Les exceptions

- En java les exceptions représentent le mécanisme de gestion des erreurs

- Pourquoi des exceptions ?

  - Le code est peu lisible, on ne distingue pas le traitement normal des traitements des cas exceptionnels (qui ont souvent une logique très différente)

  - Des traitements d'erreurs sont souvent oubliés par le programmeur.

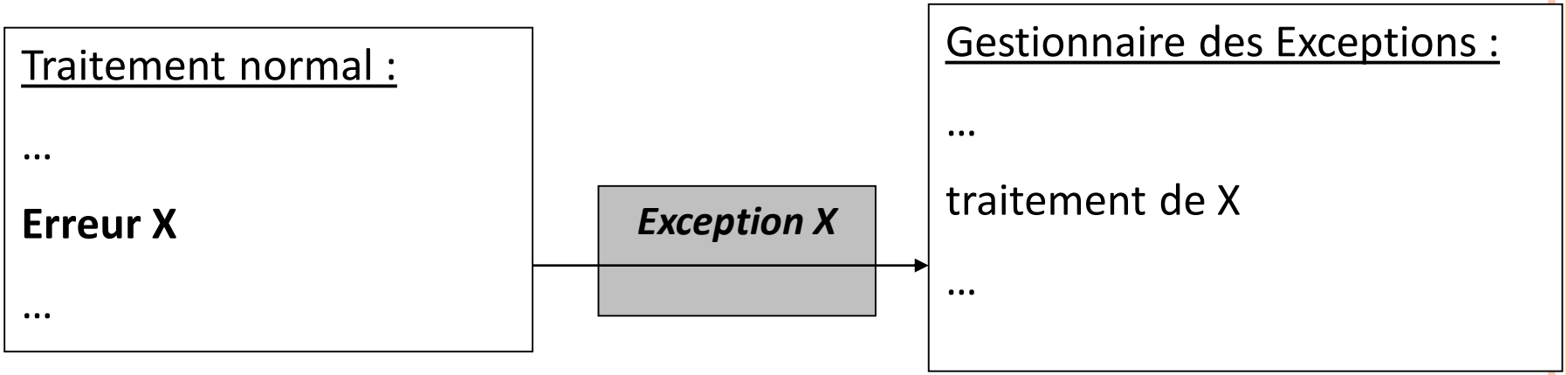
  - Il est souvent difficile de traiter de manière cohérente l'ensemble des erreurs : comment distinguer un résultat valide d'un cas d'erreur
    - Par exemple si le code retour d'une fonction est utilisé pour transmettre le résultat, où doit-on passer le code d'erreur ?

La notion d'exception

- Un mécanisme facilitant le traitement de tous les cas exceptionnels et permettant en particulier de séparer ces traitements des traitements habituels du programme.

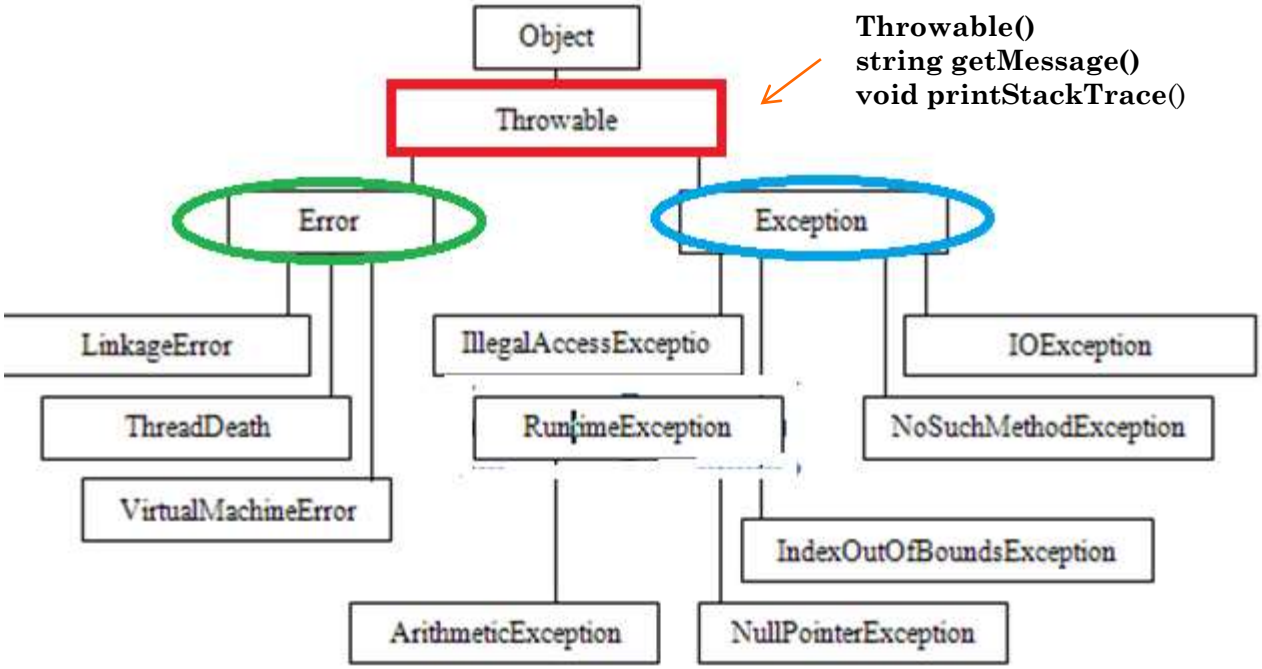
- Un cas exceptionnel est représenté **par un objet** (une exception), qui contient la description du cas exceptionnel, et qui peut être transmis de l'endroit où il a été déclenché jusqu'à celui où l'on sait le traiter (un gestionnaire d'exception).

  Dans Java, contrairement à C++, le traitement des exceptions est vérifié par le compilateur.



Représentation et déclenchement des exceptions

- **Classe Exception et Error**
  - **C'est deux classes dérivent de Throwable**
  - **La classe Error représente une erreur Grave intervenue dans la machine virtuelle Java ou dans un sous système Java**
    - **L'application java s'arête instantanément des l'apparition d'une exception de la classe Error**
  - **La classe Exception représente des erreurs moins grave**
    - Une exception dans Java est un objet appartenant à une classe dérivée de la classe **java.lang.Exception**



Lancer une exception

- Le mot clé opérateur **« throw »**
- Une exception est déclenchée avec le mot clé **throw**, après avoir créé le nouvel objet **Exception** correspondant

```
class MonException extends Exception {
   // constructeur
}
class Test{
  public static void main(String []args)
  {
    ....
    if ( erreurDétectée ) throw new MonException();
    ...
  }
```

Rattraper une exception

- Lorsqu'une exception est déclenchée dans une méthode
  - soit directement avec **throw**,
  - soit parce que une méthode appelée dans cette méthode la déclenche et la propage)
- il y a deux possibilités
  - 1 - On ne traite pas l'exception localement et on la propage.
  - 2 - On traite l'exception localement (try – catch).

1 - Propagation des exceptions

- **le mot clé throws**

  - Toute exception pouvant être propagée par une méthode doit être signalée dans la déclaration de celle-ci.

  - Pour cela on utilise le mot clé **throws** suivi de la liste des exceptions.

class PasDeSolution extends Exception { ...//Constructeur..... }

```
class Equation { /* Equation du second degré ax²+bx+c */
     private double a, b, c ;
     public Equation(double a, double b, double c) { this.a = a ; this.b = b ; this.c = c ; }

public double resultat(){
public double resultat() throws PasDeSolution { // Cette méthode propage une exception
     double discriminent  = b*b-4*a*c ;
     if (discriminent < 0) throw new PasDeSolution() ;
     return ( b + Math.sqrt(discriminent) ) / (2 * a) ;
   }
}
class  test {
    void calcul(){
    void calcul() throws PasDeSolution {
     Equation eq = new Equation(1,0,1) ;
     Eq.resultat() ;
   }
       // Cette méthode doit déclarer la propagation de l'exception PasDeSolution que
     Eq.resultat() peut déclencher, car elle ne la traite pas localement.
}
```

Capture des exceptions : catch, try et finally

- Un gestionnaire d'exception est défini en :
  - Définissant quelles instructions sont surveillées par ce gestionnaire, en plaçant celles-ci dans un bloc d'instructions préfixé par le mot clé **try**.
  - Définissant un ou plusieurs blocs de traitement d'exception préfixés par le mot-clé **catch**.

```
class PasDeSolution extends Exception { ..... }
 class Equation { /* Equation du second degré ax²+bx+c */
    private double a, b, c ;
    public Equation(double a, double b, double c) { this.a = a ; this.b = b ; this.c = c ; }
        public double resultat() throws PasDeSolution { // Cette méthode propage une exception
                        double discriminent  = b*b-4*a*c ;
                        if (discriminent < 0) throw new PasDeSolution() ;
                        return ( b + Math.sqrt(discriminent) ) / (2 * a) ;
}
class test {
   void calcul() {
    try {
            Equation eq = new Equation(1,0,1) ;
            double resultat = Eq.resultat() ;
            System.out.println("Resultat = " + resultat) ;
     }
    catch ( PasDeSolution e ) {
            System.out.println("Pas de solutions") ;
    }
 }
```

```
try {
        ···// 1
}
catch ( Exception e ) { /* permet de capturer tout type d'exception */
        ··· //2
}
```

Il est aussi possible de mettre plusieurs bloc **catch** :

```
try {
        ...
}
catch ( IOException e ) { /* permet de capturer tout type d'exception d'entrée/sorties */
        ...
}
catch ( PasDeSolution e ) {
        ...
}
```

Le bloc finally

- Il est possible de définir un bloc **finally** qui, contrairement au **catch**, n'est pas obligatoire, et qui permet de spécifier du code qui sera exécuté dans tous les cas, qu'une exception survienne ou pas.

- *Ce code sera exécuté même si une instruction **return** est exécutée dans le catch ou le try !*

```
try {
        // Ouvrir un fichier
        // Lire et écrire des données
}
catch ( IOException e ) { /* permet de capturer  tout type d'exception  d'entrée/sorties */
        System.out.println(e) ;
        return ;
}
finally {
        // Fermeture du fichier
}
…// autres traitements  sur les données lues…
```

- Nous constatons dans cet exemple qu'un objet Exception peut être directement affiché avec la méthode System.out.println().
- L'affichage consiste en une chaîne de caractères expliquant la nature de cette exception.

LA MÉTHODE **PRINTSTACKTRACE()**

```
catch ( IOException e ) { /* permet de capturer tout type
    d'exception d'entrée/sorties */
        System.out.println("Erreur du type : " + e) ;
        e.printStackTrace() ;
    }
```

- Il est aussi possible en appliquant la méthode **printStackTrace**() sur l'objet exception,
  - d'afficher la pile des appels de méthodes remontées depuis le déclenchement de cette exception

EXERCICE

1- public class EntierNaturel
{
private int Nbr ;
// constructeur
public EntierNaturel (int Nbr) **throws ExceptNeg**
{
// il existe une exception lorsque le nombre donné est négatif
// on peut générer une exception qu'on peut appeler **ExceptNeg**
if (Nbr < 0) **throw new ExceptNeg(Nbr) ;**
this.Nbr = Nbr ;
}
public int getNbr () { return Nbr ;}
}

2- **public class ExceptNeg extends Exception**
{
private int value;
// constructeur
public ExceptNeg (int value)
{ this.value = value ;}
public int getValue() { return value;}
}

```
3-  public class Testentiernaturel
 {
public static void main (String args[])
 {
// on peut tester en changeant les valeurs tels
que 100, -50…
int n;
Scanner C = new Scanner(System.in);
System.out.println("donner un entier
naturel : ") ;
n = C.lireInt() ;
try
{
 EntierNaturel Nbr1 = new EntierNaturel (n);
 System.out.println("le nombre naturel est
 valide et =" + Nbr1.getNbr()) ;
 // la question à penser : pourquoi on a utilisé
 getNbr et non pas Nbr1.Nbr ?
}
catch ( ExceptNeg e)
{
 Sytem.out.println ("**********erreur de
 construction du nombre********** ") ;}
finally
 {      System.exit(-1) ;}
}
}
```

```
4- public class EntierNaturel
{ …
 public  static EntierNaturel somme (EntierNaturel
 Nbr1, EntierNaturel Nbr2) throws ExceptSom,

 ExceptNeg
 {
 int op1 = Nbr1.Nbr; int op2= Nbr2.Nbr;
 int S = op1 + op2 ;
 if (S > Interger.Max_Value) throw new ExceptSom
 (op1,op2);
 EntierNaturel res = new EntierNaturel (S);
 return res;
}
public  static EntierNaturel diff (EntierNaturel Nbr1,
EntierNaturel Nbr2) throws ExceptDiff, ExceptNeg
{
   int op1 = Nbr1.Nbr; int op2= Nbr2.Nbr;
   int D = op1 - op2 ;
   if (D<0) throw new ExceptDiff (op1,op2);
   // autre façon en utilisant classe anonyme
   return new EntierNaturel(D);
}
```

5 - **public class ExceptSom extends Exception**

```
{
        public int value1, value2;
        // constructeur
        public ExceptSom (int value1, int value2)
        { this.value1 = value1 ;
        this.value1 = value1 ;}
}
```

public class ExceptDiff extends Exception

```
{
        public int value1, value2;
        // constructeur
        public ExceptDiff (int value1, int value2)
        { this.value1 = value1 ;
        this.value1 = value1 ;}
}
```

//une autre façon est de créer une classe Except_op et des sous classes ExceptSom et ExceptDiff, et appeler super(value1, value2)

6 - On peut définir une classe : class ExceptNaturel extends Exception {}

Les autres seront des sous-classes :

```
class ExceptNeg extends ExceptNaturel {……}
class Except_op extends ExceptNaturel{……}
class ExceptSom extends Except_op {……..}
class ExceptDiff extends Except_op {……}
```

```
public class Test2entiernaturel
{
 public static void main (String args[])
 {
  // on peut tester en changeant les valeurs tels que 100, -50…
  int n1, n2;
  System.out.println("donner un premier entier naturel : ") ;
  n1 = Clavier.lireInt() ;
  System.out.println("donner un second entier naturel : ") ;
  n2 = Clavier.lireInt() ;
  try{
      EntierNaturel Nbr1 = new EntierNaturel (n1);
      EntierNaturel Nbr2 = new EntierNaturel (n2);
      EntierNaturel Som =  EntierNaturel.somme (n1,n2) ;
      EntierNaturel Di =  EntierNaturel.diff (n1,n2) ;
 }
// exception sans différentiation
 catch ( ExceptNaturel e)
 { Sytem.out.println ("****erreur entier naturel******* ") ;}
// test avec différentiation des exceptions
  try {
         EntierNaturel Nbr1 = new EntierNaturel (n1);
        EntierNaturel Nbr2 = new EntierNaturel (n2);
        EntierNaturel Som =  EntierNaturel.somme (n1,n2) ;
        EntierNaturel Di =  EntierNaturel.diff (n1,n2) ;}
 catch ( ExceptNeg e)
 { Sytem.out.println ("erreur construction entier
                        naturel "+e.getValue()) ;   }
 catch ( ExceptSom e)
{ Sytem.out.println ("erreur somme des entiers
                      naturels "+e.value1 +" "+e.value2 ) ;  }
catch ( ExceptDiff e)
{ Sytem.out.println ("erreur difference des entiers
                       naturels "+e.value1 +" "+e.value2 ) ; }
Finally      {System.exit(-1) ;}     }}
```