Programmation Objet. Cours 5

Marie-Pierre Béal
DUT 1

Exceptions.

Une classe Stack

Voici une classe pile contenant des int, implémentée par un tableau.

```
public class Stack{
  static final int MAX=4;
  private int height = 0;
  private final int[] table = new int[MAX];

  public boolean isEmpty() {
    return height == 0;
  }
  public boolean isFull() {
    return height == MAX;
  }
```

Une classe Stack

```
  public void push(int item) {
    table[height] = item;
    height = height + 1;
  }
  public int peek() {
    return table[height - 1];
  }
  public int pop() {
    height = height - 1;
    return table[height];
  }
}
```

Il y a débordement lorsque l'on fait

- peek pour une pile vide ;
- pop pour une pile vide ;
- push pour une pile pleine.

Exceptions

Une pile ne peut pas proposer de solution en cas de débordement, mais elle doit signaler (et interdire) le débordement. Cela peut se faire par l'usage d'une exception.
Une exception est un objet d'une classe qui étend la classe `Exception`.

```
java.lang.Object
 |_ java.lang.Throwable
      |_java.lang.Error
      |_java.lang.Exception
          |_java.lang.ClassNotFoundException
          |_ ...
          |_java.lang.RuntimeException
              |_java.lang.NullPointerException
              |_java.lang.UnsupportedOperationException
              |_ ...
```

Exceptions

Pour les piles, on peut définir par exemple une nouvelle classe dérivant de la classe `Exception`.

```
public class StackException extends Exception { ... }
```

En cas de débordement, on lève une exception, par le mot `throw`.
On doit signaler la possible levée dans la déclaration par le mot `throws`.
Par exemple :

```
public void push(int item) throws StackException {
    if (isFull())
        throw new StackException("Pile pleine");
    table[height] = item;
    height = height + 1;
}
```

Exceptions

La levée d'une exception provoque

- la propagation d'un objet d'une classe d'exceptions ;
- la sortie immédiate de la méthode ;
- la remontée dans l'arbre d'appel à la recherche d'une méthode qui capture l'exception.

La capture se fait par un bloc `try / catch`. Par exemple,

```
...
Stack s = new Stack();
try {
  System.out.println("top = "+ p.peek());
} catch(StackException e) {
  System.out.println(e.getMessage());
}
```

Bloc try/catch

- Le bloc `try` lance une exécution contrôlée.
- En cas de levée d'exception dans le bloc `try`, ce bloc est quitté immédiatement, et l'exécution se poursuit par le bloc `catch`.
- Le bloc `catch` reçoit en argument l'objet créé lors de la levée d'exception.
- Plusieurs `catch` sont possibles, et le premier dont l'argument est du bon type est exécuté. Les instructions du bloc `finally` sont exécutées dans tous les cas.

```
try { ... }
catch (Type1Exception e) { .... }
catch (Type2Exception e) { .... }
catch (Exception e) { .... }
    // cas par defaut, capture les
    // exceptions non traitees plus haut
finally {....} // toujours execute
```

Exceptions

Exemple de catch mal ordonné

```
try { ... }
catch (Exception e) { .... }
catch (StackException e) { .... }
// le deuxieme jamais execute
```

Exceptions Runtime et non Runtime

Exceptions ne dérivant pas de `RuntimeException`

Une levée d'exception se produit lors d'un appel à `throw` ou d'une méthode ayant levé une exception. Ainsi l'appel à une méthode pouvant lever une exception doit être :

- ou bien être contenu dans un bloc `try / catch` pour capturer l'exception ;
- ou bien être dans une méthode propageant cette classe d'exception (avec `throws`).

`RuntimeException`

Les exceptions dérivant de la classe `RuntimeException` n'ont pas à être capturées.

Interface Stack

Voici une interface de pile d'`int`, et deux implémentations

- une implémentation par tableau de type `int []` ;
- une implémentation par `ArrayList`.

```
interface Stack {
  boolean isEmpty ();
  boolean isFull();
  void push(int item) throws StackException;
  int peek() throws StackException;
  int pop() throws StackException;
}
public class StackException extends Exception {
  public StackException(String m) {super(m);}
}
```

```
public class ArrayStack implements Stack {
  static final int MAX=4;
  private int height = 0;
  private final int[] table = new int[MAX];
  public void push(int item) throws StackException {
    if (isFull())
      throw new StackException("Pile pleine");
    table[height] = item; height++;
  }
  public int peek() throws StackException{
    if (isEmpty())
      throw new StackException("Pile vide");
    return table[height-1];
  }
  public int pop() throws StackException{
    if (isEmpty())
      throw new StackException("Pile vide");
    height--;
    return table[height];
  }
```

Implémentation avec des ArrayList

```
public class ArrayListStack implements Stack{
   private final List<Integer> list;

   public ArrayListStack() {
     list = new ArrayList<Integer>();
   }
   public boolean isEmpty() {
     return list.isEmpty();
   }
   public boolean isFull() {
     return false;
   }
   public void push(int item) throws StackException{
     list.add(list.size(),item);
   }
```

Implémentation avec des ArrayList

```
    ...
    public int peek() throws StackException{
      if (isEmpty())
        throw new StackException("Pile vide");
      return list.get(list.size()-1);
    }
    public int pop() throws StackException{
      if (isEmpty())
        throw new StackException("Pile vide");
      return list.remove(list.size()-1);
  }
}
```

Utilisation de l'interface Stack

```
public class Test{
  public static void main(String[] args) {
    //le nom de l'interface (Stack) est le type de s
    Stack s = new ArrayListStack(); //par table
    try {
      s.push(2);
      System.out.println(s.peek());
      s.pop();
      s.pop(); // ca coince
      s.pop(); // jamais atteint
    }
    catch(StackException e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
}}
```

```
public class Test{
  public static void main(String[] args) {
    Stack s;
      if ((args.length != 0) && (args[0].equals("array"))) {
         s = new ArrayStack(); //par table
    } else {
         s = new ArrayListStack(); //par ArrayList
    }
    try {
       s.push(2);
       System.out.println(s.peek());
       s.pop();
       s.pop(); // ca coince
       s.pop(); // jamais atteint
    }
    catch(StackException e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
}
```

On obtient

```
$ java Test array
2
Pile vide
StackException: Pile vide
at ArrayStack.pop(ArrayStack.java:24)
at Test.main(Test.java:13)
$ java Test
2
Pile vide
StackException: Pile vide
at ArrayListStack.pop(ArrayListStack.java:26)
at Test.main(Test.java:13)
$
```