EXCEPTIONS LOGICIELLES

Mise en œuvre en Java

Introduction

- Exemple : la simple ouverture d'un fichier peut provoquer beaucoup d'erreurs telles que
  - l'inexistence du fichier,
  - un mauvais format,
  - une interdiction d'accès,
  - une erreur de connexion au périphérique, …
- Le programme est robuste, si toutes les erreurs possibles sont détectées et traitées.
- Objectifs
  - Fiabilité et «tolérance aux pannes»
  - Séparer l'essentiel de l'exceptionnel dans les traitements

Exceptions

- En java les exceptions sont des objets
  - Elles sont toutes du type `Exception`
    - leur type "précis" est un "sous-type" de `Exception`
    - les classes d'exceptions se nomment par convention *QuelqueChose*`Exception`
- Exceptions pré-programmées :
  - exceptions du langage et de ses bibliothèques
- Programmer ses propres exceptions :
  - exceptions utilisateur

Mécanisme d'exception

- une portion de code peut émettre/lancer une exception
  - si un événement d'ordre "exceptionnel" se produit.
    - (les erreurs rentrent dans cette catégorie)
  - La méthode en question ne renvoie alors pas de valeur de retour, mais émet une exception expliquant la cause de cette émission.
- le programmeur dispose d'un moyen pour capturer une exception et proposer une alternative/solution

Processus de propagation

- La propagation d'une exception se déroule selon les étapes suivantes :

1. Une exception est générée à l'intérieur d'une méthode ;
2. Si la méthode prévoit un traitement de cette exception, on va au point 4, sinon au point 3 ;
3. L'exception est renvoyée à la méthode ayant appelé la méthode courante, on retourne au point 2 ;
4. L'exception est traitée et le programme reprend son cours après le traitement de l'exception.

- Exemple :
  - Si une méthode `a()` appelle une méthode `b()` qui appelle une méthode `c()` qui appelle une méthode `d()`, et que cette méthode `d()` lève une exception,
  - celle-ci est d'abord transmise à `c()` qui peut l'intercepter ou la transmettre à `b()`, qui peut aussi l'intercepter ou la transmettre à `a()`.

Interception (capture)

- Via une "mise sur écoute" d'une portion de code.
  - Utilisation du mot-clé `try` suivi du bloc à surveiller.
    - Si aucune exception ne se produit dans le bloc correspondant : exécution normale du programme.
    - Si une exception est levée, le traitement de l'exception est exécuté puis l'exécution du programme reprend son cours après le bloc testé.
- Les classes d'exception interceptées dans le bloc testé doivent être précisées.
  - Utilisation du mot-clé `catch` suivi de la classe concernée, d'un nom de variable correspondant à l'objet exception, puis du traitement.
- Si une exception est levée sans qu'aucune interception ne soit prévue pour sa classe, celle-ci est propagée à la méthode appelante.

Capture d'exception

```
try {
 // code susceptible de générer des exceptions
} catch (ExceptionType1 id) {
 // traitant
} catch (ExceptionType2 id) {
 // traitant
} ...
```

- Mécanisme

  Quand une exception survient dans un bloc :
  - l'exécution normale est arrêtée (pas le programme)
  - l'exception est propagée jusqu'au premier bloc capable de la capturer (catch)
  - selon l'imbrication lexicale des blocs puis de la pile des messages et éventuellement jusqu'au système si aucune capture n'est assurée (arrêt du programme).

Cas de non capture

- Si une méthode contient une portion de code susceptible de lancer une exception et que l'on ne souhaite pas (ou ne peut pas) traiter l'exception dans le corps de la méthode, il est nécessaire d'informer l'utilisateur de la méthode que celle-ci peut générer une exception

  "throws ClasseDException " dans la signature de la méthode

```
public class AClass {
  private String name;
  ...
  public int suffixe(int longueurSuffixe) throws
  IndexOutOfBoundsException {
    return this.name.substring(this.name.length()-
    longueurSuffixe);
  }
}
```

Déclaration

- Déclaration, pour chaque méthode, des classes d'exception qu'elle est susceptible d'émettre.
  - Dans la signature de la méthode
  - Mot-clé `throws` suivi des classes d'exceptions (séparées par une virgule s'il en existe plusieurs)
- Exemple (La méthode parseInt de la classe Integer) :

```
public static int parseInt(String s) throws
NumberFormatException {
...
}
```

  - Conversion d'une chaîne de caractères, contenant uniquement des chiffres, en un entier.
    - Erreur si cette chaîne de caractères ne contient pas que des chiffres.
    - Emission d'une exception de la classe `NumberFormatException`.

Emission / lever d'exception

- Une exception peut être émise dans une méthode de deux manières :
  - par une autre méthode appelée dans le corps de la première méthode ;
  - par la création d'un objet instanciant la classe Exception (ou la classe Throwable) et la levée explicite de l'exception en utilisant le mot-clé throw.
- L'exemple du second cas :

```
public class ExempleException {
   public static String month(int mois) throws
   IndexOutOfBoundsException {
      if ((mois < 1) || (mois > 12)) {
        throw new IndexOutOfBoundsException(
        "le numero du mois qui est " + mois
        + " doit être compris entre 1 et 12");
      }
      if (mois == 1) return "Janvier";
      else if (mois == 2) return "Février";
      ...
      else if (mois == 11) return "Novembre";
      else return "Décembre";
   }
}
```

Dans cet exemple, la détection et la formulation de l'erreur sont codées dans la méthode mais pas son traitement.

Représentation

- En Java les exceptions sont représentées par des objets
  - décrits par des classes telles que celles ci-dessus
  - Instanciées lorsque l'exception survient.
- L'une des principales méthodes : `printStackTrace()` qui affiche la pile des messages qui a conduit à l'exception.
- Par défaut cette méthode est appliquée si l'exception n'est pas capturée (remontée jusqu'au système).

Exception du langage : exemple

```
class Pile {
 int espace[];
 int sommet=-1;
 Pile(int taille) {
      espace = new int[taille];
 }
 void printEtat() {...}
 void empiler(int x) {
      try {
      sommet +=1;
      espace[sommet]=x;
      } catch (ArrayIndexOutOfBoundsException ex){
            printEtat();
      }
 }
}
```

Classe d'exception

- Une classe est considérée comme une classe d'exception dès lors qu'elle hérite de la classe `Throwable`.

Classe	Description
AWTException	Erreur lors d'opérations de type graphique.
ClassCastException	Erreur lors de la conversion d'un objet en une classe incompatible avec sa vraie classe.
FileNotFoundException	Tentative d'ouverture d'un fichier inexistant.
IndexOutOfBoundsException	Tentative d'accession à un élément inexistant dans un ensemble.
IOException	Erreur lors d'opérations d'entrées/sorties.
NullPointerException	Un pointeur null est reçu par une méthode n'acceptant pas cette valeur, ou appel d'une méthode ou d'une variable à partir d'un pointeur null.

Exceptions utilisateur

- Sous-classer la classe `Exception`
- Provoquer explicitement l'exception par l'instruction: `throw <objet exception>`
- Spécifier l'exception dans la déclaration de la méthode provocante :

```
<methode> throws
 <ClasseException>[,<ClasseException>]* {…}
```

- Attention : cette déclaration fait partie du profil de la méthode et doit être respectée en cas de redéfinition.

Exception utilisateur : exemple

```
class PilePleineException extends Exception {}

class Pile {
 boolean pleine() {return (sommet==espace.length-1);}
 void vider() {...}
 void empiler(int x) throws PilePleineException {
      if (pleine()) throw new PilePleineException();
      else {sommet +=1;espace[sommet]=x;}}}

class Client {
 Pile p = new Pile(N);
 void appli(int x) {
      //throws PilePleineException si non traitee
      try {
      p.empiler(x);
      } catch (PilePleineException ex) {p.vider();}}}
```


Programmation « orientée exceptions »

// Essayer d'abord vs. tester d'abord.

```
class PileVideException extends Exception {}
class Pile {
 int top() throws PileVideException {
      //ESSAYER
      try {return espace[sommet];}
      catch (ArrayIndexOutOfBoundsException ex) {
            throw new PileVideException(); //PROPAGER
      }}
 void empiler(int x) throws PilePleineException {
      //ESSAYER
      try {sommet +=1; espace[sommet]=x;}
      catch (ArrayIndexOutOfBoundsException ex) {
            sommet -=1; //REPARER LOCALEMENT
            throw new PilePleineException();//PROPAGER
      }}}
```

Atouts de la propagation d'exception

- Une facilité de programmation et de lisibilité :
  - Regroupement de la gestion d'erreurs à un même niveau.
    - Evite des redondances dans l'écriture de traitements d'erreurs
    - Encombre peu le reste du code avec ces traitements.
- Une gestion des erreurs propre et explicite :
  - Par opposition à l'utilisation de la valeur de retour des méthodes pour signaler une erreur à la méthode appelante.
    - Pas le rôle de la valeur de retour de décrire une erreur,
    - La dissociation de la valeur de retour et de l'exception permet à cette dernière de décrire précisément la ligne de code ayant provoqué l'erreur et la nature de cette erreur.