Exceptions

Un moyen de structurer le traitement des cas exceptionnels.

Principe:

1. une instruction génère une exception
   une exception est un objet
2. l'exception se propage
   a.vers les blocs englobants
   b.vers les méthodes appelantes
3. l'exception est captée

Nouvelle structure de bloc:

`try` <instruction>

`catch` (<type-exception1 ex1) <instruction>

`catch` (<type-exception2 ex2) <instruction>

…

`finally` <instruction>

Exemples

Conversion String -> Entier

```
String parStr;
int parInt;
parStr = getParameter(“LONGUEUR”);
// essaye de “lire” le paramètre comme un entier
try parInt = Integer.parseInt(parStr);
// en cas d’erreur met à -1
catch (NumberFormatException nfe) parInt = -1;
```

Accès contrôlé à un tableau

```
try z = tab[i];
catch (ArrayIndexOutOfBounds b) {
     System.out.println(b.getMessage());
     z = 0;
}
```


Captage des exceptions

```
catch (TypeException obj) <instruction>
```

Au cas ou une exception de type `TypeException` (ou d'un de ses sous-types) a été générée on exécute `<instruction>` et l'exception est référencée par `obj` (comme un paramètre de méthode)

Devoir de captage

Si la méthode `m()` peut produire une exception de type E alors il faut soit

- Appeler `m()` dans un bloc `try`

```
try obj.m(); catch (E e) recuperation;
```

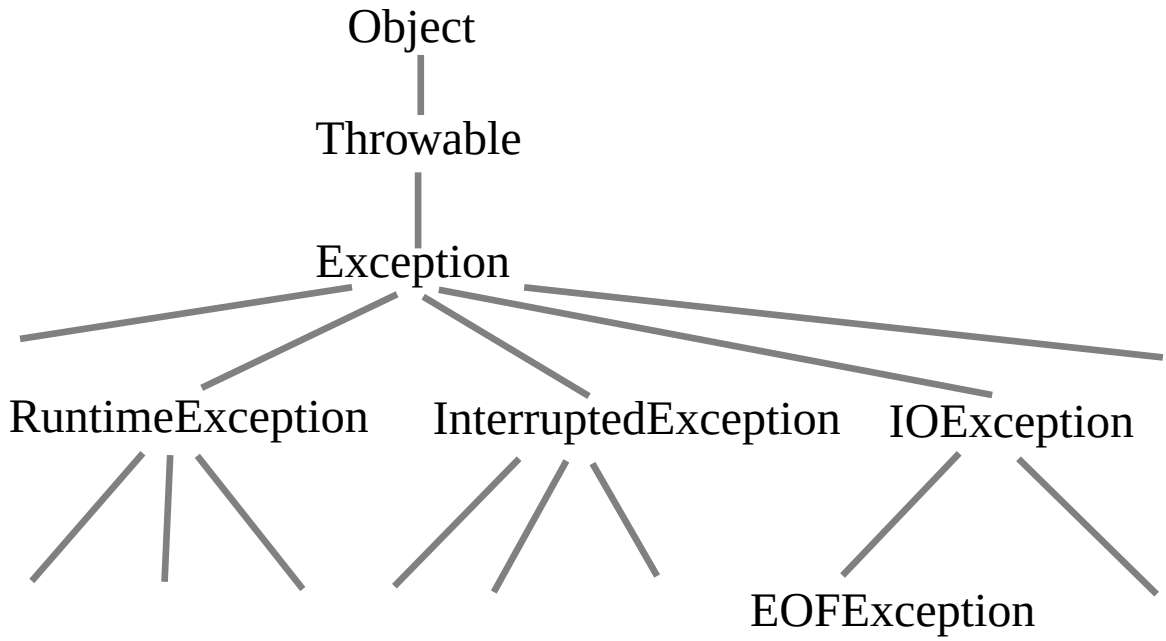
- Déclarer que la méthode courante peut générer une exception E

```
void maMethode() throws E {
     ...
     obj.m();
     ...
}
```

L'exception non captée est propagée à la méthode appelante.

Créer des types d'exceptions

Hiérarchie des classes d'exceptions



Création d'une nouvelle classe d'exceptions

```
class TotoException extends Exception {
     public TotoException() {super();}
}
```

Générer une exception

```
...
throw new TotoException()
```


Le bon usage des exceptions

- Les exceptions doivent rester … exceptionnelles
- Ce n'est pas une nouvelle structure de contrôle en plus des if, while, for, switch
- Pour éclaircir les traitements d'erreur du genre

```
if (erreur) <trait-erreur>
else {...
     if (erreur) <trait-erreur>
     else {...
          if (erreur) <trait-erreur>
          else {...
               (suite)
          }
     }
}
```

- Eviter de capter les erreurs graves non récupérables (NullPointerException, …)

Processus

La machine virtuelle Java permet l'exécution concurrent de plusieurs processus dans une programme.

Un processus (thread) est un fil d'exécution et un objet, il possède

- une instruction courante
- un environnement
- un état (actif, inactif, en attente, etc.)
- un nom (pas forcément unique)

Il faut le créer ET le démarrer

```
AnimatorThread a = new AnimatorThread();
a.start();
```

La classe Thread fournit un type de processus "minimal" qui ne fait rien =>

- la sous-classer
- ou utiliser un objet Runnable.

Processus comme sous-classe de Thread

```
class Producteur extends Thread {
public void run() {
      ...
      }

Producteur p = new Producteur();
p.start();
Producteur q = new Producteur();
q.start();
```

La méthode `start()` initialise le processus et lance l'exécution de la méthode `run()`.

Processus utilisant une autre classe

```
class Consommateur implements Runnable {
public void run() {
     ...
}

Consommateur conso = new Consommateur();
Thread t1 = new Thread(conso);
t1.start();
Thread t2 = new Thread(conso);
t2.start();
```

Dans ce cas le processus utilise un objet qui possède une méthode run() (qui implante Runnable). start() initialise le processus et lance l'exécution

Synchronisation

But: assurer la cohérence des traitements

Technique des moniteurs = garantir l'accès exclusif à une ressource (données ou instructions).

Chaque objet possède un moniteur d'accès

On obtient l'accès exclusif à un objet `o` de la classe `C`

- en exécutant une méthode synchronisée

```
class C {
synchronized void m() { ... instructions ... }
...
}

C o = new C();
o.m()
```

- en exécutant une instruction synchronisée sur o

```
synchronized (o) {... instructions ...}
```

Le moniteur bloque tout autre processus qui essaye d'obtenir l'accès à o.

Attente et notification

Lorsqu'un processus possède l'accès à un objet il peut appeler la méthode wait() pour attendre qu'un autre processus lui notifie qu'il peut continuer.

***obj*.wait()**

- relâche l'accès à l'objet
- attend une notification (ou un certain temps)
- après notification le processus redémarre dès qu'il peut réobtenir l'accès exclusif à l'objet

***obj*.notify()**

- notifie aux processus qui ont fait un wait() sur cet objet qu'ils peuvent continuer
- ne peut se faire qu'en ayant obtenu l'accès à l'objet
- ne relâche pas l'accès

Exemple Producteur-Consommateur

Un producteur produit des objets (int) et les place dans une file d'attente

Un consommateur prend ce qu'il trouve dans la file

L'accès à la file doit être atomique

La file est de taille limitée

La file d'attente

- on produit et consomme des entiers (int)
- la file est limitée à un entier

```
class Produit {
   int contenu;
   boolean vide;

   Produit() {vide = true;}

   void ajoute(int x) {contenu = x; vide = false;}

   int retire() {vide = true; return contenu;}

   boolean estVide() {return vide;}
}
```


Prod-cons - Producteur

```
class Producteur extends Thread {

Produit produit;

Producteur(Produit p) {this.produit = p;}

public void run() {
  int ip;
  while (true) {
    // dort un moment
    try {sleep((int)(Math.random() * 1000));}
    catch (InterruptedException e) {};
    synchronized (produit) {
        if (!produit.estVide()) {
            try produit.wait();
            catch (InterruptedException e);
        }
        ip =  (int)(Math.random() * 1000);
        produit.ajoute(ip);
        // signale qu'il y a qqch à prendre
        produit.notify();
    }
  }
}
}
```


Prod-cons - Consommateur

```
class Consommateur extends Thread {

Produit produit;

Consommateur (Produit p) {this.produit = p;}

public void run() {
  int jp;
  while (true) {
    try {sleep((int)(Math.random() * 1000));}
    catch (InterruptedException e) {};
    synchronized (produit) {
        if (produit.estVide()) {
            try produit.wait();
            catch (InterruptedException e);
        }
        jp =  produit.retire();
        produit.notify();
    }
  }
}

}
```


Prod-cons (suite)

Programme principal

```
public class ProdCons {

public static void main(String[] argv) {

  Produit pp = new Produit();

  Producteur pr = new Producteur(pp);

  Consommateur co = new Consommateur(pp);

  co.start(); pr.start();

}

}
```


Prod-cons Exécution

```
                    CO -- dort un moment
PR -- dort un moment
PR -- avant synchro.
PR -- synchronisé
PR -- ajoute 321
PR -- notify()
PR -- sort de la synchro
PR -- dort un moment
                    CO -- avant synchro.
                    CO -- synchronisé
                    CO -- retire 321
                    CO -- notify()
                    CO -- sort de la synchro
                    CO -- dort un moment
                    CO -- avant synchro.
                    CO -- synchronisé
                    CO -- wait()
PR -- avant synchro.
PR -- synchronisé
PR -- ajoute 995
PR -- notify()
PR -- sort de la synchro
PR -- dort un moment
                    CO -- reprend après notification
                    CO -- retire 995
                    CO -- notify()
                    CO -- sort de la synchro
                    CO -- dort un moment
```


Exécution (suite)

```
PR -- avant synchro.
PR -- synchronisé
PR -- ajoute 788
PR -- notify()
PR -- sort de la synchro
PR -- dort un moment
                    CO -- avant synchro.
                    CO -- synchronisé
                    CO -- retire 788
                    CO -- notify()
                    CO -- sort de la synchro
                    CO -- dort un moment
PR -- avant synchro.
PR -- synchronisé
PR -- ajoute 794
PR -- notify()
PR -- sort de la synchro
PR -- dort un moment
PR -- avant synchro.
PR -- synchronisé
PR -- wait()
                    CO -- avant synchro.
                    CO -- synchronisé
                    CO -- retire 794
                    CO -- notify()
                    CO -- sort de la synchro
                    CO -- dort un moment
```


Animer une applette

Principe: (modèle - vue)

- Créer un processus qui appelle régulièrement la méthode paint() de l'applette pour dessiner le modèle.
- Créer un ou des processus qui font "bouger" le modèle

Exemple:

- Le modèle est un ensemble de rectangles.
- Un processus s'occupe de chaque rectangle.
- Le processus dessine les rectangles (en appelant la méthode paint() de chaque rectangle)

class RectVivant extends Thread {

```
      Color cc;
      int rx = 0; int ry = 0;
      int deltaT; int dx, dy;

public RectVivant(String str, Color c, int dx, int dy,
int dt)
{ super(str); cc = c;  deltaT = dt; this.dx = dx;
this.dy = dy; }

    // ** le processus **
    public void run() {
       for (int i = 0; i < 100; i++) {
          try {sleep(deltaT); }
          catch (InterruptedException e) {}
          if ((rx+dx < 0) || (rx+dx > 170)) dx = -dx;
          if ((ry+dy < 0) || (ry+dy > 170)) dy = -dy;
           rx += dx;
           ry += dy;
     }}
     // ** appelé par le processus de dessin **
     public void paint(Graphics g) {
         if (this.isAlive()) {
               g.setColor(cc);
               g.fillRect(rx, ry, 30, 30);}
           else
               g.drawRect(rx, ry, 30, 30);
    }}
```


public class Anima
extends Applet implements Runnable{

```
    RectVivant a1, a2, a3, af;
    Thread dessinateur;

  // ** le processus de dessin **
   public void run() {
      for (int i = 0; i < 300; i++) {
         try {Thread.sleep(50);  }
         catch (InterruptedException e) {}
         this.paint(this.getGraphics());
      }
   }
    public void paint(Graphics g) {
        g.clearRect(0,0,200,200);
         a1.paint(g);
         a2.paint(g);
         a3.paint(g);
    }
    // ** démarrage de l'applette, création des procs
    public void start() {
a1 = new RectVivant("Jamaica", Color.pink, 2, 3, 50);
a2 = new RectVivant("Fiji", Color.white, 2, 3, 200);
a3 = new RectVivant("Bermuda", Color.blue, 3, 3, 150);
    a1.start(); a2.start(); a3.start();
    dessinateur = new Thread(this);
    dessinateur.start();
  }
 }
```


Schema des processus