

Q1a (1 pts) Ecrire la déclaration, réservation mémoire et initialisation du tableau de double suivant :

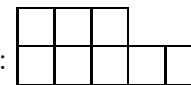
1.2	5.6	3.4
-----	-----	-----

Solution 1 : `double [] tab=new double [3]; tab[0]=1.2; tab[1]=5.6; tab[2]=3.4;`

Solution 2 : obligatoirement en une seule instruction : `double [] tab={1.2, 5.6, 3.4};`

Solution 3 : `double [] tab; tab=new double[]{1.2, 5.6, 3.4};`

Q1b (2 pts) Ecrire la déclaration et réservation mémoire du tableau d'entiers suivant :



Solution 1 : on déclare d'abord un tableau à 2 dimensions contenant 2 cases où chaque case contient une variable de type tableau à une dimension, puis on réserve l'espace mémoire pour chaque ligne

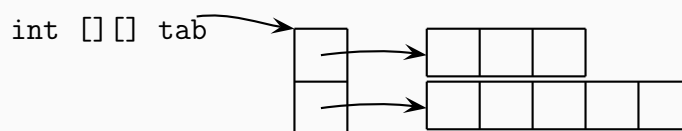
```
int [][] tab=new int[2] [];
```

```
tab[0]=new int[3];
```

```
tab[1]=new int[5];
```

Solution 2 : en une seule instruction

```
int [][] tab={new int[3],new int[5]};
```



Q2 (3 pts) On veut écrire une classe pour générer des identifiants. Chaque identifiant est une chaîne de caractères qui commence par "ID" suivi d'un nombre supérieur à 1 000 000. Le premier identifiant généré sera "ID1000001", puis "ID1000002", puis "ID1000003"... **Q2a)** Complétez la classe `GenerateurIDs` pour qu'elle contienne un attribut `cpt` initialisé à 1 000 000 et une méthode `getIDSuivant()` qui retourne le prochain identifiant (ne pas gérer le dépassement de la taille du compteur).

Générer un nouvel identifiant ne dépend pas d'une instance de la classe `GenerateursIDs`, donc l'attribut `cpt` et la méthode `getIDSuivant()` sont statiques.

```
public class GenerateurIDs {
    private static int cpt=1000000;
    public static String getIDSuivant() {
        cpt++;
        return "ID"+cpt;
    }
}
```

Remarque : si on initialise `cpt` à 0, pour obtenir les mêmes identifiants, il faut faire `"ID"+(1000000+cpt)`; et non pas `"ID1"+cpt`; qui génère des identifiants de la forme "ID11", "ID12", "ID13", ..., "ID19", "ID110", ... "ID199", "ID1100", "ID1101", ...

Q2b) Complétez les instructions ci-dessous pour afficher 100 identifiants.

```
for (int i=0;i<100;i++) {
    System.out.println(GenerateurIDs.getIDSuivant ());
}
```

Q3 (4 pts) Un jeu vidéo dispose d'une machine à copier les personnages (avec toutes leurs caractéristiques).

```

1 public class Sac {
2     private double poids;
3     public Sac(double poids) {
4         this.poids=poids;
5     }
6 }
7 public class Personnage {
8     private int numero;
9     private Sac leSac;
10    public Personnage(int numero, Sac leSac) {
11        this.numero=numero;
12        this.leSac=leSac;
13    } }

```

- Un constructeur par copie est un constructeur qui prend en paramètre l'objet à copier (le type du paramètre est donc la classe courante) et qui initialise chacun des attributs avec des valeurs similaires (affectation des valeurs, copie des instances) à celles des attributs du paramètre.

- Une méthode `clone()` est une méthode sans paramètre qui crée un objet du même type que l'objet courant, qui recopie chacun des attributs (affectation des valeurs, copie des objets) de l'objet courant, et qui a pour type de retour la classe courante.

Q3a Ecrire ci-dessous le constructeur par copie ou la méthode `clone()` de `Sac`

```

Constructeur par copie
public Sac(Sac s) {
    this.poids=s.poids;
}
// OU
public Sac(Sac s) {
    this(s.poids);
}

```

```

Méthode clone()
public Sac clone() {
    return new Sac(this.poids);
}
// OU si on appel le
// constructeur par copie
public Sac clone() {
    return new Sac(this);
}

```

Q3b Ecrire le constructeur par copie ou la méthode `clone()` de `Personnage`

```

Constructeur par copie
public Personnage(Personnage p) {
    this.numero=p.numero; //affectation
    this.leSac=new Sac(p.leSac); //copie
} // OU
public Personnage(Personnage p) {
    this(p.numero,
        new Sac(p.leSac));
}

```

```

Méthode clone()
public Personnage clone() {
    return new Personnage(numero,
        leSac.clone());
}
// OU appel le constructeur par copie
public Personnage clone() {
    return new Personnage(this);
}

```

Dans les questions **Q3a** et **Q3b**, il est important de penser à faire une copie de chacune des instances qui sont attributs. La classe `Personnage` possède un attribut `numero` qui est de type simple `int`, il est inutile de faire une copie de cet attribut, il suffit de faire une affectation de la valeur. Par contre, elle possède un attribut `leSac` qui est de type objet `Sac`, il faut faire une copie de cet instance en appelant le constructeur par copie ou la méthode `clone()`.

Q3c Créez le personnage numéro 123 avec un sac de 2 kg, puis créez une copie de ce personnage :

```

Personnage p1=new Personnage(123,
                            new Sac(2));
Personnage p2=new Personnage(p1);
// OU
Personnage p2=p1.clone();

```

Q3d Combien d'objets de la classe `Sac` ont été instanciés **en tout** à la question **Q3c** ?

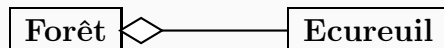
0 1 2 3 4

Quand on copie un personnage, on copie aussi son sac. Il y a donc 2 objets de la classe `Sac` (une pour chaque personnage).

Q4 (10 points) Classe et tableau d'objets

Pour un jeu vidéo, on veut modéliser une forêt avec des écureuils. Une forêt peut contenir au maximum n écureuils. Un écureuil peut être ajouté à la forêt.

Q4a Dessiner ci-contre le diagramme de classes simplifié (ne pas écrire les attributs ni méthodes).



Remarque : les noms de ces classes s'écrivent au singulier, car un objet de la classe `Foret` / `Ecureuil` représente une forêt / un écureuil et non pas plusieurs.

Q4b Complétez la classe `Ecureuil` avec : attribut `POIDS_MAX` (poids max d'un écureuil initialisé à 900), attribut `nbEcureuils` (nombre d'écureuils créés depuis le début du programme), attribut `numero` (numéro de l'écureuil : 1 pour le 1er écureuil, 2 pour le 2ième... ; le numéro ne doit pas changer et peut être connu par tous), constructeur sans paramètre qui initialise le poids dans $[100, \text{POIDS_MAX}[$ avec `Math.random()`, accesseur de `nbEcureuils`, et `toString()` qui retourne par exemple : "Ecureuil N°1 200g".

```

1 public class Ecureuil {
2     private double poids;
3     public static final double POIDS_MAX = 900 ;
4     private static int nbEcureuils=0;
5     public final int numero;
6
7     public Ecureuil() {
8         poids= 100+Math.random()*(POIDS_MAX-100) ;
9         nbEcureuils++;
10        numero=nbEcureuils;
11    }
12    public double getPoids() {
13        return poids;
14    }
15    public static int getNbEcureuils() {
16        return nbEcureuils;
17    }
18    public String toString() {
19        return "Ecureuil No "+numero+" "+poids+" g ";
20    }
21 }
  
```

Q4c Complétez la classe `Foret` pour qu'elle contienne : attribut `tabEcu` de type tableau d'écureuils, attribut `nbEcureuils` qui correspond au nombre d'écureuils présents dans la forêt, constructeur qui prend en paramètre le nombre maximal d'écureuils possible dans la forêt, méthode qui ajoute un écureuil, méthode `getPoidsTotal()` qui retourne la somme des poids de tous les écureuils de la forêt, méthode `toString()` qui retourne par exemple pour une forêt pouvant contenir jusqu'à 20 écureuils et contenant 2 écureuils :

"Forêt 2 écureuils sur 20, poids total=500g : Ecureuil N°1 200g Ecureuil N°2 300g"

```

1 public class Foret {
2     private Ecureuil [] tabEcu ;
3     private int nbEcureuils ;
4
5     public Foret(int n) {
6         tabEcu=new Ecureuil [n];
7         nbEcureuils =0;
8     }
9     public void ajouter( Ecureuil ins) {
10        if ( nbEcureuils >= tabEcu .length) {
11            System.out.println("foret_pleine");
12        } else {
13            tabEcu [ nbEcureuils ]=ins ;
14            nbEcureuils ++;
15        }
16    }
17    public double getPoidsTotal() {
18        double poidsTotal=0;
19        for(int i=0;i< nbEcureuils ;i++) {
20            poidsTotal+=tabEcu [ i ] .getPoids () ;
21        }
22        return poidsTotal;
23    }
24
25    public String toString() {
26        String s="";
27        for(int i=0;i< nbEcureuils ;i++) {
28            s+=tabEcu [ i ] .toString ()+" ";
29        }
30        return "Foret "+ nbEcureuils +" écureuils sur "+ tabEcu .length+" ,
31            poids total="+getPoidsTotal()+" g :\n"+s;
32    }
33 }

```

Remarquez que `nbEcureuils` de la classe `Ecureuil` est `static`, tandis que `nbEcureuils` de la classe `Foret` ne l'est pas. Pourquoi? Parce que le nombre d'écureuils créés depuis le début du programme ne dépend pas d'une seule instance de la classe `Ecureuil` (donc `static`), tandis que le nombre d'écureuils dans la forêt dépend de la forêt (donc pas `static`) : il peut y avoir une forêt avec 10 écureuils et une autre avec 15.