


## Créer ses premiers objets

 **Exercice 1 :** Dans une classe *Calcul*, contenant un entier  $a$  comme attribut ;

1. Ecrivez une méthode *square* qui ne prend aucun argument et qui retourne  $a^2$ .

```

1  public class Calcul {
2
3      // To do
4
5      public Calcul(int a){
6          // To do
7      }
8
9      public int square(){
10         // To do
11     }
12
13     public static void main(String[] args) {
14         Calcul c = new Calcul(10);
15         System.out.println(c.square()); // devrait afficher 100
16     }
17 }
```

2. Ecrivez une fonction *power* qui prend un entier  $b$  en paramètre, et qui retourne  $a^b$ . (utiliser une boucle *for* ou *while*).
3. Ecrivez une fonction *factorial* qui ne prend aucun paramètre et qui retourne  $a! = \prod_{i=1}^a i$ . (utiliser une boucle *for* ou *while*).
4. Ré-écrivez la fonction *factorial* de manière récursive.
5. Ecrivez une fonction *containedIn* qui prends en paramètre un tableau  $T$  d'entiers et qui retourne *true* si  $T$  contient l'élément  $a$ , renvoie *false* sinon

```

1  // rappel tableau
2  int[] tableau = {1, 2, 3};
3  int tailleDuTableau = tableau.length;
4  // /\ indice en java : tableau[0] correspond a la valeur 1,
5  // de maniere general l'element en i eme position
6  // correspond a tableau[i-1]
7
8  // condition if en java - test d'egalite
9  // /\ avec un double "=", sinon on affecte la valeur de
10 // var2 a var1
11 // /\ compare la valeur des types primitifs (dont int,
12 // double, char, boolean), mais compare
13 // les references pour des objets (par exemple String !!!
14 // On verra cela plus tard pour les objets)
```

```

15
16  if(var1 == var2){
17      // code execute uniquement si var1 == var2
18  }
19
20  // rappel : une variable de type boolean ne
21  // prend soit la valeur true soit false
22  public boolean containedIn(int[] T){
23      // To do
24  }

```

6. Ecrivez une fonction *isPrime* qui renvoie *true* si *a* est un nombre premier, et *false* sinon.

```

1  // bb % 2 correspond a bb modulo 2 (i.e. le reste de la
2  // division euclidienne de bb par 2)


```

7. Ecrivez une fonction *max* qui prends un tableau d'entiers en paramètre et qui retourne le plus grand entier dans la liste
8. Ecrivez une fonction *insertionSort* qui prend en entrée un tableau d'entiers et qui le retourne trié par ordre croissant. Ci dessous, le pseudo-code de l'algorithme du tri par insertion


```

1  // tri par insertion - pseudo code
2  procedure tri_insertion(tableau T)
3      n prend la valeur taille(T)
4      pour i de 1 a n - 1
5
6          // memoriser T[i] dans x
7          x prend la valeur T[i]
8
9          // decaler vers la droite les elements de
10         // T[0]..T[i-1] qui sont plus grands que
11         // x en partant de T[i-1]
12         j prend la valeur i
13         tant que j > 0 et T[j - 1] > x
14             T[j] prend la valeur T[j - 1]
15             j prend la valeur j - 1
16
17         // placer x dans le "trou" laisse par le decalage
18         T[j] prend la valeur x

```

 **Exercice 2 :** Ecrivez une classe *Rationnel* qui définit les nombres rationnels. La classe a comme attributs un numérateur et un dénominateur. La classe *Rationnel* doit disposer des constructeurs suivants : *Rationnel()*; *Rationnel(int numérateur, int dénominateur)*; *Rationnel(Rationnel r)*. Elle doit aussi contenir les méthodes :

- void *additionner*(*Rationnel r*);
- void *soustraire*(*Rationnel r*);
- void *multiplier*(*Rationnel r*);
- void *diviser*(*Rationnel r*);
- double *evaluer*();
- *Rationnel* *inverser*();
- void *afficher*();


 **Exercice 3 :** Dans cet exercice, nous allons implémenter un petit jeu de lancer de dés.

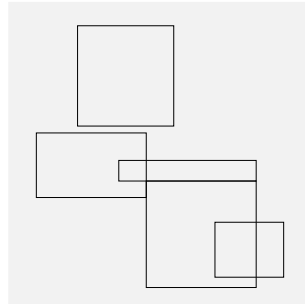
1. Créez une classe *Dice*, (dé en anglais, mieux vaut éviter de mettre des accents dans un code, cela peut mener à des erreurs) qui représentera un dé, avec un attribut *faceNumber* pour le nombre de faces du dé.
2. Ajoutez un constructeur à votre classe *Dice*
3. Créez une méthode *randomSelection* qui retournera un entier tiré aléatoirement entre 1 et *face-number*.

```

1  // il faut importer la classe Random pour pouvoir
2  // generer des nombres aleatoires
3  // import a mettre avant le mot cle class
4  import java.util.Random;
5
6  public class Dice{
7
8      // To do
9
10     public int randomSelection(){
11         // cree un objet de type Random
12         Random rand = new Random();
13         // generer un entier compris dans [0 - 49]
14         // (par exemple)
15         int n = rand.nextInt(50);
16
17         // To do
18     }
19 }
```

4. Maintenant, créez une classe *Player* qui possède en attribut un identifiant ainsi qu'un tableau de taille 5 d'objets de type *Dice*.
5. Ecrivez le constructeur de la classe *Player*, sachant que chaque joueur devra avoir un identifiant unique ainsi que 2 dés avec 6 faces et 3 dés avec 10 faces.
6. Ecrivez une méthode *play* qui simule le lancer des 5 dés et qui renvoie la somme des nombres obtenus.
7. (plus difficile) Ecrivez une méthode *playyam* qui simule le lancer des 5 dés également, mais qui retourne un entier égal à la somme des faces obtenus plus un bonus de  $i \times a$  si la face  $a$  est apparu  $i$  fois ( $i > 1$ ).

 **Exercice 4 :** On s'inspire de la création d'interface graphique. Nous n'allons pas en réaliser une aujourd'hui (c'est évidemment possible en java). Si on regarde une interface comme par exemple celle d'un gestionnaire d'emails, on voit beaucoup de rectangles : certains sont là pour donner un peu de couleurs, certains contiennent du texte, et certains contiennent du texte et sont des boutons.



On va simplifier le problème. On travaille sur un écran de résolution  $1920 \times 1080$ , donc les coordonnées vont de 0 à 1919 en abscisse et de 0 à 1079 en ordonnée. On va considérer que tous les rectangles sont horizontaux (pas de rectangles penchés). Créez une classe `Box` qui sera définie par les coordonnées du coin en haut à gauche et du coin en bas à droite.

Si la classe `Box` n'évoluera que dans un écran de résolution  $1920 \times 1080$ , on peut passer de `int` à `short` pour les attributs.

1. Créez un constructeur qui prend en paramètres les coordonnées des deux coins.
2. Créez une méthode `String toString()` qui retourne une chaîne de caractères qui indique les caractéristiques de la `Box`. Par exemple la méthode peut retourner une chaîne comme celle-ci : `"(100, 200) - (300,100)"`.
3. Ecrivez une méthode `main` qui crée deux instances de la classe `Box` et qui affichent leurs caractéristiques.
4. Ecrivez une méthode `void translate(int vx, int vy)` qui translate la box suivant le vecteur `(vx, vy)`. Testez votre code en ajoutant des tests dans la méthode `main`.
5. Créez une classe `Pixel` pour représenter un pixel de l'écran. Pour cette classe, vous pouvez également implémenter une méthode `String toString()` et `void translate(int vx, int vy)`.
6. Modifiez l'implémentation de la classe `Box` pour utiliser maintenant votre classe `Pixel` à la place des coordonnées de vos points.
7. On désire maintenant ajouter un identifiant sous la forme d'un `int`. Modifiez le constructeur pour donner de manière automatique un unique identifiant.  
Indice : on pourrait compter le nombre d'instances de la classe `Box` et faire en sorte que la  $i^{eme}$  instance créée ait pour identifiant  $i$ .
8. Créez une méthode `public int getId()` qui retourne l'identifiant de la `Box`. Modifiez la méthode `toString()` pour que l'affichage soit maintenant comme suit : `"[Box id=13] (100, 200) - (300,100)"`.
9. Pour ceux qui vont plus vite, recodez les mêmes classes mais avec des classes non-mutables.

Un exemple, d'une classe écrite en mutable puis la même en non-mutable.

```

1  public class Point {
2      private double x ;
3      private double y ;
4
5      public Point(double x, double y) {
6          this.x = x ;
7          this.y = y ;
8      }
9
10     public void translate(double dx, double dy) {

```

```
11     x += dx ;
12     y += dy ;
13 }
14 }
```

```
1  public class Point {
2      private final double x ;
3      private final double y ;
4
5      public Point(double x, double y) {
6          this.x = x ;
7          this.y = y ;
8      }
9
10     public Point translate(double dx, double dy){
11         return new Point(x + dx,y + dy);
12     }
13 }
```

**N.B.** utilisez la méthode `System.out.println(String s)` pour afficher une chaîne de caractères `s` sur la console (La classe `System` a un attribut `out` dont vous appelez la méthode `println(String s)`).