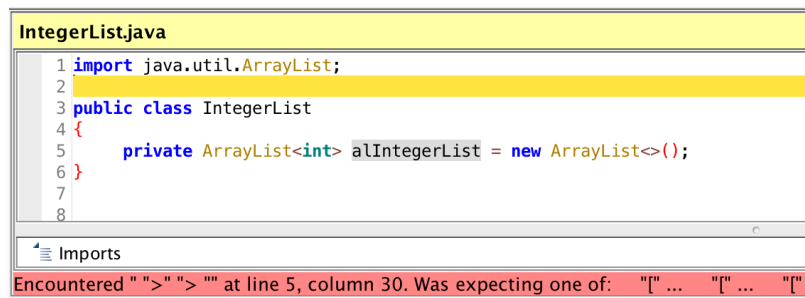


# Classes enveloppes (Wrapper classes)

Une `ArrayList` sert à regrouper plusieurs **objets**. Parfois on désire cependant regrouper des valeurs d'un type primitif c.à.d. des nombres entiers, des nombres réels ou plus rarement des booléens.

Si on essaie de déclarer une `ArrayList` contenant des valeurs primitives, des entiers dans l'exemple de l'image suivante, on obtient un message d'erreur peu expressif :



```
IntegerList.java
1 import java.util.ArrayList;
2
3 public class IntegerList
4 {
5     private ArrayList<int> alIntegerList = new ArrayList<>();
6 }
7
8
```

Imports

Encountered ">" ">" at line 5, column 30. Was expecting one of: "[" ... "[" ... "["

## Problème

On indique un type primitif entre < et > tandis qu'une `ArrayList` ne fonctionne qu'avec des objets.

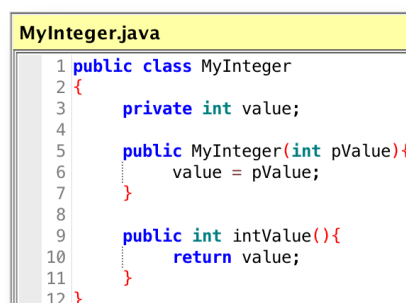
## Analogie

On peut envoyer n'importe quel contenu par voie postale tant qu'on utilise une enveloppe ou un paquet postal avec une bonne adresse.

Pour les listes en Java c'est pareil. On peut regrouper n'importe quel contenu tant qu'il a une enveloppe « objet ». Toute enveloppe, tout paquet postal possède une adresse. Les objets en Java possèdent aussi des propriétés nécessaires au bon fonctionnement d'une liste qu'un type primitif ne possède pas.

## Que faire ?

On peut créer une classe enveloppe, appelée ainsi parce qu'elle enveloppe une valeur primitive dans un objet. Souvent une telle classe ne possède qu'un seul attribut, un accesseur et un constructeur. Les objets d'une telle classe peuvent être stockés dans une liste comme illustré par les deux images qui suivent.



```
MyInteger.java
1 public class MyInteger
2 {
3     private int value;
4
5     public MyInteger(int pValue){
6         value = pValue;
7     }
8
9     public int intValue(){
10        return value;
11    }
12 }
```

```
IntegerList.java
1 import java.util.ArrayList;
2
3 public class IntegerList
4 {
5     private ArrayList<MyInteger> alIntegerList = new ArrayList<>();
6
7 }
```

### Meilleure solution

Le problème principal de la solution précédente est qu'il faudrait recréer une classe `MyInteger`, respectivement `MyDouble` ou `MyBoolean` pour chaque programme. Heureusement les développeurs Java ont pensé à créer des classes enveloppes prédéfinies à cet effet. Ces classes s'appellent `Integer` et `Double` (et `Boolean`). Une meilleure solution consiste à utiliser ces classes, qui nous permettent de déclarer nos listes comme suit, sans créer de classe additionnelle :

```
IntegerList.java
1 import java.util.ArrayList;
2
3 public class IntegerList
4 {
5     private ArrayList<Integer> alIntegerList = new ArrayList<>();
6 }
```

*Exemple 1 - Liste d'entiers*

```
DoubleList.java
1 import java.util.ArrayList;
2
3 public class DoubleList
4 {
5     private ArrayList<Double> alDoubleList = new ArrayList<>();
6 }
```

*Exemple 2 - Liste de réels*

# Auto-boxing & Auto-unboxing

On pourrait croire qu'il faut manuellement transformer les valeurs primitives en des objets en faisant appel au constructeur des classes `Integer` respectivement `Double`. De même on pourrait croire qu'il faut manuellement extraire la valeur primitive de l'objet si on veut l'accéder. Or dans la plupart des cas qu'on va rencontrer ceci se fait automatiquement.

## Auto-boxing

La conversion automatique d'un type primitif en un objet d'une classe enveloppe s'appelle **auto-boxing**. Java l'applique si

- on passe un type primitif comme paramètre à une méthode s'attendant à un objet de la classe enveloppe correspondante. On le trouve dans les deux méthodes `addNumber` des images de la page suivante.
- on retourne un type primitif dans une méthode retournant le type enveloppe correspondant.

### Exemple :

```
public Integer testMethod(){
    return 5;
    /*équivalent à
       return new Integer(5);
    */
}
```

- on affecte une valeur primitive à une variable du type de la classe enveloppe.

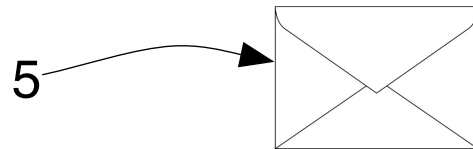
### Exemple :

```
Integer i = 5;
```

ou

```
Integer i = new Integer(5);
```

Les deux instructions sont identiques.



## Auto-unboxing

La conversion inverse d'un objet d'une classe enveloppe en type primitif s'appelle **auto-unboxing**. Java l'applique automatiquement si

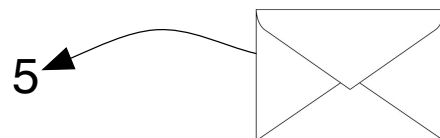
- on passe un objet d'une classe enveloppe comme paramètre à une méthode s'attendant au type primitif correspondant.
- on retourne un objet d'une classe enveloppe dans une méthode retournant le type primitif correspondant. Un exemple est donné par les deux méthodes `getNumber` des images de la page suivante.
- on affecte un objet d'une classe enveloppe à une variable du type primitif correspondant.

### Exemple :

```
Integer i = 5;
```

```
int j = i;
```

`j` finit par avoir la valeur 5 qui est automatiquement extraite de `i`



## Listes d'entiers et de réels

Grâce aux conversions automatiques expliquées ci-devant on peut plus facilement implémenter les méthodes les plus courantes pour des listes d'entiers et de réels. Ceci est illustré par l'exemple suivant.

```
IntegerList.java
1 import java.util.ArrayList;
2
3 public class IntegerList
4 {
5     private ArrayList<Integer> alIntegerList = new ArrayList<>();
6
7     public void addNumber(int pInt){
8
9         //alIntegerList.add(new Integer(pInt));
10
11         alIntegerList.add(pInt);
12     }
13
14     public int getNumber(int pIndex){
15
16         //return alIntegerList.get(pIndex).intValue();
17
18         return alIntegerList.get(pIndex);
19     }
20 }
```

Dans les deux méthodes `addNumber` et `getNumber` l'instruction commentée est absolument équivalente à l'instruction non commentée.

A la ligne 11 la méthode `add` accepte un objet de la classe enveloppe `Integer` comme paramètre. Le concept dit **auto-boxing** est appliqué à la valeur primitive `pInt` passée en paramètre.

La méthode `getNumber` retourne le type primitif `int`. A la ligne 18 la méthode `get` retourne cependant un objet de la classe enveloppe `Integer`. Par conséquent le concept dit **auto-unboxing** est appliqué à cet objet.

Il en est de même pour la liste suivante de nombres réels.

```
DoubleList.java
1 import java.util.ArrayList;
2
3 public class DoubleList
4 {
5     private ArrayList<Double> alDoubleList = new ArrayList<>();
6
7     public void addNumber(double pDouble){
8
9         //alDoubleList.add(new Double(pDouble));
10
11         alDoubleList.add(pDouble);
12     }
13
14     public double getNumber(int pIndex){
15
16         //return alDoubleList.get(pIndex).doubleValue();
17
18         return alDoubleList.get(pIndex);
19     }
20 }
```