Avant...

- Le pattern “int-enum”.

```
1  public static final int POMME_FUJI = 0 ;
2  public static final int POMME_GALA = 1 ;
3  public static final int POMME_GRANNY = 2 ;
4
5  public static final int ORANGE_NAVEL = 0 ;
6  public static final int ORANGE_TEMPLE = 1 ;
7  public static final int ORANGE_SANGUINE = 2 ;
```

- Problem(s) ?

Avant...

- Problem(s) ?

```
1  public static void decreaseStock(int orange) {
2    switch(orange) {
3    case ORANGE_NAVEL :
4      //...
5      break ;
6    case ORANGE_TEMPLE :
7      //...
8      break ;
9    case ORANGE_SANGUINE :
10     //...
11     break ;
12   default :
13     //...
14     break ;
15   }
16 }
17 public static void main(String[] args) {
18   decreaseStock(POMME_FUJI) ; //compile !
19   System.out.println(POMME_GALA == ORANGE_TEMPLE) ; // ?
20   System.out.println(POMME_GRANNY) ; //affiche 2  :( debuggage pas facile
21 }
```

Enums !

```
1   enum Pomme {Fuji, Gala, Granny}
2   enum Orange {Navel, Temple, Sanguine}
3
4   public static void decreaseStock(Orange orange) {
5     switch(orange) {
6     case Navel :
7       //...
8       break ;
9     }
10  }
11  public static void main(String[] args) {
12    decreaseStock(Pomme.Fuji) ; //compile plus !
13    System.out.println(Pomme.Gala == Orange.Temple) ; //compile pas
14    System.out.println(Pomme.Granny) ; //affiche Granny !
15
16  }
```

Type énuméré

- **Type** contenant un nombre fini de valeurs de ce type.
- Exemple (ensemble de valeurs constantes fini connu à la compilation) :
  - Jeux de cartes.
  - Options d'un appel système.
  - Points cardinaux.
  - Jours de la semaine.
  - ...
- Depuis Java 5.

Type énuméré

- Les valeurs d'une énumération sont des
  - **objets**
  - **constants** (final)
  - **uniques** (static)

  avec
  - une valeur entière unique (`ordinal()`)
  - un nom unique (`name()`)
- Une énumération connaît l'ensemble de ses valeurs (`values()`, permet d'itérer) et est capable d'associer un nom à une valeur (`valueOf()`).

Type énuméré

- Les valeurs d'une énumération sont des
  - **objets**
  - **constants** (final)
  - **uniques** (static)

  avec
  - une valeur entière unique (`ordinal()`)
  - un nom unique (`name()`)
- Une énumération connaît l'ensemble de ses valeurs (`values()`, permet d'itérer) et est capable d'associer un nom à une valeur (`valueOf()`).
- On peut en plus y ajouter des constructeurs, méthodes, champs, classes internes...

Exemple

```
1  public enum Option {
2    l, a, v ;
3
4    public static Option getOption(String s) {
5      if (s.length() !=2)
6        return null ;
7      if (s.charAt(0) !='-')
8        return null ;
9      return Option.valueOf(s.substring(1)) ;
10   }
11
12   public static void main(String[] args) {
13     for(String s :args) {
14       Option option=getOption(s) ;
15       if (option !=null)
16         System.out.println(option.name()+" "+option.ordinal()) ;
17     }
18   }
19   //java Option -a -v -l
20   // a 1 v 2 l 0
21 }
```

Champs

- Champs de l'énumération accessibles par la notation ".".
  - `Option.a`
- Un champ à le type de l'énumération.
  - `Option o = Option.a ;`
- Enumération vide interdite.
- Les champs doivent toujours être les premières instructions (avant le reste).

Constructeurs

- Possible de spécifier un ou plusieurs **constructeurs** (arguments entre parenthèses pour l'appeler)
- Parenthèses non obligatoires (constructeur par défaut).

```
1  public enum Age {
2    jeune, pasIeuv(), mur(40), âgé(60), vieux(80), cadavérique(999) ;
3
4    private final int année ; //champ
5
6    Age(int année) { //constructeur
7      this.année=année ;
8    }
9    Age() { //constructeur sans arg (car sinon plus de constructeur par défaut)
10     this(20)
11   }
12   private static Age getAge(int année) {
13     for(Age age :Age.values())
14       if (age.année>=année)
15         return age ;
16     return cadavérique ;
17   }
18   public static void main(String[] args) {
19     System.out.println(getAge(new Scanner(System.in).nextInt())) ;
20   }
21 }
```

Constructeurs

- Constructeur d'énum uniquement de visibilité de package ou privé.
- Impossible de faire `Age a = new Age(20);`.
- Car type-safe : une enum ne peut pas prendre une **valeur autre que de l'ensemble fini défini**.

Enumeration et classe interne

- Interdit de définir une énumération dans une méthode ou une classe interne non statique.

```
1  public class Outer {
2    enum Enu1 { //ok
3      toto
4    }
5    class Inner {
6      enum Enu2 {
7        // interdit
8        toto
9      }
10   }
11   void m() {
12     enum Enu3 { // interdit
13       toto
14     }
15   }
16   static class InnerStat {
17     enum Enu4 {
18       toto //ok
19     }
20   }
21 }
```

Switch

- Peut être utilisé pour faire un switch.

```
1  public enum Option {
2    l, a, v ;
3
4    public static void performs(Set<Option> set) {
5      for(Option option :set)
6        switch(option) {
7        case l : // Option.l compile pas
8          System.out.println("l") ;
9          break ;
10       case a :
11         System.out.println("a") ;
12         break ;
13       case v :
14         System.out.println("v") ;
15         break ;
16       }
17   }
18 }
```

- Mais code pas objet, peut mieux faire !

Better Switch

- Utiliser des **enumérations abstraites** (attention, pas de `abstract` dans le nom de l'enum) !
- Même règles que les classes anonymes.

```
1  public enum Option {
2    l {
3      @Override public void performs() {
4        System.out.println("l") ;
5      }
6    },
7    a {
8      @Override public void performs() {
9        System.out.println("a") ;
10     }
11   },
12   v {
13     @Override public void performs() {
14       System.out.println("v") ;
15     }
16   } ; //attention au point virgule  !
17   public abstract void performs() ; //
         oblige a definir performs
18 }
```

```
1 static void applyOpts(Set<Option> set) {
2   for(Option option :set)
3     option.performs() ; //possible
4 }
```

Assertion

```
1  enum Option {a,b,c}
2
3  public int getOpt(Option o) {
4    switch(o) {
5      case a : return 1 ;
6      case b : return 2 ;
7      case c : return 3 ;
8      default : throw new AssertionError("Invalid option") ;
9    }
10 }
```

(Sans) Assertion : mieux

```
1  public enum Option {
2    a(1),b(2),c(3) ;
3    private final int num ;
4    Option(int n) { //"constructeur"
5      this.num=n ;
6    }
7    public int getOpt() {
8      return num ;
9    }
10 }
```

```
1  public int bla(Option o) {
2    return o.getOpt() ;
3  }
```

Méthodes

- On peut déclarer des méthodes (statiques ou non) dans l'enum.

```
1  public enum A {
2    a,b ;
3    public int f(int i) {
4      return i+ordinal() ;
5    }
6    public static int g() {
7      //ordinal() pas accessible car statique !
8      return 42 ;
9    }
10
11   public static void main(String[] args) {
12     System.out.println(A.a.f(5)) ;
13     System.out.println(A.b.f(5)) ;
14     System.out.println(A.g()) ;
15   }
16 }
```

Affiche ?

Méthodes

- On peut déclarer des méthodes (statiques ou non) dans l'enum.

```
1  public enum A {
2    a,b ;
3    public int f(int i) {
4      return i+ordinal() ;
5    }
6    public static int g() {
7      //ordinal() pas accessible car statique !
8      return 42 ;
9    }
10
11   public static void main(String[] args) {
12     System.out.println(A.a.f(5)) ;
13     System.out.println(A.b.f(5)) ;
14     System.out.println(A.g()) ;
15   }
16 }
```

Affiche ?

- 5, 6, 42

Héritage

- Toutes les énumérations héritent de `java.lang.Enum`.
- Le compilateur assure que :
  - Une classe ne peut pas hériter (via extends) de `Enum`.
  - Une classe ne peut pas hériter d'une énumération.
  - Une énumération ne peut pas hériter d'une classe ou d'une énumération.

```
1 public class A extends Enum { } // erreur
2 public class A extends Option { } // erreur
3 public enum Option extends A { } // erreur
4 public enum Option extends Option { } // erreur
```

Héritage

- Une enum peut **implémenter** une ou plusieurs **interfaces**

```
1  public interface Performer {
2    void performs() ;
3  }
```

```
1  public enum Test implements Performer{
2    a {
3      @Override
4      public void performs() {
5        System.out.println("a") ;
6      }
7    },
8    b {
9      @Override
10     public void performs() {
11       System.out.println("b") ;
12     }
13   } ; // ;  !
14 }
```

Enum en résumé

- Plus que des entiers : des objets avec des méthodes associées (ordinal, name...).
- Switch possible dessus.
- Ajout de méthodes, champs possible.
- Enum abstraites et redéfinitions de méthodes.