Chapitre 14

Notions objet avancées

1. Classes enveloppes

Chacun des 8 types primitifs (**`boolean`**, **`byte`**, **`char`**, **`short`**, **`int`**, **`long`**, **`float`** et **`double`**) possède une classe correspondante, appelée **classe enveloppe**, qui généralise le type.

Les 8 classes enveloppes sont **`Boolean`**, **`Byte`**, **`Character`**, **`Short`**, **`Integer`**, **`Long`**, **`Float`** et **`Double`**.

Ces classes enveloppes sont définies dans le package **java.lang** et peuvent donc être utilisées sans importation.

Les classes enveloppes permettent de représenter un type simple par un objet lorsque c'est nécessaire.

Par exemple, on peut utiliser une classe enveloppe **Integer** dans une **ArrayList** alors que cette liste n'accepte pas le type simple **int** :

```
ArrayList<int> liste = new ArrayList<int>();          // N'est pas permis
ArrayList<Integer> liste = new ArrayList<Integer>();  // Est permis
```

Les classes enveloppes fournissent aussi des constantes ou des méthodes intéressantes, comme les valeurs minimum et maximum du type, ainsi que des méthodes de conversion vers d'autres types :

https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Integer.html

Integer.MAX_VALUE

Constante contenant la valeur maximale d'un int (2^{31}-1)

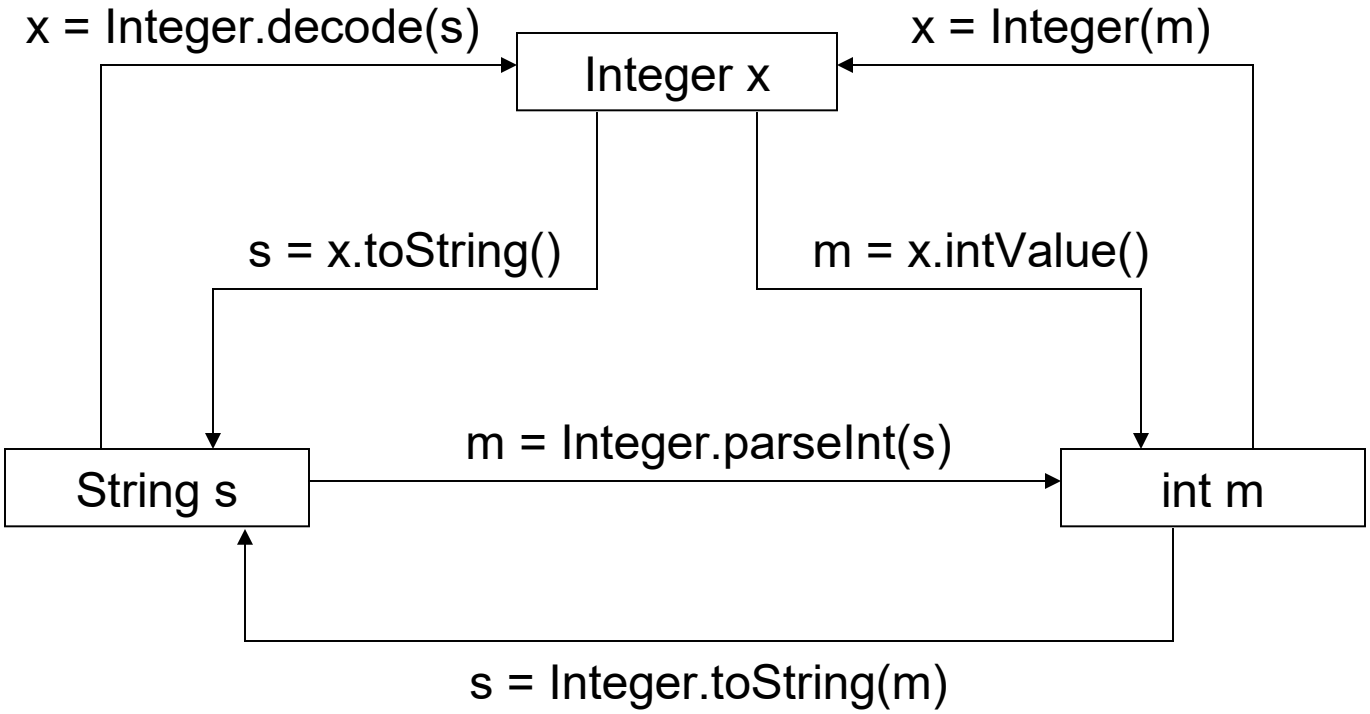
Integer.MIN_VALUE

Constante contenant la valeur minimale d'un int (-2^{31})

Integer.BYTES

Nombre d'octets utilisés pour représenter un int.

Méthodes de conversion entre le type **`int`** et les classes **`Integer`** et **`String`** :



Des méthodes similaires existent pour les 7 autres classes enveloppes.

2. Interfaces

Une interface se présente comme ci-dessous : le mot-clé **`interface`** suivi de son nom, puis, entre accolades, une liste de **constantes** (mais pas de variable) et de **prototypes** de méthodes :

```
public interface Forme
{
    static final int DIM_MAX=100;
    float surface();
}
```

Prototype d'une fonction = en-tête de la déclaration de la fonction (type retourné, nom, paramètres), sans son corps.

Une interface **`I`** peut être **implémentée** par une classe **`A`**, ce qui signifie que :

- la classe **`A`** déclare étendre l'interface **`I`** ;
- toutes les méthodes figurant dans l'interface **`I`** doivent être définies par la classe **`A`**.

Ici, l'interface **`Forme`** comporte la constante entière **`DIM_MAX`** et le prototype de la méthode **`surface()`**

→ Une classe qui implémente l'interface **`Forme`** sera dans l'obligation de définir la méthode **`surface()`** et possédera la constante **`DIM_MAX`**.

Exemple

Forme.java

```
public interface Forme
{
   public float surface();
}
```

Rectangle.java

```
public class Rectangle implements Forme
{
   float longueur, largeur;

   public float surface()
   {
      return longueur * largeur;
   }
}
```

Les attributs et les méthodes d'une interface sont automatiquement publiques, ce qui entraîne qu'une classe qui implémente l'interface devra déclarer publique (avec le modificateur **public**) les méthodes de l'interface qu'elle définira.

Une interface :

- sert à regrouper des constantes
- mais surtout sert chaque fois que l'on veut traiter des objets qui ne sont pas nécessairement tous de la même classe mais qui ont en commun une partie "interface" et que seule cette partie "interface" intervient dans le traitement que l'on veut définir.

  Lorsqu'on utilise un objet d'une classe implémentant une certaine interface, on a la garantie que cet objet dispose de toutes les méthodes annoncées par l'interface.

Remarques

Une interface :

- Ne peut pas être instanciée
- Ne contient pas de variables
- Peut hériter d'une autre interface
- Une classe peut implémenter plusieurs interfaces
  - Si une classe **`A`** implémente une interface **`I`**, les sous-classes de **`A`** implémentent aussi automatiquement **`I`**.

3. Classes abstraites

Dans une classe, une méthode est dite **abstraite** si seul son prototype figure. Elle doit alors être déclarée abstract :

```
abstract float surface();
```

Une classe est abstraite dès qu'elle contient une méthode abstraite, elle doit alors être déclarée abstract :

```
abstract class Forme
{
   abstract float surface(); // méthode abstraite
   ...
```

Une classe abstraite ne peut pas être instanciée.

```
Forme une_forme = new Forme() ;   // erreur
```

→ il faut l'étendre pour pouvoir l'utiliser. Une sous-classe d'une classe abstraite sera encore abstraite si elle ne définit pas toutes les méthodes abstraites dont elle hérite.

Exemple

Forme.java

```
abstract class Forme
{
   abstract float surface(); //méthode abstraite

   void affiche_surface()
   {
      float la_surface = surface();
      System.out.println("surface=" + la_surface);
   }
}
```

Rectangle.java

```
class Rectangle extends Forme
{
  private float longueur, largeur;

  Rectangle(float lon, float lar)
  {
    longueur = lon;
    largeur = lar;
  }

  float surface()
  {
     return longueur*largeur;
  }
}
```

EssaiFormes.java

```
class EssaiFormes
{
  public static void main(String[] argv)
  {
    Rectangle rect = new Rectangle(4,3);
    System.out.println(rect.surface());
  }
}
```