Les enums (classes limitées)

Parfois on connais déjà le domaine d'une classe et il est assez réduit. Par exemple si on a une classe `Suit`, on sait déjà que les seules valeurs possibles sont `Spade`, `Heart`, `Diamond` et `Tremol`.

On peut un peu imaginer les enums comme des collections de constantes. Contrairement à d'autres languages de programmation comme le Rust où les `enum` sont plus flexibles.

Définition d'une enum simple

Dans ce cas on peut donc créer un objet special appellé `enum` en y définissant les valeurs possibles :

```
public enum Suit {
    SPADE, HEART, DIAMOND, TREMOL;

    // On pourrait créer nos méthodes ici si on a en a
    public int getValue() {
        // On peut utiliser un swicth pour tester un enum (this fait référence à l'objet actuel)
        switch (this) {
            case SPADE: return 1;
            case HEART: return 2;
            case DIAMOND: return 3;
            case TREMOL: return 4;
            default: return 0; // Une valeur par défault est obligatoire même si tous les cas ont été traité
        }
    }

    // TODO Faire un exemple avec les positions des objets

    public void print() {
        // Les enums ont une valeur string par défault
        System.out.println(this); // Va écrire SPADE, HEART, DIAMOND ou TREMOL dans la console
    }
```

```
    // On peut aussi récupérer un membre d'un enum depuis un String
    public static Suit getSuitFromString(String name) {
        // Attention ! Le nom est case sensitive et si le nom n'est pas trouvé le programme va crash
        return Suit.valueOf(name);
    }
}
```

Définition d'une enum avec attributs

Mais on peut aussi avoir des attributs dans des enums :

```
public enum Suit {
    SPADE(false), HEART(true), DIAMOND(true), TREMOL(false);

    // On va avoir un seul attribut ici appellé "isRed" pour savoir la couleur de la carte
    // Il est condiéré être une bonne pratique de mettre les attributs en final car ils ne sont pas sensé être modifié
    private final boolean isRed;

    // On crée un constructeur pour pouvoir modifier cette valeur
    // Le constructeur n'est pas public car on ne peut pas créer de nouveaux objets
    Suit(boolean isRed) {
        this.isRed = isRed;
    }

    // On va juste faire une petite méthode pour montrer que l'on récupère cet attribut comme dans une classe
    public String getColor() {
        if (this.isRed) {
            return "Red";
        } else {
            return "Black";
        }
    }
}
```

Comment utiliser une enum

Maintenant pour l'utiliser on a plus besoin de `new` :

```
// On met l'objet SPADE dans une variable "spade" de type "Suit"
Suit spade = Suit.SPADE;

// On peut récupérer la position d'un élément dans un enum avec .ordinal()
System.out.printf("La position de HEART dans l'enum est : %d\n", Suit.HEART.ordinal());

// On peut aussi retrouver un élément par sa position en transformant l'enum en Array et en prenant la position
1
Suit heart = Suit.values()[1];

// Et on peux comparer les positions de deux avec .compareTo()
int difference = heart.compareTo(Suit.HEART);
System.out.printf("Si 0 == %d alors c'est un coeur\n", difference);

// On peut maintenant comme n'importe quel classe, appeller ses méthodes tel que "getColor()"
System.out.println(spade.getColor());
```

Convertir une enum en une autre

Mais on peut aussi transformer une enum en une autre si les noms sont compatibles.

```
// Disons une permiere enum "Foo" qui as un attribut String value
// Les attributs vont être perdu lors de la conversion en revanche
enum Foo {
  ONE("hello"), TWO("world"), THREE("foo"), FOUR("bar");

  private final String value;

  Foo(String value) {
    this.value = value;
  }
```

```
  public String getValue() {
    return this.value;
  }
}

// Voici une seconde enum "Bar", la clé ici est que les membres de Foo et de Bar ont le même nom
enum Bar {
  ONE, TWO, THREE, FOUR;
}

// Disons que l'on veut convertir des membres de Foo en Bar
Foo first = Foo.ONE;

// Pour cela on peut prendre le nom de foo avec toString
String name = first.toString();

// Ensuite on peut récupérer le membre du deuxième sur base de son nom avec valueOf
Bar second = Bar.valueOf(name);

// Et donc maintenant "second" vaut bien Bar.ONE et que first vaut toujours bien Foo.ONE
System.out.println(second.equals(Bar.ONE)); // affiche true
System.out.println(first.equals(Foo.ONE));  // affiche true
```

Résumé

Un enum *en Java* :

- Est un moyen de fixer le domaine d'une classe à quelques éléments
- Est immuable (on ne peut pas ajouter de nouveaux objets, ni modifier les attributs)
- Supporte l'utilisation de Switch et peuvent être automatiquement converti en String.
- Chaque membre possède une position et on peut comparer les différents objets par leurs positions.
- Ne nécésite pas de `new` pour être instancié.
- Ses attributs doivent être `private final` pour le rendre immuable et encapsuler ses attributs.

Les méthodes par défault des enums :

- `.compareTo(AutreEnum)` pour comparer les positions de 2 membres d'un enum
- `.ordinal()` pour avoir la position d'un membre de l'enum
- `.values()` pour convertir l'enum en tableau
- `.toString()` donne le nom du membre de l'enum

- `.valueOf(String)` pour récupérer un enum à partir d'un String (donc avec toString très utile pour convertir 2 enums qui ont les même noms de membres)

Revision #3
Created 27 April 2023 06:31:40 by SnowCode
Updated 1 May 2023 09:21:24 by SnowCode