JAV - TD 4

Exceptions

Packages et résolution de noms

Chargement de classes

Classpath et jar

Les exceptions

- Elles permettent de séparer un bloc d'instructions de la gestion des erreurs pouvant survenir dans ce bloc.

```
ptr=malloc(sizeof(struct));
if (ptr==0){
  return (ko);
}...
```

Exemple en langage C : Le retour ok de la fonction (le pointeur) et le retour d'erreur sont dans le même espace mémoire. Le programmeur peut ne pas traiter l'erreur... Ca marche tout le temps, sauf que...

- En java:
  - Les exceptions sont décrites au niveau des méthodes
  - Elles contraignent le programmeur à valider le code
  - Deux activités :
    - L'utilisateur de la méthode
    - Le programmeur de la méthode

L'utilisateur "malin" des méthodes

Ici tout le bloc de code "normal"

Ici l'ensemble des gestions d'exceptions, de la plus spécifique à la plus générale

```
void maFonction(Armoire lArmoire){
try {
  Pile maPile=lArmoire.getPileAssiettes();
  while(1){
     Assiette tmp1=(Assiette)maPile.depiler();
          Assiette tmp2=(Assiette)maPile.depiler();
          Assiette plusGrande=Assiette.getLaPlusGrande(tmp1,
tmp2);
          plusGrande.poser();
          if (plusGrande==tmp1){
           tmp2.poser();
          }else{
           tmp1.poser();
         }
  }
}catch(PileVideException e){
  System.out.println("fin de la pile d'assiettes");
}catch(PilePasTrouveeException e){
  System.out.println("pb: pas de pile d'assiettes");
}catch(IOException e){
  // code de traitement d'une éventuelle Exception d'E/S
}finally{  System.out.println ("terminé");}
}
```

Celui qui n'a rien compris...

ATTENTION
c'est
n'importe quoi!!

MAIS
POURQUOI???

```
void maFonction(Armoire lArmoire){
try {
  Pile maPile=lArmoire.getPileAssiettes();
}catch(PilePasTrouveeException e){
  System.out.println("pb: pas de pile d'assiettes");
}
while(1){
  try {
     Assiette tmp1=(Assiette)maPile.depiler();
          Assiette tmp2=(Assiette)maPile.depiler();
  }catch(PileVideException e){
    System.out.println("fin de la pile d'assiettes");
  }
    Assiette plusGrande=Assiette.getLaPlusGrande(tmp1, tmp2);
          plusGrande.poser();
          if (plusGrande==tmp1){
           tmp2.poser();
          }else{
           tmp1.poser();
         }
  }
//mais où met-on le code suivant?
//}finally{//  System.out.println ("terminé");//}
}
```

L'autre utilisateur "malin" des méthodes

```
void maFonction(Armoire lArmoire)throws PileVideException, PilePasTrouveeException{
try {
  Pile maPile=lArmoire.getPileAssiettes();
  while(1){
     Assiette tmp1=(Assiette)maPile.depiler();
          Assiette tmp2=(Assiette)maPile.depiler();
          Assiette plusGrande=Assiette.getLaPlusGrande(tmp1, tmp2);
          plusGrande.poser();
          if (plusGrande==tmp1){
           tmp2.poser();
          }else{
           tmp1.poser();
         }
  }

}catch(IOException e){
  // code de traitement d'une éventuelle Exception d'E/S
}finally{
  System.out.println ("terminé");
}
```

Ne jamais...

- Ne jamais écrire ce code, sauf si on en comprend parfaitement ses conséquences.

```
try{
  unCodeQuiLeveUneException();
}catch(Exception e){
  /* Aucune action, ce qui masque les erreurs */
}
```

- Le code doit toujours traiter l'exception. Au minimum :

```
try{
  unCodeQuiLeveUneException();
}catch(Exception e){
  e.printStackTrace();
/* affiche l'empilement des appels qui ont mené à l'erreur */
}
```

Le programmeur de la méthode

Une classe peut gérer ses propres exceptions, dans ce cas le programmeur indique que la méthode peut lever une exception **(throws)** et lève l'exception au moment voulu **(throw)**

```
public class Pile1 {
  public Object depiler () throws Exception {
    if (this.size()==0){
      throw new Exception("C'est vide!");
       ...
```

```
public class Pile2 {
  public Object depiler () throws PileVideException {
    if (this.size()==0){
          throw new PileVideException("C'est vide!");
       ...
```

```
public class PileVideException extends Exception {}
```

Exercice

- Reprendre la classe DivisionO du TD-2 et modifier le code pour lever une exception (instance de la classe Exception) quand le diviseur est 0

- Reprendre le même code de la classe cliente de DivisionO (TD-2)
  - Compiler, que se passe t'il?
  - Modifier le code, compiler et exécuter

Translation d'exception

```
public class PileAssiette {
  Pile p=new Pile();
  public Assiette depiler () throws PlusDAssietteException {
    try{
      return p.depiler();
    }catch(Exception e){
      throw new PlusDAssietteException();
    }
  }
```

```
public class Client {
  public utilise(PileAssiette pileAssiette){
    try{
      Assiette suivante=pileAssiette.depiler();
    }catch(PlusDAssietteException e3){
      System.out.println("Fini la vaisselle");
    }
  }
}
```

Ce qu'il faut retenir des Exceptions

- Ce sont des instances de classes dérivant de `java.lang.Exception`
- La levée d'une exception provoque une remontée dans l'appel des méthodes jusqu'à ce qu'un bloc `catch` acceptant cette exception soit trouvé.
- L'appel à une méthode pouvant lever une exception doit :
  - soit être contenu dans un bloc `try/catch`
  - soit être situé dans une méthode propageant (`throws`) cette classe d'exception
- Un bloc `finally` peut-être posé à la suite des `catch`. Son contenu est exécuté après un `catch` ou après un `break`, un `continue` ou un `return` dans le bloc `try`

Chargement de classes

Qui ?

Où ?

Classe et fichier

- Il est recommandé (mais pas imposé) de ne mettre qu'une classe par fichier

- Le fichier doit avoir le même nom que la classe qu'elle contient.

Les packages - définition

- Unité d'organisation des classes
  - Organisation logique : `time.clock.Watch`
  - Organisation physique : `time/clock/Watch.class`
- Espace de **nommage** hiérarchique
  - Description de la hiérarchie : `package time.clock;`
  - Notion de nom complet : `time.clock.Watch`
- Les bibliothèques java sont organisées en package
  - java.util, java.net, org.objectweb…
  - Deux classes ayant le même nom complet ne peuvent pas s'exécuter en même temps.

Nom de classe : résolution

- Pour résoudre un nom de classe dans une autre classe

```
...
time.clock.Watch toto=new time.clock.Watch();
...
```

```
import time.clock.Watch;
...
Watch toto=new Watch();
...
```

```
import time.clock.*;
...
Watch toto=new Watch();
Clock titi=new Clock();
...
```

```
import time.*;
...
Watch toto=new Watch();
Clock titi=new Clock();
...
```

Les packages : exercice à réaliser, lancer...

graph/2d/Circle.java

```
package graph.2d;
public class Circle()
{ private double rayon;
  public Circle(double r){
    this.rayon=r;
    System.out.println("je suis un nouveau cercle de rayon " +this.rayon);
  }
}
```

graph/3d/Sphere.java

```
package graph.3d;
public class Sphere()
{ private double rayon;
  public Sphere(double r){
    this.rayon=r;
    System.out.println("je suis une nouvelle sphere de rayon " +this.rayon);
}
  }
```

paintShop/MainClass.java

```
package paintShop;

public class MainClass()
{
  public static void main(String[] args) {
    graph.2d.Circle c1 = new graph.2d.Circle(50)
    graph.2d.Circle c2 = new graph.2d.Circle(70);
    graph.3d.Sphere s1 = new graph.3d.Sphere(100);
    Sphere s2 = new Sphere(40); // error: class paintShop.Sphere not found
  }
```

Les packages : utilisation des «alias»

graph/2d/Circle.java

```
package graph.2d;
public class Circle()
{ ... }
```

graph/3d/Sphere.java

```
package graph.3d;
public class Sphere()
{ ... }
```

paintShop/MainClass.java

```
package paintShop;

import graph.2d.Circle;

public class MainClass()
{
  public static void main(String[] args) {
    Circle c1 = new Circle(50)
    Circle c2 = new Circle(70);
    graph.3d.Sphere s1 = new graph.3d.Sphere(100);
    Sphere s2 = new Sphere(40); // error: class paintShop.Sphere not found
}
```

Crée un alias "Circle" pour graph.2d.Circle

Le mot-clé "import" n'a rien à voir avec "#include" du C

#include fait récupérer et compiler le code inclus

import crée seulement un alias

Où ?

Chargement de classe

```
Hell t;         /* La classe n'est pas encore chargée*/
t=new Hell(); /* Le classloader charge la classe Hell
                 en mémoire */
Vector v=new Vector(); /* Le classloader charge la
                          classe Vector en mémoire*/
```

- Les outils du système cherchent toutes les classes
  - Compilation et Exécution : typage fort C & E
  - Les classes sont recherchées sur le système de fichiers
  - Les classes standards sont automatiquement trouvées
  - Pour indiquer l'endroit sur le système de fichiers ***à partir duquel*** il faut chercher les classes, on utilise le **classpath**
  - Le classpath a un fonctionnement identique au path d'exécution

Le classpath

- Il indique à partir de quel endroit rechercher une classe

```
java -classpath /usr/local/ titi.Toto
```

- La classe titi.Toto et **toutes** les classes lancées à partir de maintenant sont recherchées à partir du répertoire

```
/usr/local /*Résolution physique*/
```

- Il faut donc que la classe soit définie dans le fichier :

```
/usr/local/titi/Toto.class /*Résolution java*/
```

- Remarque :

```
il est possible d'indiquer pendant l'exécution la localisation
de nouvelles classes
```

Le classpath : le jar

- Un jar est une archive java
- Regroupement de fichiers dans un fichier
- Extension du système de fichiers

```
jar tvf toto.jar
```

```
tutu/
tutu/ours/
tutu/ours/Grumly.class
```

```
vi Test.java
```

```
package test;
import tutu.ours.Grumly;
public class Test {
  Grumly toto=new Grumly();
}
```

```
javac ? ???
```

Exercices classpath

- Ecrire une classe test.Division,
- Ecrire la classe de test nommée test.Test
  - Compiler, exécuter à partir du repertoire au dessus de test.
  - Compiler, exécuter à partir du repertoire test, que se passe t'il ?
- Ecrire une classe de test nommée test2.Test
  - Compiler, exécuter, que se passe t'il
  - Compiler, exécuter à partir du repertoire test2

import / Classpath / Path

- **import** : alias du nom court sur un nom long
  - import java.util.Vector, permet d'utiliser l'alias Vector
  - Il est utilisé par les classes
- **Classpath** : localisation physique des classes sur le disque
  - Chemin pour trouver la racine des classes externes (repertoire ou .jar)
  - Il est utilisé par la machine virtuelle
- **Path** : localisation physique des exécutables
  - Chemin pour trouver la racine des executables
  - Il est utilisé par le SE
- Il n'y a aucun rapport entre import et classpath

Exercice jar

- Utiliser la classe toto.test.Tutu qui présente une méthode statique hello(). Cette classe est dans le jar bonjour.jar sur

  //Partage/enseignants/Info/TP-Frenot/IJA/test.jar

- Il est possible de réaliser un jar directement exécutable avec la commande

  java -jar test.jar

- Pour cela il faut écrire un fichier manifest (fichier de propriétés dans le monde java). Le fichier doit contenir le nom de la classe à lancer dans la propriété :

  Main-Class: test.Test

- Une autre propriété permet d'ajouter des archives dans le classpath

  Class-Path: lib/jmxremote.jar lib/jmxremote_optional.jar

Exo debug d'exceptions

- Récupérer Test.java dans tc-net2/ija/exos/

  L'ouvrir, que fait-il? Le compiler, Le lancer...Corriger

```
package ex;

import java.net.Socket;
import java.io.PrintWriter;
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class Test{
  public static void main(String [] arg) throws Exception {
    Socket s=new Socket("localhost", 80);
    PrintWriter out=new PrintWriter(s.getOutputStream(), true);
    BufferedReader in=new BufferedReader(new InputStreamReader(s.getInputStream()));
    out.println("GET /truc HTTP/1.0\n");
    String l=in.readLine();
    while (true){
      System.out.print(l.toString());
      l=in.readLine();
    }
  }
}
```