

Programmation Orientée Objet

Mathieu RAYNAL

mathieu.raynal@irit.fr

<http://www.irit.fr/~Mathieu.Raynal>

Programmation Orientée Objet

Enumération

Exception

Créer une énumération

- Se déclare comme une classe
 - Mot clé **enum** à la place de **class**
 - Hérite de la classe **java.lang.Enum**
- Contient une série de valeurs constantes
 - ni le type, ni la valeur réelle de chaque constante n'est précisé.
- Les valeurs sont classées par ordre de déclaration
- Elles ont un indice
 - Commence à la valeur 0 pour la première valeur déclarée

```
public enum TypeObjet
{
    RECTANGLE,
    CARRE,
    TRIANGLE,
    ELLIPSE,
    CERCLE;
}
```

Utilisation d'une énumération

- Les valeurs sont comparables avec les opérateurs classiques
- Utilisable avec l'instruction switch

```
TypeObjet t = TypeObjet.CARRE;
```

```
if(t == TypeObjet.CARRE)
{
    // Traitement ...
}
```

```
for(TypeObjet type : TypeObjet.values())
    System.out.println(type);
```

```
switch(t)
{
    case RECTANGLE:
        // Traitement ...
        break;
    case CARRE:
        // Traitement ...
        break;
    ...
}
```

Compléter une énumération

- Les énumérations peuvent avoir des attributs et des méthodes

```
public enum TypeObjet {  
    RECTANGLE("un rectangle"),  
    CARRE("un carré"),  
    TRIANGLE("un triangle"),  
    ELLIPSE("une ellipse"),  
    CERCLE("un cercle");  
  
    private String nom;  
  
    private TypeObjet(String nom){  
        this.nom = nom;  
    }  
  
    public String getNom(){  
        return nom;  
    }  
}
```

Exercice 1 – Utilisation d'une énumération

- Ecrire une classe Personne
 - Deux attributs de type String pour le prénom, et le nom
 - Un attribut de type Genre pour le genre (HOMME|FEMME)
 - Définir l'énumération GENRE
 - Un constructeur qui prend en paramètre le genre, le nom et le prénom
 - Une méthode affichage qui affiche un message du type
« Bonjour **M.** BARRIOT, vous êtes **le bienvenu** dans cette classe ! »
« Bonjour **Mme** FICHANT, vous êtes **la bienvenue** dans cette classe ! »

Programmation Orientée Objet

Enumération

Exception

Qu'est ce qu'une exception ?

- Instruction qui ne peut pas s'exécuter correctement
 - Accès à un objet qui n'existe pas (***NullPointerException***)
 - Accès à un élément du tableau hors de ses bornes (***ArrayIndexOutOfBoundsException***)
 - Division par 0 (***ArithmeticException***)
 - Demande d'accès à un fichier qui n'existe pas
 - Tout cas particulier qui empêche le bon déroulement d'une instruction

FileReader

```
public FileReader(String fileName)  
    throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading

Comment gérer une exception ?

- Possibilité de « capturer » les exceptions
 - Informer de l'erreur produite
 - Permettre au programme de continuer de fonctionner correctement

- 2 gestions possibles d'une exception
 - Gérer directement l'exception avec des blocs ***try / catch***
 - « Propager » l'exception avec ***throws***

Traitement de l'exception avec *try / catch*

- Deux blocs d'instructions
 - **try** : on exécute, dans ce bloc, les instructions qui peuvent déclencher une exception
 - **catch** : on passe dans ce bloc qu'en cas de déclenchement d'une exception
- Il est possible de gérer plusieurs exceptions à partir du même bloc try
 - Dans un même bloc catch
 - Avec un bloc catch par exception
- Les instructions après le bloc try/catch seront exécutées

```
try {  
    fileReader = new FileReader("test.txt");  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
}
```

```
try {  
    fileReader = new FileReader(new File(new URI("test.txt")));  
} catch (FileNotFoundException | URISyntaxException e) {  
    e.printStackTrace();  
}
```

```
try {  
    fileReader = new FileReader(new File(new URI("test.txt")));  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} catch (URISyntaxException e) {  
    e.printStackTrace();  
}
```

Affichage de l'exception

```
try {  
    fileReader = new FileReader("test.txt");  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
}
```

```
java.io.FileNotFoundException: test.txt (Le fichier spécifié est introuvable)  
    at java.io.FileInputStream.open0(Native Method)  
    at java.io.FileInputStream.open(Unknown Source)  
    at java.io.FileInputStream.<init>(Unknown Source)  
    at java.io.FileInputStream.<init>(Unknown Source)  
    at java.io.FileReader.<init>(Unknown Source)  
    at fr.petitsboutsdecode.Personne.lireFichier(Personne.java:16)  
    at fr.petitsboutsdecode.Personne.main(Personne.java:28)
```

Utilisation de *throws*

- On ne traite pas l'exception dans la méthode où se produit l'exception

```
public void lireFichier() throws FileNotFoundException {  
    FileReader fileReader = new FileReader("test.txt");
```

- On propage l'exception
 - L'exception sera traitée à l'appel de cette méthode

```
try {  
    obj.lireFichier();  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
}
```

Forcer l'exécution de certaines instructions

- Avec l'utilisation de **throws**, en cas d'exception
 - L'exception est propagée
 - Le bloc d'instructions est interrompu
 - Les instructions qui suivent ne sont pas exécutées
- Le bloc **finally** permet d'imposer l'exécution d'un bloc d'instructions, même après la levée d'une exception

```
public void lireFichier() throws FileNotFoundException {  
    try{  
        FileReader fileReader = new FileReader("test.txt");  
        System.out.println("Cette instruction 0 est exécutée ...");  
    }finally{  
        System.out.println("Cette instruction 1 est exécutée ...");  
    }  
}
```

```
try {  
    obj.lireFichier();  
    System.out.println("Cette instruction 2 est exécutée ...");  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
    System.out.println("Cette instruction 3 est exécutée ...");  
}
```

```
Cette instruction 1 est exécutée ...  
java.io.FileNotFoundException: test.txt (Le fichier spécifié est introuvable)  
    at java.io.FileInputStream.open0(Native Method)  
    at java.io.FileInputStream.open(Unknown Source)  
    at java.io.FileInputStream.<init>(Unknown Source)  
    at java.io.FileInputStream.<init>(Unknown Source)  
    at java.io.FileReader.<init>(Unknown Source)  
    at fr.petitsboutsdecode.Personne.lireFichier(Personne.java:28)  
    at fr.petitsboutsdecode.Personne.main(Personne.java:39)  
Cette instruction 3 est exécutée ...
```

Créer sa propre exception

- Toutes les exceptions sont définies par des classes
 - Toutes héritent de la classe **Exception**
- Il est possible de définir sa propre classe d'exception
- **throw** permet de propager une nouvelle exception

```
public class ValueException extends Exception{  
    public ValueException(int valeur){  
        super(valeur+" n'est pas une valeur positive !");  
    }  
}
```

```
public void setLongueur(int longueur) throws ValueException{  
    if(longueur<0)  
        throw new ValueException(longueur);  
    else  
        this.longueur = longueur;  
}
```

```
fr.petitsboutsdecode.geometrie.ValueException: -10 n'est pas une valeur positive !  
    at fr.petitsboutsdecode.geometrie.Rectangle.setLongueur(Rectangle.java:22)  
    at fr.petitsboutsdecode.geometrie.Rectangle.main(Rectangle.java:57)
```

Exercice

- Créez une exception ***DimensionException*** qui précise qu'une dimension doit être une valeur positive
- Dans la classe Rectangle, pour les méthodes ***setLongueur*** et ***setHauteur***, utilisez les exceptions en cas de valeur négative