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.

```java
public class Calcul {

    // To do

    public Calcul(int a){
        // To do
    }

    public int square(){
        // To do
    }

    public static void main(String[] args) {
        Calcul c = new Calcul(10);
        System.out.println(c.square()); // devrait afficher 100
    }
}
```

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

```java
// rappel tableau
int[] tableau = {1, 2, 3};
int tailleDuTableau = tableau.length;
// /!\ indice en java : tableau[0] correspond a la valeur 1,
// de maniere general l'element en i eme position
// correspond a tableau[i-1]

// condition if en java - test d'egalite
// /!\ avec un double "=", sinon on affecte la valeur de
// var2 a var1
// /!\ compare la valeur des types primitifs (dont int,
// double, char, boolean), mais compare
// les references pour des objets (par exemple String !!!
// On verra cela plus tard pour les objets)
```

```java

  if(var1 == var2){
      // code execute uniquement si var1 == var2
  }

  // rappel : une variable de type boolean ne
  // prend soit la valeur true soit false
  public boolean containedIn(int[] T){
      // To do
  }
```

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

```java
  // bb % 2 correspond a bb modulo 2 (i.e. le reste de la
  // 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 entré un tableau d'entiers et qui le retourne trié par ordre croissant. Ci dessous, le pseudo-code de l'algorithme du tri par insertion

```
// tri par insertion - pseudo code
procedure tri_insertion(tableau T)
       n prend la valeur taille(T)
       pour i de 1 a n - 1

            // memoriser T[i] dans x
            x prend la valeur T[i]

            // decaler vers la droite les elements de
            // T[0]..T[i-1] qui sont plus grands que
            // x en partant de T[i-1]
            j prend la valeur i
            tant que j > 0 et T[j - 1] > x
                     T[j] prend la valeur T[j - 1]
                     j prend la valeur j - 1

            // placer x dans le "trou" laisse par le decalage
            T[j] prend la valeur x
```

Correction Exercice 1 :

```java
    public int square(){
        return a*a;
    }

    public double power(int b){
        if (b == 0){
            return 1;
        }
```

```
9           double pow = 1.0;
10          for (int i = 0; i < Math.abs(b); i++) {
11              pow *= a;
12          }
13          if (b < 0){
14              pow = 1/pow;
15          }
16          return pow;
17      }
18
19      public int factorial(){
20          if (a < 0){
21              return -1; // pourra etre interprete comme une erreur
22          }
23          int facto = 1;
24          for (int i = 2; i < a+1; i++) {
25              facto *= i;
26          }
27          return facto;
28      }
29
30      public int recursivfactorial(int a){
31          if (a < 0){
32              return -1;
33          } else if (a == 0 || a == 1) {
34              return 1;
35          } else {
36              return a * recursivfactorial(a-1);
37          }
38      }
39
40      public boolean containedIn(int[] tab){
41          for (int i = 0; i < tab.length; i++) {
42              if (tab[i] == a){
43                  return true;
44              }
45          }
46          return false;
47      }
48
49      public boolean isPrime(){
50          if (a < 0){
51              return false;
52          }
53          for (int i = 2; i < Math.sqrt(a); i++) {
54              if ((a%i) == 0){
55                  return false;
56              }
57          }
58          return true;
```

```
59      }
60
61      public int max(int[] tab){
62          if (tab.length > 0) {
63              int max = tab[0];
64              for (int i = 1; i < tab.length; i++) {
65                  if (max < tab[i]) {
66                      max = tab[i];
67                  }
68              }
69              return max;
70          }
71          return -1;
72      }
73
74      public int[] insertionSort(int[] tab){
75          for (int i = 1; i < tab.length; i++) {
76              int x = tab[i];
77              int j = i;
78              while((j > 0) && (tab[j-1] > x)){
79                  tab[j] = tab[j-1];
80                  j = j-1;
81              }
82              tab[j] = x;
83          }
84          return tab;
85      }
```

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 numerateur, int denominateur);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();`

Correction Exercice 2 :

```
1   public class Rationnel {
2           private int numerateur;
3           private int denominateur;
4
5           public Rationnel() {
6                   numerateur = 0;
7                   denominateur = 1;
```

```
8           }
9
10          public Rationnel(int numerateur, int denominateur) {
11                  this.numerateur = numerateur;
12                  if (denominateur == 0) {
13                          System.out.println("0 is not allowed as denominateur,
14                          I replaced it by 1");
15                          this.denominateur = 1;}
16                  else{this.denominateur = denominateur;}
17          }
18
19          public Rationnel(Rationnel r) {
20                  this.numerateur = r.numerateur;
21                  this.denominateur = r.denominateur;
22          }
23
24          public void additionner(Rationnel r) {
25                  this.numerateur = this.numerateur*r.denominateur
26                                  + r.numerateur*this.denominateur;
27                  this.denominateur = this.denominateur*r.denominateur;
28          }
29
30          public void soustraire(Rationnel r) {
31                  this.numerateur = this.numerateur*r.denominateur
32                                  - r.numerateur*this.denominateur;
33                  this.denominateur = this.denominateur*r.denominateur;
34          }
35
36          public void multiplier(Rationnel r) {
37                  this.numerateur = this.numerateur*r.numerateur;
38                  this.denominateur = this.denominateur*r.denominateur;
39          }
40
41          public void diviser(Rationnel r) {
42                  this.numerateur = this.numerateur*r.denominateur;
43                  this.denominateur = this.denominateur*r.numerateur;
44          }
45
46          public double evaluer() {
47                  return this.numerateur/(double)this.denominateur;
48          }
49
50          public Rationnel inverser() {
51                  return new Rationnel(this.denominateur,this.numerateur);
52          }
53
54          public void printRationnel() {
55                  System.out.println("Rationnel "+ this.numerateur
56                      + "/" + this.denominateur);
57          }
```

```java

}


public class Test {

        public static void main(String[] args){
        Rationnel r1 = new Rationnel(24,32);
        Rationnel r2 = new Rationnel(-12,5);
        //Tests sur la classe Rationnel
        ...
        }
}
```

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*.

```java
// il faut importer la classe Random pour pouvoir
// generer des nombres aleatoires
// import a mettre avant le mot cle class
import java.util.Random;

public class Dice{

    // To do

    public int randomSelection(){
        // cree un objet de type Random
        Random rand = new Random();
        // generer un entier compris dans [0 - 49]
        // (par exemple)
        int n = rand.nextInt(50);

        // To do
    }
}
```

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.

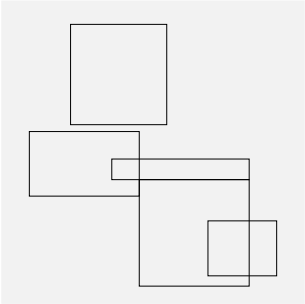
7. (plus difficile) Ecrivez une méthode *playyam* qui simule le lancer des 5 dés également, mais qui retourne un entier égal à la somme des faces obtenus plus un bonus de $i \times a$ si la face a est apparu i fois ($i > 1$).

Correction Exercice 3 :

```java
//Dans un fichier Dice.java
package TP_2;
import java.util.Random;

public class Dice {
    private int faceNumber;

    public Dice(int faceNumber){
        this.faceNumber = faceNumber;
    }

    public int randomSelection() {
        Random random = new Random();
        return random.nextInt(faceNumber - 1) + 1;
    }
}

//Dans un fichier Player.java
package TP_2;
public class Player {
    private int id;
    private static int count = 1;
    private Dice[] hand;

    public Player(){
        this.id = count;
        count++;
        hand = new Dice[5];
        hand[0] = new Dice(6);
        hand[1] = new Dice(6);
        hand[2] = new Dice(10);
        hand[3] = new Dice(10);
        hand[4] = new Dice(10);
    }

    public int play(){
        int sum = 0;
        for (int i = 0; i < 5; i++) {
            sum += hand[i].randomSelection();
        }
        return sum;
    }

    public int playyam(){
        int sum = 0;
```

```
46            int[] play = new int[5];
47            for (int i = 0; i < 5; i++) {
48                play[i] = hand[i].randomSelection();
49                sum += play[i];
50            }
51            for (int i = 1; i < 10; i++) {
52                int count = 0;
53                for (int j = 0; j < 5; j++) {
54                    if (play[j] == i){
55                        count++;
56                    }
57                }
58                if (count > 1){
59                    sum += i*count;
60                }
61            }
62            return sum;
63        }
64
65    }
```

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 1920×1080, 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.

```
1   // Dans un fichier Box.java
2   public class Box {
3           //CHG pour coin en haut a gauche
4           //CBD pour coin en bas a droite
5           private int absCHG;
6           private int ordCHG;
7           private int absCBD;
8           private int ordCBD;
9
```

```
}
```

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

1. Créez un constructeur qui prend en paramètres les coordonnées des deux coins.

```
public class Box {
        private int absHG;
        private int ordHG;
        private int absBD;
        private int ordBD;

        public Box() {}

        public Box(int absHG, int ordHG, int absBD, int ordBD) {
                this.absHG = absHG;
                this.absBD = absBD;
                this.ordHG = ordHG;
                this.ordBD = ordBD;
        }
}
```

Dans le code ci-dessus il faut rappeler que c'est une bonne pratique d'écrire un constructeur par défaut (il n'est plus généré par défaut si on en écrit un autre). On peut aussi présenter le mot-clé **this**.

2 Créez une méthode `String toString()` qui retourne une chaîne de caractères qui indique les caractéristique de la `Box`. Par exemple la méthode peut retourner une chaîne comme celle-ci : `"(100, 200) - (300,100)"`.

```
public class Box {
...
        public String toString() {
                return "(" + absHG + ", " + ordHG +
                                ") -- (" + absBD + ", " + ordBD + ")";
        }
}
```

Dans le code ci-dessus, les étudiants ne connaissent pas l'annotation `@Override`. Il est surement préférable de l'oublier pour le moment.

3 Ecrivez une méthode `main` qui crée deux instances de la classe `Box` et qui affichent leur caractéristiques.

```
//Dans un fichier Main.java
public class Main {

        public static void main(String[] args) {
                Box box1 = new Box(100,200,300,100);
                Box box2 = new Box(20,400,200,300);

                System.out.println(box1.toString());
```

```
                System.out.println(box2.toString());
        }

}
```

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`.

```
public class Box {
...
        void translate(int vx, int vy) {
                this.absBD+=vx;
                this.absHG+=vx;
                this.ordBD+=vy;
                this.ordHG+=vy;
        }
}
```

```
public class Main {

        public static void main(String[] args) {
                Box box1 = new Box(100,200,300,100);
                Box box2 = new Box(20,400,200,300);

                box1.translate(30, 52);
                box2.translate(12, -100);

                System.out.println(box1.toString());
                System.out.println(box2.toString());
        }
}
```

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)`.

```
//Dans un fichier Pixel.java
public class Pixel {
        private abs;
        private ord;

        public Pixel() {}

        public Pixel(int abs, int ord) {
                this.abs = abs; this.ord = ord;
        }

        public String toString() {
                return "("+abs+", "+ord+")";
        }
```

```
15
16          public void translate(int vx, int vy) {
17                  this.abs+=vx;
18                  this.ord+=vy;
19          }
20  }
```

6 Modifiez l'implémentation de la classe Box pour utiliser maintenant votre classe Pixel à la place des coordonnées de vos points.

```
1   public class Box {
2           private Pixel pixelHG;
3           private Pixel pixelBD;
4
5           ...
6
7           public Box() {
8                   pixelHG = new Pixel(0,0);
9                   pixelBD = new Pixel(0,0);
10          }
11
12          public Box(int absHG, int ordHG, int absBD, int ordBD) {
13                  pixelHG = new Pixel(absHG,ordHG);
14                  pixelBD = new Pixel(absBD,ordBD);
15          }
16
17          public String toString() {
18                  return pixelHG.toString()
19                          +  " -- " + pixelBD.toString();
20          }
21
22          void translate(int vx, int vy) {
23                  pixelHG.translate(vx, vy);
24                  pixelBD.translate(vx, vy);
25          }
26  }
```

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)"`.

```
1   public class Box {
2           ...
3           private Pixel pixelHG;
4           private Pixel pixelBD;
5           private static int nbBox;
```

```
6           private int idBox;
7
8           ...
9
10          public Box() {
11                  pixelHG = new Pixel(0,0);
12                  pixelBD = new Pixel(0,0);
13                  idBox = ++nbBox;
14          }
15
16
17          public Box(int absHG, int ordHG, int absBD, int ordBD) {
18                  pixelHG = new Pixel(absHG,ordHG);
19                  pixelBD = new Pixel(absBD,ordBD);
20                  idBox = ++nbBox;
21          }
22
23          public String toString() {
24                  return "[Box id="+ getId() + "]" + pixelHG.toString()
25                          +  " -- " + pixelBD.toString();
26          }
27
28          public int getId() {
29                  return this.idBox;
30          }
31
32          ...
33  }
```

Vous pouvez réfléchir ensuite à la non-robustesse du code. Peut-être que `absHG` $>$ 1919 ou que `absHG`$>$`absDB` lors de la création d'une `Box` ou après une translation. Essayer de rajouter des tests pour prendre ces différents cas en compte.

9 Pour ceux qui vont plus vite, recodez les mêmes classes mais avec des classes non-mutables.

Un exemple, d'une classe écrit 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  }
```

```java
public class Point {
    private final double x ;
    private final double y ;

    public Point(double x, double y) {
        this.x = x ;
        this.y = y ;
    }

    public Point translate(double dx, double dy){
        return new Point(x + dx,y + dy);
    }
}
```

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)`).