Les énumérations

Dans une application, il est très utile de pouvoir représenter des listes finies d'éléments. Par exemple, si une application a besoin d'une liste de niveaux de criticité, elle peut créer des constantes dans une classe utilitaire quelconque.

```
package com.cgi.udev;


public class ClasseUtilitaireQuelconque {


  public static final int CRITICITE_FAIBLE = 0;
  public static final int CRITICITE_NORMAL = 1;
  public static final int CRITICITE_HAUTE = 2;
}
```

Ce choix d'implémentation est très imparfait. Même si le choix du type **int** permet de retranscrire une notion d'ordre dans les niveaux de criticité, il ne permet pas de créer un vrai type représentant une criticité.

Il existe un type particulier en Java qui permet de fournir une meilleure implémentation dans ce cas : l'énumération. Un énumération se déclare avec le mot-clé **enum**.

```
package com.cgi.udev;


public enum Criticite {
  FAIBLE,
  NORMAL,
  HAUTE
}
```

> Les éléments d'une énumération sont des constantes. Donc, par convention, ils sont écrits en majuscules et les mots sont séparés par _.

Comme une classe, une énumération a une portée et elle est définie dans un fichier qui porte son nom. Pour l'exemple ci-dessus, le code source sera dans le fichier *Criticite.java*.

Une énumération peut également être définie dans une classe, une interface et même dans une énumération.

```
package com.cgi.udev;

public class Zone {

  public enum NiveauDeSecurite {FAIBLE, MOYEN, FORT}

  private NiveauDeSecurite niveau;

  // ...

}
```

L’énumération permet de définir un type avec un nombre fini de valeurs possibles. Il est possible de manipuler ce nouveau type comme une constante, comme un attribut, un paramètre et une variable.

```
package com.cgi.udev;

public class RapportDeBug {

  private Criticite criticite = Criticite.NORMAL;

  public RapportDeBug() {
  }

  public RapportDeBug(Criticite criticite) {
    this.criticite = criticite;
  }
}
```

```
RapportDeBug rapport = new RapportDeBug(Criticite.HAUTE);
```

> Une variable, un attribut ou un paramètre du type d’une énumération peut avoir la valeur **null**.

Les méthodes d’une énumération

Nous verrons bientôt qu'une énumération est en fait une classe particulière. Donc une énumération fournit également des méthodes de classe et des méthodes pour chacun des éléments de l'énumération.

valueOf

Une méthode de classe qui permet de convertir une chaîne de caractères en une énumération. Attention toutefois, si la chaîne de caractères ne correspond pas à un nom d'un élément de l'énumération, cette méthode produit une IllegalArgumentException.

```
Criticite criticite = Criticite.valueOf("HAUTE");
```

values

Une méthode de classe qui retourne un tableau contenant tous les éléments de l'énumération dans l'ordre de leur déclaration. Cette méthode est très pratique pour connaître la liste des valeurs possibles par programmation.

```
for(Criticite c : Criticite.values()) {
  System.out.println(c);
}
```

Chaque élément d'une énumération possède les méthodes suivantes :

name

Retourne le nom de l'élément sous la forme d'une chaîne de caractères.

```
String name = Criticite.NORMAL.name(); // "NORMAL"
```

ordinal

Retourne le numéro d'ordre d'un élément. Le numéro d'ordre est donné par l'ordre de la déclaration, le premier élément ayant le numéro 0.

```
int ordre = Criticite.HAUTE.ordinal(); // 2
```

> Une énumération implémente également l'interface Comparable. Donc, une énumération implémente la méthode compareTo qui réalise une comparaison en se basant sur le numéro d'ordre.

Égalité entre énumérations

Par définition, chaque élément d'un énumération n'existe qu'une fois en mémoire. Un énumération garantit que l'unicité de la valeur est équivalente à l'unicité en mémoire. Cela signifie que l'on peut utiliser l'opérateur **==** pour comparer des variables, des attributs et des paramètres du type d'énumération. L'utilisation de l'opérateur **==** est même considérée comme la bonne façon de comparer les énumérations.

```
if (criticite == Criticite.HAUTE) {
  // ...
}
```

Support de switch

Une énumération peut être utilisée dans une structure **switch** :

```
switch (criticite) {

case Criticite.FAIBLE:
  // ...
  break;

case Criticite.NORMAL:
  // ...
  break;

case Criticite.HAUTE:
  // ...
  break;

}
```

Génération d'une énumération

Le mot-clé **enum** est en fait un sucre syntaxique. Les énumérations en Java sont des classes comme les autres. Ainsi, l'énumération :

```
package com.cgi.udev;

public enum Criticite {

  FAIBLE,
  NORMAL,
  HAUTE

}
```

est transcrite comme ceci par le compilateur :

```
package com.cgi.udev;

public final class Criticite extends Enum<Criticite> {

  public static final Criticite FAIBLE = new Criticite("FAIBLE", 0);
  public static final Criticite NORMAL = new Criticite("NORMAL", 1);
  public static final Criticite HAUTE = new Criticite("HAUTE", 2);

  public static Criticite valueOf(String value) {
    return Enum.valueOf(Criticite.class, value);
  }

  public static Criticite[] values() {
    return new Criticite[] {FAIBLE, NORMAL, HAUTE};
  }

  private Criticite(String name, int ordinal) {
    super(name, ordinal);
  }
}
```

> Malheureusement le code ci-dessus ne compile pas car le compilateur Java n'autorise pas à créer soi-même une énumération.

Le code ci-dessus nous permet de remarquer que :

- Les valeurs d'une énumération sont en fait des attributs de classe du type de l'énumération elle-même.

- Une énumération est déclarée **final** donc il n'est pas possible d'hériter d'une énumération (sauf en créant une classe interne anonyme).
- Le constructeur d'une énumération est privé, empêchant ainsi de créer de nouvelle instance.
- Une énumération hérite de la classe Enum.

Ajout de méthodes et d'attributs

Lorsque l'on a bien compris qu'une énumération est une classe particulière, il devient évident qu'il est possible d'ajouter des attributs et des méthodes à une énumération.

```
package com.cgi.udev;

public enum Couleur {

 ROUGE, ORANGE, JAUNE, VERT, BLEU, MAGENTA;

 private static final List<Couleur> COULEURS_CHAUDES = Arrays.asList(ROUGE, ORANGE, JAUNE);

 public boolean isChaude() {
  return COULEURS_CHAUDES.contains(this);
 }

 public boolean isFroide() {
  return !isChaude();
 }

 public Couleur getComplementaire() {
  Couleur[] values = Couleur.values();
  int index = this.ordinal() + (values.length / 2);
  return values[index % values.length];
 }
}
```

> Notez l'utilisation du point-virgule à la fin de la liste des couleurs. Ce point-virgule n'est obligatoire que lorsque l'on veut ajouter une déclaration dans l'énumération afin de séparer la liste des éléments du reste.

Les énumérations peuvent devenir des objets complexes qui fournissent de nombreux services.

```
Couleur couleur = Couleur.ROUGE;


System.out.println(couleur.isChaude()); // true
System.out.println(couleur.isFroide()); // false
System.out.println(couleur.getComplementaire()); // VERT
```

Ajout de constructeurs

Il est également possible d’ajouter un ou plusieurs constructeurs dans une énumération. Attention, ces constructeurs doivent impérativement être de portée **private** sous peine de faire échouer la compilation. L’appel au constructeur se fait au moment de la déclaration des éléments de l’énumération.

```
package com.cgi.udev;


public enum Polygone {

  TRIANGLE(3), QUADRILATERE(4), PENTAGONE(5);

  private final int nbCotes;

  private Polygone(int nbCotes) {
    this.nbCotes = nbCotes;
  }

  public int getNbCotes() {
    return nbCotes;
  }


}
```

Revision #1
Created 10 February 2025 19:55:36 by Nicolas
Updated 10 February 2025 19:57:05 by Nicolas