Programmation Objet. Cours 3

Marie-Pierre Béal

DUT 1

Types primitifs et enveloppes. Tableaux. ArrayList. Objects.requireNonNull().

Types primitifs

Nom	Taille	Exemples
`byte`	8	1, -128, 127
`short`	16	2, 300
`int`	32	234569876
`long`	64	2L
`float`	32	3.14, 3.1E12, 2e12
`double`	64	0.5d
`boolean`	8	`true` ou `false`
`char`	16	`'a'`, `''\n'`, `'\u0000'`

- Les caractères sont codés sur deux octets en Unicode.
- Les types sont indépendants du compilateur et de la plateforme.
- Tous les types numériques sont signés sauf les caractères.
- Un booléen n'est pas un nombre.
- Les opérations sur les entiers se font modulo, et sans erreur :

```
byte b = 127; b += 1; // b = -128
```

Enveloppes des types primitifs

Il existe un type objet correspondant à chaque type primitif.

`boolean`	java.lang.Boolean
`char`	java.lang.Character
`byte`	java.lang.Byte
`short`	java.lang.Short
`int`	java.lang.Integer
`long`	java.lang.Long
`float`	java.lang.Float
`double`	java.lang.Double

Enveloppes des types primitifs

- Une instance de la classe enveloppe encapsule une valeur du type de base correspondant.
- Chaque classe enveloppe possède des méthodes pour extraire la valeur d'un objet (par exemple `o.intValue()` appliquée sur un objet `o` de la classe `Integer` renvoie une valeur de type `int`).
- Une méthode statique de chaque classe enveloppe renvoie un objet enveloppant le primitif correspondant (par exemple `Integer.valueOf(int p)` renvoie un objet enveloppant le primitif correspondant.
- Un objet enveloppant est immutable : la valeur contenue ne peut être modifiée.
- On transforme souvent les valeurs primitives en objets pour les mettre dans les collections.

Conversions automatiques

Auto-boxing

```
Integer i = 3; // int -> Integer
Long l = 3L; // long -> Long
Long l = 3; // erreur, int -> Integer -X-> Long
```

Auto-unboxing

```
Integer i = new Integer(3);
int x = i; // Integer -> int
Long lo = null;
long l = lo; //erreur : java.lang.NullPointerException
```

Les conversions se font aussi pour les paramètres lors d'appel de méthodes et pour les valeurs de retour des méthodes.

Auto-boxing et égalité

Sur des objets, == teste l'égalité des références. Les enveloppes obtenues par auto-boxing ont la même référence.

```
public static void main(String[] args){
        Integer i = 6;
        Integer j = 6;
        Integer k = new Integer(6);
        System.out.println(i == j); //true, i,j petits
        System.out.println(i == k); //false
        i = i+1;//Integer -> int +1 -> int -> Integer
        System.out.println(i); // 7
        System.out.println(j); // 6
}
```

Conclusion : ne jamais tester l'égalité de références.

Tableaux

Les tableaux sont des objets particuliers. L'accès se fait par référence et la création par `new`. Un tableau

- se *déclare*,
- se *construit*,
- et s'*utilise*.

Tableaux de dimension 1

Déclaration

```
int[] tab; // vecteur d'entiers
```

Construction

```
int n = 10;
tab = new int[n] ;
```

Utilisation

```
for (i = 0; i < tab.length; i++)
  System.out.print(tab[i]);
//ou bien boucle "for each"
for (int x : tab) System.out.println(x);
```

Tableaux de dimension 2

Déclaration

```
double[][] mat; // matrice de doubles
```

Construction

```
mat = new double[n][p]; // n lignes, p colonnes
```

Utilisation

```
mat[i][j] = 10; // ligne i, colonne j
for (i = 0; i < mat.length; i++) {
   for (j = 0; j < mat[0].length; j++)
       System.out.print(mat[i][j]);
   System.out.print("\n");
}
```

Tableaux suite

On peut fusionner déclaration et construction et faire une initialisation énumérative :

```
String[] jours = {"Lundi", "Mardi", "Mercredi",
  "Jeudi", "Vendredi", "Samedi", "Dimanche"};
```

Les instructions suivantes provoquent chacune toujours une exception (de la classe `ArrayIndexOutOfBoundsException`) :

```
tab[tab.length] = 20;
tab[-1] = 20;
```

Listes tableaux (ArrayList)

Le paquetage `java.lang.util` contient une classe `ArrayList` qui permet de créer des listes tableaux dont les éléments sont tous du même type.
Dans le code ci-dessous on crée une liste qui contient des objets `Pixel`.

```
import java.util.ArrayList;
class Test {
  public static void main(String[] args) {
    ArrayList<Pixel> list = new ArrayList<Pixel>();
    Pixel p1 = new Pixel(0,0);
    Pixel p2 = new Pixel(1,1);
    list.add(p1);
    list.add(p2);
    System.out.println(list);
  }
}
```

Quelques prototypes de méthodes sur les listes

```
public boolean add(E element);
public boolean add(int index, E element);
public int size();
public E get(int);
public boolean contains(Object o);
```

où E est le type des objets contenus dans la liste.

```
class Test {
  public static void main(String[] args) {
   // suite
    System.out.println(list.size());
    Pixel p = list.get(0);
    System.out.println(p);
  }
}
```

S'assurer que les arguments sont bien des objets ?

On reprend la classe `Segment` avec un constructeur.

```
public class Segment {
  private Pixel start;
  private Pixel end;
  public Segment(Pixel start, Pixel end){
      this.start = start;
      this.end = end;
  }
}
```

S'assurer que les arguments sont bien des objets ?

Qu'affiche le code suivant ?

```
public class SegmentTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(0,0);
       Pixel p2 = null;
       Segment s1 = new Segment(p1,p2);
       System.out.println(s1.getEnd().getX());
   }
}
```

S'assurer que les arguments sont bien des objets ?

On désire s'assurer que les segments créés sont bien formés de deux objets `Pixel` (pas de référence `null`).

On peut utiliser la méthode statique `requireNonNull`, méthode outil de la classe `java.util.Objects`

```
import java.util.Objects;

public class Segment {
  private Pixel start;
  private Pixel end;
  public Segment(Pixel start, Pixel end){
      this.start = Objects.requireNonNull(start);
      this.end = Objects.requireNonNull(end);
  }
}
```

S'assurer que les arguments sont bien des objets ?

Dans ce cas l'erreur se fait à la création du segment.

```
public class SegmentTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(0,0);
       Pixel p2 = null;
       Segment s1 = new Segment(p1,p2); //erreur ici
   }
}
```

```
$ java SegmentTest
Exception in thread "main" java.lang.NullPointerException
at java.util.Objects.requireNonNull(Objects.java:201)
at Segment.<init>(Segment.java:8)
at SegmentTest.main(SegmentTest.java:5)
$
```