Enum et documentation

Arnaud Labourel arnaud.labourel@univ-amu.fr

3 novembre 2021

Section 1

Les énumérations

Énumérations en Java

En programmation, une *énumération* est un type spécial qui permet de définir des variables pouvant prendre des valeurs parmi un ensemble prédéfini de constantes.

Il est possible de définir des énumérations en Java grace au mot-clé `enum`

```
enum Suit{
  SPADES,
  HEARTS,
  DIAMONDS,
  CLUBS;
}
```

Énumérations en Java

```
enum Suit {
  SPADES, HEARTS, DIAMONDS, CLUBS;
}
```

Une énumération est une classe avec des éléments prédéfinis et statiques.

On peut donc tester directement l'égalité avec l'opérateur égal car il n'y qu'un objet et donc qu'une référence qui correspond à chaque valeur possible.

```
Suit suit = Suit.SPADES;
/* ... */
if (suit == Suit.SPADES)
{
  /* .... */
}
```

Définition de champs, de méthodes et d'un constructeur

```
public enum Suit {
  SPADES("Pique", "Pi"), HEARTS("Cœur", "Co"),
  DIAMONDS("Carreau", "Ca"), CLUBS("Trèfle", "Tr");

  private final String frenchName;
  private final String frenchSymbol;

  Suit(String name, String symbol) {
    this.frenchName = name;
    this.frenchSymbol = symbol;
  }
  public String frenchName() { return frenchName; }
  public String frenchSymbol() { return frenchSymbol; }
}
```

Classe `Enum`

Toutes les énumération de java étendent la classe `Enum`.

Quelques méthodes utiles :

- `String name()` : retourne le nom de la constante tel que déclaré dans l'`enum`.
- `int ordinal()` : retourne la position de la constante dans la déclaration de l'`enum` (commençant par 0).

Tout `enum` définit aussi une méthode statique `values()` renvoyant un tableau contenant les constantes dans l'ordre de leurs déclarations.

Exemple d'utilisation d'énumération (1/2)

```
public static void main(String[] args) {
  Suit[] values = Suit.values();
  for (Suit suit : values)
  System.out.printf("Le symbole de %s est %s.\n",
          suit.frenchName(), suit.frenchSymbol());
}
```

Affichage :

```
Le symbole de Pique est Pi.
Le symbole de Cœur est Co.
Le symbole de Carreau est Ca.
Le symbole de Trèfle est Tr.
```

Exemple d'utilisation d'énumération (2/2)

```
public static void main(String[] args) {
  for (Suit suit : Suit.values())
  System.out.printf("The position of %s is %d.\n",
                    suit.name(), suit.ordinal());
}
```

Affichage :

```
The position of SPADES is 0
The position of HEARTS is 1
The position of DIAMONDS is 2
The position of CLUBS is 3
```

Mot-clé `switch` et `enum`

Supposons qu'on est une `enum` pour les jour de la semaine :

```
enum Day
{
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY;
}
```

Mot-clé switch et enum

```
public void dayIsLike() {
  switch (day) {
    case MONDAY:
      System.out.println("Mondays are bad.");
      break;
    case FRIDAY:
      System.out.println("Fridays are better.");
      break;
    case SATURDAY:
    case SUNDAY:
      System.out.println("Weekends are best.");
      break;
    default:
      System.out.println("Midweek days are so-so.");
      break;
  }
```

Section 2

Commentaires et documentation

Commentaires inutiles

```
String get(String[] source, int index) {
    // Teste si l'index est dans les limites du tableau.
    if (index < 0 || index >= source.length)
        return null;
    return source[index];
}
```

Si un commentaire semble nécessaire, le remplacer par une méthode :

```
boolean indexIsInBounds(String[] source, int index) {
    return index >= 0 && index < source.length;
}
String get(String[] source, int index) {
    if (!indexIsInBounds(source, index)) return null;
    return source[index];
}
```

Commentaires inutiles

Les commentaires se désynchronisent souvent du code :

```
/* doit toujours retourner true. */
boolean isAvailable() {
    return true;
}
```

risque de devenir un jour :

```
/* doit toujours retourner true. */
boolean isAvailable() {
    return false;
}
```

Commentaires inutiles = répétition

Commentaires inutiles

Des commentaires qui peuvent sembler utiles :

```
/* une méthode qui retourne que les carrés : */
List<Rectangle> get(List<Rectangle> list) {
    /* le résultat sera stocké dans cette liste : */
    List<Rectangle> list2 = new ArrayList<Rectangle>();
    for (Rectangle x : list)
        if (x.w == x.h /* un carré ? */)
            list2.add(x);
    return list2;
}
class Rectangle {
    public int w; /* largeur */
    public int h; /* hauteur */
}
```

Commentaires inutiles

On peut se passer de commentaire en rajoutant une méthode et en nommant correctement les éléments du code.

```
List<Rectangle> findSquares(List<Rectangle> rectangles) {
    List<Rectangle> squares = new ArrayList<Rectangle>();
    for (Rectangle rectangle : rectangles)
        if (rectangle.isSquare())
            squares.add(rectangle);
    return squares;
}
class Rectangle {
    private int width, height;
    boolean isSquare() {
        return width == height;
    }
}
```

Commentaires **utiles**

Des commentaires pour décrire les tâches à réaliser peuvent être utiles

```
void processElement(Stack<Formula> stack,
                    String element) {
    // TODO : prendre en compte les signes '-' et '/'
    switch (element) {
        case "+": processSum(stack); break;
        case "*": processProduct(stack); break;
        default : processInteger(stack, element);
        break;
    }
}
```

Commentaires **utiles**

Documentation ou spécification du comportement d'une méthode :

```
/**
 * Returns <tt>true</tt> if this list contains the
 * specified element. More formally, returns
 * <tt>true</tt> if and only if this list contains
 * at least one element <tt>e</tt> such that
 * <tt>(o==null ? e==null : o.equals(e))</tt>.
 *
 * @param o element whose presence in this list is
 * to be tested
 * @return <tt>true</tt> if this list
 * contains the specified element
 */
public boolean contains(Object o) {
    return indexOf(o) >= 0;
```

JavaDoc

JavaDoc permet de générer automatiquement une documentation du code à partir de commentaires associés aux classes, méthodes, propriétés, ...

La documentation contient :

- Une description des membres : attributs et méthodes (publics et protégés) des classes
- Une description des classes, interfaces, ...
- Des liens permettant de naviguer entre les classes
- Des informations sur les implémentations et extensions

Un bloc de commentaire Java commençant par /** deviendra un bloc de commentaire Javadoc qui sera inclus dans la documentation du code source.

JavaDoc

Tag	Description
@author	pour préciser l'auteur de la fonctionnalité
@deprecated	indique que l'attribut, la méthode ou la classe est dépréciée
@return	pour décrire la valeur de retour
{@code `literal`}	Formate `literal` en code
{@link `reference`}	permet de créer un lien

Pour générer la javadoc en IntelliJ

Tools → generate Javadoc

JavaDoc

```
/**
 * The {@code Byte} class wraps a value of primitive
 * type {@code byte} in an object.  An object of type
 * {@code Byte} contains a single field whose type is
 * {@code byte}.
 *
 * <p>In addition, this class provides several methods
 * for converting a {@code byte} to a {@code String}
 * and a {@code String} to a {@code byte}, as well as
 * other constants and methods useful when dealing
 * with a {@code byte}.
 *
 * @author  Nakul Saraiya
 * @author  Joseph D. Darcy
 */
```