Exceptions logicielles

Mise en œuvre en Java

Exceptions

- Evènement qui suspend l'exécution normale (interruption soft)
- Récupérable par programme, à l'exécution
- Objectifs
  - Fiabilité et «tolérance aux pannes»
  - Séparer l'essentiel de l'exceptionnel dans les traitements
- Exceptions pré-programmées : exceptions du langage et de ses bibliothèques
- Programmer ses propres exceptions : exceptions utilisateur

Exemples : langage et bibliothèques

```
Exception
   RuntimeException // erreurs du noyau
        ArithmeticException // division par zero
        ClassCastException // echec de downcast
        IndexOutOfBoundsException // i<0 | i>=taille
               ArrayIndexOutOfBoundsException
               StringIndexOutOfBoundsException
        NullPointerException
        SecurityException
   IOException // bibliothèque d'e/s
        EOFException
        FileNotFoundException
   AWTException // interface graphique
   ...
```


Capture d'exception

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

Mécanisme: quand une exception survient

- l'exécution normale est arrêtée
- l'exception est propagée jusqu'au premier bloc capable de la capturer (`catch`) en remontant la pile des appels de méthodes ...
- ... « au pire » jusqu'à la méthode `main` qui :
  - soit la capture
  - sinon : `exit` de programme avec compte-rendu (`printStackTrace()`)

Représentation

- En Java les exceptions sont représentées par des objets
  - décrits par une hiérarchie de classes de racine `Exception`
  - 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();
       }
   }
}
```

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 de la signature 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
        }}}
```