Les énumérations

Les énumérations constituent une notion nouvelle depuis Java 5. Ce sont des structures qui définissent une liste de valeurs possibles. Cela vous permet de créer des types de données personnalisés. Nous allons par exemple construire le type `Langage` qui ne peut prendre qu'un certain nombre de valeurs : JAVA, PHP, C, etc.

Le principe est très simple, vous allez voir !

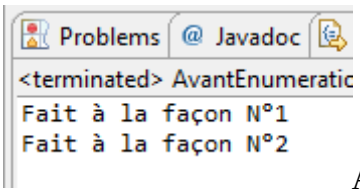
Avant les énumérations

Vous aurez sans doute besoin, un jour ou l'autre, de données permettant de savoir ce que vous devez faire. Beaucoup de variables statiques dans Java servent à cela, vous le verrez bientôt dans une prochaine partie.

Voici le cas qui nous intéresse :

```
public class AvantEnumeration {

  public static final int PARAM1 = 1;
  public static final int PARAM2 = 2;

  public void fait(int param){
    if(param == PARAM1)
      System.out.println("Fait à la façon N°1");
    if(param == PARAM2)
      System.out.println("Fait à la façon N°2");
  }

  public static void main(String args[]){
    AvantEnumeration ae = new AvantEnumeration();
    ae.fait(AvantEnumeration.PARAM1);
    ae.fait(AvantEnumeration.PARAM2);
    ae.fait(4);
  }
}
```

Voyons le rendu de ce test en figure suivante.



Avant les énumérations, des erreurs étaient possibles

Je viens de vous montrer non seulement le principe dont je vous parlais, mais aussi sa faiblesse. Vous voyez que rien ne vous empêche de passer un paramètre inattendu à une méthode : c'est ce qui s'est passé à la dernière ligne de notre test. Ici, rien de méchant, mais vous conviendrez tout de même que le comportement de notre méthode est faussé !

Bien sûr, vous pourriez créer un objet qui vous sert de paramètre de la méthode. Eh bien c'est à cela que servent les `enum` : fabriquer ce genre d'objet de façon plus simple et plus rapide.

Une solution : les enum

Une énumération se déclare comme une classe, mais en remplaçant le mot-clé `class` par `enum`. Autre différence : les énumérations héritent de la classe `java.lang.Enum`.

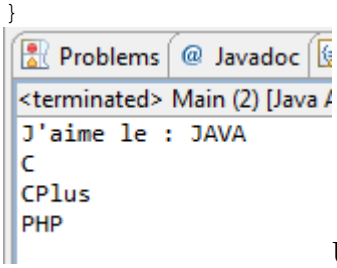
Voici à quoi ressemble une énumération :

```
public enum Langage {
  JAVA,
  C,
  CPlus,
  PHP;
}
```

Rien de difficile ! Avec cela, vous obtenez une structure de données qui encapsule quatre « objets ». En fait, c'est comme si vous aviez un objet `JAVA`, un objet `C`, un objet `CPlus` et un objet `PHP` partageant tous les mêmes méthodes issues de la classe `java.lang.Object` comme n'importe quel autre objet : `equals()`, `toString()`, etc.

Vous constatez aussi qu'il n'y a pas de déclaration de portée, ni de type : les énumérations s'utilisent comme des variables statiques déclarées `public` : on écrira par exemple `Langage.JAVA`. De plus, vous pouvez recourir à la méthode `values()` retournant la liste des déclarations de l'énumération dont vous trouverez un exemple à la figure suivante et sur son code :

```
public class Main {
  public static void main(String args[]){
    for(Langage lang : Langage.values()){
      if(Langage.JAVA.equals(lang))
        System.out.println("J'aime le : " + lang);
      else
        System.out.println(lang);
    }
  }
}
```



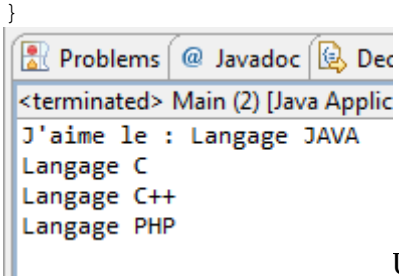
Utilisation d'une enum

Vous disposez ainsi d'un petit aperçu de l'utilisation des énumérations. Vous aurez pu constater que la méthode `toString()` retourne le nom de l'objet défini dans l'énumération.

À présent, étoffons tout cela en redéfinissant justement cette méthode. Pour ce faire, nous allons ajouter un paramètre dans notre énumération, un **constructeur** et ladite méthode redéfinie. Voici notre nouvelle énumération (résultat en figure suivante) :

```
public enum Langage {
  //Objets directement construits
  JAVA ("Langage JAVA"),
  C ("Langage C"),
```

```
  CPlus ("Langage C++"),
  PHP ("Langage PHP");

  private String name = "";

  //Constructeur
  Langage(String name){
    this.name = name;
  }

  public String toString(){
    return name;
  }
}
```



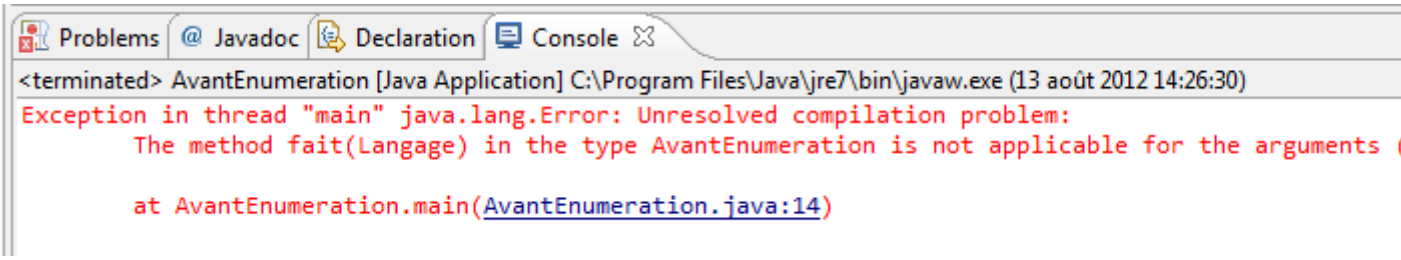
Utilisation d'un constructeur avec une enum

Même remarque pour le constructeur : pas de déclaration de portée, pour une raison simple ; il est toujours considéré comme `private` afin de préserver les valeurs définies dans l'`enum`. Vous noterez par ailleurs que les données formant notre énumération sont directement construites dans la classe.

Voici le code du début de chapitre, revu pour préférer les énumérations aux variables statiques :

```
public class AvantEnumeration {

  public void fait(Langage param){
    if(param.equals(Langage.JAVA))
      System.out.println("Fait à la façon N°1");
    if(param.equals(Langage.PHP))
      System.out.println("Fait à la façon N°2");
  }

  public static void main(String args[]){
    AvantEnumeration ae = new AvantEnumeration();
    ae.fait(Langage.JAVA);
    ae.fait(Langage.PHP);
    ae.fait(4);
  }
}
```

La figure suivante nous montre ce que cela donne.



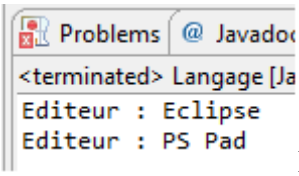
Code du début de chapitre avec une enum

Une belle exception ! Normal, puisque la méthode attend un certain type d'argument, et que vous lui en passez un autre : supprimez la dernière ligne, le code fonctionnera très bien. Maintenant, nous avons un mécanisme protégé : seuls des arguments valides peuvent être passés en paramètres de la méthode.

Voici un exemple plus complet :

```
public enum Langage {
  //Objets directement construits
  JAVA("Langage JAVA", "Eclipse"),
  C ("Lanage C", "Code Block"),
  CPlus ("Langage C++", "Visual studio"),
  PHP ("Langage PHP", "PS Pad");

  private String name = "";
  private String editor = "";

  //Constructeur
  Langage(String name, String editor){
    this.name = name;
    this.editor = editor;
  }

  public void getEditor(){
    System.out.println("Editeur : " + editor);
  }

  public String toString(){
    return name;
  }

  public static void main(String args[]){
    Langage l1 = Langage.JAVA;
    Langage l2 = Langage.PHP;

    l1.getEditor();
    l2.getEditor();
  }
}
```

Voyons le résultat de cet exemple à la figure suivante.



Exemple plus complet

Vous voyez ce que je vous disais : les énumérations ne sont pas très difficiles à utiliser et nos programmes y gagnent en rigueur et en clarté.

- Une énumération est une classe contenant une liste de sous-objets.
- Une énumération se construit grâce au mot clé `enum`.
- Les `enum` héritent de la classe `java.lang.Enum`.
- Chaque élément d'une énumération est un objet à part entière.
- Vous pouvez compléter les comportements des objets d'une énumération en ajoutant des méthodes.