Formation

JAVA Exceptions

m2iformation.fr

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

JAVA

les Exceptions

Exceptions

- 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. Cette situation a deux inconvénients majeurs :
  - Le programmeur peut omettre de tester une condition
  - Le code devient vite illisible car la partie utile est masquée par les tests.

Exceptions

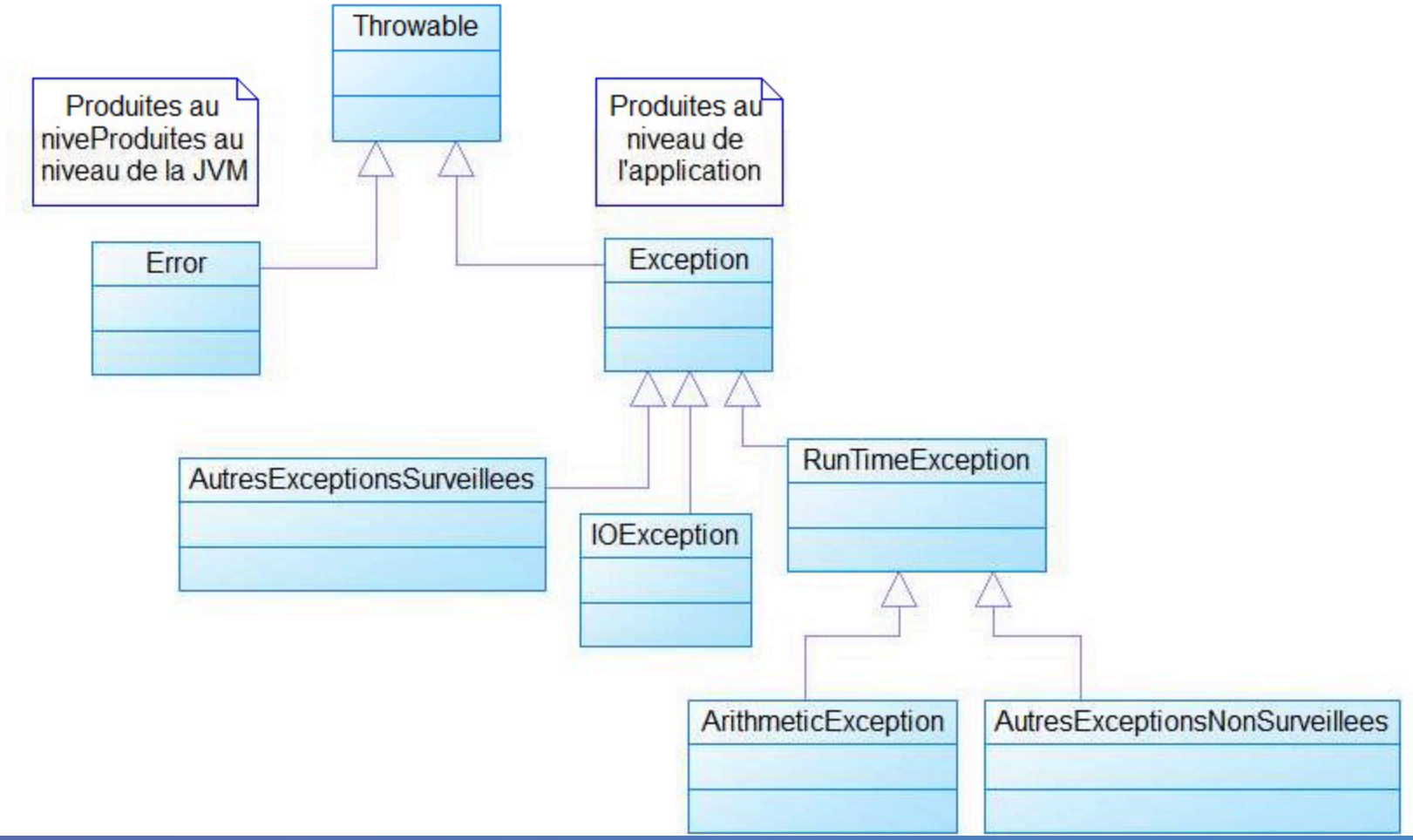
- Java remédie à cela en introduisant un Mécanisme de gestion des exceptions.
- Grâce à ce mécanisme, on peut améliorer grandement la lisibilité du code en séparant le code utile (Code métier) de celui qui traite des situations exceptionnelles.

Hiérarchie des exceptions

Java peut générer deux types d'erreurs au moment de l'exécution :

- Des erreurs produites par l'application dans des cas exceptionnels que le programmeur devrait prévoir et traiter dans son application. Ce genre d'erreur sont de **type Exception**.
- Des erreurs qui peuvent être générée au niveau de la JVM et que le programmeur ne peut prévoir dans son application. Ce type d'erreurs sont de **type Error**.

Hiérarchie des exceptions



Les Exceptions

- En Java, on peut classer les exceptions en deux catégories :
  - **Les exeptions surveillées.**
  - **Les exeptions non surveillées.**
- Java **oblige** le programmeur à **traiter les erreurs surveillées**. Elles sont signalées par le compilateur.
- Les erreurs **non surveillées peuvent être traitées ou non**. Et ne sont **pas signalées par le compilateur**.

Exemple

Considérons une application qui permet de :

- Saisir au clavier deux entiers a et b
- Faire appel à une fonction qui permet de. calculer et de retourner a divisé par b.
- Affiche le résultat.

```java
import java.util.Scanner;
public class App1 {

  public static int calcul(int a,int b){
    int c = a/b;
    return c;
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Donner a: ");
    int a = scanner.nextInt();
    System.out.print("Donner b: ");
    int b = scanner.nextInt();
    int resultat = calcul(a, b);
    System.out.println("Resultat = "+resultat);
  }
}
```

Exécution

- Nous constatons que le compilateur ne signale pas le cas ou b est égal à zero.
- Ce qui constitue un cas fatal pour l'application.
- Voyons ce qui se passe au moment de l'exécution.

Exécution

- Scénario 1 : Le cas normal

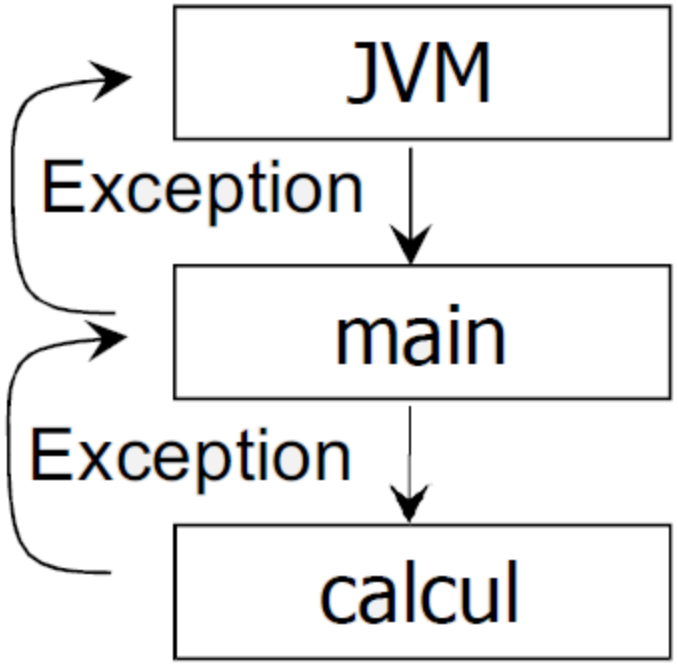
```
Donner a: 12
Donner b: 6
Resultat = 2
```

- Scénario 2: cas ou b=0

```
Donner a:12
Donner b:0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at App1.calcul(App1.java:4)
at App1.main(App1.java:11)
```

Un bug dans l'application

- Le cas du scénario 2 indique que qu'une erreur fatale s'est produite dans l'application au moment de l'exécution.
- Cette exception est de type ArithmeticException. Elle concerne une division par zero
- L'origine de cette exception étant la méthode calcul dans la ligne numéro 4.
- Cette exception n'a pas été traité dans calcul.
- Elle remonte ensuite vers main à la ligne numéro 11 dont elle n'a pas été traitée.
- Après l'exception est signalée à la JVM.
- Quand une exception arrive à la JVM, cette dernière arrête l'exécution de l'application, ce qui constitue un bug fatale.
- Le fait que le message « Resultat= » n'a pas été affiché, montre que l'application ne continue pas son exécution normal après la division par zero



Traiter l'exception

Dans java, pour traiter les exceptions, on doit utiliser le bloc **try catch** de la manière suivante:

```java
import java.util.Scanner;
  public class App1 {
    public static int calcul(int a,int b){
      int c=a/b;
      return c;
    }
  public static void main(String[] args) {
      Scanner clavier=new Scanner(System.in);
      System.out.print("Donner a:");
      int a=clavier.nextInt();
      System.out.print("Donner b:");
      int b=clavier.nextInt();
      int resultat=0;
      try{
        resultat=calcul(a, b);
      }
      catch (ArithmeticException e) {
        System.out.println("Division par zero");
      }
      System.out.println("Resultat="+resultat);
  }
}
```

Scénario 1

```
Donner a:12
Donner b:6
Resultat=2
```

Scénario 1

```
Donner a:12
Donner b:0
Division par zero
Resultat=0
```

Principale Méthodes d'une Exception

Tous les types d'exceptions possèdent les méthodes suivantes :

- getMessage() : retourne le message de l'exception

```
System.out.println(e.getMessage()); // / by zero
```

- toString() : retourne une chaine qui contient le type de l'exception et le message de l'exception.

```
System.out.println(e.toString()); // java.lang.ArithmeticException: / by zero
```

- printStackTrace: affiche la trace de l'exception

```
System.out.println(e.printStackTrace());
/*
Résultat affiché :
java.lang.ArithmeticException: / by zero
at App1.calcul(App1.java:4)
at App1.main(App1.java:13)
*/
```

Exemple : Générer, Relancer ou Jeter une exception surveillée de type Exception

- Considérons le cas d'un compte qui est défini par un code et un solde et sur lequel, on peut verser un montant, retirer un montant et consulter le solde.

```
package metier;
public class Compte {
  private int code;
  private float solde;
  public void verser(float mt){
    solde=solde+mt;
  }
  public void retirer(float mt)throws Exception{
    if(mt>solde) throw new Exception("Solde Insuffisant");
    solde=solde-mt;
  }
  public float getSolde(){
    return solde;
  }
}
```

Utilisation de la classe Compte

- En faisant appel à la méthode retirer, le compilateur signale que cette dernière peut générer une exception de type Exception.
- C'est une Exception surveillée. Elle doit être traitée par le programmeur

```
package pres;
import java.util.Scanner;
import metier.Compte;
public class Application {
  public static void main(String[] args) {
    Compte cp=new Compte();
    Scanner clavier=new Scanner(System.in);
    System.out.print("Montant à verser:");
    float mt1=clavier.nextFloat();
    cp.verser(mt1);
    System.out.println("Solde Actuel:"+cp.getSolde());
    System.out.print("Montant à retirer:");
    float mt2=clavier.nextFloat();
    cp.retirer(mt2); // Le compilateur signal l'Exception
  }
}
```

Traiter l'exception

- Deux solutions :
  - Soit utiliser le bloc try catch

```
try {
  cp.retirer(mt2);
} catch (Exception e) {
  System.out.println(e.getMessage());
}
```

- Ou déclarer que cette exception est ignorée dans la méthode main et dans ce cas là, elle remonte vers le niveau supérieur. Dans notre cas la JVM.

```
public static void main(String[] args) throws Exception {
 // code ...
}
```

Exécution de l'exemple

```
package pres;
import java.util.Scanner;
import metier.Compte;
public class Application {
  public static void main(String[] args) {
    Compte cp=new Compte();
    Scanner clavier=new Scanner(System.in);
    System.out.print("Montant à verser:");
    float mt1=clavier.nextFloat();
    cp.verser(mt1);
    System.out.println("Solde Actuel:"+cp.getSolde());
    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("Solde Final="+cp.getSolde());
  }
}
```

Scénario 1

```
Montant à verser:5000
SoldeActuel:5000.0
Montant à retirer:2000
Solde Final=3000.0
```

Scénario 2

```
Montant à verser:5000
SoldeActuel:5000.0
Montant à retirer:7000
Solde Insuffisant
Solde Final=5000.0
```

Exemple : Générer une exception non surveillée de type RuntimeException

```
package metier;
public class Compte {
  private int code;
  private float solde;
  public void verser(float mt){
    solde=solde+mt;
  }
  public void retirer(float mt){
    if(mt>solde) throw new RuntimeException("Solde Insuffisant");
    solde=solde-mt;
  }
  public float getSolde(){
    return solde;
  }
}
```

Utilisation de la classe Compte

- En faisant appel à la méthode retirer, le compilateur ne signale rien
- C'est une Exception non surveillée. On est pas obligé de la traiter pour que le programme soit compilé.

```
package pres;
import java.util.Scanner;
import metier.Compte;
public class Application {
  public static void main(String[] args) {
    Compte cp=new Compte();
    Scanner clavier=new Scanner(System.in);
    System.out.print("Montant à verser:");
    float mt1=clavier.nextFloat();
    cp.verser(mt1);
    System.out.println("Solde Actuel:"+cp.getSolde());
    System.out.print("Montant à retirer:");
    float mt2=clavier.nextFloat();
    cp.retirer(mt2); // Le compilateur ne signale rien
  }
}
```

Personnaliser les exception métier.

- L'exception générée dans la méthode retirer, dans le cas ou le solde est insuffisant est une exception métier.
- Il est plus professionnel de créer une nouvelle Exception nommée SoldeInsuffisantException de la manière suivante :

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

- En héritant de la classe Exception, nous créons une exception surveillée.
- Pour créer une exception non surveillée, vous pouvez hériter de la classe RuntimeException

Utiliser cette nouvelle exception métier

```
package metier;
public class Compte {
  private int code;
  private float solde;
  public void verser(float mt){
    solde=solde+mt;
  }
  public void retirer(float mt)throws SoldeInsuffisantException{
    if(mt>solde) throw new SoldeInsuffisantException("Solde Insuffisant");
    solde=solde-mt;
  }
  public float getSolde(){
    return solde;
  }
}
```

Application

```
package pres;
import java.util.Scanner;
import metier.Compte;
import metier.SoldeInsuffisantException;
public class Application {
  public static void main(String[] args) {
    Compte cp=new Compte();
    Scanner clavier=new Scanner(System.in);
    System.out.print("Montant à verser:");
    float mt1=clavier.nextFloat();
    cp.verser(mt1);
    System.out.println("Solde Actuel:"+cp.getSolde());
    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("Solde Final="+cp.getSolde());
  }
}
```

Scénario 1

```
Montant à verser:5000
Solde Actuel:5000.0
Montant à retirer:2000
Solde Final=3000.0
```

Scénario 2

```
Montant à verser:5000
Solde Actuel:5000.0
Montant à retirer:7000
Solde Insuffisant
Solde Final=5000.0
```

Scénario 3

```
Montant à verser:azerty
Exception in thread "main"
java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:840)
at java.util.Scanner.next(Scanner.java:1461)
at
java.util.Scanner.nextFloat(Scanner.java:2319)
at pres.Application.main(Application.java:9)
```

Améliorer l'application

- Dans le scénario 3, nous découvrons qu'une autre exception non surveillée est générer dans le cas ou on saisie une chaine de caractères et non pas un nombre.
- Cette exception est de type InputMismatchException, générée par la méthode nextFloat() de la classe Scanner.
- Nous devrions donc faire plusieurs catch dans la méthode main.

Application

```
import org.example.metier.Compte;
import org.example.metier.SoldeInsuffisantException;
import java.util.InputMismatchException;
import java.util.Scanner;
public class Application {
    public static void main(String[] args) {
        Compte cp=new Compte();
            try {
                Scanner clavier = new Scanner(System.in);
                System.out.print("Montant à verser:");
                float mt1 = clavier.nextFloat();
                cp.verser(mt1);
                System.out.println("Solde Actuel:" + cp.getSolde());
                System.out.print("Montant à retirer:");
                float mt2 = clavier.nextFloat();
                cp.retirer(mt2);
            }
            catch (SoldeInsuffisantException e) {
                System.out.println(e.getMessage());
            }
            catch (InputMismatchException e){
                System.out.println("Probléme de saisie");
            }
        System.out.println("Solde Final "+cp.getSolde());
    }
}
```

Un autre cas exceptionnel dans la méthode retirer

- Supposons que l'on ne doit pas accepter un montant négatif dans la méthode retirer.
- On devrait générer une exception de type MontantNegatifException.
- Il faut d'abord créer cette nouvelle exception métier.

Le cas MontantNegatifException

- L'exception métier MontantNegatifException

```
package metier;
public class MontantNegatifException extends Exception {
   public MontantNegatifException(String message) {
      super(message);
   }
}
```

- La méthode retirer de la classe Compte

```
public void retirer(float mt)throws SoldeInsuffisantException,MontantNegatifException {
  if(mt<0) throw new MontantNegatifException("Montant "+mt+" négatif");
  if(mt>solde) throw new SoldeInsuffisantException("Solde Insuffisant");
  solde=solde-mt;
}
```

Application : Contenu de la méthode main

```
Compte cp=new Compte();
try {
  Scanner clavier=new Scanner(System.in);
  System.out.print("Montant à verser:");
  float mt1=clavier.nextFloat();
  cp.verser(mt1);
  System.out.println("Solde Actuel:"+cp.getSolde());
  System.out.print("Montant à retirer:");
  float mt2=clavier.nextFloat();
}
catch (SoldeInsuffisantException e) {
  System.out.println(e.getMessage());
}
catch (InputMismatchException e) {
  System.out.println("Problème de saisie");
}
catch (MontantNegatifException e) {
  System.out.println(e.getMessage());
}
System.out.println("Solde Final=" + cp.getSolde());
```

Scénario 1

```
Montant à verser:5000
SoldeActuel:5000.0
Montant à retirer:2000
Solde Final=3000.0
```

Scénario 2

```
Montant à verser:5000
SoldeActuel:5000.0
Montant à retirer:7000
Solde Insuffisant
Solde Final=5000.0
```

Scénario 3

```
Montant à verser:5000
Solde Actuel:5000.0
Montant à retirer:-2000
Montant -2000.0 négatif
Solde Final=5000.0
```

Scénario 4

```
Montant à verser:azerty
Problème de saisie
Solde Final=0.0
```

Le bloc finaly

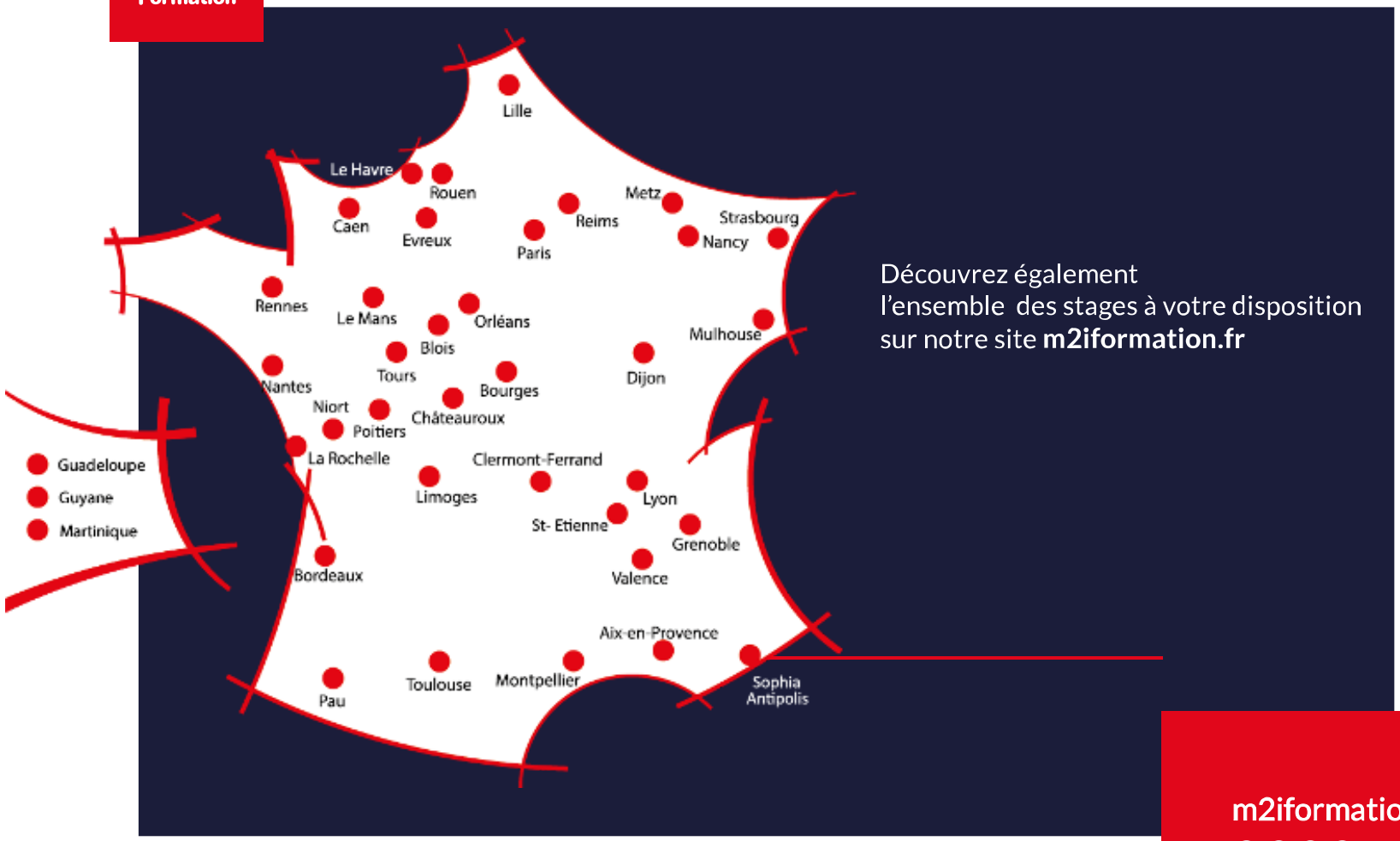
- La syntaxe complète du bloc try est la suivante :

```
try {
  System.out.println("Traitement Normale");
}
catch (SoldeInsuffisantException e) {
  System.out.println("Premier cas Exceptionnel");
}
catch (NegativeArraySizeException e) {
  System.out.println("dexuième cas Exceptionnel");
}
finally{
  System.out.println("Traitement par défaut!");
}
System.out.println("Suite du programme!");
```

- Le bloc finaly s'exécute quelque soit les différents scénarios.

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr