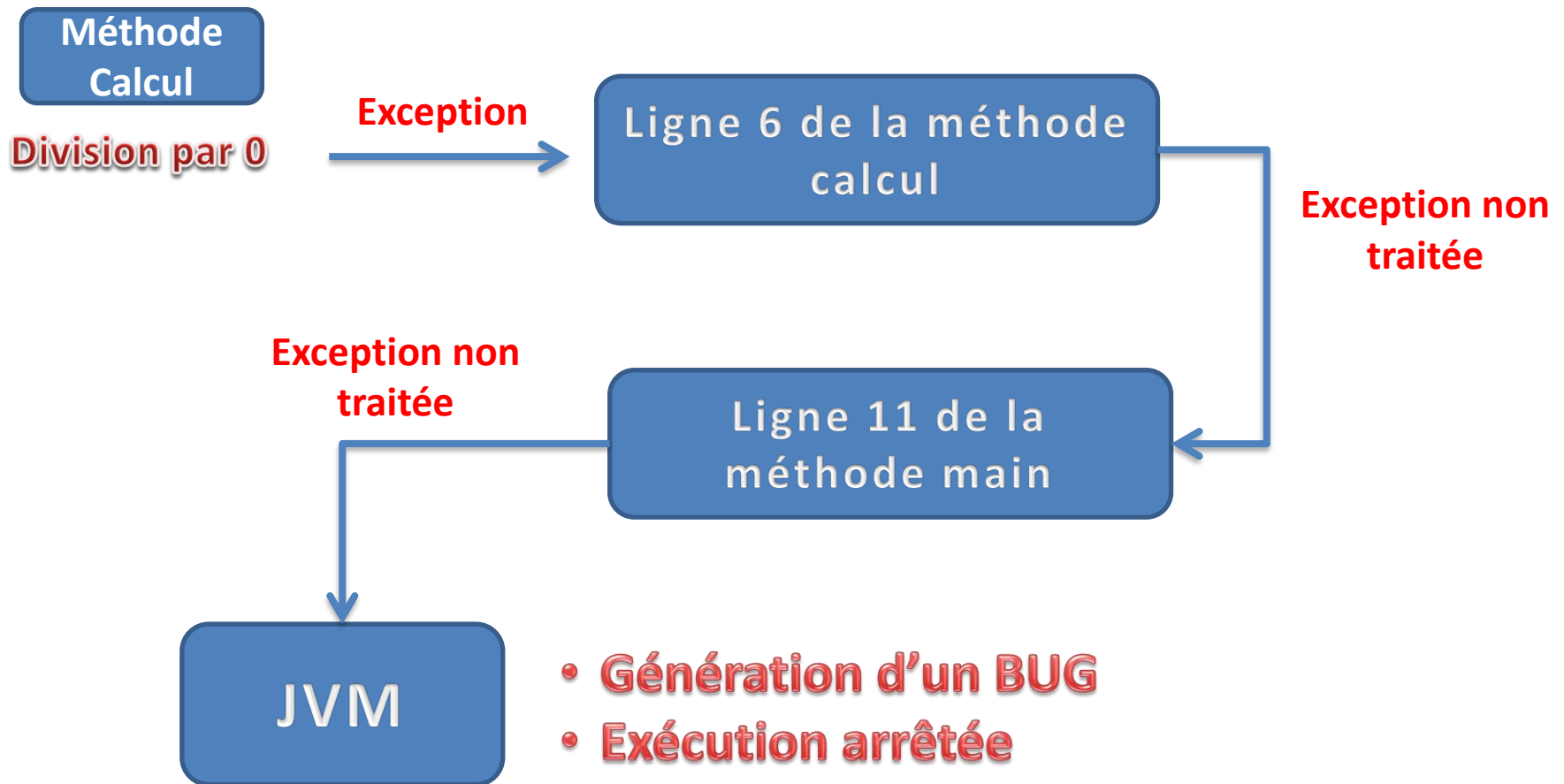
### Le résultat de l'exécution

**Erreur d'exécution à la ligne 6 et la ligne 11**

```
<terminated> DivPar0 [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (16 n  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at test.com.exceptions.DivPar0.calcul(DivPar0.java:6)  
    at test.com.exceptions.DivPar0.main(DivPar0.java:11)
```

# Exemples des Exceptions

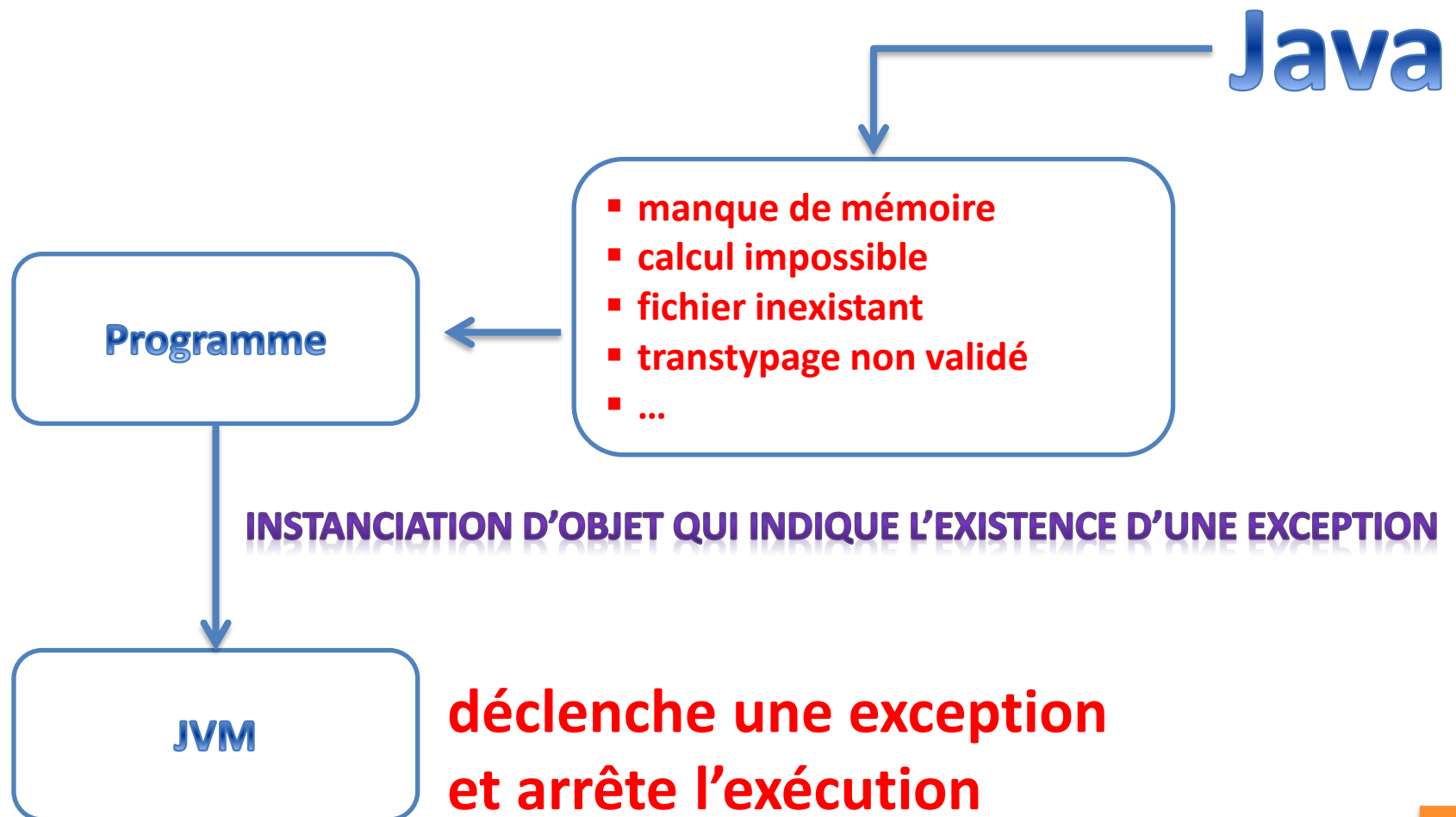
## Exemple 2



*" le résultat est: " +resultat } est non affiché*

# Comportement de JAVA avec les exceptions

## Les exceptions

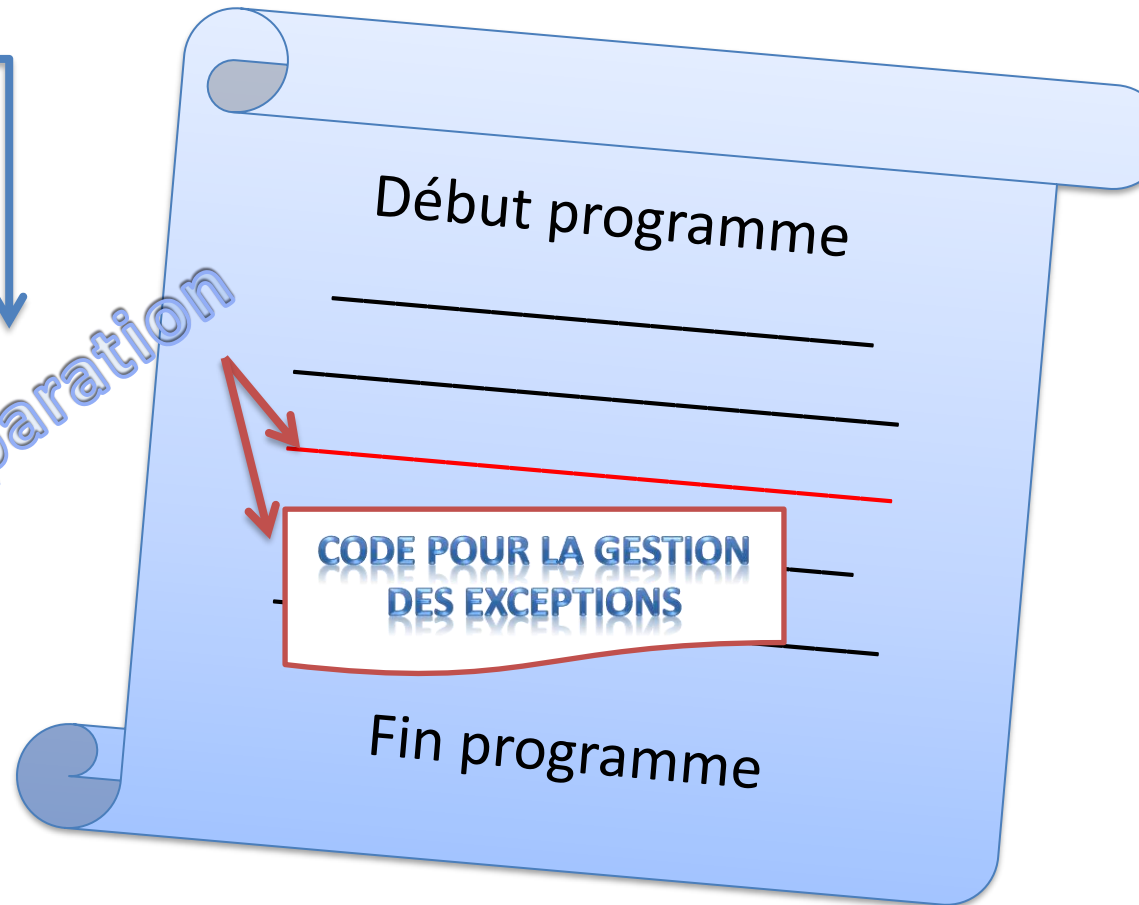


# Gestion des Exceptions



**Gestion des  
exceptions**

Séparation

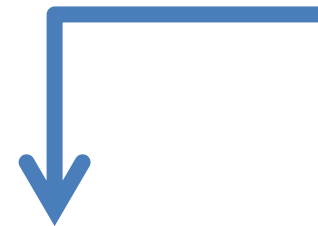


**Améliorer grandement la lisibilité du code**

# Gestion des Exceptions



Gestion des  
exceptions



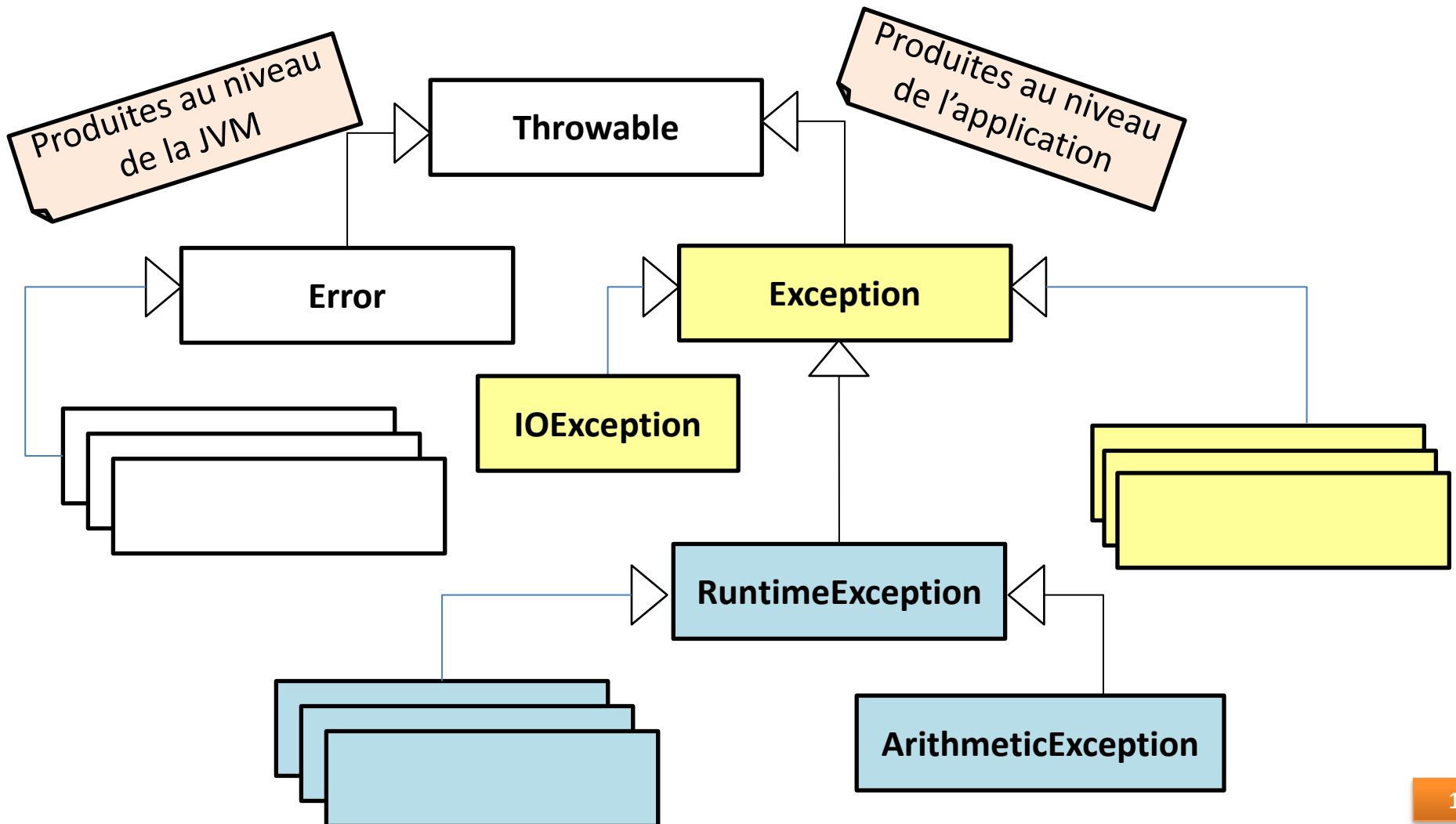
Situation stable

# Gestion des Exceptions





# La hiérarchie des exceptions



# Traitement des Exceptions

## Avec quoi?

- 5 mots clés pour la gestion des Exception

- throws
- throw
- try
- catch
- finally

- Comment les combiner pour gérer une exception?

# Traitement des Exceptions

## Comment? try ... catch...finaly

```
try {  
  <lignes de code à protéger>  
}catch ( UneException E )  
  {<lignes de code réagissant à l'exception UneException > }  
catch ( UneAutreException E )  
  {<lignes de code réagissant à l'exception UneAutreException > }  
catch ( ... )  
  {<...> }  
[finally ( ... )  
  {<...> }]
```

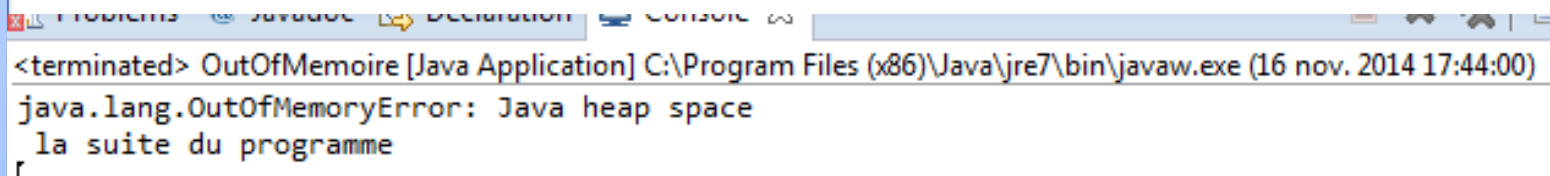
Les types **UneException**, **UneAutreException** sont obligatoirement des classes qui **héritent de la classe Throwable**.

# Traitement des Exceptions

## Traitement de l'exception de l'exemple 1

```
public class OutOfMemoire {  
    public static void main(String[] args) {  
        try{  
            String t[]=new String [100000000];  
        }  
        catch(Error err){  
            System.out.println(err.toString());  
        }  
        System.out.println(" la suite du programme");  
    }  
}
```

### Le résultat de l'exécution



```
<terminated> OutOfMemoire [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (16 nov. 2014 17:44:00)  
java.lang.OutOfMemoryError: Java heap space  
la suite du programme  
r
```

# Traitement des Exceptions

## Traitement de l'exception de l'exemple 2

```
public class DivPar0 {  
    public static float calcul(int a,int b){  
        float resultat;  
        resultat=a/b;  
        return resultat;  
    }  
    public static void main(String args[]){  
        int b=0;int a=4;float resultat=0;  
        try{  
            resultat=calcul(a,b);  
        }  
        catch (Exception ex){  
            System.out.println(ex.toString());  
        }  
        System.out.println("le résultat est "+resultat);  
        System.out.println("la suite du programme");  
    }  
}
```

### Le résultat de l'exécution

```
<terminated> DivPar0 [Java Application] C:\Program Files  
java.lang.ArithmeticException: / by zero  
le résultat est 0.0  
la suite du programme
```

# Les principales méthodes d'une Exception

```
[...]  
catch (Exception ex){  
System.out.println(ex.toString());  
}  
[...]
```

## Le résultat de l'exécution

```
<terminated> DivPar0 [Java Application] C:\Program Files  
java.lang.ArithmeticException: / by zero  
le résultat est 0.0  
la suite du programme
```

```
[...]  
catch (Exception ex){  
System.out.println(ex.getMessage());  
}  
[...]
```

## Le résultat de l'exécution

```
<terminated> DivPar0 [Java Application] C:\Program Files  
/ by zero  
le résultat est 0.0  
la suite du programme
```

```
[...]  
catch (Exception ex){  
ex.printStackTrace();  
}  
[...]
```

## Le résultat de l'exécution

```
<terminated> DivPar0 [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.  
java.lang.ArithmeticException: / by zero  
    at test.com.exceptions.DivPar0.calcul(DivPar0.java:6)  
    at test.com.exceptions.DivPar0.main(DivPar0.java:12)  
le résultat est 0.0  
la suite du programme
```

# Gestion des Exceptions: Application

## Exemple 3: gestion de comptes bancaires

L'objectif est de mettre en place une application de gestion de comptes.

Le compte est caractérisé par un numéro de compte et un solde courant. Et les opérations possibles sur un compte sont:

- La création (initialisation possible pendant la création)
- La consultation de solde
- Le versement d'un montant
- Le retrait d'un montant
  - Les retraits ne sont possible seulement si le solde le permet (le solde doit être supérieur au montant à retirer).
- Questions:
  - modéliser la classe Compte et implémenter les méthodes nécessaires pour sa gestion.
  - Ajouter une classe de test pour mettre en œuvre les méthodes implémentées
- Scénario de mise en œuvre:
  - Création d'un compte numéro 256B301077 avec un montant 3000 DH
    - Versement de 4000 DH puis afficher le solde
    - Retrait de 2500 DH puis afficher le détail du compte
    - Ajouter un autre Retrait de 6000 DH puis afficher le détail du compte

# Gestion des Exceptions: Application

## Exemple 3: Classe Compte sans gestion des exceptions (méthode classique)

```
public class Compte {  
    Private String numero;  
    private float solde;  
    public Compte(String numero,float solde){  
        this.numero=numero;  
        this.solde=solde;  
    }  
    public void verser(float mt) {  
        solde = solde + mt;  
    }  
    public void retirer(float mt) {  
        if (solde > mt)  
            solde = solde - mt;  
        else  
            System.out.println("solde insuffisant!");  
    }  
    public float getSolde() {  
        return solde;  
    }  
    public String toString(){  
        return "["+numero+", "+ getSolde() +"]";  
    }  
}
```

```
public class Main_c {  
    public static void main(String[] args) {  
        Compte cp=new Compte("256B301077",3000);  
        Scanner clavier=new Scanner(System.in);  
        System.out.print("Montant à verser:");  
        float mt1=(float)clavier.nextFloat();  
        cp.verser(mt1);  
        System.out.println(cp.toString());  
        System.out.print("Montant à retirer:");  
        float mt2=clavier.nextFloat();  
        cp.retirer(mt2);  
        System.out.println(cp.toString());  
        System.out.println("la suite du programme!");  
    }  
}
```


**Tester le résultat de ce programme !!**



# Gestion des Exceptions: Application

## Exemple 3: Classe Compte avec gestion des exceptions (méthode 1)

```
public class Compte {  
    private String numero;  
    private float solde;  
    public Compte(String numero,float solde){  
        this.numero=numero;  
        this.solde=solde;  
    }  
    public void verser(float mt) {  
        solde = solde + mt;  
    }  
    public void retirer(float mt) throws Exception {  
        if (solde < mt) throw new Exception("solde insuffisant !");  
        solde = solde - mt;  
        System.out.println("retrait effectué!");  
    }  
    public float getSolde() {  
        return solde;  
    }  
    public String toString(){  
        return "["+numero+", "+ getSolde()+"]";  
    }  
}
```



```
public class Main_c {  
    public static void main(String[] args){  
        Compte cp=new Compte("256B301077",3000);  
        Scanner clavier=new Scanner(System.in);  
        System.out.print("Montant à verser:");  
        float mt1=(float)clavier.nextFloat();  
        cp.verser(mt1);  
        System.out.println(cp.toString());  
        System.out.print("Montant à retirer:");  
        float mt2=clavier.nextFloat();  
        try {  
            cp.retirer(mt2);  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
        System.out.println(cp.toString());  
        System.out.println("la suite du programme!");  
        //exécution continuée  
    }  
}
```

**Le traitement de l'erreur est jeté à la méthode appelante de la méthode retirer, dans ce cas il s'agit de la méthode main**

# Gestion des Exceptions: Application

## Exemple 3: Classe Compte avec gestion des exceptions (méthode 1)

```
public class Compte {
    private String numero;
    private float solde;
    public Compte(String numero,float solde){
        this.numero=numero;
        this.solde=solde;
    }
    public void verser(float mt) {
        solde = solde + mt;
    }
    public void retirer(float mt) throws Exception {
        if (solde < mt) throw new Exception("solde insuffisant !");
        solde = solde - mt;
        System.out.println("retrait effectué!");
    }
    public float getSolde() {
        return solde;
    }
    public String toString(){
        return "["+numero+", "+getSolde()+"]";
    }
}
```

```
public class Main_c {
    public static void main(String[] args){
        Compte cp=new Compte("256B301077",3000);
        Scanner clavier=new Scanner(System.in);
        System.out.print("Montant à verser:");
        float mt1=(float)clavier.nextFloat();
        cp.verser(mt1);
        System.out.println(cp.toString());
        System.out.print("Montant à retirer:");
        float mt2=clavier.nextFloat();
        try {
            cp.retirer(mt2);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        System.out.println(cp.toString());
        System.out.println("la suite du programme!");
    }
}
```

## Le résultat de l'exécution

```
<terminated> Main_c (3) [Java Application] C:\
Montant à verser:3000
[256B301077, 6000.0]
Montant à retirer:8000
solde insuffisant !
[256B301077, 6000.0]
la suite du programme!
```

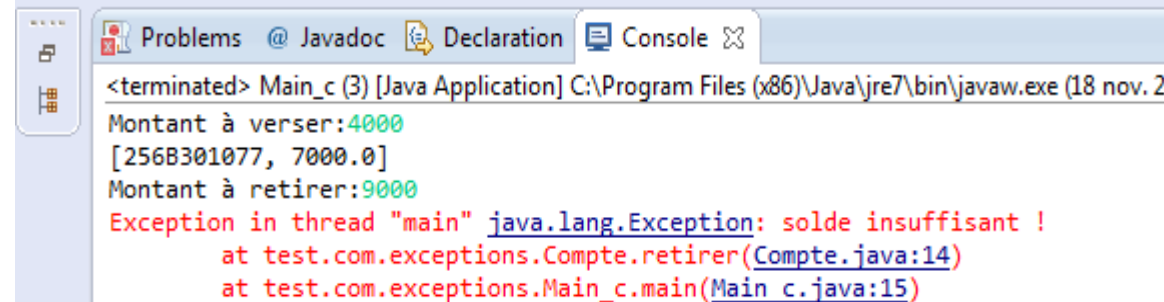
# Gestion des Exceptions: Application

## Exemple 3: Classe Compte avec gestion des exceptions (méthode 2)

```
public class Compte {  
    private String numero;  
    private float solde;  
    public Compte(String numero,float solde){  
        this.numero=numero;  
        this.solde=solde;  
    }  
    public void verser(float mt) {  
        solde = solde + mt;  
    }  
    public void retirer(float mt) throws Exception {  
        if (solde < mt) throw new Exception("solde insuffisant !");  
        solde = solde - mt;  
        System.out.println("retrait effectué!");  
    }  
    public float getSolde() {  
        return solde;  
    }  
    public String toString(){  
        return "["+numero+", "+ getSolde() +"]";  
    }  
}
```

```
public class Main_c {  
    public static void main(String[] args) throws Exception {  
        Compte cp=new Compte("256B301077",3000);  
        Scanner clavier=new Scanner(System.in);  
        System.out.print("Montant à verser:");  
        float mt1=(float)clavier.nextFloat();  
        cp.verser(mt1);  
        System.out.println(cp.toString());  
        System.out.print("Montant à retirer:");  
        float mt2=clavier.nextFloat();  
        cp.retirer(mt2);  
        System.out.println(cp.toString());  
        System.out.println("la suite du programme!");  
    }  
}
```

## Le résultat de l'exécution



```
Problems @ Javadoc Declaration Console  
<terminated> Main_c (3) [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (18 nov. 2  
Montant à verser:4000  
[256B301077, 7000.0]  
Montant à retirer:9000  
Exception in thread "main" java.lang.Exception: solde insuffisant !  
    at test.com.exceptions.Compte.retirer(Compte.java:14)  
    at test.com.exceptions.Main_c.main(Main_c.java:15)
```

# Gestion des Exceptions: Application

## Exemple 3: Classe Compte avec gestion des exceptions (méthode 2)

```
public class Compte {
    private String numero;
    private float solde;
    public Compte(String numero,float solde){
        this.numero=numero;
        this.solde=solde;
    }
    public void verser(float mt) {
        solde = solde + mt;
    }
    public void retirer(float mt) throws Exception {
        if (solde < mt) throw new Exception("solde insuffisant !");
        solde = solde - mt;
        System.out.println("retrait effectué!");
    }
    public float getSolde() {
        return solde;
    }
    public String toString(){
        return "["+numero+", "+ getSolde() +"]";
    }
}
```

À son tour, la méthode main peut **jeter** le traitement de l'exception à un niveau supérieur, dans ce cas il s'agit de la JVM

```
public class Main_c {
    public static void main(String[] args) throws Exception{
        Compte cp=new Compte("256B301077",3000);
        Scanner clavier=new Scanner(System.in);
        System.out.print("Montant à verser:");
        float mt1=(float)clavier.nextFloat();
        cp.verser(mt1);
        System.out.println(cp.toString());
        System.out.print("Montant à retirer:");
        float mt2=clavier.nextFloat();
        cp.retirer(mt2);
        System.out.println(cp.toString());

        System.out.println("la suite du programme!");
        //exécution arrêtée !!
    }
}
```

# Catégorie des Exceptions

## Exception vérifiée (Checked)

- **appelée aussi exception explicite**

**Erreur de l'utilisateur ou d'un problème qui ne peut être prévu par le programmeur.**

### Exemple

**si un fichier doit être ouvert, mais le fichier ne peut être trouvé, une exception se produit.**

**Ces exceptions ne peuvent pas être ignorés au moment de la compilation, mais il doivent être traitées par le programmeur !!**

# Catégorie des Exceptions

## Exception vérifiée (Checked)

- Dans l'exemple de Gestion de compte, nous avons utilisé la classe **Exception** pour la gestion des exceptions. Il s'agit d'une exception vérifiée.
- Pourquoi cette exception est vérifiée: car le compilateur oblige le programmeur à traiter cette exception dans les méthodes appelantes

# Catégorie des Exceptions

## Exception vérifiée (Checked)

- L'exemple génère une erreur de compilation:

```
public class Compte {
    private String numero;
    private float solde;
    public Compte(String numero,float solde){
        this.numero=numero;
        this.solde=solde;
    }
    public void verser(float mt) {
        solde = solde + mt;
    }
    public void retirer(float mt) throws Exception {
        if (solde < mt) throw new Exception("solde insuffisant !");
        solde = solde - mt;
        System.out.println("retrait effectué!");
    }
    public float getSolde() {
        return solde;
    }
    public String toString(){
        return "["+numero+", "+ getSolde() +"]";
    }
}
```

```
public class Main_c {
    public static void main(String[] args){
        Compte cp=new Compte("256B301077",3000);
        Scanner clavier=new Scanner(System.in);
        System.out.print("Montant à verser:");
        float mt1=(float)clavier.nextFloat();
        cp.verser(mt1);
        System.out.println(cp.toString());
        System.out.print("Montant à retirer:");
        float mt2=clavier.nextFloat();
        cp.retirer(mt2);
        System.out.println(cp.toString());
        System.out.println("la suite du programme!");
    }
}
```

```
E:\tps_java>javac Compte.java
E:\tps_java>javac Main_c.java
Main_c.java:13: error: unreported exception Exception; must be caught or declared to be thrown
        cp.retirer(mt2);
                ^
1 error
```

**Le compilateur oblige le programmeur à traiter l'exception Exception, elle est de type vérifié !!!!**

# Catégorie des Exceptions

## Exception non vérifiée (unchecked)

- **Exception non vérifiée (Unchecked):** appelée aussi exception implicite
  - **java.lang.ArithmeticException** ou ses classes dérivées



# Catégorie des Exceptions

- **Exception non vérifiée (unchecked):**

- **L'exemple génère une erreur de compilation:**

```
public class Compte {  
    private String numero;  
    private float solde;  
    public Compte(String numero,float solde){  
        this.numero=numero;  
        this.solde=solde;  
    }  
    public void verser(float mt) {  
        solde = solde + mt;  
    }  
    public void retirer(float mt) throws ArithmeticException{  
        if (solde < mt) throw new ArithmeticException("solde insuffisant !");  
        solde = solde - mt;  
        System.out.println("retrait effectué!");  
    }  
    public float getSolde() {  
        return solde;  
    }  
    public String toString(){  
        return "["+numero+", "+ getSolde() +"]";  
    }  
}
```

```
public class Main_c {  
    public static void main(String[] args){  
        Compte cp=new Compte("256B301077",3000);  
        Scanner clavier=new Scanner(System.in);  
        System.out.print("Montant à verser:");  
        float mt1=(float)clavier.nextFloat();  
        cp.verser(mt1);  
        System.out.println(cp.toString());  
        System.out.print("Montant à retirer:");  
        float mt2=clavier.nextFloat();  
        cp.retirer(mt2);  
        System.out.println(cp.toString());  
        System.out.println("la suite du programme!");  
    }  
}
```

```
E:\tps_java>javac Compte.java  
E:\tps_java>javac Main_c.java  
E:\tps_java>_
```

**Le compilateur ne nous oblige pas de traiter l'exception ArithmeticException, elle est de type non vérifié !!!! Et l'erreur ne sera générée que pendant l'exécution**

# Ordre d'interception d'exceptions hiérarchisées

## Exemple d'application

```
java.lang.Exception
|
+--java.lang.RuntimeException
    |
    +--java.lang.ArithmeticException
        |
        +--java.lang.ArrayStoreException
            |
            +--java.lang.ClassCastException
```

# Ordre d'interception d'exceptions hiérarchisées

## Exemple d'application

```
public class Hirarchie {  
    public static void main(String arg[]){  
        int a=4;  
        int b=0;  
        try{  
            int c=a/b;  
        }catch(RuntimeException ex)  
        {  
            System.out.println(ex.getMessage());  
        }  
        catch(ArithmeticException ex){  
            System.out.println(ex.getMessage());  
        }  
        catch(ArrayStoreException ex){  
            System.out.println(ex.getMessage());  
        }  
        catch(ClassCastException ex){  
            System.out.println(ex.getMessage());  
        }  
    }  
}
```

Ordre d'appel non accepté

# Personnaliser les Exceptions

**Objectif: Créer une classe** personnalisée `SoldeInsuffisantException`

L'exception générée dans la méthode retirer, dans le cas où le solde est insuffisant est une exception métier.

Pour assurer une évolutivité davantage, et améliorer la lisibilité et la maintenance du code, il est plus professionnel de :

- Étendre la classe **Exception** en créant la classe `SoldeInsuffisantException` **pour personnaliser** le traitement de l'exception **d'une manière vérifiée**
- Ou étendre la classe **ArithmeticException** pour une **exception non vérifiée**

```
public class SoldeInsuffisantException extends Exception {  
    public SoldeInsuffisantException(String message) {  
        super(message);  
    }  
}
```

# Personnaliser les Exceptions

## Utilisation de la classe SoldeInsuffisantException

```
public class Compte {
    private String numero;
    private float solde;
    public Compte(String numero,float solde){
        this.numero=numero;
        this.solde=solde;
    }
    public void verser(float mt) {
        solde = solde + mt;
    }
    public void retirer(float mt) throws SoldeInsuffisantException{
        if (solde < mt) throw new SoldeInsuffisantException("solde insuffisant !");
        solde = solde - mt;
        System.out.println("retrait effectué!");
    }
    public float getSolde() {
        return solde;
    }
    public String toString(){
        return "["+numero+", "+getSolde()+"]";
    }
}
```

```
public class Main_c {
    public static void main(String[] args){
        Compte cp=new Compte("256B301077",3000);
        Scanner clavier=new Scanner(System.in);
        System.out.print("Montant à verser:");
        float mt1=(float)clavier.nextFloat();
        cp.verser(mt1);
        System.out.println(cp.toString());
        System.out.print("Montant à retirer:");
        float mt2=clavier.nextFloat();
        try {
            cp.retirer(mt2);
        } catch (SoldeInsuffisantException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(cp.toString());
        System.out.println("la suite du programme!");
        //exécution continuée
    }
}
```

# Personnaliser les Exceptions

## Notre solution peut générer d'autres exceptions ...

```
E:\tps_java>java Main_c
Montant à verser:azerty
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextFloat(Scanner.java:2388)
    at Main_c.main(Main_c.java:8)

E:\tps_java>_
```

- Quelles sont les modifications à apporter à la solution pour prendre en compte cette exception
- Discuter les cas possibles?
- Est-ce que cette exception est vérifiée ou non vérifiée???
- Comment gérer le cas de versement des montant négatifs? Ajouter une exception de type vérifié pour traiter ce cas.

### Discussions