La gestion des exceptions

La gestion des exceptions

- Principe, type d'exceptions, propagation, création d'exception personnalisée

Anomalies potentielles

Problèmes liés au matériel : par exemple la perte subite d'une connexion à un port, un disque défectueux, ...

Actions imprévues de l'utilisateur, entrainant par exemple une division par zéro, un calcul impossible, un fichier inexistant, un transtypage non valide, ...

Débordements de stockage dans les structures de données, manque de mémoire, ...

➔ Prévoir une réponse adaptée à chaque type de situation

Notion de logiciel « robuste »

Intérêt de la gestion des exceptions

- Exemple : vous écrivez une fonction qui réalise un calcul, elle renvoie –1 si ce calcul est faux ou impossible, et la valeur résultat si le calcul est juste.
  - La même variable de retour est donc utilisée pour un résultat calculé et/ou une erreur.
- Le mécanisme d'exception est un mécanisme puissant offrant **deux canaux d'information** lors de l'exécution d'une méthode :
  - la valeur de retour correspond au régime normal de fonctionnement
  - l'exception correspond au régime d'erreur.

Rôle des exceptions

- Une exception est chargée de signaler un comportement ***exceptionnel*** (mais prévu) d'une partie spécifique d'un logiciel.
- Les exceptions font partie du langage lui-même. Dans Java, les exceptions constituent une classe particulière: la classe **Exception**.
  - Cette classe contient un nombre important de classes dérivées
- **Comment agit une exception ?**
- Dès qu'une erreur se produit, un objet de la classe adéquate dérivée de la classe **Exception** est instancié
  - Le logiciel " ***déclenche une exception*****"** et la traite ensuite

DEUX TYPES EXCEPTIONS

- Exceptions **implicites**, directement gérées par la MVJ
  - Anomalies qui interviennent lors de l'exécution '`RunTime Exception`') ou liées au matériel/logiciel '`Error Exception`')
  - Division par zéro, dépassement de tableau, etc.

- Exceptions **explicites**
  - Anomalies liées à l'application
  - Gestion à prévoir dans le code

→ Avantage : séparation nette du **traitement d'erreur** du code normal

Premier exemple d'exception

Comment déclencher une exception avec throw

une classe ***Point***

- Supposons que l'on ne souhaite manipuler que des points ayant des coordonnées non négatives.

- Nous pouvons, au sein du constructeur, vérifier la validité des paramètres fournis. Lorsque l'un d'entre eux est incorrect, nous "déclenchons" une exception à l'aide de l'instruction ***throw***.

- Nous créons donc (un peu artificiellement) une classe que nous nommerons ***ErrCoord***. Java impose que cette classe dérive de la classe standard ***Exception***.

```
class ErrConst extends Exception
{ }
```

Premier exemple d'exception

- Pour lancer une exception de ce type au sein de notre constructeur, nous fournirons à l'instruction *throw* un objet de type *Errconst*, par exemple de cette façon :

```
throw new ErrConst() ;
```

Le constructeur de notre classe *Point* peut se présenter ainsi :

class Point

```
{ public Point(int x, int y) throws ErrConst
    { if ( (x<0) || (y<0)) throw new ErrConst() ; // lance une exception de
    this.x = x ; this.y = y ; // type ErrConst
     }
```

Notez la présence de *throws ErrConst*, dans l'en-tête du constructeur, qui précise que la méthode est susceptible de déclencher une exception de type *ErrConst*. Cette indication est obligatoire en Java

Premier exemple d'exception

Exemple d'une classe Point *dont le constructeur déclenche une exception* ErrConst

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst() ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + " " + y) ;
}
private int x, y ;
}
class ErrConst extends Exception
{ }
```

Premier exemple de gestion d'exception

Utilisation d'un gestionnaire d'exception

- pour gérer convenablement les éventuelles exceptions de type *ErrConst* que son emploi peut déclencher. Pour ce faire, il faut :

1) Inclure dans un bloc particulier dit "bloc *try*" les instructions dans lesquelles on risque de voir déclenchée une telle exception

```
try
{
// instructions
}
```

2) Faire suivre ce bloc de la définition des différents gestionnaires d'exception

Chaque définition de gestionnaire est précédée d'un en-tête introduit par le mot-clé ***catch***

```
catch (ErrConst e)
{ System.out.println ("Erreur construction ") ;
System.exit (-1) ;
}
```

Premier exemple de gestion d'exception

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst() ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x +
" " + y) ;
}
private int x, y ;
}
class ErrConst extends Exception
{ }
```

```
public class Except1
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println ("Erreur construction
") ;
System.exit (-1) ; }}}
```

```
coordonnees : 1 4
Erreur construction
```

Gestion de plusieurs exceptions

un exemple plus complet dans lequel on peut déclencher et traiter deux types d'exceptions

- Du constructeur précédent, déclenchant toujours une exception ***ErrConst***,
- Méthode *deplace* qui s'assure que le déplacement ne conduit pas à une coordonnée négative ; si tel est le cas, elle déclenche une exception *ErrDepl*

(on crée donc, ici encore, une classe *ErrDepl*) :

```
public void deplace (int dx, int dy) throws
ErrDepl
{ if ( ((x+dx)<0) || ((y+dy)<0)) throw new
ErrDepl() ;
x += dx ; y += dy ;
}
```

Nous pouvons détecter les deux exceptions potentielles ***ErrConst*** et ***ErrDepl*** (ici, nous nous contentons comme précédemment d'afficher un message et d'interrompre l'exécution) :

Gestion de plusieurs exceptions

```
try
{ // bloc dans lequel on souhaite detecter les exceptions ErrConst et ErrDepl
}

catch (ErrConst e) // gestionnaire de l'exception ErrConst
{ System.out.println ("Erreur construction ") ;
System.exit (-1) ;
}

catch (ErrDepl e) // gestionnaire de l'exception ErrDepl
{ System.out.println ("Erreur deplacement ") ;
System.exit (-1) ;
}
```

Gestion de deux exceptions

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst() ;
this.x = x ; this.y = y ;
}
public void deplace (int dx, int dy) throws ErrDepl
{ if ( ((x+dx)<0) || ((y+dy)<0)) throw new
ErrDepl() ;
x += dx ; y += dy ;
}

public void affiche()
{ System.out.println ("coordonnees : " + x + " "
+ y) ;}
private int x, y ;
}

class ErrConst extends Exception
{ }
class ErrDepl extends Exception
{ }
```

```
public class Except2
{ public static void main (String
args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a.deplace (-3, 5) ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println ("Erreur
construction ") ;
System.exit (-1) ;
}
catch (ErrDepl e)
{ System.out.println ("Erreur
deplacement ") ;
System.exit (-1) ;
}}}
```

```
coordonnees : 1 4
Erreur deplacement
```

Transmission d'information au gestionnaire d'exception

On peut transmettre une information au gestionnaire d'exception :

- par le biais de l'objet fourni dans l'instruction *throw*,
- par l'intermédiaire du constructeur de l'objet exception.

Par l'objet fourni à l'instruction throw

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst(x, y)
;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + "
" + y) ;
}
private int x, y ;
}
class ErrConst extends Exception
{ ErrConst (int abs, int ord)
{ this.abs = abs ; this.ord = ord ;
}
public int abs, ord ;
}
```

Exemple de transmission d'information à un gestionnaire d'exception 1

```
public class Exinfo1
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println ("Erreur
construction Point") ;
System.out.println (" coordonnees
souhaitees : " + e.abs + " " + e.ord) ;
System.exit (-1) ;
}}}
```

```
coordonnees : 1 4
Erreur construction Point
coordonnees souhaitees : -3 5
```

Par le constructeur de la classe exception

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0))
throw new ErrConst("Erreur construction avec
coordonnees " + x + " " + y) ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + "
" + y) ;
}
private int x, y ;
}
class ErrConst extends Exception
{ ErrConst (String mes)
{ super(mes) ;
}
}
```

Exemple de transmission d'information au gestionnaire d'exception 2

```
public class Exinfo2
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println (e.getMessage())
;
System.exit (-1) ;
}
}
}
```

```
coordonnees : 1 4
Erreur construction avec coordonnees -3 5
```

Mécanisme de gestion des exceptions

- la poursuite de l'exécution après le traitement d'une exception par le gestionnaire,

- le choix du gestionnaire,

- le cheminement des exceptions, c'est-à-dire la manière dont elles peuvent remonter d'uneméthode à une méthode appelante,

- les règles d'écriture de la clause *throws*,

- les possibilités de redéclencher une exception,

- l'existence d'un bloc particulier dit *finally.*

Mécanisme de gestion des exceptions

Dans les exemples précédents; le gestionnaire d'exception mettait fin à l'exécution du programme en appelant la méthode ***System.exit***.

Cela n'est pas une obligation ; en fait, après l'exécution des instructions du gestionnaire, l'exécution se poursuit simplement avec les instructions suivant le bloc *try,*

Exemple:

Mécanisme de gestion des exceptions

Poursuite de l'exécution

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0))
throw new ErrConst("Erreur construction avec
coordonnees " + x + " " + y) ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + "
" + y) ;
}
private int x, y ;
}
class ErrConst extends Exception
{ ErrConst (String mes)
{ super(mes) ;
}
}
```

```
avant bloc try
coordonnees : 1 4
Erreur deplacement
apres bloc try
```

```
public class Suitexecusion
{ public static void main (String args[])
{ System.out.println ("avant bloc try") ;
try
{
Point a = new Point (1, 4) ;
a.affiche() ;
a.deplace (-3, 5) ;
a.affiche() ;}

catch (ErrConst e)
{ System.out.println ("Erreur
construction ") ;}

catch (ErrDepl e)
{ System.out.println ("Erreur
deplacement ") ;}

System.out.println ("apres bloc try") ;
}}
```

Mécanisme de gestion des exceptions

Cheminement des exceptions

Lorsqu'une méthode déclenche une exception, on cherche tout d'abord un gestionnaire dans l'éventuel bloc *try* contenant l'instruction *throw* correspondante.

Si l'on n'en trouve pas ou si aucun bloc *try* n'est prévu à ce niveau, on poursuit la recherche dans un éventuel bloc *try* associé à l'instruction d'appel dans une méthode appelante, et ainsi de suite….

Le gestionnaire est rarement trouvé dans la méthode qui a déclenché l'exception puisque l'un des objectifs fondamentaux du **traitement d'exception** est précisément de **séparer déclenchement et traitement** !

Mécanisme de gestion des exceptions

Redéclenchement d'une exception

Dans un gestionnaire d'exception, il est possible de demander que, malgré son traitement, l'exception soit retransmise à un niveau englobant, comme si elle n'avait pas été traitée.

```
try
{ .....
}
catch (Excep e) // gestionnaire des exceptions de type
Excep
{ .....
throw e ; // on relance l'exception e de type Excep
}
```

Exemple:

Mécanisme de gestion des exceptions

Redéclenchement d'une exception

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<=0) || (y<=0)) throw new ErrConst()
;
this.x = x ; this.y = y ;
}
public void f() throws ErrConst
{ try
{ Point p = new Point (-3, 2) ;
}
catch (ErrConst e)
{ System.out.println ("dans catch (ErrConst)
de f") ;
throw e ; // on repasse l'exception a un
niveau superieur
}}
private int x, y ;
}
class ErrConst extends Exception
{ }
```

```
public class Redecl
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.f() ;
}
catch (ErrConst e)
{ System.out.println ("dans catch
(ErrConst) de main") ;
}
System.out.println ("apres bloc try
main") ;
}
}
```

```
dans catch (ErrConst) de f
dans catch (ErrConst) de main
apres bloc try main
```