
# LU2IN002 : Éléments de programmation par objets avec JAVA

## Licence de Sciences et Technologies Mention Informatique

Fascicule de TD/TME

Année 2021-2022

# 1 Classe : définition, syntaxe (TD)

## Exercice 1 (Cours) – Premier programme Java

Dans le fichier `Bonjour.java`, écrire une classe `Bonjour` qui affiche `"Bonjour Monde"`. Quel est le rôle de la méthode `main` ? Aide : pour la syntaxe, on se reportera à l'annexe page 61.

## Exercice 2 – Planète

Soit la classe `Planete` suivante située dans le fichier `Planete.java` :

```
1 public class Planete {
2         private String nom;
3         private double rayon; // en kilometre
4
5         public Planete(String n,double r) {
6                 nom=n;
7                 rayon=r;
8         }
9         public String toString() {
10                String s="Planete␣"+nom;
11                s+="␣de␣rayon␣"+rayon;
12                return s;
13        }
14        public double getRayon() {
15                return rayon;
16        }
17 }
```

**Q 2.1** Dans cette classe, quelles sont (a) les variables d'instance ? (b) les variables qui sont des paramètres de méthode ? (c) les variables qui sont des variables locales à une méthode ?

**Q 2.2** Où est le constructeur ? Comment le reconnaît-on ? Quel est le rôle des constructeurs en général ? Quand sont-ils appelés ?

**Q 2.3** Quelles sont les méthodes de cette classe ?

**Q 2.4** Ecrire une nouvelle classe appelée `SystemeSolaire`. On souhaite que cette classe soit le point d'entrée du programme, que doit-elle contenir ? Créer un objet (ou instance) de la classe `Planete` pour la planète Mercure qui a un rayon de 2439.7 km et un autre objet pour la planète Terre qui a un rayon de 6378.1 km. Afficher la valeur de retour de la méthode `toString()` pour la planète Mercure, puis afficher le rayon de la planète Terre.

**Q 2.5** Quel doit être le nom du fichier contenant la classe `SystemeSolaire` ? Quelles sont les commandes pour compiler les classes `Planete` et `SystemeSolaire` ? Quelle est la commande pour exécuter ce programme ?

**Q 2.6** Dans le `main`, est il possible d'accéder (en lecture) au rayon d'une planète précédemment instanciée ? est il possible d'accéder (en lecture) au nom de cette planète ? Est-il possible de modifier des attributs d'une planète ?

## Exercice 3 – Se présenter

**Q 3.1** Une personne est représentée par son nom et son âge. Ecrire la classe `Personne` qui contient :
- les variables d'instance `nom` et `age`,
- un constructeur dont la signature est : `public Personne(String n, int a)`.

**Q 3.2** Ecrire une nouvelle classe appelée `Presentation` avec une méthode `main` qui crée un objet (ou instance) d'une personne appelée Paul qui a 25 ans, et d'une autre personne appelée Pierre qui a 37 ans.

**Q 3.3** On souhaite maintenant avoir des méthodes qui nous permettent d'obtenir des informations sur les objets de la classe `Personne`. Ajouter dans la classe `Personne`, les méthodes suivantes :
- la méthode standard `public String toString()` dont le but est de *retourner* une chaîne de caractères au format suivant : "Je m'appelle <*nom*>, j'ai <*age*> ans" où <*nom*> et <*age*> doivent être remplacés par le nom et l'âge de la personne courante. Dans la classe `Presentation`, ajouter une instruction qui utilise cette méthode pour afficher le nom et l'âge de Pierre.
- la méthode `public void sePresenter()` dont le but est *d'afficher* la chaîne de caractères retournée par la méthode `toString()`. Dans la classe `Presentation`, ajouter une instruction qui utilise cette méthode pour afficher le nom et l'âge de Paul.
- Quelle différence y-a-t-il entre la méthode `toString()` et la méthode `sePresenter()` ?

**Q 3.4** Que se passe-t-il si, dans la classe `Personne`, on modifie la signature de la méthode `sePresenter()` pour que cette méthode soit privée ?

**Q 3.5** Peut-on connaître l'âge de Pierre dans la classe `Presentation` ? Pourquoi ? Ajouter un accesseur `getAge()` pour la variable age. Quel est le type de retour de `getAge()` ?

**Q 3.6** Ajouter dans la classe `Personne`, la méthode `vieillir()` qui ajoute un an à la personne. Dans la classe `Presentation`, faites vieillir Paul de 20 ans (utiliser une boucle `for`), et Pierre de 10 ans (utiliser une boucle `while`), puis faites se présenter Paul et Pierre. Aide : voir la syntaxe des boucles page 62

## Exercice 4 – Constructeurs multiples, méthodes multiples

**Rappel :** en JAVA, une méthode est identifiée par son nom ET ses arguments. Ainsi, deux méthodes avec le même nom et des arguments différents (nombre ou type des arguments) sont différentes. Le même principe prévaut avec les constructeurs.
En repartant de l'exercice 2 :
**Q 4.1** Ajouter un second constructeur qui prend en argument seulement le nom de la planète et fixe son rayon à 1000km (arbitrairement).

**Q 4.2** Écrire un programme de test construisant deux planètes en utilisant les deux constructeurs pour vérifier le bon fonctionnement de cette approche.

## Exercice 5 – Alphabet

**Q 5.1** Ecrire la classe `Alphabet` qui ne contient que la méthode `main` qui réalise le traitement suivant, en utilisant une boucle `for` :

**Q 5.1.1** Afficher les chiffres de 0 à 9, ainsi que leur code ASCII.

**Q 5.1.2** Afficher les lettres de l'alphabet de 'A' à 'Z', ainsi que leur code ASCII.

**Q 5.2** Recommencer en utilisant une boucle while.

## Exercice 6 – Rétro-engineering

**Q 6.1** Écrire le programme permettant d'obtenir l'affichage suivant (en utilisant des boucles) :

```
1 2 3 5 8 13 21 34 45
```

## Quizz 1 – Génération de nombres aléatoires

Pour générer un nombre aléatoire, on peut utiliser `Math.random()` qui rend un `double` dans l'intervalle $[0, 1[$.
Par exemple, pour générer :
- un réel dans [MIN,MAX[ : `double val=Math.random()*(MAX-MIN)+MIN;`
- un entier dans [MIN,MAX] : `int val=(int)(Math.random()*(MAX-MIN+1)+MIN);`
- un booléen vrai dans 5% des cas : `boolean val=Math.random()<0.05;`

Générer aléatoirement :

a) un réel dans [10,30[

b) un entier dans [50,150]

c) un booléen vrai dans 25% des cas

d) une lettre de l'alphabet comprise entre 'a' et 'z'.
*Aide* : on peut utiliser la même formule que pour les entiers, mais avec une variable de type `char`.

## Quizz 2 – Conventions de nommage

Les identificateurs suivants respectent les conventions de nommage de Java. Indiquer pour chaque identificateur : si c'est une variable (V), un appel de méthode (AM), le nom d'une classe (NC), un appel à un constructeur (AC), un mot réservé (R) ou une constante (CST).

| | | | |
|---|---|---|---|
| abcDefg() | true | abcd | Abcd |
| Abc() | String | False | ABCD |

## Quizz 3 – Syntaxe des expressions

**QZ 3.1** L'instruction suivante provoque-t-elle une erreur ? `float val=1.12;`

**QZ 3.2** Soient : `int x=2, y=5; double z=x/y;` Quelle est la valeur de z ?

**QZ 3.3** Sachant que le code ascii du caractère '1' est 49, quel est le type et la valeur des expressions suivantes :

a) `1+"1"` ?

b) `1+'1'` ?

c) `'1'+"1"` ?

**QZ 3.4** Sachant qu'en Java l'opérateur + est évalué de la gauche vers la droite, quelle est la valeur des expressions suivantes :

a) `1+'1'+"1"` ?

b) `"1"+'1'+1` ?

**QZ 3.5** Quel est le type et la valeur de l'expression suivante ? `true && false == false`
Aide : voir la table de priorité des opérateurs page 63.

**QZ 3.6** Qu'affiche : `System.out.println("Bonjour \nvous \ttous !")` ?

## Quizz 4 – Compilation et exécution

**QZ 4.1** Un fichier source Java est sauvegardé avec l'extension ...

**QZ 4.2** Un fichier source Java contient ...

**QZ 4.3** Une classe est composée de ... et de ...

**QZ 4.4** Les instructions Java sont toujours situées à l'intérieur de ...

**QZ 4.5** Les lignes composant une méthode sont soit des ... soit des ...

**QZ 4.6** Si vous n'utilisez pas l'environnement intégré, quelle commande tapez-vous pour compiler un fichier ? Pour exécuter un `.class` correspondant à un `main` ?

**QZ 4.7** Quel est le nom de la méthode par lequel un programme Java commence son exécution ?

**QZ 4.8** Quel est l'en-tête de la méthode main ?

# TME 1 : Introduction à Java – premiers pas

**Remarque** : les intitulés des exercices de cette séance sont aussi disponibles en ligne sur le site Moodle de l'UE : https://moodle-sciences.upmc.fr/moodle-2021/course/view.php?id=3409

## Exercice 7 – Préliminaires

**Q 7.1** En salle de TME, si vous n'avez pas accès à internet :
- chercher l'outil de configuration d'accès dans Mozilla Firefox.
- passer en configuration manuelle en mettant proxy (port = 3128) sur tous les champs

Mettez ce site Moodle de l'UE dans les favoris de votre navigateur pour que vous puissiez y accéder rapidement à chaque séance de TME.

**Q 7.2** Pour bien organiser vos fichiers, créer le répertoire `LU2IN002` dans votre répertoire de travail, puis dans ce répertoire, créer un répertoire `TME1`, Tous vos fichiers du `TME1` devront se trouver dans ce répertoire. Pour cela, ouvrir un nouveau terminal, puis taper les commandes qui permettent de réaliser les instructions suivantes :

1. créer le répertoire `LU2IN002` (commande `mkdir` *`nomDuRepertoire`*),
2. se déplacer dans ce répertoire `LU2IN002` (commande `cd` *`nomDuRepertoire`*),
3. créer le répertoire `TME1`,
4. lister les fichiers du répertoire `LU2IN002` pour vérifier que le répertoire `TME1` a bien été créé (commande `ls`),
5. se déplacer dans le répertoire `TME1` ,
6. afficher le nom du répertoire courant (commande `pwd`).

*Aide* : l'annexe du poly rappelle la liste des instructions utilisables pour organiser vos fichiers et répertoires sous linux.

**Q 7.3** Pour ouvrir un éditeur de texte (par exemple, l'éditeur `gedit`) afin d'écrire du code Java ou du texte, il est possible d'utiliser la commande : `gedit &` ou bien `gedit` *`MaClasse.java`* `&` pour ouvrir le fichier *`MaClasse.java`*.
*Attention* : ne pas oublier le '`&`' à la fin de la commande pour séparer le terminal et l'éditeur de texte.
Il y a différents éditeurs de texte possibles : `gedit`, `vim`, `geany` ou `emacs`.
*Remarque* : vous ne devez PAS UTILISER d'IDE avant la semaine 5 (c'est-à-dire, PAS d'eclipse, PAS de netbean, ...)
Avec l'éditeur que vous avez choisi, chercher les options pour colorer la syntaxe, indenter les lignes et les numéroter.

**Q 7.4**
- Pourquoi est-il important d'indenter vos programmes ?
- Pourquoi est-il important de sauvegarder et de compiler régulièrement vos programmes sans attendre d'avoir écrit le programme en entier ?

## Exercice 8 – Premier programme

**Q 8.1** Écrire la classe `Bonjour` (fichier `Bonjour.java`) dont la méthode main affiche le message `"Bonjour !"`.

**Q 8.1.1** Quelle est la commande pour compiler cette classe ? Compiler la classe. Quel est le nom du fichier créé par la compilation ?

**Q 8.1.2** Quelle est la commande pour exécuter cette classe ?

**Q 8.1.3** Supprimer le fichier `Bonjour.class` et, sans recompiler, taper à nouveau la commande pour exécuter la classe. La classe est-elle exécutée ?

**Q 8.2** Observer les erreurs de compilation :

**Q 8.2.1** Introduire un espace au milieu du mot `static`. Compiler. D'après le message d'erreur, à quelle ligne se trouve l'erreur ? à quel endroit est détectée l'erreur ?

*Remarque* : pour cette erreur, l'explication de l'erreur par le compilateur ne correspond pas à la correction à effectuer : les diagnostics du compilateur ne doivent donc pas être suivis à la lettre, mais indiquent seulement l'échec de l'analyse.

**Q 8.2.2** Rétablir le mot `static` correctement et supprimer le `"` terminant le mot `Bonjour`. Compiler et observer.

**Q 8.2.3** Après avoir supprimé du répertoire courant le fichier `Bonjour.class`, transformer la méthode `main` en `Main` et recompiler. La compilation réussit-elle ? Peut-on exécuter le programme obtenu ? Expliquer.

**Q 8.2.4** Après avoir remis le bon nom à la fonction `main`, supprimer l'accolade { qui suit le `main`. Compiler et lire les messages.

**Q 8.2.5** Après avoir remis l'accolade, supprimer le mot-clé `public`. La compilation réussit-elle ? Peut-on exécuter le programme ?

**Q 8.2.6** Après avoir remis le mot clé `public`, supprimer le mot-clé `static`. La compilation réussit-elle ? Peut-on exécuter le programme ?

## Exercice 9 – Segment de droite


On veut écrire des classes Java afin de pouvoir comparer la longueur de plusieurs segments de droite (sur une seule dimension). On se limite dans cet exercice à des valeurs entières.

**Q 9.1** Un segment est une portion de droite délimitée par 2 extrémités. Écrire la classe `Segment` qui contient :
- les variables d'instance `x` et `y` correspondant aux valeurs des deux extrémités (entiers),
- un constructeur : `public Segment(int extX,int extY)` qui initialise la variable `x` avec la valeur de `extX` et la variable `y` avec la valeur de `extY`,
- une méthode `public int longueur()` qui retourne la longueur du segment. Si `x` est plus petite que `y` alors la longueur est `y-x`, sinon la longueur est `x-y`.
- la méthode `toString()` dont le but est de retourner une chaîne de caractères au format suivant : `"Segment [<x>,<y>]"` où *<x>* et *<y>* doivent être remplacés par les valeurs des extrémités `x` et `y` du segment courant.

**Q 9.2** Écrire une classe `TestSegment` dont la méthode `main` crée le segment [6,8] et le segment [12,5], puis compare la longueur de ces 2 segments. Si le premier segment est plus long, ce programme affiche que le premier segment est plus long, sinon il affiche que le deuxième segment est plus long.

## Exercice 10 – Solidarité villageoise

Un énorme rocher est tombé dans la nuit sur un petit village de l'ouest de la France bloquant l'unique axe routier sortant du village. Il est décidé de former une équipe de villageois pour tenter de déplacer le rocher (qui pèse 100 kg).

**Q 10.1** Dans la classe `Villageois`, définir les variables suivantes :
- `nom` (le nom du villageois, de type `String`),
- `poids` (le poids (kg) du villageois, type `double`),
- `malade` (type `boolean`, sa valeur est `true` si le villageois est malade, `false` sinon).

Quel doit être le nom du fichier contenant cette classe ?

**Q 10.2** Ajouter dans `Villageois` le constructeur `public Villageois(String nomVillageois)` qui initialise :
- le nom du villageois avec la valeur de `nomVillageois`,
- la variable `poids` avec un poids compris entre 50 et 150 kg (150 exclu),
- la variable `malade` à `true` dans 20% des cas et à `false` sinon.

*Aide* : voir les formules du quizz 1. Voir aussi la documentation de la classe `Math` page 63.

**Q 10.3** Dans une nouvelle classe `TestVillageois`, ajouter une méthode `main`, qui crée 4 instances de la classe `Villageois`. Quel est le nom du fichier contenant cette classe `TestVillageois` ? Compiler et exécuter ce programme.

**Q 10.4** On n'a pas encore ajouté de méthode `toString` dans la classe `Villageois` pourtant cette méthode qui est

une méthode standard existe pour chaque objet. Vérifiez-le en ajoutant dans le `main` les instructions pour afficher la méthode `toString` de chacun des villageois. Compilez et exécutez.

**Q 10.5** On rappelle que le but de la méthode standard `public String toString()` est de retourner une chaîne de caractères qui représente l'objet. En général, elle contient une concaténation facile à lire des variables d'instance. Il est recommandé de définir la méthode `toString` dans chaque classe. Ajouter maintenant dans la classe `Villageois` la méthode `toString()` qui doit retourner une chaîne décrivant les caractéristiques d'un villageois. Par exemple :
`"villageois : Eustache, poids : 95 kg, malade : non"`
Attention : on veut `oui` ou `non`, et non pas `true` ou `false`.
Aide : `String.format("%.2f",123.4567);` retourne la chaîne de caractères `"123.45"` (deux chiffres après la virgule).
Compilez et exécutez à nouveau votre programme. Comparez avec le résultat de la question précédente.

**Q 10.6** Ajouter dans la classe `Villageois` et utiliser dans la classe `TestVillageois` les accesseurs suivants :
- `public String getNom()` qui retourne le nom de ce villageois,
- `public double getPoids()` qui retourne le poids du villageois,
- `public boolean getMalade()` accesseur de la variable malade,

**Q 10.7** Ajouter dans la classe `Villageois` la méthode `double poidsSouleve()` qui retourne le poids soulevé par ce villageois : le tiers de son poids s'il est en bonne santé, le quart s'il est malade.

**Q 10.8** Modifier la méthode `toString()` pour qu'elle retourne en plus le poids soulevé. Par exemple :
`"villageois : Eustache, poids : 95 kg, malade : non", peut soulever 31.7 kg`

**Q 10.9** Ajouter dans la classe `TestVillageois`, les instructions pour calculer le poids total que peuvent soulever les 4 villageois définis dans le `main`, et afficher un message pour indiquer s'ils réussissent à soulever le rocher ou pas.

## Exercice 11 – Affichage avec passage à la ligne

Soit la classe `Lettre` suivante dont le but est de gérer des caractères.

```
1 public class Lettre {
2         private char carac;
3
4         public Lettre(char c) {
5                 carac=c;
6         }
7         public char getCarac() {
8                 return carac;
9         }
10        public int getCodeAscii() {
11                return (int)carac;
12        }
13 }
```

**Q 11.1** Dans la méthode main d'une classe `TestLettre`, écrire les instructions qui, pour chaque caractère de 'a' à 'z', affiche son code ascii (utiliser la méthode `getCodeAscii()`).
*Aide* : utiliser une boucle `for` avec un compteur de type `char`.

**Q 11.2** On veut maintenant afficher l'alphabet comme ceci :

```
a b c d e
f g h i j
k l m n o
p q r s t
u v w x y
z
```

Pour cela, il suffit de répéter l'affichage d'un caractère en passant à la ligne tous les cinq caractères. A la suite dans le `main`, en utilisant la méthode `getCarac()` de la classe `Lettre`, effectuer cet affichage.
*Aide* : utiliser l'opérateur % (modulo, c-à-d reste de la division) et l'instruction : `System.out.print(`*`chaine`*`);` qui affiche *`chaine`* sans passer à la ligne (contrairement à `System.out.println()`).

### Exercice 12 – Formule de Newton

La suite de Newton définie ci-dessous converge vers la racine carrée du nombre `x` :
$U_0 = \frac{x}{2}$ et $U_i = \frac{1}{2}(U_{i-1} + \frac{x}{U_{i-1}})$ pour tout $i \geq 1$
Écrire une classe `SuiteNewton` qui étant donné un nombre $x$ et un réel $\varepsilon$ calcule la valeur de la racine de $x$ avec une précision de $\varepsilon$ en utilisant la suite de Newton.

### Exercice 13 – Libellé d'un chèque (*exercice long et fastidieux...*)

Écrire une classe `Libelle` qui traduit en toutes lettres un nombre entier inférieur à 1000 selon les règles usuelles de la langue française.
*Par exemple* : 123 s'écrit "cent vingt-trois" et 321 s'écrit "trois cent vingt et un".
Le programme doit prendre en compte les règles d'orthographe essentielles :
- les nombres composés inférieurs à cent prennent un trait d'union sauf vingt et un, trente et un, ..., soixante et onze.
- les adjectifs numéraux sont invariables sauf cent et vingt qui se mettent au pluriel quand ils sont multipliés et non suivis d'un autre nombre.

## 2 Encapsulation, surcharge

### Exercice 14 – Classe Bouteille (surcharge de constructeurs, this)

Soit la classe `Bouteille` suivante :

```
1 public class Bouteille{
2         private double volume; // Volume du liquide dans la bouteille
3
4         public Bouteille(double volume){
5                 this.volume = volume;
6         }
7         public Bouteille() {
8                 this(1.5);
9         }
10        public void remplir (Bouteille b){
11                // A compléter
12        }
13        public String toString(){
14                return("Volume du liquide dans la bouteille ="+volume);
15        }
16 }
```

**Q 14.1** Combien y-a-t-il de constructeurs dans cette classe ? Quelle est la différence entre ces constructeurs ? Pour chaque constructeur, donner les instructions qui permettent de créer un objet utilisant ce constructeur.

**Q 14.2** Expliquer l'affectation de la ligne 5 : que représente `this.volume` ? `volume` ?

**Q 14.3** Expliquer la ligne 8.

**Q 14.4** Compléter la méthode d'instance `remplir(Bouteille b)` qui ajoute le contenu de `b` à la bouteille courante.

**Q 14.5** Peut-on rajouter une méthode portant le même nom que la méthode précédente, mais prenant un paramètre de type `double` ? Si oui, écrire cette méthode.

**Q 14.6** Quel va être le résultat de l'affichage des lignes 5, 6, 8 et 9 du programme ci-après ? Expliquer.

```
1 public class TestBouteille{
```

```
2     public static void main (String[] args){
3         Bouteille b1=new Bouteille(10);
4         Bouteille b2=new Bouteille();
5         System.out.println(b1.toString());
6         System.out.println(b2.toString());
7         b1.remplir(b2);
8         System.out.println(b1.toString());
9         System.out.println(b2.toString());
10    }
11 }
```

## Exercice 15 – Addition de couples d'entiers

```
1 public class Couple {
2     private int x,y;
3     public Couple(int x, int y) {
4         this.x=x; this.y=y;
5     }
6     public String toString() {
7         return "("+x+","+y+")";
8     }
9 }
```

```
10 public class TestCouple {
11     public static void main(String [] args){
12       Couple cA=new Couple(1,5);
13       Couple cB=new Couple(3,7);
14
15       Couple cAPlusCB = ..................
16     }
17 }
```

Écrire la méthode `addition` qui permet d'additionner deux couples d'entiers (bien réfléchir aux paramètres et au type de retour), puis compléter la méthode `main` pour créer un nouveau couple résultat de l'addition de `cA` et `cB`.

## Exercice 16 – Sélection de méthode

Soit une classe `Truc` contenant un constructeur sans argument... Et 4 méthodes portant le même nom :

```
1 public class Truc{
2     public Truc(){ }
3     public void maMethode(int i){
4         System.out.println("je␣passe␣dans:␣maMethode(int␣i)");
5     }
6     public void maMethode(double d){
7         System.out.println("je␣passe␣dans:␣maMethode(double␣d)");
8     }
9     public void maMethode(double d1, double d2){
10        System.out.println("je␣passe␣dans:␣maMethode(double␣d1,␣double␣d2)");
11    }
12    public void maMethode(int i1, int i2, int i3){
13        System.out.println("je␣passe␣dans:␣maMethode(int␣i1,␣int␣i2,␣int␣i3)");
14    }
15 }
```

**Q 16.1** Selon le principe de base de JAVA qui interdit deux signatures identiques pour des méthodes (sans prise en compte du retour), cette classe compile-t-elle ?

**Q 16.2** Donner les affichages associés à l'exécution du programme suivant. Certaines lignes ne compilent pas : indiquer brièvement pourquoi.

```
1 public class TestTruc{
2     public static void main(String[] args){
3         Truc t = new Truc();
4         Truc t2 = new Truc(2);
5         double deux = 2;
6         int i = 2.5;
7         t.maMethode(2);
8         t.maMethode(deux);
```

```
9           t.maMethode(2.);
10          t.maMethode(1, 2);
11          t.maMethode(1, 2, 3);
12          t.maMethode(1., 2, 3);
13      }
14  }
```

---

**Exercice 17 – Point : sur l'égalité**

---

Soit le programme suivant :

```
1  // TestPoint.java
2  public class TestPoint {
3      public static void main(String[] args){
4          Point p1 = new Point(1,2);
5          Point p2 = new Point(1,2);
6          Point p3 = new Point(2,3);
7          Point p4 = p1;
8          Point p5 = null;
9          Point p6 = p5;
10         p6 = new Point(3,4);
11     if(p1==p2)
12       System.out.println("p1 egale p2");
13     if(p1==p3)
14       System.out.println("p1 egale p3");
15     if(p1==p4)
16       System.out.println("p1 egale p4");
17     if(p1==null)
18       System.out.println("p1 egale null");
19     if(p1.equals(p2))
20       System.out.println("p1 egale p2 (2)");
21     if(p1.equals(p4))
22       System.out.println("p1 egale p4 (2)");
23     }
24 }
```

**Q 17.1** Donner le nombre d'instances de `Point` crées lors de l'exécution. Dessiner l'état de la mémoire après l'exécution de la première colonne de code.

**Q 17.2** Quels sont les affichages à l'issue de l'exécution de la seconde colonne.

**Q 17.3** Donner les sorties associées aux commandes suivantes :

```
25 System.out.println(p5);   System.out.println(p6);
26 System.out.println(p5.toString());   System.out.println(p6.toString());
```

**Q 17.4** On ajoute encore 2 lignes au programme principal. Quel est l'impact de chacune des deux lignes sur le nombre total d'instances présentes en mémoire ?

```
27 p3=p1;
28 p1=p4;
```

## Quizz 5 – Fleur (constructeur, `this`)

Etudier le programme ci-dessous puis répondre aux questions.

```
1  public class Fleur {
2          private String nom;
3          private String couleur;
4
5          public Fleur (String name, String couleur) {
6                  nom = name;
7                  this.couleur = couleur;
8          }
9          public Fleur (String nom) {
10                 this(nom,"rouge");
11         }
12         public String toString() {
13                 return nom + " de couleur " + couleur ;
14         }
15         public String getNom() { return nom; }
16 }
```

```
17
18 public class TestFleur {
19         public static void main (String[] args ) {
20                 Fleur tulipe = new Fleur("Tulipe", "Jaune");
21                 System.out.println(tulipe.getNom());
22         }
23 }
```

**QZ 5.1** Pourquoi a-t-on déclaré `private` les variables `nom` et `couleur` ?

**QZ 5.2** La variable d'instance `nom` aurait-elle pu être déclarée après la variable `couleur` ? après la méthode `getNom()` ? Si oui, est-ce que cela aurait fait une différence ? Peut-on intervertir les lignes 20 et 21 ?

**QZ 5.3** Dans la classe `TestFleur`, quelle différence faites-vous entre `tulipe` et `"Tulipe"` ?

**QZ 5.4** Quel est le rôle de la méthode `getNom()` ?

**QZ 5.5** Dans le constructeur de la classe `Fleur`, aurait-on pu écrire `this.nom = name` ?

**QZ 5.6** Si dans la méthode main, on rajoute l'instruction : `tulipe.toString();` Quel est le résultat produit par cette instruction ?

**QZ 5.7** Un étudiant rajoute le constructeur suivant. Quelle erreur est signalée à la compilation ?

```
1 public Fleur (String couleur) {
2         this("Marguerite",couleur);
3 }
```

**QZ 5.8** Un autre étudiant rajoute dans la classe `Fleur` le constructeur suivant. Quelle erreur est signalée à la compilation ?

```
1 public Fleur () {
2         couleur="jaune";
3         this("Jonquille");
4 }
```

**QZ 5.9** Un troisième étudiant propose le constructeur suivant. Le programme compile et fait ce qui est demandé, pourtant il y a un problème avec ce constructeur. Quel est-il ?

```
1 public Fleur () {
2         this("Rose");
3         couleur="rouge";
4 }
```

## Quizz 6 – Encapsulation

```
1 public class Point {
2         private int x;
3         public int y;
4         public void f1 () {}
5         private void f2 () {}
6 }
7 public class TestPoint {
8         public static void main(String[] args) {
9                 Point p1=new Point();
10                System.out.println(p1.x);
11                System.out.println(p1.y);
12                p1.f1();
13                p1.f2();
14        }
15 }
```

Parmi les instructions de la méthode `main`, quelles sont celles qui provoquent une erreur ? Expliquez.

### Quizz 7 – Méthode `toString()`

**QZ 7.1** `int k=3; System.out.println("k="+k.toString());` Ces instructions sont-elles correctes ?

**QZ 7.2** Soit la classe suivante :

```
1 class Fleur {
2     public String toString() {
3         return "Je␣suis␣une␣fleur";
4     }
5 }
```

Soit la déclaration : `Fleur f1=new Fleur();` Qu'affiche : (a) `System.out.println(f1.toString())` ? (b) `System.out.println(f1)` ? (c) `System.out.println("Affichage de :\n\t"+f1)` ?

# TME 2 : Exercices simples et composition

**Remarque** : les intitulés des exercices de cette séance sont aussi disponibles en ligne sur le site Moodle de l'UE : https://moodle-sciences.upmc.fr/moodle-2021/course/view.php?id=3409.

## Exercice 18 – Adresse Web (Surcharge et appel de constructeurs)

On suppose qu'une adresse web est composée de 3 éléments :
- un protocole (par exemple : `http`, `https`, `ftp`,...),
- un nom de domaine (par exemple : `supersite.fr` )
- et un chemin commençant par `"/"` (par exemple : `/rep1/rep2/index.html`).

L'URL correspondante est de la forme : `http://www.supersite.fr/rep1/rep2/index.html`
c'est-à-dire : protocole, suivi de `"://www."`, suivi du domaine, suivi du chemin (qui peut être vide).

**Q 18.1** Ecrire la classe `AdresseWeb` qui contiendra les variables et méthodes suivantes :
- `protocole`, `domaine`, `chemin` : des chaînes de caractères,
- un premier constructeur qui prend en paramètre un protocole, un domaine et un chemin,
- un deuxième constructeur qui prend en paramètre un domaine et un chemin. On suppose que toutes les adresses web créées avec ce constructeur auront le protocole `http`. Ce constructeur doit appeler le constructeur à 3 paramètres.
- un troisième constructeur qui prend en paramètre un domaine. On suppose que toutes les adresses web créées avec ce constructeur auront le protocole `http` et auront pour chemin la chaîne de caractères vide. Ecrivez ce constructeur avec le moins d'instructions possibles.
- une méthode `String toString()` qui retourne l'URL de l'adresse web. Par exemple, pour l'adresse web de protocole *https*, de domaine *site.fr* et de chemin */dir/page1.html*, la chaîne retournée est : `"https://www.site.fr/dir/page1.html"`.

**Q 18.2** Ecrire la classe `TestAdresseWeb` qui crée 3 adresses Web en appelant à chaque fois un constructeur différent, puis affiche les URLs correspondantes. A quoi sert la surchage de constructeurs ? Quel est l'intérêt d'utiliser `this(...)` au lieu de réécrire l'initialisation de chaque variable d'instance dans chaque constructeur ?

## Exercice 19 – Course de relais 4 fois 100m

On veut modéliser la course de relais quatre fois cent mètres avec passage de témoin.

**Q 19.1** Écrire une classe `Coureur` comportant les variables d'instance suivantes :
- `numDossard` de type `int` (numéro du dossard du coureur),
- `tempsAu100` de type `double` (nombre de secondes pour un 100m),
- `possedeTemoin` de type `boolean` (vrai si et seulement si le coureur possède le témoin).

**Q 19.2** Ajouter dans la classe `Coureur`, les constructeurs suivants :

— un constructeur prenant un seul paramètre correspondant au numéro du dossard, qui initialise `tempsAu100` avec un nombre aléatoire choisi dans l'intervalle $[12, 16[$, et `possedeTemoin` avec `false`,
— un constructeur sans paramètre qui appelle le constructeur à un paramètre et qui initialise `numDossard` avec un entier choisi aléatoirement entre 1 et 1000.

**Q 19.3** Dans un autre fichier, écrire une classe `TestCoureur` contenant la méthode `main`, point d'entrée du programme. Cette méthode crée 4 instances de la classe `Coureur` : `c1`, `c2`, `c3` et `c4`. Vérifier que la méthode `main` compile.

**Q 19.4** Ajouter et tester au fur et à mesure dans la méthode `main()` de `TestCoureur` les méthodes suivantes :
— les accesseurs : `int getNumDossard()`, `double getTempsAu100()`, `boolean getPossedeTemoin()`,
— le modifieur : `void setPossedeTemoin(boolean b)` qui change la valeur de la variable `possedeTemoin`,
— la méthode `toString()` qui retourne une chaîne de caractères décrivant les caractéristiques de ce coureur. Exemple : `Coureur 56 tempsAu100 : 13,7s au 100m possedeTemoin : non`.

**Q 19.5** Ajouter dans la classe `Coureur` les méthodes suivantes :
— `void passeTemoin(Coureur c)` qui affiche : `"moi, coureur xx, je passe le témoin au coureur yy"`, enlève le témoin à ce coureur et le donne au coureur `c` passé en paramètre.
— `void courir()` qui simule la course du coureur sur 100 mètres en affichant le message `"je suis le coureur xx et je cours"`.

**Q 19.6** Ajouter dans la méthode main les instructions qui permettent de :
— faire courir en relais 4 fois 100m les quatre coureurs dans l'ordre c1, c2, c3, c4.
— calculer et afficher le temps total mis par les coureurs pour faire les 400m.

### Exercice 20 – Gestion des complexes

La classe `Complexe` possède :
— deux attributs double `reelle` et `imag`,
— un constructeur à 2 arguments initialisant les deux attributs
NB : signature obligatoire : `public Complexe(double reelle, double imag)`,
— un constructeur sans argument, qui initialise les arguments aléatoirement entre -2 et 2.
NB : utiliser obligatoirement `this()` dans ce second constructeur.

**Q 20.1** Donner le code de la classe `Complexe`.

**Q 20.2** Ajouter les méthodes suivantes (à vous de déterminer les signatures) :
— `toString` qui génère une chaine de caractère de la forme : (*reelle* `+` *imag* `i`)
— `estReel` qui teste si le complexe est en fait réel (dans le cas où la partie imaginaire est nulle).
— `addition` de deux complexes. Aide : $(a + bi) + (a' + b'i) = (a + a') + (b + b')i$
— `multiplication` de deux complexes. Aide : $(a + bi) \times (a' + b'i) = (aa' - bb') + (ab' + ba')i$
Pour vérifier, tester : $i^2 = -1$, $(1 + i) \times (2 + 2i) = 4i$

**Q 20.3** Donner le code de la classe `TestComplexe` qui, dans un `main`, effectue les opérations suivantes :
— créer 3 complexes, les afficher,
— tester s'ils sont réels ou pas,
— les additionner, multiplier et afficher les résultats

## 3 Composition, copie d'objets

### Exercice 21 – Composition/agrégation et modélisation

Dessiner le diagramme de classes montrant seulement les relations de composition pour les problèmes suivants.

1. appartement / immeuble / pièce
2. camion poubelle / déchet / poubelle
3. aéroport / avion / piste

## Exercice 22 – Pion (copie d'objets et composition)

**Q 22.1** Soient les classes suivantes :

```
1 public class Pion {
2   private String nom ;
3   private double posx ; //position du pion
4
5   public Pion(String n) {
6     nom=n;
7     posx=Math.random();
8   }
9   public void setNom(String n) { nom=n; }
10  public String getNom() { return nom; }
11 }
```

```
12 public class TestPion {
13   public static void main(String [] args) {
14     Pion unPion=new Pion("Atchoum");
15     Pion autrePion=unPion;
16     autrePion.setNom("Dormeur");
17     System.out.println(unPion.getNom());
18   }
19 }
```

**Q 22.1.1** Que s'affiche-t-il ? Quel est le problème ? Aide : combien y-a-t-il d'objets `Pion` créés ?

**Q 22.1.2** Une solution possible est d'utiliser un constructeur par copie. Ecrire le constructeur par copie de `Pion`.

**Q 22.1.3** Modifier la méthode `main` pour résoudre le problème.

**Q 22.2** On ajoute la classe `Point` ci-dessous, et on modifie la classe `Pion` pour que l'attribut `posx` soit maintenant non plus de type de base `double`, mais de type `Point`.

```
1 public class Point {
2   private double x, y;
3   public Point() {
4     x=Math.random();
5     y=Math.random();
6   }
7   public void bouger() {
8     x=Math.random();
9     y=Math.random();
10  }
11 }
```

```
1 public class Pion {
2   private String nom ;
3   private Point posx ; // modification
4
5   public Pion(String n) {
6     nom=n;
7     posx=new Point(); // modification
8   }
9  ...
10 }
```

**Q 22.2.1** La classe `Pion` (avec le constructeur par copie) et la classe `TestPion` modifiée compilent-t-elles toujours ?

**Q 22.2.2** On ajoute dans la classe `Pion`, la méthode `seDeplacer()` ci-dessous, quel est le résultat de l'instruction : `autrePion.seDeplacer()` placée dans le `main` après la ligne 17 ? Expliquez le problème.
*Aide* : combien y-a-t-il d'objets `Point` créés ?

```
public void seDeplacer() {
  posx.bouger();
}
```

**Q 22.2.3** Proposez une solution pour résoudre le problème.

**Q 22.3** (optionnel) Si vous avez déjà vu les méthodes `clone()` en cours, recommencez l'exercice en utilisant non pas un constructeur par copie, mais une méthode `clone()` qui est la façon standard de faire de la copie d'objets en Java.

## Exercice 23 – Feu tricolore

Un feu tricolore est composé de 3 lampes : une verte, une orange et une rouge. Soit la classe `Lampe` suivante :

```
1 class Lampe {
2     private boolean etat; // true allumee, false eteinte
3     public Lampe() { etat=false; }
4 }
```

**Q 23.1** Dessiner le diagramme de classe.

**Q 23.2** Écrire la classe `FeuTricolore` avec les constructeurs suivants :
- un constructeur sans paramètre qui crée un feu tricolore où toutes les lampes sont éteintes.
- un constructeur qui prend 3 lampes en paramètre. Donnez 2 façons de créer un objet de la classe `FeuTricolore` en utilisant ce constructeur.

**Q 23.3** Pourquoi le constructeur suivant est-il erroné ? Faire un schéma des objets en mémoire.

```
1 public FeuTricolore(Lampe l) {
2     verte=l;
3     orange=l;
4     rouge=l;
5 }
```

**Q 23.4** Trouver et expliquer les erreurs dans les instructions ci-après. Faire un schéma des objets en mémoire.

```
1 Lampe lp1=new Lampe();
2 Lampe lp2=lp1;
3 FeuTricolore ft=new FeuTricolore(lp1,lp2,lp1);
```

## Exercice 24 – Mariage (composition récursive)

On veut écrire un programme qui modélise le mariage et le divorce. Pour simplifier, on suppose que les personnes s'appellent `"pers"` suivi de 3 lettres. Voici une possibilité pour écrire la classe `Personne` :

```
1  public class Personne {
2          private String nom;
3
4          public Personne() {
5                  this("pers");
6                  nom = nom +  tirageLettre()+ tirageLettre()+ tirageLettre();
7          }
8          public Personne(String nom) {
9                  this.nom=nom;
10         }
11         private char tirageLettre(){
12                 return (char) ((int) (Math.random()*26) + 'A');
13         }
14 }
```

**Q 24.1** Compléter et modifier la classe `Personne` pour avoir le conjoint de cette personne (qui est une `Personne`). Par défaut une personne est célibataire. Écrire aussi la méthode `toString()` qui retourne le nom de la personne auquel est ajouté "célibataire" ou "marié(e)" suivant le cas. Par exemple : `"persATD, marié(e)"`.

**Q 24.2** Écrire la méthode `void epouser(Personne p)` qui marie cette personne et la personne `p`. Si l'une des 2 personnes est déjà mariée, le mariage est impossible, on affiche alors le message `"Ce mariage est impossible!"`.

**Q 24.3** Écrire la méthode `void divorcer()` qui fait divorcer cette personne si cela est possible.

**Q 24.4** Écrire une méthode `main` créant trois célibataires `p1`, `p2`, et `p3`, qui marie `p1` à `p2`, puis `p1` à `p3` (impossible), puis fait divorcer `p1` et `p2`. Voici une exécution possible :

```
Les personnes :
persATD , celibataire
persZIG , celibataire
persTHX , celibataire
Mariage de persATD , celibataire  et de persZIG , celibataire :
persATD , celibataire  se marie avec persZIG , celibataire
Les personnes apres mariage :
persATD , marie(e)
```

```
persZIG , marie(e)
Essai de mariage de persATD , marie(e)  et de persTHX , celibataire :
Ce mariage est impossible!
Divorce de persATD , marie(e) et de persZIG , marie(e) :
persATD , marie(e)  divorce de persZIG , marie(e)
Les personnes apres divorce :
persATD , celibataire
persZIG , celibataire
```

## Exercice 25 – Tracteur (composition d'objets et copie d'objets)

Un tracteur agricole est composé de 4 roues et d'une cabine.

**Q 25.1** Donner le diagramme de classe correspondant.

**Q 25.2** Écrire une classe `Roue` ayant un attribut privé de type `int` définissant son diamètre. Écrire deux constructeurs, l'un avec un paramètre, et l'autre sans paramètre qui appelle le premier pour mettre le diamètre à 60 cm (petite roue). Écrire aussi la méthode `toString()`.

**Q 25.3** Créer une classe `TestTracteur` pour tester la classe `Roue` dans une méthode `main` dans laquelle sont créées 2 grandes roues de 120 cm et 2 petites roues. Compiler et exécuter.

**Q 25.4** Écrire une classe `Cabine` qui a un volume (en mètres cubes (m3)) et une couleur de type `String`. Écrire un constructeur avec paramètres et la méthode `toString()` qui rend une chaîne de caractères donnant le volume et la couleur. Ajouter le modifieur `setCouleur(String couleur)`.

**Q 25.5** Ajouter dans la méthode `main` la création d'une cabine bleue.

**Q 25.6** Écrire la classe `Tracteur` où celui-ci est constitué d'une cabine et de quatre roues, avec un constructeur avec 5 paramètres (la cabine et les 4 roues), d'une méthode `toString()`, d'une méthode `peindre(String couleur)` qui change la couleur de la cabine du tracteur.

**Q 25.7** Créer un tracteur `t1` dans la méthode `main` avec les 4 roues et de la cabine bleue créées précédemment. Afficher ensuite ce tracteur.

**Q 25.8** Ajouter l'instruction `Tracteur t2=t1;` puis modifier la couleur de la cabine du tracteur `t2`. Quelle est la couleur de la cabine de `t1` ? Expliquer pourquoi la couleur a changée. Que faut-il faire pour que `t1` et `t2` soient deux objets distincts qui ne contiennent pas les mêmes objets ? Expliquer et appliquer cette correction.

## Exercice 26 – Classe triangle

**Q 26.1** Écrire une classe `Point` à deux variables d'instance entières `posx` et `posy`, respectivement l'abscisse et l'ordonnée du point. Cette classe comprendra :

- Un constructeur par défaut (sans paramètre).
- Un constructeur a deux paramètres entiers : l'abscisse et l'ordonnée.
- Les modifieurs et accesseurs `setPosx`, `setPosy`, `getPosx`, `getPosy` qui permettent respectivement de modifier ou récupérer les coordonnées d'un objet de la classe `Point`.
- La méthode `public String toString()` qui retourne une chaîne de caractères décrivant le point sous la forme `(x, y)`. Par exemple, (3, 5) pour le point d'abscisse 3 et d'ordonnée 5.
- La méthode `distance(Point p)` recevant en paramètre un objet de la classe Point et retournant sa distance à cet objet (c'est-à-dire l'objet sur lequel est invoquée cette méthode).
- La méthode `deplaceToi(int newx, int newy)` qui déplace le point en changeant ses coordonnées.

**Q 26.2** Tester cette classe en écrivant la méthode `main` qui crée des points et affiche leurs coordonnées.

**Q 26.3** Écrire une classe `Triangle` à 3 variables d'instance prenant leur valeur dans la classe `Point`. Elle comprendra :

- Un constructeur par défaut.
- Un constructeur à trois paramètres : les trois sommets du triangle.
- Une méthode `getPerimetre()` qui retourne le périmètre du triangle.

— Une redéfinition de la méthode `public String toString()` qui retourne une chaîne de caractères décrivant le triangle (en utilisant la méthode `toString()` de la classe `Point`).

**Q 26.4** Écrire une classe `TestTriangle`, contenant une méthode `main` dans laquelle on crée 3 points, puis un triangle composé de ces 3 points. On affichera ensuite les caractéristiques du triangle (les 3 points, la longueur de ses côtés et son périmètre).

**Q 26.5** Comment tester l'égalité structurelle entre deux triangles ? Réfléchir à l'organisation du code et aux signatures des méthodes puis proposer une implémentation dans les différentes classes.

### Quizz 8 – Instanciation

Soient la classe `public class A {}` et les instructions suivantes :

```
1 A a1=new A();
2 A a2=a1;
3 A a3=new A();
4 A a4=null;
```

**QZ 8.1** La classe `A` contient-elle un constructeur ?

**QZ 8.2** Combien d'objets sont-ils créés ?

**QZ 8.3** Combien de références (appelées aussi *handles*) utilisés ?

**QZ 8.4** Que se passe-t-il si on rajoute l'instruction `a3=null;` ? `a2=null;` puis `a1=null;` ?

# 4 Tableaux

## Exercice 27 – Base syntaxique

**Q 27.1** Donner deux façons pour créer le tableau `tab` d'entiers suivant :

| 1 | 2 | 3 |
|---|---|---|

**Q 27.2** Donner deux façons pour créer le tableau `mat` d'entiers à deux dimensions suivant :

| 4 | 5 | 6 |
|---|---|---|
| 7 | 8 | 9 |

**Q 27.3** Soit le code suivant :

```
1 double[] tabD = new double[10];
2 for(int i=0; i<tabD.length; i++) {
3         tabD[i] = Math.random();
4 }
```

(a) Quel est l'intérêt d'écrire `tabD.length` au lieu d'écrire 10 ? (b) En utilisant la variante de la boucle `for` qui n'utilise pas les indices du tableau, afficher ce tableau. (c) Peut-on utiliser cette variante de la boucle `for` pour afficher seulement les cases du tableau dont l'indice est pair ? (d) Réécrire le code des lignes 2 à 4 en utilisant la variante de la boucle `for`. Obtient-on le même résultat ?

**Q 27.4 Tableau d'objets**. En supposant une classe `Point` existante avec un constructeur sans paramètre, créer un tableau contenant 10 instances de `Point`, puis afficher le tableau avec la variante de la boucle `for`.

**Q 27.5** Quel affichage correspond aux lignes suivantes ?

```
int[] t1 = {1,2,3};
int[] t2 = {1,2,3};
int[] t3 = t1;
System.out.println(t1 == t2); System.out.println(t1 == t3);
```

### Exercice 28 – Tableau triangulaire

Créer le tableau d'entiers ci-contre.
Aide : la déclaration s'effectue en 2 étapes :
- d'abord, on déclare un tableau de 3 lignes où chaque ligne est un tableau d'entiers (sans préciser la taille de la deuxième dimension)
- puis, pour chaque ligne, on déclare un tableau d'entiers à la bonne taille

| | | |
|---|---|---|
| 1 | | |
| 2 | 2 | |
| 3 | 3 | 3 |

### Exercice 29 – N-uplets (classe avec attribut de type tableau)

On souhaite écrire des classes qui permettent de gérer des n-uplets. Par exemple, le triplet (7,8,9), le 5-uplet (3,3,3,3,3).

**Q 29.1** Ecrire la classe `NUplet` qui contient pour seul attribut un tableau `tab` d'entiers et les constructeurs :
- un constructeur : `NUplet(int n)` qui réserve `n` cases mémoires pour le tableau référencé par `tab`
- un constructeur : `NUplet(int n, int x)` qui crée un tableau de $n$ cases mémoires et initialise toutes les cases du tableau à la même valeur $x$ (exemple : le 5-uplet (3,3,3,3,3)). Attention : on demande que vous appeliez le constructeur à un paramètre
- un constructeur : `NUplet(int a, int b, int c)` qui crée le triplet `(a,b,c)`. Attention : on demande que vous appeliez le constructeur à un paramètre

**Q 29.2** On suppose que l'on est dans une méthode `main`, donnez les instructions pour créer le 5-uplet (3,3,3,3,3) et le triplet (7,8,9).

**Q 29.3** Ajouter à la classe `NUplet` les méthodes suivantes :
- une méthode `toString()` qui retourne la chaîne de caractères correspondant au n-uplet. Par exemple, pour le triplet (7,8,9), cette méthode doit retourner la chaîne de caractères : `"(7,8,9)"`. Remarque : si la taille du tableau est 0, cela doit retourner `"()"`.
- une méthode `int somme()` qui retourne la somme des éléments du n-uplet. Par exemple, pour le triplet (7,8,9), cette méthode retourne l'entier 24.

**Q 29.4** On ajoute dans la classe `NUplet` le constructeur ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème, puis proposez une solution.

```
public NUplet(int [] tab) {
   this.tab=tab;
}
```

```
int [] t123={1,2,3};
NUplet u3=new NUplet(t123);
t123[0]=50;
System.out.println(u3.toString());
```

**Q 29.5** On ajoute dans la classe `NUplet` la méthode `getTab` ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème, puis proposez une solution.

```
public int [] getTab() {
  return tab;
}
```

```
NUplet u4=new NUplet(4,5,6);
int [] t456=u4.getTab();
t456[0]=70;
System.out.println(u4.toString());
```

**Q 29.6** Écrire une méthode `public boolean egal(NUplet n2)` qui rend vrai si `n2` est égal au n-uplet courant.

### Exercice 30 – Représentation mémoire d'objets et de tableaux

Soit une classe `Truc` possédant un constructeur sans argument.

**Q 30.1** Donner la représentation mémoire correspondant à l'exécution du code suivant. Combien d'instances de `Truc` ont été créées à l'issue de l'exécution de ces lignes ?

```
1 Truc o = new Truc();
2 Truc o2 = o;
3 Truc[] tabO = new Truc[3];
4 tabO[0] = new Truc(); tabO[1] = o; tabO[2] = o2;
```

**Q 30.2** Donner les instructions nécessaires pour dupliquer le tableau `tab0`. Le résultat est-il satisfaisant ?

## Exercice 31 – Tableau d'entiers

**Q 31.1** Écrire une classe `TableauInt` qui comporte une variable d'instance `tab` de type tableau de 10 entiers. Cette classe contient deux constructeurs :

- un constructeur sans paramètre qui initialise le tableau avec des nombres entiers compris entre 0 et 100, générés aléatoirement (à l'aide de la méthode statique random() de la classe `Math` qui génère une valeur aléatoire de type double comprise entre 0.0 inclus et 1.0 exclu).
- un constructeur à un paramètre entier n qui initialise le tableau avec des valeurs consécutives à partir de n : (n, n+1,..., n+9).

**Q 31.2** Ajouter dans cette classe les trois méthodes suivantes :

- une méthode `public String toString()` qui rend une chaîne représentant les valeurs du tableau sous la forme : `"[a0, a1, a2, ...]"`.
- une méthode `rangMax` qui renvoie le rang du maximum du tableau.
- une méthode `somme` qui renvoie la somme des éléments du tableau.

**Q 31.3** Tester ces méthodes au fur et à mesure dans la méthode `main` d'une classe `TestTableau`.

## Exercice 32 – Triangle de Pascal (tableau à 2 dimensions)

Le triangle de Pascal est une représentation des cœfficients binomiaux dans un triangle. Voici une représentation du triangle de Pascal en limitant le nombre de lignes à 5 :

| | | | | |
|---|---|---|---|---|
| 1 | | | | |
| 1 | 1 | | | |
| 1 | 2 | 1 | | |
| 1 | 3 | 3 | 1 | |
| 1 | 4 | 6 | 4 | 1 |

Chaque élément du triangle de Pascal peut être défini ainsi :
C(i,j) = 1 si j=0 ou si j=i
C(i,j) = C(i-1,j-1) + C(i-1,j) sinon.

**Q 32.1** Écrire une classe `TrianglePascal` qui réserve uniquement la place mémoire nécessaire pour stocker le triangle de Pascal dont le nombre de lignes est passé en paramètre du constructeur.

**Q 32.2** Ajouter à la classe `TrianglePascal` une méthode `remplirTriangle()` qui calcule les valeurs du triangle de Pascal et une méthode `toString()` qui rend la chaîne de caractères représentant le tableau sous la forme d'un triangle.

**Q 32.3** Écrire une classe `TestTrianglePascal` qui crée plusieurs instances de la classe `TrianglePascal` et les affiche.

**Q 32.4** Ajouter dans la classe `TableauInt` une méthode `boolean egal(TableauInt t)` qui teste si cet objet de type `TableauInt` a les mêmes entiers aux mêmes places que le tableau `t` passé en paramètre.

## Exercice 33 – Histogramme de notes

Dans cet exercice, il s'agit d'écrire un programme qui permet de représenter un histogramme de notes entières comprises entre 0 et 20 (c'est-à-dire il y a 21 notes possibles).
Par exemple, si dans une classe, il y a 10 étudiants qui ont obtenus les notes suivantes : 2, 3, 4, 3, 0, 0, 2, 3, 3, 2 (c'est-à-dire 2 étudiants ont obtenu la note 0, 0 étudiant ont obtenu la note 1, 3 étudiants la note 2, 4 étudiants la note 3, 1 étudiant la note 4), le tableau représentant l'histogramme sera : [2, 0, 3, 4, 1].
L'affichage de l'histogramme correspondant donnera :

```
0 | **
1 |
2 | ***
3 | ****
4 | *
```

**Q 33.1** On souhaite écrire une classe `Histo` qui affiche un histogramme des notes. Pour cela, vous définirez :

- un attribut tableau `hist` représentant l'histogramme,

— un constructeur sans paramètre qui initialise le tableau hist et met toutes les cases du tableau à la valeur 0,
— une méthode d'ajout d'une note,
— un constructeur qui prend en paramètre un tableau de notes et qui initialise l'histogramme à partir des notes.

**Q 33.2** Ajouter à cette classe, une méthode `afficheHistogrammeTableau()` qui affiche l'ensemble des valeurs du tableau histogramme.

**Q 33.3** Ajouter à cette classe, une méthode `afficheHistogramme()` qui affiche le résultat sous forme d'un histogramme, c'est-à-dire en associant à chaque élément du tableau une ligne comprenant autant de `*` que la valeur de cet élément. (comme dans l'exemple de l'énoncé)

**Q 33.4** Écrire une classe `TestHisto` dont la méthode `main` crée un tableau de notes aléatoires (150 étudiants), une instance de `Histo`, puis qui affiche le résultat sous les deux formes proposées.

## Exercice 34 – Pile de machins (tableau d'objets)

Écrire une classe `Pile` permettant de gérer une pile d'objets de type `Machin` au moyen d'un tableau.
La pile devra avoir les opérations suivantes :

— `boolean estVide()` qui indique si la pile est vide.
— `boolean estPleine()` qui indique si la pile est pleine.
— `void empiler(Machin m)` qui, si possible, ajoute l'élément au sommet de la pile.
— `Machin depiler()` qui, si possible, retire l'élément au sommet de la pile.
— `String toString()` qui retourne une chaîne représentant le contenu de la pile, à raison d'un nom par ligne, le sommet de pile étant la première valeur affichée.

**Q 34.1** Définir la classe `Machin` qui se caractérise par un nom et une valeur (*remarque* : `Machin` pourrait un objet plus sophistiqué, mais là n'est pas l'objet de l'exercice).

**Q 34.2** Définir la classe `Pile` avec ses attributs, un constructeur qui a comme paramètre la taille maximale de la pile, ainsi que les méthodes données ci-dessus. Aide : pour gérer la pile, pensez à ajouter une variable d'instance qui indique à tout moment le nombre d'éléments actuellement présents dans la pile (ce nombre correspond au sommet de la pile). Attention à ne pas confondre le nombre d'éléments présents dans la pile et la taille maximale de la pile.

**Q 34.3** Tester cette classe en écrivant une méthode `main` qui empile trois `Machin` précédemment initialisés, dépile une fois, puis empile deux autres `Machin`, puis dépile 4 fois. Afficher le contenu de la pile après chacune de ces opérations.

## Quizz 9 – Tableaux (révision)

**QZ 9.1** Soit le tableau : `int [][] tabX=new int [2][5];` Comment obtenir la taille de la première dimension du tableau ? Comment obtenir la taille de la deuxième dimension ?

**QZ 9.2** Créer le tableau d'entiers suivant :

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | |
| 1 | 2 | | |
| 1 | 2 | 3 | 4 |

**QZ 9.3** On considère la classe `Bouteille` vue dans l'exercice 14 page 8. Créer un tableau de 2 bouteilles, la première bouteille aura un volume de 3 litres et la deuxième de 1,5 litre.

## Quizz 10 – Tableaux d'objets

Qu'affichent les instructions suivantes ?

```
1 int [] tabSimple=new int[5];
2 Integer [] tabObjet=new Integer[5];
3 System.out.println(tabSimple[3]);
4 System.out.println(tabObjet[3]);
5 System.out.println(tabObjet[3].toString());
6 tabObjet[3]=new Integer(10);
7 System.out.println(tabObjet[3].toString());
```

### Quizz 11 – `final`

Rappels : le mot clef `final` indique qu'un élément (variable, méthode, classe...) ne peut être modifié
- une *variable* (attribut, paramètre ou variable locale) `final` ne peut être modifiée après initialisation
- une *variable d'instance* `final` ne peut être initialisée que lors de la déclaration ou dans le constructeur

**QZ 11.1** Pour chaque instruction ci-dessous, indiquez si l'instruction compile ou pas. Expliquez brièvement.

```
1 final int a=25;
2 a=17;
3 final int b;
4 b=10;
5 b=20;
```

**QZ 11.2** Soit la classe ci-dessous, indiquez les lignes qui ne compilent pas et expliquez.

```
1 public class Bidule {
2   private final double x;
3   private final double y=Math.random();
4   private final double z;
5   public Bidule(double x, double y) {
6       this.x=x;
7       this.y=y;
8   }
9   public void setZ(double z) {
10      this.z=z;
11  }
12 }
```

# 5 Variables et méthodes de classes

## Exercice 35 – Membres d'instance ou de classe

**Q 35.1** On rappelle qu'un membre d'une classe est soit une variable (V) soit une méthode (M). On considère les classes ci-dessous. Pour chacune des expressions sous la classe, dire si ce sont des membres d'instance (I) ou de classe (C) de cette classe :

**Classe** `Chien`
- `nom`
- `nbChiots`
- `nbChiens`
- `getNbChiens()`
- `siteWebDuChien`
- `siteWebSPA`
- `aboyer()`
- `chercherLivreSurLesChiens()`
- `regarderDVD()`

**Classe** `Chenil`
- `nbChiots`
- `nbChiens`
- `getNbChiens()`

**Classe** `Maison`
- `nbPièces`
- `prix`
- `prixMoyenEnFrance`
- `listeClassesEnergetiques`
- `classeEnergetique`
- `cptVentesDeCetteMaison`
- `cptVentesEnFrance`
- `getCptVentesEnFrance()`

**Classe** `Stylo`
- `taille`
- `TAILLE_STANDARD`
- `cptStylosProduits`

## Exercice 36 – Compter les trucs (compteur, variables d'instance et variables de classes)

Soit la classe `Truc` suivante :

```
1 public class Truc {
2         private static int cpt=0;
3         private int num;
4
5         public Truc() {
```

```
6                   cpt++;
7                   num=cpt;
8           }
9           public Truc(int x) {
10                  num=x;
11          }
12          public static int getCpt() { return cpt; }
13          public int getNum() { return num; }
14  }
```

**Q 36.1** Quel est le nom de la variable de classe ? Comment la reconnaît-on ?

**Q 36.2** Pourquoi la variable cpt a-t-elle été initialisée ?

**Q 36.3** Quel est l'affichage obtenu par l'exécution des lignes 4, 6, 8 et 9 du programme suivant :

```
1 public class TestTruc{
2           public static void main (String[] args){
3                   Truc n1=new Truc();
4                   System.out.println(n1.getCpt());
5                   Truc n2=new Truc(25);
6                   System.out.println(n1.getCpt()+"␣"+n2.getCpt());
7                   Truc n3=new Truc();
8                   System.out.println(n1.getNum()+"␣"+n2.getNum()+"␣"+n3.getNum());
9                   System.out.println(n1.getCpt()+"␣"+n2.getCpt()+"␣"+n3.getCpt());
10          }
11  }
```

**Q 36.4** Peut-on ajouter une instruction à la fin du programme de la question précédente afin d'afficher la valeur de la variable `cpt` sans utiliser d'instance ? Même question pour la variable `num`.

## Exercice 37 – Vecteur (& questions static)

```
1 public class Vecteur {
2     public final int id;
3     private static int cpt = 0;
4     public final double x,y;
5
6     public Vecteur(double x, double y) {
7             id = cpt; cpt++;
8             this.x = x; this.y = y;
9     }
10    public static int getCpt(){return cpt;}
11 }
```

**Q 37.1** A-t-on commis une *faute de conception* en déclarant plusieurs attributs comme public ? Justifier brièvement.

**Q 37.2** Les propositions suivantes sont-elles correctes du point de vue syntaxique (compilation) ? Donner les affichages pour les lignes correctes.

```
12 // dans la classe Vecteur
13 public int getCpt2(){return cpt;}
14 public static int getId(){return id;}
15 public static String format(Vecteur v){
16          return String.format("[%5.2f,␣%5.2f]", v.x, v.y);
17 }
18
19 // dans le main
20 Vecteur v1 = new Vecteur(1, 2); v2 = new Vecteur(1, 2);
21 if(v1.x == v2.x && v1.y == v2.y) System.out.println("v1␣egale␣v2");
22 if(v1.id == v2.id) System.out.println("les␣points␣ont␣le␣meme␣identifiant");
```

```
System.out.println("Compteur:␣"+v1.getCpt());
System.out.println("Compteur␣(2):␣"+Vecteur.getCpt());
System.out.println("Compteur␣(3):␣"+v1.cpt);
```

## Exercice 38 – Génération d'adresses IP

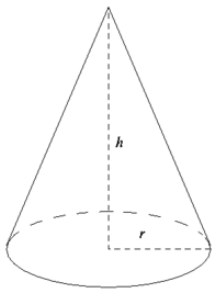
Une adresse IP est un numéro d'identification qui est attribué à chaque branchement d'appareil à un réseau informatique. Elle est composée d'une suite de 4 nombres compris entre 0 et 255 et séparés par des points. Dans le réseau privé d'une entreprise, les adresses IP commencent par 192.168.X.X où X est remplacé par un nombre entre 0 et 255. Par exemple : `"192.168.25.172"`. On souhaite écrire une classe dont le but est de générer des adresses IP. Chaque appel à la méthode `getAdresseIP()` retourne une nouvelle adresse IP. La première adresse générée sera : `"192.168.0.1"`, la deuxième `"192.168.0.2"`, ... puis `"192.168.0.255"`, `"192.168.1.0"`, `"192.168.1.1"`.

**Q 38.1** Ecrire la classe `GenerateurIP` qui contiendra les variables et méthodes suivantes :
- un constructeur `private`, car on ne veut pas créer d'instance de cette classe. NB : ce construceur ne fait rien
- `tab` : une variable de classe de type tableau de 4 entiers où chaque case correspond à une partie de l'adresse IP. Ce tableau est initialisé à l'adresse IP : 192.168.0.0
- une méthode de classe `String getAdresseIP()` qui retourne la prochaine adresse IP. Cette méthode incrémente d'abord le 4ième nombre de l'adresse IP. Si ce nombre est supérieur à 255 alors le 3ième nombre est incrémenté, et le 4ième est remis à 0. Remarque : cette méthode s'occupe seulement des 3ième et 4ième nombres de l'adresse IP, elle ne s'occupe pas du cas où la prochaine IP est celle après 192.168.255.255.

**Q 38.2** Dans une méthode `main` d'une classe `TestGenerateurIP`, afficher 257 adresses IP pour vérifier que votre méthode fonctionne.

## Exercice 39 – Cône de révolution

Un cône de révolution est défini par son rayon r et par sa hauteur h (voir figure). On souhaite écrire une classe `Cone` qui permet de calculer le volume d'un cône de révolution.




**Q 39.1** Écrire la classe `Cone` qui contient les variables ci-après.
*Attention* : certaines de ces variables sont des variables de classe.
- `r` : le rayon du cône de type double,
- `h` : la hauteur du cône de type double,
- `PI` : une constante de type double dont la valeur est 3.14159,
- `nbCones` : le nombre de cônes créés depuis le début du programme.

**Q 39.2** Ajouter les méthodes ci-après. *Attention* : certaines de ces méthodes sont des méthodes de classe.
- le constructeur dont la signature est : `public Cone(double r, double h)`,
- le constructeur sans paramètre qui initialise le rayon et la hauteur du cône entre 0 et 10 (non compris). Ce constructeur doit appeler le premier constructeur. Aide : utiliser `Math.random()`.
- la méthode `double getVolume()` qui retourne le volume $V = \frac{1}{3}\pi r^2 h$ du cône,
- la méthode `String toString()` qui retourne une chaine de caractère qui, pour un cône de rayon 5.4 et de hauteur 7.2, a le format : `"Cone r=5.4 h=7.2 V=219.854736"`,
- l'accesseur de la variable `nbCones` (aide : comment devrait être déclaré l'accesseur d'un attribut static ?)

**Q 39.3** Écrire une classe `TestCone` qui contient une méthode `main` qui commence par afficher le nombre de cônes créés depuis le début du programme (cela doit afficher 0), puis qui crée deux instances de la classe `Cone` en appelant une fois chaque constructeur et enfin qui affiche à nouveau le nombre de cônes (cela doit afficher 2).

## Exercice 40 – Chaines aléatoires (Méthodes de classe)

**Q 40.1** Écrire la classe `Alea` qui contient les deux méthodes de classe suivantes :
- la méthode de classe `lettre()` qui retourne un caractère choisi aléatoirement parmi les 26 lettres de l'alphabet (c'est-à-dire entre 'a' et 'z'). Aide : utiliser `Math.random()`

— la méthode de classe `chaine()` qui retourne une chaîne de caractères construit à partir de la concaténation de 10 lettres de l'alphabet choisis aléatoirement (appeler la méthode `lettre()`).

**Q 40.2** Pour quelle raison les méthodes `lettre()` et `chaine()` sont-elles des méthodes de classes ?

**Q 40.3** La classe `Alea` est une boite à outils : il n'y a pas besoin de créer d'instance pour l'utiliser. Afin d'ôter toute ambiguïté, proposer une solution pour interdire la création d'instance de cette classe.

**Q 40.4** Dans la méthode `main()` de la classe `Alea`, afficher le résultat retourné par la méthode `chaine()`. Même question pour la méthode `main()` d'une classe `TestAlea`.

## Exercice 41 – Projet avec trio de personnes (static)

**Q 41.1** Classe `Personne`

**Q 41.1.1** On veut écrire une classe `Personne` où chaque personne aura un nom de la forme `"Individu"` suivi d'un nombre : la première personne aura le nom `Individu1`, la deuxième `Individu2`, la troisième `Individu3`... Écrire la classe `Personne` qui possède :

— un attribut `nom`,
— un compteur `nbPersonnes` qui compte le nombre de personnes créés depuis le début du programme,
— un constructeur sans paramètre qui initialise le nom de la personne à `Individu` suivi du nombre,
— une méthode `toString()` qui retourne le nom de la personne.

**Q 41.1.2** Dans la méthode `main` d'une classe `TestProjet.java`, créer deux personnes, afficher-les et vérifier que les noms générés sont correctes (en particulier, le premier nom doit être `Individu1` et non pas `Individu0`).

**Q 41.1.3** Ajouter une méthode `getNbPersonnes()` qui retourne le nombre de personnes créés depuis le début du programme. Appeler cette méthode tout au début de votre méthode `main` (l'affichage à obtenir est 0 puisqu'il n'y a encore aucune personne créée), puis appeler cette méthode à la fin de la méthode `main`.

**Q 41.1.4** On veut modifier la classe pour que les noms des personnes soient maintenant `"Individu"` suivi d'une lettre : la 1ère personne aura le nom `IndividuA`, la 2ième `IndividuB`, la 3ième `IndividuC`... Ajouter dans votre classe `Personne` une variable `lettre` initialisée à `'A'` (qui devra prendre successivement les valeurs 'A', 'B', 'C'...), puis modifier légèrement votre classe pour générer les noms demandés.
Vérifier que l'affichage obtenu par votre méthode `main` (inutile de la modifier) ressemble à :
`nbPersonnes=0, IndividuA, IndividuB, nbPersonnes=2`.

**Q 41.2** Un trio est composé de 3 personnes (tableau de 3 personnes) et d'un numéro (entier) déterminé en fonction d'une variable compteur qui est incrémentée à chaque création d'un trio. Écrire une classe `Trio` avec un constructeur sans paramètre et une méthode `toString()`.
Par exemple, pour le trio 1, cette méthode retourne : `Trio 1 : IndividuC IndividuD IndividuE`.

**Q 41.3** Un projet est composé d'un nom de projet et d'un objet `Trio`. Ecrire une classe `Projet` avec un constructeur ayant le nom du projet comme unique paramètre et une méthode `toString()`.
Par exemple, pour le projet dont le nom est `P3X-774`, cette méthode retourne :
`Projet P3X-774 Trio 1 : IndividuC IndividuD IndividuE`

**Q 41.4** Ajouter dans la méthode `main` de la classe `TestProjet`, les instructions nécessaires pour obtenir l'affichage suivant :

```
Projet P3X-774 Trio 1 : IndividuC IndividuD IndividuE
Projet P3R-233 Trio 2 : IndividuF IndividuG IndividuH
```

**Q 41.5** Dans les classes `Trio` et `Projet`, rajouter les variables et méthodes nécessaires pour afficher dans la méthode `main` le nombre de personnes créées, le nombre de trios créés et le nombre de projets créés.

## Exercice 42 – Somme de 2 vecteurs

On veut faire la somme de deux vecteurs dans l'espace, c'est-à dire créer un nouveau vecteur résultant de la somme des deux vecteurs. Un vecteur est caractérisé par un triplet (x, y, z) de nombres réels, appelés coordonnées. Soient AB=(x1, y1, z1) et BC=(x2, y2, z2) deux vecteurs, alors le vecteur AC a pour coordonnées (x1+x2, y1+y2, z1+z2). Pour cela, on donne le début de la classe `Vecteur` :

```
1 public class Vecteur {
2         private double x, y, z;
3
4         public Vecteur(double c1, double c2, double c3) {
5                 x = c1; y = c2; z = c3;
6         }
7         public Vecteur() {
8                 this(Math.random()*10,Math.random()*10,Math.random()*10);
9         }
10        public String toString() {
11                return "(" + x + ",␣" + y + ",␣" + z + ")";
12        }
13 }
```

**Q 42.1** Ajouter à la classe `Vecteur` une méthode d'instance qui fait la somme de deux vecteurs.

**Q 42.2** Ajouter à la classe `Vecteur` une méthode de classe `qui` fait la somme de deux vecteurs.

**Q 42.3** Dans une classe `TestVecteur`, écrire une méthode main qui initialise deux vecteurs, puis fait la somme des 2 vecteurs an utilisant la méthode d'instance et en utilisant la méthode de classe.

## Exercice 43 – Génération de noms (tableau de caractères, méthode de classe)

On veut écrire une classe `Nom` qui offrira une *méthode de classe* générant des noms de façon aléatoire. On écrira cette classe avec les variables et méthodes suivantes qu'on testera au fur et à mesure :

**Q 43.1** Écrire une méthode de classe `rendAlea(int inf, int sup)` qui rend un entier naturel aléatoire entre inf et sup compris. Aide : lisez la documentation Java (voir site web de l'UE) de la classe `Random`.

**Q 43.2** Écrire une méthode de classe `boolean estPair(int n)` qui vérifie que `n` est pair.

**Q 43.3** Déclarer en variable `static` deux tableaux de `char` de noms voyelles et consonnes. Initialiser lors de la déclaration le premier avec les consonnes et le second avec les voyelles.

**Q 43.4** Écrire les méthodes `rendVoyelle()` et `rendConsonne()` qui rendent respectivement une voyelle et une consonne de façon aléatoire.

**Q 43.5** Écrire une méthode `genereNom()` qui rend un nom de longueur aléatoire comprise entre 3 et 6 caractères en générant alternativement une consonne et une voyelle.

**Q 43.6** Écrire une classe `TestNom` dont la méthode `main` génère et affiche, dans une boucle, une dizaine de noms générés.

## Quizz 12 – Variables et méthodes de classes

On considère les classes `Cercle` et `TestCercle` suivantes :

```
1 public class Cercle {
2     public static final double PI=3.14159;
3     private static int nbCercles=0;
4     public final int numero;
5     private int rayon;
6     public Cercle(int r) {
7         rayon=r;
8         nbCercles++;
9         numero=nbCercles;
10    }
```

```
11      public double surface() { return PI*rayon*rayon;  }
12      public static int getNbCercles() { return nbCercles; }
13  }
14  //==================
15  public class TestCercle {
16      public static void main(String [] args) {
17          Cercle c=new Cercle(3);
18          System.out.println(EXPRESSION);
19      }
20  }
```

**QZ 12.1** Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

| | | |
|---|---|---|
| c.PI | c.nbCercles | c.numero |
| c.rayon | c.surface(); | c.getNbCercles(); |

**QZ 12.2** Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

| | | |
|---|---|---|
| Cercle.PI | Cercle.nbCercles | Cercle.numero |
| Cercle.rayon | Cercle.surface(); | Cercle.getNbCercles(); |

**QZ 12.3** Soit la classe `Test2Cercle` suivante :

```
1  public class Test2Cercle {
2          public static void main(String [] args) {
3                  Cercle c1=new Cercle(2);
4                  Cercle c2=new Cercle(3);
5                  Cercle c3=c2;
6                  Cercle c4=new Cercle(4);
7          }
8  }
```

- Qu'affiche `System.out.println(c2.getNbCercles())` ?
- Qu'affiche `System.out.println(c4.getNbCercles())` ?

# 6 Héritage et modélisation

### Exercice 44 – Héritage et modélisation

Dessiner le diagramme de classes correspondant aux problèmes suivants.

1. Une voiture est un véhicule qui contient 4 roues
2. Les vélos, voitures et camions sont des véhicules roulants, tandis qu'un char d'assaut est un véhicule à chenille
3. Un cartable contient des fournitures (trousses, stylos....). Une trousse peut contenir des stylos.
4. Les animaux (renard, lièvre...) d'une forêt sont soit herbivores, soit carnivores.

### Exercice 45 – Personne (héritage)

Soient les classes `Personne` et `Etudiant` suivantes :

```
1  public class Personne {
2    protected final String nom;
3    protected String numTel;
4    private int nbEnfants;
5    public Personne(String nom, String numTel){
6        this.nom=nom; this.numTel=numTel; nbEnfants=0;
7    }
8    public Personne(String nom){
9        this(nom,null);
```

```
10   }
11   public String getNom() { return nom; }
12   public String getNumTel() { return numTel; }
13   protected int getNbEnfants() { return nbEnfants; }
14   public void ajouterEnfant() { nbEnfants++; }
15 }

16 public class Etudiant extends Personne {
17   private String cursus;
18   public Etudiant(String n, String t, String c) {
19       super(n,t);
20       cursus=c;
21   }
22   public boolean estEnL2 () { return cursus.equals("L2"); }
23 }
```

**Q 45.1** On ajoute dans la classe `Etudiant`, les méthodes suivantes. Pour chaque instruction de ces méthodes, indiquez si l'instruction compile ou pas. Justifiez par un mot.

```
1   public void afficherInfo() {
2     System.out.println("Nom␣:"+nom);
3     System.out.println("NumTel␣:"+numTel);
4     System.out.println("NbEnfants␣:"+nbEnfants);
5     System.out.println("Cursus␣:"+cursus);
6   }
```

```
7   public void modifierInfo() {
8     nom="toto";
9     numTel="0102030405";
10    nbEnfants=-1;
11    cursus="L0";
12  }
```

**Q 45.2** Un salarié a un salaire. Écrire la classe `Salarie` qui hérite de `Personne` et qui possède un constructeur ayant comme paramètre le nom et le salaire, et qui possède un accesseur pour le salaire.

**Q 45.3** Écrire une méthode `prime()` qui retourne le montant de la prime accordée pour les enfants, à savoir 5% du salaire par enfant. Dans quelle classe mettre cette méthode ?

**Q 45.4** Écrire une méthode `modifierNumTel(String numTel)` qui permet de modifier le numéro de téléphone de l'employé, et qui affiche, par exemple, pour l'employé `Albert` : `"Le salarié Albert a pour numéro 012345678"`.

**Q 45.5** Trouver et expliquer les erreurs dans la méthode `main` ci-dessous :

```
1 public class TestPersonne {
2    public static void main(String[] args) {
3       Personne p = new Personne("Albert");
4       p.ajouterEnfant();
5       p.prime();
6       p.estEnL2();
7
8       Etudiant e = new Etudiant("Ahmed",null,"L2");
9       e.ajouterEnfant();
10      e.prime();
11      e.estEnL2();
12
13      Salarie s1 = new Salarie("Amelle");
14      Salarie s2 = new Salarie("Pauline","0122334455");
15      Salarie s3 = new Salarie("Yves","0123401234",2000);
16   }
17 }
```

## Exercice 46 – Botanique (héritage et redéfinition de méthodes)

**Q 46.1** Dessiner l'arbre d'héritage et dire ce qu'affiche le programme suivant :

```
1 public class Plante {
2    public String toString () { return "Je␣suis␣une␣Plante"; }
```

```
3 }
4 public class Arbre extends Plante { }
5 public class Fleur extends Plante {
6     public String toString () { return "Je␣suis␣une␣Fleur"; }
7 }
8 public class Marguerite extends Fleur {
9     public String toString () { return "Je␣suis␣une␣Marguerite"; }
10 }
11 public class Chene extends Arbre { }
12 public class Rose extends Fleur { }
13 public class MainPlante {
14     public static void main(String[] args) {
15       Plante p = new Plante(); System.out.println(p);
16       Arbre a = new Arbre();System.out.println(a);
17       Fleur f = new Fleur();System.out.println(f);
18       Marguerite m = new Marguerite();System.out.println(m);
19       Chene c = new Chene();System.out.println(c);
20       Rose r = new Rose();System.out.println(r);
21     }
22 }
```

**Q 46.2** En tirer des conclusions sur l'héritage et la redéfinition de méthode.

**Q 46.3** Qu'affiche le programme suivant ? Rappel : le corps de méthode appelé est celui de l'objet, et non pas celui du type de la variable qui référence l'objet.

```
1  Plante p2 = new Arbre(); System.out.println(p2);
2  Plante p3 = new Fleur(); System.out.println(p3);
3  Plante p4 = new Marguerite(); System.out.println(p4);
4  Plante p5 = new Rose(); System.out.println(p5);
5  Plante p6 = new Chene(); System.out.println(p6);
```

## Exercice 47 – Orchestre

On souhaite modéliser le déroulement d'un orchestre. Un orchestre est composé d'un ensemble d'instruments. On instanciera des guitares, pianos, trompettes.

**Q 47.1** Dessiner le diagramme de classes.

**Q 47.2** Écrire une classe `Instrument` contenant deux variables d'instance de type double pour stocker le poids et le prix de l'instrument, respectivement. Munir la classe d'un constructeur à deux paramètres pour initialiser les variables d'instance, ainsi que de la méthode toString(). Quelle est la particularité de la méthode toString() d'un point de vue de l'héritage ?

**Q 47.3** Écrire les classes `Piano`, `Guitare`, `Trompette`. Ces classes comporteront une méthode `jouer()` qui affichera, par exemple pour `Guitare` : `"La guitare joue"`.

**Q 47.4** Un orchestre sera composé d'un tableau d'instruments. Écrire la classe `Orchestre` correspondante, contenant une variable pour stocker le nombre d'instruments courant. Écrire une méthode `ajouterInstrument(Instrument i)` qui ajoute un instrument à l'orchestre lorque ceci est possible.

**Q 47.5** Ajouter à la classe `Orchestre` une méthode `jouer()` qui fait jouer l'ensemble des intruments le constituant. Quel est le problème dans le code actuel et comment remédier à ce problème ?

**Q 47.6** Écrire une classe `TestOrchestre` avec la méthode `main()` qui créer un orchestre composé d'une guitare, d'un piano et d'une trompette, et fait jouer cet orchestre.

**Q 47.7** Pour comprendre l'intérêt d'utiliser de l'héritage quand cela est possible, répondez à la question suivante : si l'on souhaite ajouter un nouvel instrument (*e.g.* batterie), quelle(s) classe(s) doit-on modifier ?

## Exercice 48 – Véhicules à moteurs

On considère un parc de véhicules. Chacun a un numéro d'identification (attribué automatiquement à l'aide d'un compteur statique, ce numéro ne doit pas pouvoir être modifié et doit être visible dans les classes filles), une marque (`String`) et une distance parcourue (initialisée à 0). Parmi eux on distingue les véhicules à moteur qui ont une capacité de réservoir et un niveau d'essence (initialisé à 0) et les véhicules sans moteur qui n'ont pas de caractéristique supplémentaire. Les vélos ont un nombre de vitesses, les voitures ont un nombre de places, et les camions ont un volume transporté.

**Q 48.1** Construire le graphe hiérarchique des classes décrites ci-dessus.

**Q 48.2** Écrire le code java des classes `Vehicule`, `AMoteur`, et `SansMoteur` avec tous les constructeurs nécessaires et les méthodes `toString()`. Rappel : en général, la méthode `toString()` des classes filles contient l'appel à la méthode `toString()` de la classe mère : `super.toString()`.

**Q 48.3** Écrire la classe `Velo` avec un consructeur, une méthode `toString()` qui retourne par exemple : `"Vélo 1 de marque MyVTT sans moteur 17 vitesses"` et une méthode `void transporter(String depart, String arrivee)` qui affiche par exemple `"Le vélo 1 se déplace de Paris à Lyon"`.

**Q 48.4** Écrire une méthode `rouler(double distance)` qui fait avancer de `distance` km un véhicule et qui affiche par exemple : `"Vélo 1 de marque MyVTT sans moteur 17 vitesses a roulé 10.0 km"`. A quel niveau de la hiérarchie faut-il l'écrire ?

**Q 48.5** Dans la méthode `main` d'une classe `TestVehicule`, créer un vélo de marque `"MyVTT"` avec 17 vitesses, et tester les méthodes de la classe `Vélo` (y compris la méthode héritée `rouler`).

**Q 48.6** Écrire les méthodes `void approvisionner(double nbLitres)`, et `boolean enPanne()` (en panne s'il n'y a plus d'essence). A quel niveau de la hiérarchie faut-il les écrire ?

**Q 48.7** Ecrire la classe `Voiture` avec un constructeur, une méthode `toString()` et une méthode `void transporter(int nbPers, int km)` qui affiche par exemple `"La voiture 2 transporte 5 personnes sur 200 km"` ou bien `"La voiture 2 n'a plus d'essence !"` suivant le cas. Ajouter une voiture dans le main et tester ses méthodes.

**Q 48.8** Écrire la classe `Camion` avec un constructeur, une méthode `toString()` et une méthode `void transporter(String materiau, int km)` qui affiche par exemple `"Le camion 3 n'a plus plus d'essence!"` ou bien `"Le camion 3 a transporté des tuiles sur 500 km"`.

**Q 48.9** Dans le main, déclarer un tableau de 3 véhicules, y ajouter le vélo, la voiture et un camion. Est-il possible de faire rouler 10 km tous les véhicules du tableau ? Si oui, faites-le.

**Q 48.10** Peut-on factoriser la déclaration de la méthode `transporter` ? Si oui, à quel niveau ?

# 7 Héritage et classe abstraite

### Exercice 49 – Chien et Mammifère (transtypage d'objet)

*Rappel de cours* : Le cast (conversion de type ou transtypage) consiste à forcer un changement de type si les types sont compatibles. Pour cela, il suffit de placer le type entre parenthèses devant l'expression à convertir. Attention : un cast ne change pas l'objet ou la variable sur laquelle il s'applique, mais cela change seulement la façon dont le compilateur considère l'expression castée.

Pour chaque ligne de la méthode `main` suivante, indiquez si il se produit une erreur à la compilation ? à l'exécution ? Expliquez chaque erreur.

```
1 public class Mammifere { }
2 public class Chien extends Mammifere {
3   public void aboyer() { System.out.println("Ouaff"); }
4   public static void main(String[] args) {
5     Chien c1 = new Chien();
```

```
6       Mammifere m1 = c1;
7       Chien c2 = (Chien) m1;
8       Chien c3 = m1;
9       Mammifere m2 = new Mammifere();
10      Chien c4 = (Chien) m2;
11      m1.aboyer();
12      ((Chien)m1).aboyer();
13    }
14 }
```

## Exercice 50 – Figures (méthode et classe abstraite)

Soit le programme Java constitué des classes suivantes :

```
1 public abstract class Shape {
2     protected double x, y ; // ancrage de la figure
3     public Shape() { x = 0 ; y = 0 ; }
4     public Shape(double x, double y) { this.x = x ; this.y = y ; }
5     public String toString() { return "Position␣:␣(" + x + "," + y + ")" ; }
6     public abstract double surface() ;
7 }
8
9 public class Circle extends Shape {
10    private double radius ;
11    public Circle() {
12      super();   // pas obligatoire (appel implicite) mais très recommandé
13      radius = 1 ;
14    }
15    public Circle(double x, double y, double r) {
16      super(x,y) ;
17      radius = r ;
18    }
19    public String toString() {
20      return super.toString() + "␣Rayon␣:␣" + radius ;
21    }
22 }
23
24 public class MainShape {
25    public static void main(String [] args) {
26      Circle c1,c2 ;
27      c1 = new Circle(1,1,3) ;
28      c2 = new Circle() ;
29      System.out.println(c1.toString() + "\n" + c2.toString());
30    }
31 }
```

**Q 50.1** A quels membres (variables d'instance et méthodes) de `Shape` la classe `Circle` a-t-elle accès ?

**Q 50.2** La compilation de la classe `Circle` échoue, expliquer pourquoi.

**Q 50.3** Pourquoi la méthode `surface()` a-t-elle était déclarée abstraite dans la classe `Shape` ? Ajouter une méthode `surface()` à la classe `Circle` et modifier en conséquence la méthode `toString`.

**Q 50.4** Créer une classe `Rectangle` qui hérite de `Shape`.

**Q 50.5** Donner le code d'un `main` qui instancie un tableau de `Shape`, le remplit avec différents types de forme puis calcule l'aire totale de la figure composite (sans prendre en compte les recouvrements).

## Exercice 51 – Ménagerie (tableaux, héritage, constructeur)

On veut gérer une ménagerie dont les animaux ont chacun un nom (`String`) et un âge (`int`). Parmi ceux-ci on distingue les animaux à pattes (variable `nbPattes`) et les animaux sans pattes. On s'intéresse uniquement aux vaches, boas, saumons, canards et mille-pattes.

**Q 51.1** Établir graphiquement la hiérarchie des classes ci-dessus. Déterminer celles qui peuvent être déclarées abstraites ?

**Q 51.2** Écrire la classe `Animal` avec deux constructeurs (un prenant en paramètre le nom et l'âge, l'autre prenant en paramètre le nom et qui fixe l'âge à 1 an), la méthode `toString`, une méthode vieillir qui fait vieillir l'animal d'une année, et une méthode `crier()` qui affichera le cri de l'animal. Peut-on écrire ici le corps de cette méthode ?

**Q 51.3** Écrire toutes les sous-classes de la classe `Animal` en définissant les méthodes `toString()` et les méthodes `crier()` qui affichent le cri de l'animal.

**Q 51.4** Écrire une classe `Menagerie` qui gère un tableau d'animaux, avec la méthode `void ajouter(Animal a)` qui ajoute un animal au tableau, et la méthode `toString()` qui rend la liste des animaux.
Aide : dans ce genre de problème (classe avec attribut de type tableau d'objets et ajout d'éléments dans le tableau), pensez à utiliser une variable `nbAnimaux` qui compte le nombre d'animaux actuellement présents dans la ménagerie.

**Q 51.5** Ajouter une méthode `void midi()` qui fait crier tous les animaux de cette ménagerie.

**Q 51.6** Écrire la méthode `vieillirTous()` qui fait vieillir d'un an tous les animaux de cette ménagerie.

**Q 51.7** Écrire la méthode `main` qui crée une ménagerie, la remplit d'animaux, les affiche avec leur âge, déclenche la méthode `midi()` et les fait vieillir d'un an.

## Exercice 52 – Figure2D (Extrait de l'examen de janvier 2009)

On veut écrire les classes correspondant à la hiérarchie ci-contre (le niveau d'indentation correspond au niveau de la hiérarchie) :

```
Figure (classe abstraite)
    |____Figure2D (classe abstraite)
            |____Rectangle
                    |____Carre
            |____Ellipse
                    |___Cercle
```

Ces classes devront respecter les principes suivants :

- Toutes les variables d'instance sont de type `double` et caractérisent uniquement la taille des objets, pas leur position.
- Chaque objet sera créé par un constructeur qui recevra les paramètres nécessaires (par exemple la longueur et la largeur d'un rectangle).
- Toutes les instances devront accepter les méthodes `surface()`.
- Toutes les instances d'objets de type 2D devront accepter la méthode `perimetre()`.

*Rappel sur les ellipses :* une ellipse est caractérisée par la longueur $a$ du demi-grand axe et la longueur $b$ du demi-petit axe. Sa surface est $\pi * a * b$ et son périmètre est $2\pi\sqrt{\frac{(a^2+b^2)}{2}}$.
*Rappel :* dans la classe `Math`, il existe la constante `Math.PI` et la méthode `Math.sqrt()` qui retourne la racine carrée d'un nombre (voir annexe page 63).

**Q 52.1** Écrire le diagramme de classe en indiquant les classes abstraites et les méthodes abstraites. Quelles sont les particularités d'une méthode abstraite et les conséquences pour la classe et les classes dérivées ?

**Q 52.2** Donner pour chacune des classes, en utilisant correctement les notions d'héritage et de classe abstraite :

- la définition de la classe,
- la déclaration des variables d'instance,
- le constructeur,
- les méthodes de la classe.

**Q 52.3** Écrire une méthode `main` dans une classe `TestFigure` qui stocke dans un tableau un objet de chacun des types précédemment créés, puis qui affiche la surface et le périmètre de chaque objet du tableau. Aide : quel doit être le type du tableau ?

## Exercice 53 – Retro engineering

Soit le programme principal suivant permettant d'effectuer des opérations mathématiques très simples dans un nouvel univers objet. Comme le précise le `main` suivant, une expression est soit une valeur réelle, soit une opération mathématique. Pour ne pas complexifier la situation, nous n'envisageons que des opérations réelles (sur des `double`).

```
1 public static void main(String args[]){
2         Expression v1=new Valeur(4.);
3         Expression v2=new Valeur(1.);
4         Expression v3=new Valeur(7.);
5         Expression v4=new Valeur(5.);
6         Expression v5=new Valeur(3.);
7         Expression v6=v5;
8         Operation p1=new Plus(v1,v2);
9         Operation m2=new Moins(v3,v4);
10        Operation mult=new Multiplie(p1,v5);
11        Operation p2=new Plus(v6,mult);
12        Operation d=new Divise(p2,m2);
13        System.out.println(d+"="+d.getVal());
14 }
```

**Q 53.1** Donner la hiérarchie des classes (avec les **signatures** de méthodes abstraites et concrètes et la signature du constructeur lorsqu'il est nécessaire) à définir pour que ce programme puisse compiler et s'exécuter.
**Attention :** on veut que la dernière ligne du `main` affiche le calcul à effectuer dans le détail (cf question suivante)

**Q 53.2** La hiérarchie de classes proposée définit une expression arithmétique qui peut être évaluée pour donner un résultat (méthode `getVal()`). Donner l'expression arithmétique (avec parenthèses) correspondant à l'objet `d` du programme donné ci-dessus.

**Q 53.3** Donner le code des classes nécessaires pour que le programme s'exécute et affiche la formule évaluée et son résultat en ligne 13 (le code des classes `Plus`, `Moins`, `Multiplie` et `Divise` étant très proche, on ne donnera le code que de `Divise`).

**Q 53.4** Donner le diagramme de l'état de la mémoire à la fin du programme (ligne 13).

**Q 53.5** On souhaite maintenant pouvoir modifier l'attribut d'un objet `Valeur`. On ajoute alors la fonction `void setVal(double v)` à la classe `Valeur` qui fixe à `v` l'attribut de la classe. Soit la ligne de code suivante :

```
1 v6.setValeur(4);
```

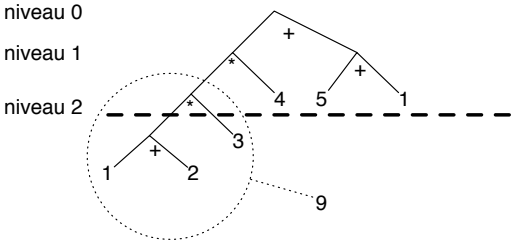
En l'état, le programme ne compile pas. Pourquoi ? Donner deux manières de remédier au problème. Discuter brièvement des avantages / inconvénients de ces deux manières de faire.

**Q 53.6** Quel problème survient dans l'exécution du programme si l'on remplace la ligne 5 par :

```
1 Expression v4=new Valeur(7);
```

**Q 53.7** Dire comment y remédier pour que le programme affiche le message d'erreur approprié et évite un arrêt brutal du programme. Décrire brièvement les méthodes à modifier et les éventuelles classes à créer.

**Q 53.8** En voyant une expression comme un arbre, on souhaite développer une méthode `simplifie(int profondeur)` permettant la simplification d'une expression à partir d'une profondeur donnée, avec `profondeur` un entier supérieur à 0. Lorsque la profondeur désirée est atteinte, cette simplification consiste à remplacer l'expression concernée par un objet `Valeur` de valeur équivalente.




Par exemple, l'objet `Expression` dont la formule est $((((1+2)*3)*4)+(5+1))$ se simplifie en $((9*4)+(5+1))$ par l'appel de `simplifie(2)`. Donner le code permettant cette simplification.

## Exercice 54 – `final` : les différentes utilisations

**Q 54.1** Questions de cours

**Q 54.1.1** A quoi sert un attribut `final` ? Où peut-il être initialisé ? Citer des cas d'utilisation.

**Q 54.1.2** Dans quel cas déclarer une méthode comme `final` ?

**Q 54.1.3** Dans quel cas déclarer une classe comme `final` ?

**Q 54.1.4** Etant donné les usages répertoriés ci-dessus, à quoi sert le mot clé `final` en général ?

**Q 54.2** Application sur la classe Point

```
1 public class Point {
2     private double x,y;
3     private static int cpt = 0;
4     private int id;
5
6     public Point(double x, double y) {
7         this.x = x;  this.y = y; id = cpt ++;
8     }
9     public double getX() { return x;}
10    public double getY() { return y;}
11    public String toString() {
12        return "Point [x=" + x + ", y=" + y + "]";
13    }
14    public void move(double dx, double dy){ x+=dx; y+=dy;}
```

**Q 54.2.1** Au niveau des attributs, serait-il intéressant d'ajouter le modifier `final` sur certains champs ? Pourquoi ?

**Q 54.2.2** A quelle condition pourrait-on mettre `x` et `y` en mode `final` ? Proposer une solution pour conserver les fonctionnalités de la classe.

**Q 54.2.3** Quelles fonctions pourraient être `final` ? Quel serait l'intérêt de la manipulation ?

**Q 54.2.4** Quel serait l'intérêt de déclarer la classe `final` ? Cela empêche-t-il tout enrichissement futur ?

**Q 54.2.5** Proposer un code pour la classe `PointNomme` (point ayant un attribut `nom`) après avoir déclaré Point en `final`.

## Quizz 13 – Classe et méthode abstraite

**QZ 13.1** Les instructions suivantes sont-elles correctes ? Expliquez.

```
1 public abstract class Z {}
2 public class TestQuizzAbstract {
3         public static void main(String [] args) {
4                 Z z=new Z();
5         }
6 }
```

**QZ 13.2** Les instructions suivantes sont-elles correctes ? Expliquez chaque erreur.

```
1 public class Z {
2         public abstract void f();
3         public abstract void g() { } ;
4         public void h();
5 }
```

**QZ 13.3** Les instructions suivantes sont-elles correctes ? Expliquez et proposez deux solutions.

```
1 public abstract class A {
2         public abstract void f();
3 }
4 public class B extends A {}
```

### Quizz 14 – Vocabulaire sur l'héritage

En utilisant quelques verbes de l'ensemble ci-après, écrire trois courtes phrases caractérisant l'héritage : implémenter, instancier, importer, réemployer, ajouter, encapsuler, étendre, spécifier, redéfinir.
L'héritage permet de : ...

# 8 Héritage et interface

### Exercice 55 – Redéfinition de la méthode `equals`

Soit la classe `Point` ci-dessous :

```
1 public class Point {
2   private int x, y; // coordonnees
3   public Point(int a, int b) {x=a; y=b;}
4   public Point() {x=0; y=0;}
5   public Point (Point p) { x=p.x; y=p.y;}
6
7   public static void main(String [] args) {
8     Point p1 = new Point(5,2);
9     Point p2 = new Point(5,2);
10    Point p3 = p1;
11    Point p4 = new Point(1,1);
12    System.out.println("p1=p2 : "+ p1.equals(p2));
13    System.out.println("p1=p3 : "+ p1.equals(p3));
14    System.out.println("p1=p4 : "+ p1.equals(p4));
15  }
16 }
```

**Q 55.1** Qu'affiche l'exécution du main ?

**Q 55.2** Rédéfinir la méthode `boolean equals(Object ob)` de la classe `Object` dans la classe `Point`, de façon qu'elle teste l'égalité des coordonnées et non des références. Les instructions de test sont fournies dans la méthode `main`.

**Q 55.3** Que se passe-t-il si dans la méthode `main`, on rajoute à la suite les instructions suivantes ? Comment résoudre le problème rencontré ?

```
1 String s1=new String("Bonjour");
2 System.out.println("p1=s1 : "+ p1.equals(s1));
```

### Exercice 56 – Interface `Submarine`

Soient les trois classes suivantes :

```
public abstract class Animal {}
public abstract class Mammifere extends Animal {}
public class Chat extends Mammifere {}
```

On considère la propriété de pouvoir se déplacer sous-l'eau. On suppose que les poissons, les dauphins (mammifères) et les bateaux sous-marins ont la propriété de pouvoir se déplacer sous l'eau. Les merlus sont des poissons.

**Q 56.1** Dessiner le diagramme de classe pour les classes `Animal`, `Chat`, `Dauphin`, `Mammifere`, `Merlu`, `Poisson`, `SousMarin` et l'interface `Submarine`.

**Q 56.2** Écrire l'interface `Submarine` qui contient une méthode `seDeplacer`.

**Q 56.3** Les poissons forment une famille d'animaux qui peuvent se déplacer sous l'eau. Écrire la classe `Poisson`.

**Q 56.4** Soit la classe suivante : `public class Merlu extends Poisson {}`. Un merlu a-t-il la propriété de nager sous l'eau ?

L'écholocation consiste à envoyer des sons et à écouter leur écho pour localiser des éléments d'un environnement. Soit l'interface `Echolocation` suivante :

```
public interface Echolocation {
        public void envoyerSon();
        public void ecouterSon();
}
```

**Q 56.5** Un dauphin est un mammifère qui peut se déplacer sous l'eau et faire de l'écholocation. Écrire la classe `Dauphin`.

**Q 56.6** Un sous-marin peut se déplacer sous l'eau et faire de l'écholocation. Écrire la classe `SousMarin`.

**Q 56.7** Écrire une classe `Ocean` qui contient un attribut de type `ArrayList` de `Submarine`, qui contient une méthode pour ajouter un élément dans la liste et une autre méthode pour déplacer tous les éléments.

**Q 56.8** On suppose que l'on est dans une méthode `main` d'une classe `TestSubmarine`, créer un océan, y ajouter un merlu, un dauphin et un sous-marin, puis déplacer-les dans l'océan.

**Q 56.9** Peut-on ajouter un chat dans l'océan ?

**Q 56.10** On suppose qu'une certaine espèce de chats très particuliers aime nager sous-l'eau dans l'océan. Peut-on ajouter une classe `ChatSub` correspondante à cette espèce ? Faut-il modifier la classe `Ocean` pour cela ?

## Exercice 57 – Interface `Motorise`

On suppose que tous les véhicules motorisés (voiture, moto...) doivent faire le plein régulièrement à la station service.

```
1 public abstract class Vehicule {
2    public abstract void rouler();
3 }
4 public class Velo extends Vehicule {
5    public void rouler() { System.out.println("Le vélo roule"); }
6 }
7 public class Voiture extends Vehicule {
8    public void rouler() { System.out.println("La voiture roule"); }
9    public void faireLePlein() {System.out.println("Le plein de la voiture est fait");};
10 }
11 public class Moto extends Vehicule {
12    public void rouler() { System.out.println("La moto roule"); }
13    public void faireLePlein() {System.out.println("Le plein de la moto est fait");};
14 }
15 public class StationService {
16    public void remplirReservoir(Vehicule v) {
17       if ( v instanceof Voiture ) ((Voiture)v).faireLePlein();
18       if ( v instanceof Moto ) ((Moto)v).faireLePlein();
19       else System.out.println("Inutile pas de réservoir");
20    }
21 }
```

**Q 57.1** Expliquez pourquoi la méthode `remplirReservoir` n'est pas bien programmée (aide : cette méthode est-elle toujours correcte, si on ajoute une classe `Camion` ?). Proposez une solution sans utiliser d'interface Java.

**Q 57.2** Créer un tableau avec un vélo, une voiture et une moto, et faire le plein de tous ces éléments à la station

service.

**Q 57.3** Maintenant, on suppose qu'il existe des engins qui ne sont pas des véhicules (par exemple, tondeuse, tronçonneuse...), mais dont on veut quand même pouvoir faire le plein à la station service. La solution précédente fonctionne-t-elle ? Proposez une autre solution.

**Q 57.4** Créer un tableau avec un vélo, une voiture et une tondeuse et faire le plein de tous éléments du tableau à la station service.

## Exercice 58 – Roulant, Volant, Flottant (Héritage d'interfaces)

Nous souhaitons gérer une grande liste de véhicule à moteur, chacun d'eux ayant comme propriété de pouvoir : `demarrer` et s'`arreter`. Pour clarifier l'organisation des véhicules, nous introduisons une hiérarchie incluant les `Roulant` (possédant une méthode `void rouler()`), les `Volant` (méthode `voler()`) et les `Flottant` (méthode `naviguer()`).

**Q 58.1** Donner la hiérarchie d'interface à créer.

**Q 58.2** Donner la signature de la classe `Voiture` et les méthodes à coder impérativement.

**Q 58.3** Donner la signature de la classe `Hydravion` et les méthodes à coder impérativement.

## Exercice 59 – Promenade en forêt (`interface`, `ArrayList`)

Une forêt peut contenir différents objets. Pour l'instant, on considère seulement les arbres et les champignons. Les champignons sont ramassables. On peut ramasser les champignons et les mettre dans un panier tant que ce qui a été ramassé n'est pas trop lourd.

**Q 59.1** Écrire une classe `Foret` avec les attributs et méthodes suivantes :

- attribut `terrain` : un tableau d'`Object` à deux dimensions représentant le terrain de la forêt. Chaque case peut donc contenir au plus un objet.
- constructeur prenant en paramètre la taille du terrain. Pour simplifier, on peut supposer que le terrain est carré.
- `boolean placer(Object obj)` qui calcule aléatoirement des coordonnées (x,y) dans le terrain et essaye de placer l'objet `obj` à ces coordonnées. Cette méthode retourne vrai si l'objet a pu être placé dans le terrain, faux si la case était déjà occupée.
- méthode `toString()` qui retourne une chaîne de caractères représentant le terrain. Chaque objet est représenté par le premier caractère de la chaîne retournée par sa méthode `toString()`. Par exemple : 'P' pour l'arbre appelé "Pin" et 'C' pour le champignon appelé "Cèpe". Aide : utilisez la méthode `char charAt(int index)` de la classe `String`. Voir ci-contre un exemple de représentation d'un terrain de taille `5x5`.

```
|   C     P |
| C   P C   |
|       C   |
|   C   C P |
| C         |
```

**Q 59.2** Écrire une classe `Arbre` qui possède un seul attribut le nom de l'arbre, et dont la méthode `toString` retourne ce nom. Écrire aussi la classe `Champignon` qui possède les attributs `nom` et `poids`. Le poids du champignon de type `double` est initialisé aléatoirement entre 0 et 3kg. Cette classe contient aussi une méthode `getPoids()` et une méthode `toString()` qui retourne par exemple `"Cèpe 1.4kg"`.

**Q 59.3** Dans une classe `TestForet`, créez une forêt, ajoutez-y si possible 10 objets (aléatoirement environ 30% de pins et 70% de cèpes), puis affichez la forêt.

**Q 59.4** On veut ramasser tous les champignons de la forêt. Écrire dans la classe `Foret` une méthode dont la signature est : `ArrayList<Champignon> ramasserChampignons()` qui crée une liste de champignons, parcourt tout le terrain pour ramasser seulement les champignons, puis retourne la liste.

**Q 59.5** Maintenant, pour mieux modéliser une forêt, on veut pouvoir ajouter de nombreuses autres classes, certaines correspondent à des objets qui sont ramassables (baies, bois morts, pierres, fleurs...), d'autres non (plantes, rivières, rochers...). Proposez une façon de modéliser le problème en Java afin de pouvoir écrire dans `Foret` une méthode `ramasserTout` qui ramasse, non pas seulement les champignons, mais tous les objets ramassables sachant que ces objets pèsent un certain poids. Écrire les instructions nécessaires, dont la méthode `ramasserTout`.

**Q 59.6** Certains objets de la forêt peuvent être toxiques. Par exemple, des baies toxiques, des champignons toxiques... Proposer une façon de modéliser ce problème en Java. Écrire la classe `ChampignonToxique`, puis ajouter quelques

amanites (nom d'un champignon toxique) dans la forêt.

**Q 59.7** Un panier peut contenir au maximum `poidsMax` kilos. Écrire une classe `Panier` qui hérite d'une `ArrayList` de ramassables et qui contient :

- un attribut `poidsMax`
- un constructeur avec en paramètre le poids maximal que peut contenir le panier
- une méthode `getPoids()` qui retourne la somme totale des poids des objets contenus dans le panier
- une redéfinition de la méthode `add` de `ArrayList` qui ajoute dans la liste l'objet ramassable en paramètre seulement si le poids maximal du panier n'est pas dépassé. Cette méthode affiche, par exemple, `"Cèpe 1.4kg est ajouté au panier"` ou `"Cèpe 1.4kg n'est pas ajouté au panier"` en fonction du cas. Elle retourne vrai si l'objet a été ajouté au panier, faux sinon. Remarque : vous pouvez demander au compilateur de vérifier que c'est bien une redéfinition en utilisant `@Override` devant la signature de la méthode.
- une méthode `compterToxiques` qui compte le nombre d'objets dans le panier qui sont toxiques
- une méthode `toString()` qui retourne par exemple `"Panier contenant 5 choses, dont 2 toxiques (7,1kg sur 8,0kg)"`.

**Q 59.8** Dans la classe `Foret`, écrire une méthode `void ramasser(Panier p)` qui comme précédemment parcourt toute la forêt pour ramasser des objets ramassables, mais si un objet est trop lourd pour être ajouté dans le panier, il n'est pas ramassé. Créer un panier de 8kg, ramasser avec le panier des objets dans la forêt, puis afficher le panier.

## Exercice 60 – Interface `Reversible`

Une interface correspond à une propriété. Nous envisageons dans cet exercice la propriété de reversibilité. Pour une chaine de caractères, il s'agit de pouvoir la lire à l'envers lorsqu'on le souhaite, pour un tableau, de prendre les éléments dans l'ordre opposé. Nous choisissons arbitrairement de définir cette propriété sans argument et sans retour : il s'agit juste de modifier l'élément invoquant la méthode.

**Q 60.1** Donner le code de l'interface `Reversible`.

**Q 60.2** Donner le code de la classe `StringReversible`. Nous rappelons que la classe String est immutable et `final`. Vous aurez besoin des méthodes de `String` suivantes : `int length()` qui renvoie la longueur de la chaine et `char charAt(int i)` qui renvoie le caractère à la position `i`.

**Q 60.3** Donner deux exemples d'utilisation dans un `main`. Est-il possible de déclarer une variable de type `Reversible` ?

**Q 60.4** Nous souhaitons maintenant créer une structure de type `ArrayList` réversible. Donner le code étendant l'`ArrayList<Object>`, ajoutant les méthodes nécessaire (dont `toString()` et surchargeant la méthode `get`.
NB : ajouter un attribut booléen indiquant si la structure est renversée ou pas.

**Q 60.5** Ajouter quelques lignes de code pour rendre la réversibilité récursive quand c'est possible dans la structure précédente. Par exemple, quand la liste contient des `StringReversible`, nous souhaitons renverser la liste ET renverser les éléments de la liste si c'est possible.

**Q 60.6 (Option)** Proposer une seconde implémentation de la structure de données récursive basée sur la composition et non plus sur l'héritage (attribut `ArrayList` au lieu de `extends ArrayList`)

## Exercice 61 – Interface `Comparable`

Nous nous plaçons maintenant comme utilisateur d'un cadre défini pour les interfaces. Nous avons besoin de trier une liste de vecteurs en fonction de leur norme. Nous disposons de la classe de base :

```
1 public class Vecteur {
2     private double x,y;
3     public Vecteur(double x, double y){
4         this.x = x;
5         this.y = y;
6     }
7     public double norme(){return Math.sqrt(x*x+y*y);}
8 }
```

La Javadoc indique : (1) dans la classe `Collections` :

```
1 static <T extends Comparable<? super T>> void sort(List<T> list)
2 // Sorts the specified list into ascending order, according to the natural ordering of its
      elements.
3 static <T> void sort(List<T> list, Comparator<? super T> c)
4 // Sorts the specified list according to the order induced by the specified comparator.
```

(2) Interface Comparable

```
1 int      compareTo(T o) // Compares this object with the specified object for order.
2 // si x<y alors, x.compareTo(y) < 0
3 // si x.equals(y) alors  x.compareTo(y) == 0
4 // sinon x.compareTo(y) > 0
```

(3) Interface Comparator

**Q 61.1** Indiquer les modifications à effectuer dans la classe `Vecteur` pour utiliser `Comparable`

**Q 61.2** Donner le code d'un main effectuant le tri d'une liste de `Vecteur` générée aléatoirement par rapport à leurs normes.

**Q 61.3** Donner la procédure et le code pour utiliser un `Comparator`. Quel est l'avantage de cette approche ?

## Exercice 62 – Attribut de type `ArrayList`

Soient les classes suivantes :

```
1 import java.util.ArrayList;
2
3 public abstract class A {
4         public abstract void afficher();
5 }
6 public class B extends A {
7         public void afficher() {
8                 System.out.println("je suis un B");
9         }
10        public void methodeDeB() {
11                System.out.println("Methode de B");
12        }
13 }
14 public class C extends A {
15        public void afficher() {
16                System.out.println("je suis un C");
17        }
18        public void methodeDeC() {
19                System.out.println("Methode de C");
20        }
21 }
```

On souhaite créer une classe qui gère une liste d'objets dont la classe mère est `A`.

**Q 62.1** Expliquez la ligne 1.

**Q 62.2** Écrire la classe `ListeDeA` qui possède une seule variable d'instance appelée `liste` qui est de type `ArrayList` de `A` (voir la documentation de la classe `ArrayList` à la page 64). Ajoutez-y un constructeur qui prend en paramètre le nombre `n` d'objets à créer à l'initialisation de la liste. Ce constructeur crée aléatoirement 50% d'objets de type `B` et 50% d'objets de type `C` et les ajoute à la liste.

**Q 62.3** Ajoutez à la classe `ListeDeA` une méthode `afficherListe()` qui appelle la méthode `afficher()` de chacun des objets de la liste. Utilisez cette méthode dans une méthode `main`.

**Q 62.4** Ajoutez à la classe `ListeDeA` une méthode `afficherMethode()` qui pour chaque objet de la liste appelle la méthode `methodeB()` si cet objet est un objet de type `B`, et appelle la méthode `methodeC()` si cet objet est un objet de type `C`. Utilisez cette méthode dans une méthode `main`.

### Exercice 63 – Compagnie de chemin de fer (`ArrayList`, `instanceof`)

Une compagnie de chemin de fer veut gérer la formation de ses trains, à partir de la description suivante. Un train est formé d'éléments de train. Un élément de train possède un numéro de série et une marque. Un élément de train est soit une motrice, soit un wagon. Une motrice a une puissance. Un wagon a un nombre de portes. Un wagon peut être soit un wagon voyageurs, auquel cas il possède un nombre de places, soit un wagon de marchandise, auquel cas il possède un poids maximum représentant la charge maximale qu'il peut transporter.

**Q 63.1** Dessiner la hiérarchie des classes `Train`, `ElemTrain`, `Motrice`, `Wagon`, `WVoyageur` et `WMarchandise`. **Q 63.2**

Écrire les classes `ElemTrain` (abstraite), `Wagon` (abstraite), `Motrice`, `WVoyageur` et `WMarchandise` avec au moins un constructeur avec paramètres et une redéfinition de la méthode `public String toString()` qui retourne pour un élément son type et son numéro de série, par exemple : « Wagon Marchandise 10236 (Alstom)». Pour plus de propreté, vous utiliserez un compteur `static` pour générer les numéros de série commençant à 10000.

**Q 63.3** Un Train possède une motrice et une une suite de wagons (on gèrera cette suite obligatoirement par la classe `ArrayList` (voir la documentation page 64). Écrire la classe `Train` avec au minimum un constructeur a un paramètre de type `Motrice` qui construit un train réduit à cette motrice, et ayant donc un ensemble vide de wagons.

**Q 63.4** Ajouter une méthode `void ajoute(Wagon w)` qui ajoute un wagon au vecteur de wagons du train.

**Q 63.5** Redéfinir la méthode `public String toString()` qui retourne la composition de ce train.

**Q 63.6** Écrire une méthode `poids()` qui retourne le poids maximum de marchandise que peut transporter le train. *Indication* : On peut utiliser l'opérateur `instanceof` qui rend vrai si et seulement si un objet est instance d'une classe. Exemple d'utilisation : `if (a instanceof A)...`

**Q 63.7** Écrire la méthode principale `public static void main(String[] args)` dans une classe `MainTrain`. Cette méthode crée une motrice, des wagons de voyageur et des wagons de marchandise, crée un train formé de ces éléments, affiche la composition de ce train ainsi que le poids transporté.

# 9 Héritage et liaison dynamique

### Exercice 64 – Des fourmis à tous les étages

```
1 public class Fourmi{
2    protected String nom;
3    public Fourmi(String nom){ this.nom = nom; }
4 }
5
6 public class Ouvriere extends Fourmi{
7    public Ouvriere(String nom){ super(nom); }
8 }
9
10 public class Reine extends Fourmi{
11    private int cpt;
12    public Reine(String nom){ super(nom); cpt=0; }
13    public Fourmi engendrer(){
14       cpt ++;
15       return new Ouvriere(nom+cpt);
16    }
17 }
```

**Q 64.1** Vrai/Faux général sur l'héritage. Parmi les instructions suivantes, identifier celles qui sont incorrectes et expliquer succinctement le problème (en précisant s'il survient au niveau de la compilation ou de l'exécution). Donner le nom des fourmis qui ont effectivement été engendrées par une reine.

```
Fourmi f1    = new Fourmi("f1");
Fourmi f2    = new Ouvriere("ouv1");
Ouvriere f3 = new Ouvriere("ouv2");
Fourmi f4    = new Reine("majeste1");
Ouvriere f5 = new Reine("majeste2");
Reine f6     = new Reine("majeste3");
```

```
Fourmi[] fourmilliere = new Fourmi[100];
fourmilliere[0]= f4.engendrer();
fourmilliere[1]= f5.engendrer();
fourmilliere[2]= f6.engendrer();
fourmilliere[3]= ((Reine) f2).engendrer();
fourmilliere[4]= ((Reine) f4).engendrer();
fourmilliere[5]= ((Reine) f6).engendrer();
```

**Q 64.2** Sélection de méthodes. On souhaite maintenant nourrir nos fourmis... en fonction de leur hiérarchie. Nous introduisons à cet effet les classes suivantes :

```
public class Nourriture{
   private String description;

   public Nourriture(String description){   this.description = description;  }
   public String toString(){   return description;  }
}

public class GeleeRoyale extends Nourriture{
   public GeleeRoyale(){  super("gelee pour la reine"); }
}

// et modification des classes existantes:
public class Fourmi{
...
   public void manger(Nourriture n){   System.out.println(nom+" mange "+n);  }
}
public class Reine extends Fourmi{
...
   public void manger(GeleeRoyale g){
      System.out.println(nom+ " (Reine) mange de "+g);
   }
}
```

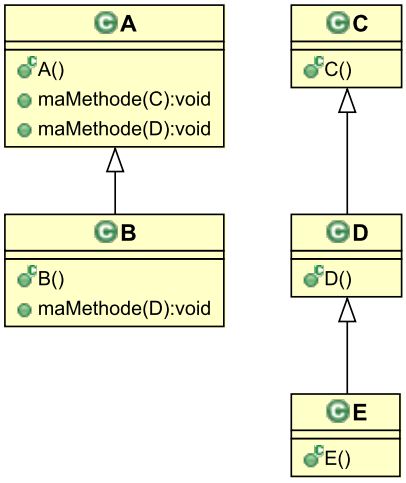
Donner les affichages lors de l'exécution du code suivant :

```
Reine r1 =  new Reine("majeste1");
Fourmi r2 =  new Reine("majeste2");
r1.manger(new Nourriture("un peu de sucre"));
r1.manger(new GeleeRoyale());
r2.manger(new Nourriture("un peu de viande"));
r2.manger(new GeleeRoyale());
```

## Exercice 65 – Sélection de méthode

Soit une hiérarchie de classes (figure ci-dessous).

**Q 65.1** Pour chaque ligne de code appelant `maMethode`, dire quelle méthode est effectivement appelée.




```
A a = new A();
B b = new B();
A ab = new B();

C c = new C();
D d = new D();
E e = new E();
C cd = new D();
```

```
a.maMethode(c);
a.maMethode(d);
a.maMethode(cd);
a.maMethode((D) cd);
a.maMethode(e);

b.maMethode(c);
b.maMethode(d);
b.maMethode(cd);
b.maMethode((D) cd);
b.maMethode(e);

ab.maMethode(c);
ab.maMethode(d);
ab.maMethode(cd);
ab.maMethode((D) cd);
ab.maMethode(e);
```

**Q 65.2** Conversions implicites (ou pas)

On envisage 3 ajouts de méthodes :

**Cas 1 :**

```
// dans A
public void meth(double d)
// rien dans B
```

**Cas 2 :**

```
// dans A
public void meth(int i)
// dans B
public void meth(double d)
```

**Cas 3 :**

```
// dans A
public void meth(int i)
// rien dans B
```

Le code à exécuter est maintenant le suivant :

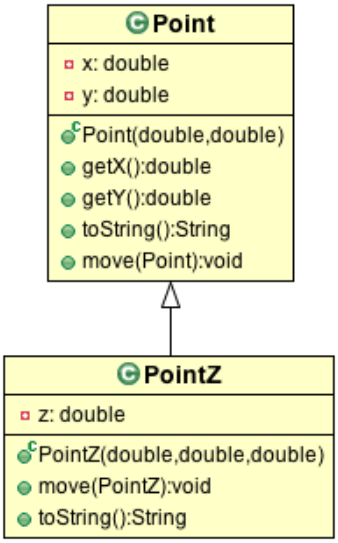
```
a.meth(2);
a.meth(2.);
b.meth(2);
b.meth(2.);
ab.meth(2);
ab.meth(2.);
```

En envisageant les 3 cas ci-dessus :

- Quelles sont les lignes posant des problèmes de compilation ?
- Quelles sont les méthodes sélectionnées (pour le cas 2) ?

## Exercice 66 – Redéfinition piégeuse

Soit la structure hiérarchique décrite dans le schéma UML ci-dessous :




```
1 public class Point {
2     private double x,y;
3     public Point(double x, double y) {
4         this.x = x;     this.y = y;
5     }
6     public double getX() { return x; }
7     public double getY() { return y; }
8     public String toString() {return "["+ x + "␣" + y +"]";}
9     public void move(Point p){ x+=p.x; y+=p.y; }
10 }
11 ///
12 public class PointZ extends Point {
13     private double z;
14     public PointZ(double x, double y, double z) {
15         super(x, y);
16         this.z = z;
17     }
18     public void move(PointZ p){
19         super.move(p);
20         z += p.z;
21     }
22     public String toString(){
23         return "["+getX()+"␣"+getY()+"␣"+z+"]";
24     }
25 }
```

**Q 66.1** Pourquoi cette hiérarchie de classe est-elle discutable ?

**Q 66.2 Syntaxe** : les lignes 15, 19 et 23 sont-elles correctes ? Sinon, proposez des modifications. En ligne 23, peut-on utiliser directement `x` et `y` sans passer par les accesseurs ? Pourquoi ?

**Q 66.3** Que pensez-vous du programme suivant ?

```
1 Point p = new Point(1,2);
2 Point p3d = new PointZ(1,2,3);
3 PointZ depl = new PointZ(1,1,1);  // deplacement a effectuer
4
5 System.out.println(p); // affichage avant modif
6 System.out.println(p3d);
7 p.move(depl); // modif
8 p3d.move(depl); // modif
9 System.out.println(p); // affichage apres modif
10 System.out.println(p3d);
```

**Q 66.3.1** Qu'est-ce qui s'affiche ?

**Q 66.3.2** Est-ce que ça vous semble logique ?

**Q 66.3.3** Expliquer en détail ce qui s'est passé au niveau de la compilation et de l'exécution.

## Exercice 67 – Documentation Java

*Rappel : Java est fourni avec un ensemble de classes. Par exemple, les classes `String`, `Math`, `System`. Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles appelés packages. Cet exercice a pour but de vous familiariser avec la documentation fournie avec Java, ainsi qu'avec les packages.*
Allez sur le site de l'UE, puis cherchez le lien vers la "Documentation Java".

**Q 67.1** Recherchez la classe `Random`. Combien a-t-elle de constructeurs ? Combien a-t-elle de méthodes ? A quel package appartient cette classe `Random` ? La classe `Math` appartient-elle au même package que la classe `Random` ? Aide : les packages sont écrits tout en minuscule.

**Q 67.2** Recherchez la classe `ArrayList`. D'après la documentation, combien a-t-elle de champs ? Combien a-t-elle

de constructeurs ? Combien environ a-t-elle de méthodes ? De quelles classes hérite-t-elle ? A quel package appartient cette classe `ArrayList` ?

**Q 67.3** Il est possible de créer une documentation pour les classes que vous créez. Pour cela, il faut utiliser la commande javadoc. Récupérez sur le site web de l'UE le fichier `Clavier.java`. Placez ce fichier dans un répertoire vide, puis tapez la commande : javadoc `Clavier.java`, puis : `firefox index.html` Comparez les commentaires du fichier `Clavier.java` et la page web affichée.

## Exercice 68 – Package Java

*Rappel : pour qu'une classe appartienne à un package, il suffit de mettre l'instruction :* `package nomdupackage;` *au début du fichier contenant la classe. Si l'on souhaite utiliser une classe d'un package dans une classe d'un autre package, il faut importer la classe :* `import nomdupackage.NomDeLaClasse;`

**Q 68.1** Créez 3 classes A, B et C chacune dans un fichier différent. Déclarez ces classes public. Mettez la classe `A` dans le package `pack1` et les classes B et C dans le package `pack2`. Ajoutez rapidement une méthode avec des commentaires à chaque classe (pour cela, il faut mettre les commentaires entre `/** ... */` avant le nom de la méthode ou de la classe). Générez une (et une seule) documentation pour ces 3 classes.

**Q 68.2** Dans la méthode `main` de la classe `B`, créez un objet de la classe `A`. Quelle instruction faut-il ajouter pour pouvoir utiliser la classe `A` ?

**Q 68.3** Afin d'organiser les fichiers `.class` créés à la compilation, ces fichiers vont être placés automatiquement dans des répertoires dont le nom est le nom du package. Pour cela, il faut utiliser l'option `-d chemin` de la commande `javac`. Quelles sont les lignes de commandes à taper si on veut que tous les fichiers `.class` se trouvent organisés dans le répertoire `build/` du répertoire courant ? Quels sont les répertoires et les fichiers créés ? Quelle est la commande pour exécuter le main de la classe `B` ?

## Exercice 69 – Visibilité et package

*Rappel :* En java, il existe 3 modificateurs de visibilité : `private`, `protected` et `public`. Lorsqu'il n'y a pas de modificateur, on dit que la visibilité est la visibilité par défaut.

Une classe est :

- soit `public` : elle est alors visible de partout.
- soit a la visibilité par défaut (sans modificateur) : elle n'est alors visible que dans son propre paquetage.

Si un champ d'une classe `A` :

- est `private`, il est accessible uniquement depuis sa propre classe ;
- est sans modificateur, il est accessible de partout dans le paquetage de `A`, mais de nulle part ailleurs ;
- est `protected`, il est accessible de partout dans le paquetage de `A` et, si `A` est publique, dans les classes héritant de `A` dans d'autres paquetages ;
- est `public`, il est accessible de partout dans le paquetage de `A` et, si `A` est publique, de partout ailleurs.

On considère les classes `A`, `B`, `C` qui sont dans le package `abc`, et les classes `D` et `E` qui sont dans le package `de`. Les classes `B` et `D` héritent de la classe `A`. On donne la classe `A` suivante :

```
1 package abc;
2 public class A {
3         private int champPrive;
4         int champSansModificateur;
5         protected int champProtected;
6         public int champPublique;
7 }
```

**Q 69.1** Donner la déclaration des classes `B`, `C`, `D` et `E`, et faire un schéma.

**Q 69.2** Compléter le tableau ci-dessous en cochant les cases pour lesquelles les variables d'instance de la classe `A` sont visibles.

| | Classe A | Classe B | Classe C | Classe D | Classe E |
|---|---|---|---|---|---|
| champPrive | | | | | |
| champSansModifieur | | | | | |
| champProtege | | | | | |
| champPublic | | | | | |

**Q 69.3** Si la classe A n'était pas déclarée public, est-ce que cela change la visibilité des variables ?

## Quizz 15 – Héritage et liaison dynamique

Soient les 4 classes suivantes :

```
1 public class Animal {
2         public void f() { }
3         public String toString() {return "Animal";}
4 }
5 public class Poisson extends Animal {
6         public void g() { }
7         public String toString() {return "Poisson";}
8 }
9 public class Cheval extends Animal { }
10 public class Zoo { }
```

et les déclarations suivantes :

```
Animal a1=new Animal(); Poisson p1=new Poisson(); Cheval c1=new Cheval(); Zoo z1=new Zoo();
```

**QZ 15.1** Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.

`a1.f();` `p1.f();` `a1.g();` `p1.g();`

**QZ 15.2** Que retournent les instructions suivantes ?

`a1.toString()` `p1.toString()` `c1.toString()` `z1.toString()`

**QZ 15.3** Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? à l'exécution ? Expliquez.

- `Animal a2=p1;`
- `Poisson p2=a1;`
- `Poisson p3=(Poisson)a1;`
- `Poisson p4=a2;`
- `Poisson p5=(Poisson)a2;`
- `a2.g();`

# 10 Exceptions

*Rappel :* Les exceptions sont un mécanisme de gestion des erreurs. Il existe 3 catégories d'exceptions : les exceptions qui étendent la classe `Exception` qui doivent obligatoirement être gérées par le programme, les exceptions qui étendent la classe `RunTimeException` qui peuvent être gérées par le programme, et les erreurs critiques qui étendent la classe `Error` qui ne sont pas censées être gérées en temps normal.
Toute instance de la classe `Exception` doit obligatoirement être capturée ou bien signalée comme étant propagée par toute méthode susceptible de la lever.

- Pour capturer une exception :

```
1 try {
2     instructions qui peuvent lever une exception
3 } catch (MonException me) {
4     System.out.println(me.getMessage());  // affiche le message de l'exception
5 } catch (AutreException ae) {
6     System.out.println(ae.toString());    // affiche "NomClasse:getMessage()"
7 } finally {
8     instructions toujours executees
9 }
```

- Pour signaler une erreur, on va lever / lancer une exception, pour cela il faut créer un nouvel objet : `throw new MonException();`
- Pour définir un nouveau type d'exception, il faut écrire une classe qui hérite de la classe `Exception` : `public class MonException extends Exception {...}`
- Pour déléguer / transmettre / propager une exception pour qu'elle soit capturée par une autre méthode : `public void maMethode () throws MonException {...}`

## Exercice 70 – Trop lourd pour ma balance ! (capture d'une exception)

Une balance est capable de peser un poids sauf si ce poids dépasse un poids maximal. Pour gérer le problème de dépassement de la capacité de la balance, on va utiliser le mécanisme des exceptions. Soient les classes suivantes qui compilent sans erreur :

```
1 public class TropLourdException extends Exception {
2         public TropLourdException() {
3                 super("Trop␣lourd␣!");
4         }
5 }
6 public class Balance {
7         private static final int MAX=150;
8         public void peser (int poids) throws TropLourdException {
9                 if (poids >= MAX) {
10                        throw new TropLourdException();
11                }
12                System.out.println("Poids␣:␣"+poids+"␣kg");
13        }
14 }
```

**Q 70.1** Expliquez brièvement ce que fait la ligne 10.

**Q 70.2** Le programme suivant compile-t-il ? Si oui, provoque-il une erreur à l'exécution ? Expliquez. Si non, expliquez pourquoi.

```
15 public class TestBalance {
16        public static void main(String [] args) {
17                Balance b1=new Balance();
18                b1.peser(50);
19                System.out.println("FIN");
20        }
21 }
```

**Q 70.3** En ajoutant seulement des instructions dans la méthode `main`, proposez deux solutions qui compilent et s'exécutent, et donnez l'affichage obtenu. Aide : le message d'erreur affiché par le compilateur indique les deux solutions possibles :

```
error: unreported exception TropLourdException; must be caught or declared to be thrown
```

**Q 70.4** On modifie maintenant l'instruction `b1.peser(50)` en `b1.peser(200)`. Quel est l'affichage obtenu pour les deux solutions proposées à la question précédente ? Quelle solution est la plus adaptée dans cette application qui correspond à une balance qui pèse des poids ?

**Q 70.5** On modifie `TropLourdException` en remplaçant `extends Exception` par `extends RuntimeException`. Qu'est-ce que cela change ?

## Exercice 71 – Capture dans le main d'une exception prédéfinie (`try catch`)

**Q 71.1** Soit classe `TestAttrapePas0` ci-dessous. Que se passe-t-il lors de l'exécution ?

```
1 public class TestAttrapePas0{
2     public static void main(String[] args){
```

```
3         int [] tab= {1,2,3,4,5};
4         for (int i=0; i<15; i++)
5             System.out.print(tab[i] + "␣");
6         System.out.println("Fin");
7     }
8 }
```

**Q 71.2** La méthode `getMessage()` de l'exception `ArrayIndexOutOfBoundsException` retourne la position dans le tableau à laquelle l'erreur s'est produite. Modifier la classe `TestAttrapePas0` pour capturer cette exception et afficher le texte : `"Exception : dépassement des bornes position 5"` quand l'exception se produit.

## Exercice 72 – Moyenne de notes valides (`try catch, throw, throws` et création d'exception)

Soit le programme suivant qui, contient une méthode `moyenne` qui, étant donné un tableau de chaînes de caractères représentant des notes (entiers entre 0 et 20), retourne la moyenne entière de ces notes.

```
1 public class TestMoyenne {
2         public static int moyenne(String [] tab) {
3                 int note, somme=0, n=0;
4                 for (int i=0;i<tab.length; i++) {
5                         note=Integer.parseInt(tab[i]);
6                         somme=somme+note;
7                         n=n+1;
8                 }
9                 return (somme/n);
10        }
11        public static void main(String[] args) {
12                System.out.println("Moyenne␣:␣"+moyenne(args));
13        }
14 }
```

*Indications :*

- La méthode `Integer.parseInt` transforme une chaîne de caractères en entier et lève une exception `NumberFormatException` si la chaîne n'est pas un entier. Remarque : l'exception `NumberFormatException` hérite de `RuntimeException`, il n'est donc pas obligatoire de la capturer.
- Les arguments qui sont passés en ligne de commande sont récupérables par le tableau `String[] args` passé en paramètre de la méthode `main`.

**Q 72.1** Est-ce que le programme compile ?

**Q 72.2** Que donnent les exécutions suivantes ?

- `java TestMoyenne 10 12 16 18`
- `java TestMoyenne 11 1j 13`
- `java TestMoyenne`

*Aide : pour les questions suivantes, si vous faites cet exercice sur une feuille en salle de TD, pensez à laisser de la place entre les différentes instructions quand vous recopiez le code de la méthode* ***`moyenne`*** *pour pouvoir ensuite rajouter différentes instructions de gestion des exceptions (`try catch, throw, throws...`).*

**Q 72.3** Quand une note n'est pas un nombre entier, on ne veut pas que le programme s'arrête. Pour cela, on va capturer l'exception. La position du `try catch` pour capturer l'exception a de l'importance et dépend du résultat que l'on veut obtenir. Où faut-il capturer l'exception `NumberFormatException` et quelles instructions faut-il écrire si on veut que, quand au moins une des notes n'est pas un nombre entier :

1. le programme n'affiche pas la moyenne, mais affiche seulement `"Erreur : note pas entière"` ?
2. le programme affiche la moyenne des notes qui sont des nombres entiers ? Pour chaque note X qui n'est pas un nombre entier, le message `"Note X pas entière"` sera affiché (où X est remplacé par la note). Quel est maintenant le résultat de l'exécution de : `java TestMoyenne 11 1j 13` ?

**Q 72.4** Pour la suite de l'exercice, on garde la deuxième solution de la question précédente. Quel est le résultat de l'exécution de : `java TestMoyenne 10a e5` ? Expliquez.

**Q 72.5** On ne veut pas que le programme s'arrête quand aucune des notes n'est entière. Pour cela, on va gérer le problème en créant notre propre exception.

- Écrire une classe `ANEException` (abréviation de `AucuneNoteEntiereException`). De quelle classe doit-elle héritée ? Cette classe contiendra un constructeur qui initialise le message de l'exception avec `"Aucune note entière"` et un autre constructeur `ANEException(String msg)` qui initialise le message de l'exception avec `msg`. Rappel : la classe `Exception` possède un constructeur prenant en paramètre une chaîne de caractères, ce paramètre étant utilisé pour initialiser le message de l'exception.
- A quel endroit faut-il lancer cette exception ? Quelles instructions et information faut-il rajouter ?
- Capturer l'exception et afficher son message dans le main.
- Quel est maintenant le résultat de l'exécution de :
  - `java TestMoyenne` ?
  - `java TestMoyenne 10a e5` ?

**Q 72.6** On veut maintenant gérer les cas où la note est négative et où la note est strictement supérieure à 20.

- Ecrire une classe `PasEntre0et20Exception` avec un seul constructeur `PasEntre0et20Exception(int note, String info)` où `info` peut être par exemple : `"négative"`, `">20"`. On veut que le message de l'exception soit par exemple `"La note -5 est négative"` ou bien `"La note 23 est >20"`.
- Où faut-il lever ces exceptions et où faut-il les capturer si on veut quand même calculer la moyenne des notes qui sont valides ?
- Quel est le résultat de : `java TestMoyenne 14 25 7c 10 -3` ?

## Exercice 73 – throw, throws, finally

**Q 73.1** Donnez l'affichage produit par le programme ci-après. Expliquez les résultats.

```
1 public class MonException extends Exception {
2    public MonException(String s) {
3       super(s);
4       System.out.println("\n␣MonException␣:␣constructeur");
5    }
6 }
7 public class TestFinally {
8    /** Exception deleguee a la methode appelante (ici main).*/
9    public static void test1() throws MonException {
10      if (true) throw new MonException("lancee␣dans␣test1");
11      System.out.println("test1␣:␣fin␣de␣la␣methode");
12   }
13 /** Exception capturee (et pas deleguee) dans la methode test2 */
14 public static void test2() {
15   try {
16      if (true) throw new MonException("lancee␣dans␣test2");
17   } catch (MonException e) {
18      System.out.println("test2␣:␣capture␣de␣l'exception␣:␣"+e);
19   }
20   System.out.println("test2␣:␣fin␣de␣la␣methode");
21 }
22 /** Exception capturee (et pas deleguee) dans la methode test3 avec finally */
23 public static void test3() {
24   try {
25      if (true) throw new MonException("lancee␣dans␣test3");
26   } catch (MonException e) {
27      System.out.println("test3␣:␣capture␣de␣l'exception␣:␣"+e);
28   } finally {
29      System.out.println("test3␣:␣finally␣est␣effectue");
30   }
31   System.out.println("test3␣:␣fin␣de␣la␣methode");
```

```
32 }
33
34 /** Exception deleguee a la methode appelante (ici main) avec finally */
35 public static void test4() throws MonException {
36    try {
37       if (true)
38          throw new MonException("lancee dans test4");
39    } finally {
40       System.out.println("test4 : finally est effectue");
41    }
42    System.out.println("test4 : fin de la methode");
43 }
44
45 /** Meme cas que le test4, mais ici l'exception n'est pas levee */
46 public static void test5() throws MonException {
47    try {
48       if (false) throw new MonException("lancee dans test5");
49    } finally {
50       System.out.println("test5 : finally est effectue");
51    }
52    System.out.println("test5 : fin de la methode");
53 }
54
55 public static void main(String [] args){
56    try {
57       test1();
58    } catch (MonException e) {
59       System.out.println("main : test1 : capture de l'exception "+e);
60    }
61    test2();
62    test3();
63    try {
64       test4();
65    } catch (MonException e) {
66       System.out.println("main : test4 : capture de l'exception "+e);
67    }
68    System.out.println();
69    try {
70       test5();
71    } catch (MonException e) {
72       System.out.println("main : test5 : capture de l'exception "+e);
73    }
74    System.out.println("Fin du programme");
75 }
76 }
```

## Exercice 74 – Utilisation de `Scanner` et de `Thread.sleep(n)`

**Q 74.1** Écrire dans une classe `TestSleep` une méthode `main` qui demande à l'utilisateur de saisir un nombre $x$ de secondes, puis qui fait "s'endormir" le programme pendant $x$ secondes. Le programme doit afficher `"Le nombre est mal formé"` si le nombre saisi n'est pas un entier. Exemples d'exécution du programme :

```
Entrer un nombre de secondes : 7
Attente de 7 secondes
Fin de l'attente
```

```
Entrer un nombre de secondes : 1a
Le nombre est mal formé
```

Documentations utiles :

— Exemple d'utilisation de la classe `Scanner` du package `java.util` :

```
Scanner scanner = new Scanner(System.in);
System.out.print( "Entrer un entier : " );
```

```
int x = scanner.nextInt();
System.out.print( "Entrer une chaîne : " );
String chaine = scanner.next();
```

La méthode `nextInt()` peut lever l'exception `InputMismatchException` du package `java.util` si la valeur saisie n'est pas un entier.

— La méthode `Thread.sleep(n)` arrête pendant $n$ millisecondes l'exécution du programme. Cette méthode peut lever une exception de type `InterruptedException`. Rappel : 1 seconde = 1000 millisecondes.

**Q 74.2** Si on ne veut pas capturer l'exception `InterruptedException`, que faut-il faire pour que le programme continue à compiler ?

## Exercice 75 – EntierBorne (throw,throws)

Le but de l'exercice est de définir une classe `EntierBorne` qui représente tous les entiers entre -10 000 et +10 000 et se prémunisse des dépassements de ces bornes. On testera au fur et à mesure les méthodes écrites. Note : toutes les exceptions seront capturées dans le main.

**Q 75.1** Écrire la classe `EntierBorne` qui est une classe « enveloppe » du type simple int, i.e. qui "enveloppe" une variable d'instance de type int dans un objet de cette classe. Écrire le constructeur à un paramètre de type `int` qui peut lever l'exception `HorsBornesException` si la valeur qui est passée en paramètre est plus grande que 10000 ou plus petite que $-10000$, et la méthode `toString()`. On définira pour cela la classe `HorsBornesException`.

**Q 75.2** Définir la méthode `EntierBorne somme(EntierBorne i)` qui rend un objet `EntierBorne` dont la valeur est la somme de cet élément et du paramètre. Elle pourra lever sans la capturer l'exception `HorsBornesException` si la somme est trop grande.

**Q 75.3** Définir la méthode `EntierBorne divPar(EntierBorne i)` qui rend un objet EntierBorne dont la valeur est la division entière de cet élément par le paramètre i. Elle pourra lever l'exception `HorsBornesException` ou l'exception `DivisionParZeroException`.

**Q 75.4** On définira ensuite la méthode `EntierBorne factorielle()` qui calcule la factorielle de cet élément. Elle pourra, en plus de l'exception `HorsBornesException`, lever l'exception `IllegalArgumentException` dans le cas où $n$ serait négatif.

**Q 75.5** Créer un jeu de tests pour ce programme, en réfléchissant aux différents cas possibles et les tester dans le main. Aide : vous pouvez utiliser la classe `java.util.Scanner` pour demander à l'utilisateur de saisir un nombre (voir la documentation de l'exercice 74).

## Exercice 76 – MonTableau

Le but de l'exercice est de définir une classe `MonTableau`, gérant des « tableaux » ayant une longueur maximum fixée pour tous les éléments de la classe, et qui se prémunisse des dépassements de capacité de ses objets.

**Q 76.1** Définir une classe `MonTableau` qui possède les variables `tab` (tableau d'entiers) et `lgReelle` (entier) donnant le nombre de cases de `tab` réellement utilisées dans le tableau. Au départ, `lgReelle` vaut 0. Ecrire un constructeur prenant en paramètre la taille du tableau, et une méthode `ajouter(int n)` qui ajoute la valeur $n$ à la suite du tableau sans vérifier s'il reste de la place.

**Q 76.2** Ecrire la méthode `main` qui crée un objet `MonTableau` de 3 cases et y ajoute 10 entiers. Exécutez le programme. Que se passe-t-il ?

**Q 76.3** Capturer dans la méthode `main` l'exception précédemment levée, et afficher le texte `"Depassement des bornes a la position 3"` en utilisant la méthode `getMessage()` de l'exception levée.

**Q 76.4** Définir un nouveau type d'exception appelée `TabPleinException`.

**Q 76.5** Modifier la méthode `ajouter` pour lever cette exception quand le tableau est plein. Capturer cette exception dans la méthode `main`. Que retourne les méthodes `getMessage()` et `toString()` de cette exception ?

## Exercice 77 – Extrait de l'examen de 2007-2008 S1

On veut écrire une classe `Etudiant` dont les instances décrivent un étudiant ayant un nom et une liste de notes entières (au maximum 5 notes) implantée par un tableau.
*Rappel de cours :* toute instance de la classe `Exception` doit obligatoirement être attrapée ou signalée comme étant propagée par toute méthode susceptible de la lever.

**Q 77.1** Écrire la classe `Etudiant` correspondant à la description ci-dessus avec un constructeur à un paramètre, le nom. La méthode `toString()` rend le nom de l'étudiant suivi de ses notes.

**Q 77.2** Ajouter la méthode `void entrerNote(int note)` qui entre la note dans la liste des notes de cet étudiant. Elle lèvera une exception `TabNotesPleinException` (à définir) dans le cas où le tableau de notes de cet étudiant serait plein. Cette exception sera capturée dans le `main`.

**Q 77.3** En supposant que la classe qui contient le `main` s'appelle `TestEtudiants`, on veut passer sur la ligne de commande une liste d'étudiants avec leurs notes, par exemple :

```
java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12 Karim 15 8 11 12 10 Melissa 12 6 18 10 12 6
```

On supposera que chaque donnée est correcte (pas de mélange entre lettres et chiffres), et que la première donnée est un nom.
Ces données sont de deux types : chaîne de caractères et entier. On va utiliser le fait qu'un entier ne fait pas lever d'exception à la méthode `Integer.parseInt` alors qu'une chaîne de caractères lui fait lever l'exception `NumberFormatException`.
*Rappel* : la méthode `int Integer.parseInt(String s)` rend l'entier représenté par la chaîne `s`, ou bien lève une exception `NumberFormatException` si la chaîne `s` ne repésente pas un entier.
Écrire le code du `main` qui récupère les données et affiche pour chacune "est une note" ou bien "est un nom" suivant le cas. On utilisera obligatoirement le mécanisme d'exception pour ce faire.
Voici une exécution possible :

```
>java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12
Anna est un nom,
12 est une note, 13 est une note, 7 est une note, 15 est une note,
Tom est un nom,
Arthur est un nom,
9 est une note, 12 est une note, 15 est une note, 0 est une note, 13 est une note, 12 est une note
```

**Q 77.4** On souhaite gérer dans la classe `Etudiant` une liste au sens `ArrayList` d'étudiants. Une liste d'étudiants ne dépend pas d'un étudiant en particulier. Qu'en concluez-vous sur le type de variables que doit être la liste d'étudiants ? Ajouter les instructions nécessaires dans la classe `Etudiant`.

**Q 77.5** Enrichir/modifier le code précédent pour qu'il traite les données de la façon suivante :
- si c'est une chaîne de caractères, il crée une nouvelle instance d'étudiant portant ce nom.
- si c'est une note, il ajoute cette note à la liste des notes de l'étudiant créé précédemment, puis affiche la liste des étudiants. On pensera à traiter les différentes exceptions levées (on rappelle qu'un étudiant a au maximum 5 notes).

Voici une exécution possible :

```
>java TestEtudiants Anna 12 13 7 15 Tom Arthur 9 12 15 0 13 12 Karim 15 8 11 12 10
Melissa 12 6 18 10 12 6
le tableau de notes de l'etudiant Arthur est plein
le tableau de notes de l'etudiant Melissa est plein
les 5 etudiants :
[Anna 12 13 7 15, Tom, Arthur 9 12 15 0 13, Karim 15 8 11 12 10, Melissa 12 6 18 10 12 ]
```

# 11 Patterns, manipulation de flux entrée / sortie

**Exercice 78** (Examen 2016) – **Gestion d'un système de tirage de boules de couleur**

```java
public class Boule {
    private String couleur;

    public Boule(String couleur) {
        this.couleur = couleur;
    }
}
```

```java
public static void main(String[] args){
    Boule b1 = new Boule("rouge");
    Boule b2 = new Boule("jaune");
    Boule b3 = new Boule("bleue");
    Boule b4 = b1;
}
```

**Q 78.1** A l'issue de l'exécution du code ci-dessus, combien y a-t-il de variables et d'instances de `Boule` ?

**Q 78.2** Donner la ligne de code pour créer un tableau de `Boule` (nommé `urne`) contenant les 4 références précédentes.

**Q 78.3** Donner la ligne de code pour choisir aléatoirement une boule dans le tableau `urne` et la ranger dans une nouvelle variable de nom `choix`. Quelle est la probabilité de tirer une boule rouge ?

**Q 78.4** La classe suivante (que l'on appelle une "factory") permet de générer des boules dont la couleur est choisie aléatoirement. Deux erreurs se sont glissées dans cette classe : la première empêche la compilation, la seconde provoque un dysfonctionnement (la méthode de génération de boule renvoie toujours des boules *rouges*).
Donner les corrections à effectuer.

```java
public class BouleFactory {
    public final String[] couleurs = {"rouge","jaune","bleue","verte","orange","violette"};
    public static Boule build(){
        return new Boule(couleurs[(int) Math.random()*couleurs.length]);
    }
}
```

**Q 78.5** En supposant que le code précédent a été corrigé, donner les lignes de code pour générer 1000 boules à l'aide de la classe précédente et les stocker dans une `ArrayList` (nommée `gdeUrne`).

**Q 78.6** Nous voulons faire des statistiques (comptages) sur les boules dans `gdeUrne`.

**Q 78.6.1** Donner le code de la méthode **standard** `equals` de la classe `Boule`.
Remarque : l'attribut `couleur` n'est jamais `null` (mais c'est un objet).

**Q 78.6.2** Compter le nombre de boules de chaque couleur et afficher le résultat dans la console.
Algorithme proposé (mais pas imposé) :

1. Pour toutes les couleurs du tableau `couleurs` (de la classe BouleFactory)
   (a) Créer une boule de la couleur courante
   (b) Initialiser un compteur à 0
   (c) Pour toutes les boules de `gdeUrne`
      i. Tester l'égalité avec la boule courante et incrémenter le compteur en cas de correspondance
   (d) Afficher la couleur courante et le nombre de boules trouvées

**Q 78.7** Un second développeur propose une nouvelle architecture (correcte) pour remplacer la classe `Boule` et la factory associée (sans les aspects aléatoires) :

```java
public class BouleV2 {
    private String couleur;

    public final static BouleV2 ROUGE = new BouleV2("rouge");
    public final static BouleV2 JAUNE = new BouleV2("jaune");
    public final static BouleV2 BLEUE = new BouleV2("bleue");
    public final static BouleV2 VERTE = new BouleV2("verte");
    public final static BouleV2 ORANGE = new BouleV2("orange");
    public final static BouleV2 VIOLETTE = new BouleV2("violette");

    private BouleV2(String couleur) {
```

```
12          this.couleur = couleur;
13      }
14  }
```

**Q 78.7.1** Donner le code permettant de stocker 1000 `BouleV2` tirées aléatoirement dans une `ArrayList`.

**Q 78.7.2** Le développeur prétend que son architecture est
— tout autant sécurisée que la précédente,
— plus rapide,
— ne nécessite pas d'ajouter un code `equals`
Que penser de ces 3 affirmations ?

---

**Exercice 79** (Examen 2017) – **Quelques notes de musique**

---

Dans cet exercice, on se propose de gérer une partition de musique. Pour gagner du temps, nous utilisons des classes existantes comme `Note` qui modélise une gamme grave de piano et la possibilité de transposer les notes dans les gammes au dessus (pour info : l'opération correspond à une multiplication de la fréquence).

```
1  public final class Note {
2      // chaque note correspond à une fréquence
3      public static final Note do_ = new Note(65.4064);
4      public static final Note re_ = new Note(73.4162);
5      public static final Note mi_ = new Note(82.4069);
6      public static final Note fa_ = new Note(87.3071);
7      public static final Note sol_ = new Note(97.9989);
8      public static final Note la_ = new Note(110);
9      public static final Note si_ = new Note(123.471);
10     public static final Note silence_ = new Note(0);
11     // coefficient multiplicateur pour les demi ton (dièse/bémol)
12     private static final double demiTon = 1.05946;
13
14     public final double frequence;
15
16     private Note(double frequence) { this.frequence = frequence; }
17
18     // pour les générer les demis tons
19     public Note diese(){ return new Note(frequence*demiTon); }
20     public Note bemol(){ return new Note(frequence/demiTon); }
21     // pour passer dans une gamme au dessus (facteur = nb de gamme au dessus)
22     public Note transpose(int facteur){ return new Note(Math.pow(2., facteur)*frequence); }
23 }
```

**Q 79.1** Parmi les opérations suivantes, toutes effectuées en dehors de la classe `Note`, lesquelles posent problème ? Expliquer très brièvement.

```
1 Note n1 = new Note(220);
2 Note n2 = Note.do_;
3 Note n3 = n2.re_;
4 Note si_bemol = Note.si_.bemol;
5 System.out.println(n2.frequence);
6 Note.mi_.frequence = 12;
7 Note do_aigu = Note.do_.transpose(1);
8 Note do_diese = Note.do_.diese();
9 Note do_diese2 = Note.do_ * Note.demiTon;
10 Note do_aigu2 = Note.do_.transpose(2.5);
```

**Q 79.2** Quels sont les points forts/faibles de l'architecture de cette classe ?

**Q 79.3** Pour ajouter une notion de rythme (noire, croche, blanche...), nous allons développer une nouvelle classe abstraite `Rythme` et des classes filles concrètes. Est-il possible de faire hériter `Rythme` de `Note` ?

**Q 79.4** Donner le code de la classe `Rythme`, qui gère une note et sa durée (`double`) et possède deux accesseurs vers la

durée et la fréquence. Donner aussi le code de la classe `Noire` qui dure `1.0` temps.

**Q 79.5** Donner le code de la classe `Partition` qui étend la classe `ArrayList<Rythme>` et qui gère un attribut `double tempo` (pour indiquer à quelle vitesse jouer cette partition). Cette classe doit (évidemment) gérer l'ajout et l'accès à la i$^e$ note du tableau. Elle possède aussi un accesseur pour le `tempo`.

**Q 79.6** En cherchant sur internet, nous avons trouvé une classe permettant de générer des sons : `Player`... Cette classe n'est pas compatible avec notre architecture : elle attend pour sa construction un argument de type `Iterator<double[]>`. Un itérateur est une interface qui impose l'implémentation des méthodes suivantes :

```
1 public interface Iterator<double[]>{ // (version simplifiée pour éviter les génériques)
2   public boolean hasNext(); // rend True s'il existe un élément suivant
3   public double[] next(); // retourne l'élément suivant
4 }
```

Chaque élément `double[]` correspondra à un triplet contenant le temps de départ du son (en seconde), sa durée (en seconde) et sa fréquence.
Donner le code de la classe `Traducteur`, qui répond à la spécification `Iterator<double[]>` et qui prend en argument une `Partition`.
**Note :** le `Traducteur` doit gérer le défilement du temps en secondes. Au début, le temps est à 0 ; à chaque fois qu'une note est récupérée dans la partition, le compteur est incrémenté de : $duree_{note} * tempo_{partition}/60$. De la même manière, la durée d'une note en seconde vaut : $duree_{note} * tempo_{partition}/60$.

**Q 79.7** Proposer une classe de test qui construit une partition de 3 notes, la donne à un traducteur puis vérifie le bon fonctionnement de celui-ci en faisant défiler les triplés et en les affichant dans la console.

---

## Exercice 80 – `String` versus `StringBuilder`

**String immutable** *La classe* ***`String`*** *est une classe immutable, ce qui veut dire que quand on crée une chaîne de caractères de type* ***`String`****, on ne peut plus changer sa valeur. Par conséquent, lorsque l'on veut concaténer la chaîne s2 à la chaîne s1 par s1+=s2, une nouvelle chaîne de caractères est créée dans laquelle le contenu de s1 est d'abord recopié, puis le contenu de s2 est recopié à la suite. La nouvelle chaîne produite est alors référencée par s1.*

**StringBuilder mutable** La classe ***`StringBuilder`*** *permet de modifier la chaîne de caractères. Un objet de type* ***`StringBuilder`*** *a une capacité initiale de caractères qui sera agrandi automatiquement si besoin. Elle contient notamment une méthode* ***`append`*** *qui permet de faire de la concaténation :* ***`s1.append(s2)`*** *ajoute les caractères de s2 à la fin de s1.*

Soit le programme suivant dont l'objectif est de comparer le temps d'exécution des concaténations de chaînes de caractères soit en utilisant la classe `String` soit en utilisant la classe `StringBuilder` :

```
1 public class TestStringBuilder {
2     public static String testString(int nbIterations,String chaine) {
3         String s="";
4         for(int i=0;i<nbIterations;i++) {
5             s+=chaine;
6         }
7         return s;
8     }
9     public static StringBuilder testStringBuilder(int nbIterations,String chaine) {
10        StringBuilder sb=new StringBuilder();
11        for(int i=0;i<nbIterations;i++) {
12            sb.append(chaine);
13        }
14        return sb;
15    }
16    public static void main(String [] args) {
17        int nbIterations=100000;
18        if (args.length==1) nbIterations=Integer.parseInt(args[0]);
19
20        String [] tab={"1","1234567890","12345678901234567890123456789012345678901234567890"};
21
22        for(String chaine : tab) {
```

```
23              System.out.println("### nbIter="+nbIterations+" longueur="+chaine.length());
24
25              long debut1 = System.currentTimeMillis();
26              testString(nbIterations,chaine);
27              long fin1 = System.currentTimeMillis();
28              System.out.println("String Durée : "+(fin1-debut1)+" ms");
29
30              long debut2 = System.currentTimeMillis();
31              testStringBuilder(nbIterations,chaine);
32              long fin2 = System.currentTimeMillis();
33              System.out.println("StringBuilder Durée : "+(fin2-debut2)+" ms");
34      }
35  }
```

L'instruction `System.currentTimeMillis()` retourne le temps courant en millisecondes. Voici ci-dessous un exemple de résultat de l'exécution de ce programme (si vous exécutez ce programme sur votre ordinateur, les temps peuvent varier en fonction de la puissance de votre machine).

```
### nbIter=100000 longueur=1
String Durée : 781 ms
StringBuilder Durée : 7 ms
### nbIter=100000 longueur=10
String Durée : 9420 ms
StringBuilder Durée : 2 ms
### nbIter=100000 longueur=30
String Durée : 37167 ms
StringBuilder Durée : 3 ms
```

**Q 80.1** Expliquez pourquoi il y a une si grande différence de temps de calcul entre `String` et `StringBuilder`.

**Q 80.2** La longueur de la chaîne concaténée a-t-elle une forte influence pour `String` ? pour `StringBuilder` ?

**Q 80.3** Quand il y a beaucoup de concaténations de chaînes de caractères à faire, quelle classe faut-il mieux utiliser ?

## La classe File

Le package `java.io` définit un grand nombre de classes pour gérer les entrées / sorties d'un programme. Parmi elles, la classe `File` permet de manipuler des fichiers ou des répertoires. Une instance de `File` est une représentation logique d'un fichier ou d'un répertoire qui peut ne pas exister physiquement sur le disque. La classe `File` définit notamment :

| | |
|---|---|
| - `File(String path)` | construit un objet `File` pointant sur l'emplacement passé en paramètre |
| - `boolean canRead()` | indique si le fichier peut être lu |
| - `boolean canWrite()` | indique si le fichier peut être modifié |
| - `boolean createNewFile()` | crée un nouveau fichier vide à l'emplacement pointé par l'objet `File`, `createNewFile()` peut lever l'exception `java.io.IOException`, par exemple, quand le répertoire n'existe pas |
| - `boolean delete()` | détruit le fichier ou le répertoire |
| - `boolean exists()` | indique si le fichier existe physiquement |
| - `String getAbsolutePath()` | renvoie le chemin absolu du fichier |
| - `File getParentFile()` | renvoie un objet `File` pointant sur le chemin parent de celui de l'objet `File` courant |
| - `boolean isDirectory()` | indique si l'objet `File` pointe sur un répertoire |
| - `boolean isFile()` | indique si l'objet `File` pointe sur un fichier |
| - `File[] listFiles()` | si l'objet `File` est un répertoire, renvoie la liste des fichiers qu'il contient |
| - `boolean mkdir()` | création du répertoire |
| - `boolean mkdirs()` | création de toute l'arborescence du chemin |
| - `boolean renameTo(File f)` | renomme le fichier |

**Exercice 81 – Flux et fichiers**

Dans cet exercice, on s'initie à l'utilisation de la classe `File` pour manipuler des fichiers et répertoires.

**Q 81.1** Soit la classe `TestLitRepertoire` ci-dessous.

```
1 import [** COMPLETER **]
2
3 public class TestLitRepertoire{
4     public static void main(String[] args){
5         String nameF = [** METTRE ICI UN NOM DE REPERTOIRE ou de FICHIER **];
6         File f = new File(nameF) ;
7         if (f.isDirectory()) {
8             System.out.println(nameF + " est un répertoire, il contient : ");
9             for (File e : [** COMPLETER **]) {
10                if ([** COMPLETER **])
11                    System.out.println("<repertoire>: "+e.getName());
12                else
13                    System.out.println("<fichier>: "+e.getName());
14            }
15        }
16    }
17 }
```

Remplacer les "`[** COMPLETER **]`" par le code nécessaire à la bonne exécution du programme : si `nameF` est le nom d'un répertoire, on souhaite lister son contenu en donnant le type et le nom des fichiers et répertoires qu'il contient.

**Q 81.2** Dans la classe `TestLitRepertoire`, proposer la modification à apporter à la ligne 13 pour qu'après le nom du fichier on affiche aussi sa taille en nombre d'octets.
Remarque : il peut être utile de consulter la documentation Java de la classe `File`.

## Exercice 82 – Manipulation de fichiers et d'arborescences

Soit la classe `TestFile` suivante :

```
1 import java.io.File;
2 import java.io.IOException;
3
4 public class TestFile{
5     public static void main(String[] args){
6         try {
7             File f=new File(args[0]);
8             f.delete();
9             System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
10            f.createNewFile();
11            System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
12            System.out.println(f.getAbsolutePath());
13        } catch(IOException e){
14            System.out.println(e);
15        }
16    }
17 }
```

**Q 82.1** Dire ce qu'affiche l'execution suivante : `java TestFile "./lu2in002/TME11/Files/fichier1.txt"`
- Si le répertoire `"./lu2in002/TME11/Files"` existe
- Si le répertoire `"./lu2in002/TME11/Files"` n'existe pas

**Q 82.2** Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier

**Q 82.3** Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la classe `File`

**Q 82.4** Écrire une méthode `ls(File f)` permettant d'afficher tous les noms de fichiers contenus dans le répertoire passé en paramètre (ne pas afficher les répertoires)

**Q 82.5** Écrire une méthode `lsRecursif(File f)` permettant d'afficher tous les noms de fichiers contenus dans

l'arborescence prenant sa racine au niveau du répertoire passé en paramètre (ne pas afficher les répertoires)

## Les flux

Outre la classe `File`, le paquetage `java.io` (i pour input, o pour output) définit une multitude de classes permettant la manipulation de flux de lecture/écriture. Ces flux permettent des échanges de données entre le programme et d'autres entités, qui peuvent être :

- une variable du programme (par exemple, pour la construction de chaînes de caractères)
- la console de l'utilisateur (`System.in` : entrée standard, `System.out` : sortie standard)
- un fichier (création, lecture, écriture, modifications, ...)
- la mémoire
- ...

Deux catégories de flux :

- Les flux entrants pour la lecture
  - `InputStream` pour lire des octets
  - `Reader` pour lire des caractères
- Les flux sortants pour l'écriture
  - `OutputStream` pour écrire des octets
  - `Writer` pour écrire des caractères

Ces classes de flux sont néanmoins des classes abstraites. Les classes à utiliser sont préfixées par :

- la source pour les flux entrants (`FileInputStream`, `FileReader`, `InputStreamReader`, `StringReader`...)
- la destination pour les flux sortants (`FileOutpuStream`, `FileWriter`, `OutputStreamWriter`, `StringWriter`...)

La classe `Reader` définit principalement les méthodes suivantes :

| | |
|---|---|
| - `void close()` | Ferme le flux |
| - `int read()` | Lit le caractère suivant du flux et le retourne. Retourne -1 si la fin du fichier est atteinte. |
| - `int read(char[] cbuf)` | Lit un ensemble de caractères et les place dans le tableau passé en paramètre. Retourne le nombre d'entiers lus, -1 si la fin du fichier est atteinte. |
| - `long skip(long n)` | Passe un nombre donné de caractères. |

La classe `Writer` définit quant à elle les méthodes suivantes :

| | |
|---|---|
| - `void close()` | Ferme le flux après avoir écrit l'ensemble des caractères en mémoire, `close()` peut lever l'exception `java.io.IOException` |
| - `void flush()` | Vide la mémoire du flux (force l'écriture de l'ensemble des caractères en mémoire) |
| - `void write(char c)` | Écrit le caractère c dans le flux. |
| - `void write(char[] cbuf)` | Écrit l'ensemble des caractère du tableau dans le flux. |
| - `void write(char[] cbuf, int debut, int nb)` | Écrit nb des caractères du tableau dans le flux en commencant par celui d'index debut. |
| - `void write(String s)` | Écrit la chaîne de caractère dans le flux. |

Il est à noter que l'appel aux méthodes `write()` n'écrit en fait pas les données directement dans la destination pointée par le flux mais passe par une mémoire nommée mémoire tampon. Ce n'est que lorsque celle-ci est pleine ou lors de l'appel à la méthoode `flush()` que l'écriture effective des données est réalisée. Si l'on travaille sur un fichier, l'inscription des données dans ce fichier n'est alors garantie qu'après appel à la méthode `flush()`.

La classe `PrintWriter` simplifie l'utilisation de la classe `Writer` en définissant les méthodes suivantes :

| | |
|---|---|
| - `PrintWriter(Writer out)` | Construction d'un objet `PrintWriter` sur un flux passé en paramètre |
| - `void close()` | Ferme le flux |
| - `void flush()` | Vide la mémoire du flux (force l'écriture de l'ensemble des caractères en mémoire) |
| - `void print(String s)` | Écrit la chaîne `s` dans le flux. Appel automatique à la methode `flush()`. |
| - `void println(String s)` | Écrit la chaîne `s` dans le flux avec passage à la ligne. Appel automatique à la methode `flush()`. |

Important : pensez à fermer les flux en fin d'utilisation (méthode `close()`).

## Exercice 83 – Manipulations simples de flux

Cet exercice à pour but de s'exercer à la sauvegarde de variables d'objets en utilisant une représentation structurée pour accéder au flux vers un fichier.
Pour cela, on considère des pixels, repérés par 2 coordonnées `x` et `y` (de type `double`), par un entier `num` et par un booléen `allume` qui dit si le pixel est allumé ou non. La classe `Pixel` est donnée ci-après.

```
1 public class Pixel {
2         private double x,y;
3         private boolean allume;
4         private int num;
5         public Pixel(double x, double y, boolean allume, int num) {
6                 this.x = x;
7                 this.y = y;
8                 this.allume = allume;
9                 this.num = num;
10        }
11        public String toString() {
12                return "["+num+"␣("+x+",␣"+y+")␣"+allume+"]";
13        }
14 }
```

Pour utiliser cette classe, on considère la classe `Test` suivante.

```
1 public class Test {
2         public static void main(String[] args) {
3                 Pixel p0 = new Pixel(0.0, 0.0, true,1);
4                 Pixel p1 = new Pixel(4.2, 11.38, false,2);
5                 Pixel p2 = new Pixel(6.6, 4.51, true,3);
6
7                 System.out.println("Création␣de␣"+p0);
8                 System.out.println("Création␣de␣"+p1);
9                 System.out.println("Création␣de␣"+p2);
10        }
11 }
```

**Q 83.1** On souhaite doter la classe `Pixel` d'un moyen de sauvegarder dans un fichier les informations associées à un pixel (c'est-à-dire les valeurs de ses variables d'instance). Cette sauvegarde se fait donc en utilisant une classe d'accès de haut niveau au flux vers le fichier, la classe `DataOutputStream`.
Rajouter à classe la méthode `public void save(DataOutputStream f)` qui a pour but de sauvegarder les 4 variables qui définissent le pixel, l'une après l'autre, dans le flux de sortie dont le descripteur est donné en argument. Rappels : cette méthode est susceptible de lever une exception de type `IOException` qu'il faut laisser passer. Penser aussi à ajouter les `import` nécessaires dans la classe.

**Q 83.2** Le flux en sortie du programme (`FileOutputStream`) vers le fichier où l'on va sauvegarder les informations doit être utilisé pour créer un objet instance de la classe `DataOutputStream` qui sera le moyen d'accéder au flux par des méthodes de haut niveau (cf. cours). L'ouverture de ce flux sera fait dans la méthode `main` de la classe `Test` :

```
1         String nameRep = [** COMPLETER **] ;
2         DataOutputStream fOut = null;
3         try {
```

```
4                   fOut = new [** COMPLETER **](new [** COMPLETER **](nameRep+"p0.bin"));
5                   p0.save(fOut);
6                   p2.save(fOut);
7                   p1.save(fOut);
8           }
9           catch (IOException e) {
10                  System.err.println("Erreur d'acces fichier: "+e);
11          }
12          try {
13                  if (fOut != null)
14                          fOut.close();
15          }
16          catch (IOException e) {
17                  System.err.println("Erreur de fermeture du fichier: "+e);
18          }
```

Remplacer les "`[** COMPLETER **]`" par le code nécessaire à la bonne exécution du programme.

**Q 83.3** Afin de pouvoir récupérer un pixel sauvegardé dans un fichier, rajouter dans la la classe `Pixel` un nouveau constructeur qui prend en argument le descripteur d'un flux structuré en lecture : `DataInputStream f` afin d'initialiser variable par variable l'objet Pixel lu dans le flux (attention à bien respecter l'ordre dans lequel chaque variable a été sauvegardée). Cette méthode est susceptible de lever une exception de type `IOException` qu'il faut laisser passer.

**Q 83.4** Rajouter dans le `main()` de la classe `Test` les instructions pour lire les 3 points précédents. Pour cela, il faut
- créer un flux en lecture (Input) sur le fichier de nom `"pixels.bin"` dans le répertoire courant ;
- associer à ce flux de lecture un flux structuré (Data) permettant de lire des entiers, des doubles ou des booléens ;
- lire un par un chacun des 3 pixels sauvegardés précédemment et les afficher au fur et à mesure ;
- fermer et gérer proprement (avec une gestion correcte de l'exception `IOException`) la fin de la lecture.

**Q 83.5** On considère la classe `ArrayPixels` suivante qui permet de stocker un ensemble d'objets `Pixel`.

```
1 import java.util.ArrayList;
2
3 public class ArrayPixels {
4           private ArrayList<Pixel> l;
5           public ArrayPixels( ) {
6                   l = new ArrayList<Pixel>();
7           }
8           public void add(Pixel p) {
9                   l.add(p);
10          }
11          public String toString() {
12                  String res = "";
13                  for (Pixel p: l) {
14                          res += p.toString()+"\n";
15                  }
16                  return res;
17          }
18 }
```

Rajouter à cette classe `ArrayPixels` la méthode `public void charge(String f)` qui permet de charger un ensemble de pixels (un par un) à partir d'un fichier de nom `f`. Dans cette méthode, la liste doit être initialisée afin qu'elle ne contienne que les objets du fichiers.
La lecture du fichier se fait pixel par pixel. Étant donné que l'on ne connaît pas à l'avance le nombre de pixels qui sont stockés dans le fichier, on utilise alors le fait qu'un accès en lecture sur un fichier par une méthode de `DataInputStream` lève une exception de type `EOFException` lorsqu'il n'y a plus rien à lire dans le fichier. Dans la méthode `charge()`, on lit donc des pixels dans le fichier jusqu'à ce que cette exception se produise. La méthode `charge()` affichera alors à l'écran le nombre de pixels qui ont été lus.
Remarque : cette méthode est aussi susceptible de lever une exception de type `IOException` qu'il faut laisser passer.

```
1 public void charge(String f) throws [** COMPLETER **] {
2           l = new ArrayList<Pixel>();
3           int compte = 0;
```

```
4           [** COMPLETER **] fIn = null;
5           Pixel pLu = null;
6           try {
7               fIn = new  [** COMPLETER **];
8               while (true) {
9                   pLu = new Pixel(fIn);
10                   [** COMPLETER **]
11                  compte++;
12              }
13          }
14          catch ( [** COMPLETER **] e) {
15              System.out.println("Fin de lecture: "+compte+" points chargés.");
16          }
17          if (fIn != null)
18                  fIn.close();
19  }
```

Remplacer les "`[** COMPLETER **]`" par le code nécessaire à la bonne exécution du programme.

**Q 83.6** Rajouter dans le `main()` de la classe `Test` les instructions pour créer une liste de pixels à partir du fichier `"pixels.bin"` et l'afficher.

## Exercice 84 – Traitement de texte

*Rappel :* `String` est une classe immutable, c'est-à-dire qu'une variable de type `String` ne peut pas être modifiée. Lorsque l'on pense modifier un objet `String`, en vérité, on crée un nouvel objet `String` à partir de l'ancien.

**Q 84.1** Écrire une méthode `String saisie()` qui demande à l'utilisateur de saisir une ligne de texte tant que la ligne entrée par l'utilisateur est différente de la chaîne "_fin_". Cette méthode retourne une chaîne de caractères contenant la concaténation de toutes les lignes saisies. Proposez une première solution utilisant des concaténations de `String`. Puis proposez une deuxième solution utilisant un seul objet `StringWriter`.

**Q 84.2** Écrire une méthode `affiche(String fichier)` affichant le contenu du fichier dont le nom est passé en paramètre.

**Q 84.3** Écrire une méthode `afficheLignes(String fichier)` affichant, en numérotant les lignes, le contenu du fichier passé en paramètre.

**Q 84.4** Écrire une méthode `ecrireTexte(String fichier)` permettant de créer un nouveau fichier contenant un texte saisi par l'utilisateur.

**Q 84.5** Écrire une méthode `ajouteTexte(String fichier)` permettant d'ajouter, en fin de fichier passé en paramètre, du texte saisi par l'utilisateur.

**Q 84.6** Écrire une méthode `replace(int num, String newLigne, String fichier)` permettant de remplacer, dans le fichier passé en paramètre, la ligne numéro `num` par la nouvelle ligne `newLigne`.

**Q 84.7** Écrire un programme proposant à l'utilisateur un menu lui permettant d'éditer un fichier dont le chemin est passé en argument. Exemple :

```
Fichier "Texte.txt"
1. Ajouter texte
2. Afficher fichier
3. Remplacer ligne
4. Quitter
```

## Exercice 85 – Copie de fichiers binaires

**Q 85.1** Écrire un programme permettant de copier un fichier binaire dont le nom est donné en premier argument sous

le nom donné en second argument.

## La mise en mémoire tampon

La mise en mémoire tampon des données lues permet d'améliorer les performances des flux sur une entité. Par l'utilisation directe d'un objet `Reader`, les caractères sont lus un par un dans le flux, ce qui est très peu efficace. La classe `BufferedReader` (existe aussi pour `BufferedInputStream` pour les octets) permet la mise en mémoire tampon des données lues avant leur transmission au programme.
En outre, elle simplifie l'utilisation du `Reader` en définissant notamment une méthode `String readLine()` permettant de lire les données ligne après ligne plutôt que caractère après caractère (toutes les méthodes de `Reader` sont disponibles dans cette classe mais avec une meilleure gestion de la mémoire).

### Exercice 86 – Mise en mémoire tampon

**Q 86.1** Sachant que la construction d'un `BufferedReader` se fait en passant un flux `Reader` en paramètre, écrivez l'ouverture d'un flux de lecture avec utilisation de la mémoire tampon sur un fichier "text.txt" du répertoire courant.

**Q 86.2** Écrire une méthode `afficheLignesFichier(String fichier)` qui affiche ligne après ligne le texte du fichier dont le chemin est passé en paramètre.

**Q 86.3** Sachant qu'il est également recommandé par souci d'efficacité d'encapsuler tout flux en écriture dans un objet `BufferedWriter` (resp. `BufferedStream` pour l'écriture d'octets), écrire une classe `Ecrivain` ouvrant un flux en écriture sur un fichier à sa construction et disposant des méthodes données ci-dessus pour la classe `PrintWriter` (sauf méthode `flush()`). On pourra donner une version avec héritage et une version sans.

### Exercice 87 – Production automatique de compte rendu TME

L'objectif de cet exercice est d'utiliser les connaissances acquises sur la lecture et l'écriture de fichier pour programmer un outil de production automatique de compte rendu de TME.
On considère que l'utilisateur dispose d'une arborescence de fichiers (telle que celle de votre répertoire) prenant racine en un répertoire LU2IN002. Ce répertoire contient un répertoire par TME (numérotés de TME1 à TME11), chacun d'entre eux contenant eux mêmes un répertoire par exercice (Exo1, Exo2, ... ExoN). On considère également que l'on dispose d'un fichier "etudiants.txt" dans le répertoire LU2IN002 contenant les prenoms, noms et numeros d'etudiants des utilisateurs du programme (une ligne par étudiant). Le fichier doit se terminer par une ligne "Groupe : <numero du groupe>". Enfin, chaque répertoire d'exercice contient deux fichiers "intitule.txt" et "executions.txt", le premier contenant l'énoncé de l'exercice, le second contenant les résultats d'exécution des programmes ainsi que les observations qui ont pu avoir été faites.

**Q 87.1** Écrire un programme `RenduTMEProducer` prenant en argument le chemin du répertoire de TME concerné par le compte rendu et produisant en racine de ce répertoire un fichier "compteRenduTME.txt" de la forme de celui que vous avez l'habitude de rendre en fin de TME.

## Entrée / Sortie standard

Nous avons vu la manière d'écrire ou lire dans des fichiers. L'écriture sur la sortie standard (tel qu'on l'a souvent pratiqué par `System.out.println` sans trop savoir à quoi cela correspondait) ou la lecture à partir de l'entrée standard (comme ce que l'on fait avec la classe `Clavier` pour interagir avec l'utilisateur) utilisent également des flux en lecture/écriture :

- La sortie standard `System.out` correspond à un flux `PrintWriter` (c'est pourquoi on peut utiliser la méthode `println` sur cet objet)
- L'entrée standard `System.in` correspond à un flux `InputStream` (flux permettant de lire des octets à partir d'une source)

Pour la sortie, aucun problème, on sait déja le faire : `System.out.println("texte a afficher");`
Pour l'entrée, c'est un peu plus compliqué : il s'agit de transformer les octets lus à partir de l'objet `InputStreamReader` en caractères que l'on sait manipuler.

### Exercice 88 – Classe Clavier

**Q 88.1** Sachant que le paquetage `java.io` contient une classe de flux `InputStreamReader` permettant de lire des caractères à partir d'un flux entrant d'octets, réécrire le code de la classe `Clavier`, notamment :

- La fonction statique `String SaisirLigne(String message)`
- La fonction statique `int SaisirEntier(String message)`

## Aide mémoire

### Site Moodle

Sur le site Moodle 2021, accéder à la page de l'UE `LU2IN002-S1` :

https://moodle-sciences.upmc.fr/moodle-2021/

### Convention d'écriture

https ://www-ppti.ufr-info-p6.jussieu.fr/

- Le nom des classes (et des constructeurs) commence par une majuscule ;
- Le nom des méthodes, des variables ou des instances commence par une minuscule ;
- Les mots réservés sont obligatoirement en minuscules ;
- Les constantes sont généralement en majuscules.

### En-tête du main

```
public static void main(String[] args)
```

### Grandes lignes de la structure d'une classe

```
class MaClasse [extends ClasseMere] {
    private int maVariable;      // Variables (appelees aussi champs ou attributs)
    private static int maVariableStatique=0; // Variables de classe
    private static final int CONSTANTE=3.1415;  // Constantes
    public MaClasse () { // Constructeurs
        ....
    }
    public int getMaVariable() {// Accesseurs (methodes get)
        return maVariable;
    }
    public void setMaVariable(int v) {// Modificateurs (methodes set)
        maVariable=v;
    }
    public String toString() {
        ....
        return chaine;
    }
    public void methode() {      // Autres methodes
        ....
    }
}
```

### Commentaires

- `//` commentaire sur une ligne
- `/*` commentaire sur plusieurs lignes `*/`

### Divers

| | |
|---|---|
| Afficher une chaine dans le terminal | `System.out.println(chaine);` |
| Déclaration de variable | `type identificateur ;` |
| Déclaration/création de tableau | `type [] identificateur = new type [taille] ;` |
| Création d'un objet (instanciation) | `new AppelConstructeur(...);` |
| Référence à l'objet courant | `this` |
| Importation d'une bibliothèque | `import nompackage.* ;` |
| Test du type de l'objet | *var* `instanceof` *NomClasse* : retourne `true` si *var* est de type *NomClasse* |

## Principales instructions

*Instruction*

expression ;
l'instruction vide ;
**{** instructions **}** aussi appelé bloc d'instruction
une instruction de contrôle

*Instruction de contrôle - Conditionnels*

`if`
**if** (condition) **{**
    instructions
**}**

`if else`
**if** (condition) **{**
    instructions 1
**} else {**
    instructions 2
**}**

*Instruction de contrôle - Boucles*

`for`
**for** (initialisation **;** condition **;** expression) **{**
    instructions
**}**

`while`
**while** (condition) **{**
    instructions
**}**

`do`
**do {**
    instructions
**} while** (condition) **;**

`switch`
**switch** (sélecteur) **{**
    **case** constante1 **:** instructions ;**break ;**
    **case** constante2 **:** instructions ;**break ;**
    ...
    **default :**
        instructions ;
**}**

## Tableau de codage des types simples

| type java | type de codage | bits | min et max | valeur par défaut |
|---|---|---|---|---|
| boolean | `true/false` | 1 | | false |
| char | Unicode | 16 | \u0000 à \uFFFF | \u0000 |
| byte | entier signé | 8 | -128 à 127 | 0 |
| short | entier signé | 16 | -32 768 à 32767 | 0 |
| int | entier signé | 32 | -2 147 483 648 à +2 147 483 647 | 0 |
| long | entier signé | 64 | -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 | 0 |
| float | flottant IEEE 754 | 32 | $\pm 1.4e^{-45}$ à $\pm 3.4028235e^{+38}$ | 0.0f |
| double | flottant IEEE 754 | 64 | $\pm 4.9e^{-324}$ à $\pm 1.7976931348263157e^{308}$ | 0.0d |

## Table de priorité des opérateurs

Les opérateurs sont classés suivant l'ordre des priorités décroissantes. Les opérateurs d'une ligne ont la même priorité, tous les opérateurs de même priorité sont évalués de la gauche vers la droite sauf les opérateurs d'affectation.

| | |
|---|---|
| opérateurs postfixés | `[ ]  .  expr++  expr--` |
| opérateurs unaires | `++expr  --expr  +expr  -expr  ~  !` |
| création ou cast | `new  ( type )  expr` |
| opérateurs multiplicatifs | `*  /  %` |
| opérateurs additifs | `+  -` |
| décalages | `<<  >>  >>>` |
| opérateurs relationnels | `<  >  <=  >=` |
| opérateurs d'égalité | `==  !=` |
| et bit à bit | `&` |
| ou exclusif bit à bit | `^` |
| ou ( inclusif ) bit à bit | `\|` |
| et logique | `&&` |
| ou logique | `\|\|` |
| opérateur conditionnel | `?  :` |
| affectations | `=  +=  -=  *=  /=  %=  &=  ^=  \|=  <<=  >>=  >>>=` |

## La classe `Math` (standard)

La classe `Math` est une classe standard de Java qui prédéfinie un certain nombre de variables et de méthodes. Pour utiliser une méthode de cette classe, il faut faire précéder l'appel de la méthode par `Math`, car les méthodes de cette classe sont des méthodes de classe (déclarées `static`).
*Exemple :* pour calculer la surface d'un cercle de rayon 3.2cm, on peut calculer $\pi r^2$ ainsi :
`double r=3.2; double s = Math.PI*Math.pow(r,2);`
Voici quelques extraits des champs et méthodes de cette classe.

| | | |
|---|---|---|
| `static double` | `E` | The double value that is closer than any other to e, the base of the natural logarithms. |
| `static double` | `PI` | The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter. |

| | | |
|---|---|---|
| `static double` | `random()` | Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. |
| `static double` | `sqrt(double a)` | Returns the correctly rounded positive square root of a double value. |
| `static double` | `pow(double a, double b)` | Returns the value of the first argument raised to the power of the second argument. |
| `static double` | `abs(double a)` | Returns the absolute value of a double value (idem pour float, int, long). |
| `static double` | `ceil(double a)` | Returns the smallest (closest to negative infinity) double value that is >= to the argument and is equal to a mathematical integer. |
| `static double` | `floor(double a)` | Returns the largest (closest to positive infinity) double value that is <= to the argument and is equal to a mathematical integer. |
| `static long` | `round(double a)` | Returns the closest long to the argument (idem pour float). |

## La classe `String` (standard)

| | | |
|---|---|---|
| `int` | `length()` | Returns the length of this string. |
| `boolean` | `equals(Object o)` | Compares this string to the specified object. |
| `int` | `compareTo(String s)` | Compares two strings lexicographically. |
| `String` | `replace(char old, char newChar)` | Returns a new string resulting from replacing all occurrences of old with newChar. |
| `String[]` | `split(String regex)` | Splits this string around matches of the given regular expression. |
| `String` | `substring(int begin, int end)` | Returns a new string that is a substring of this string. |
| `String` | `trim()` | Returns a copy of the string without leading and trailing whitespace. |
| `char` | `charAt(int index)` | Returns the char value at the specified index. |
| `int` | `indexOf(int ch)` | Returns the index within this string of the first occurrence of ch. |
| `int` | `lastIndexOf(int ch)` | Returns the index within this string of the last occurrence of ch. |
| `char[]` | `toCharArray()` | Converts this string to a new character array. |
| `static String` | `copyValueOf(char[] data)` | Returns a String that represents the character sequence in the array specified. |
| `static String` | `valueOf(double d)` | Returns the string representation of the double argument (idem pour boolean, char, char[], float, int, long et Object) |

## La classe `Arraylist` (standard)

La classe `ArrayList` est une une classe prédéfinie en java qui se trouve dans le package `java.util` (rajouter en haut de votre fichier : `import java.util.ArrayList;`). L'utilisation de cette classe nécessite de préciser le type E des objets qui sont dans la liste. Pour cela, on indique le type des objets entre `<...>`.

| | | |
|---|---|---|
| | `ArrayList<E> ()` | Construit une liste vide ; les objets insérés devront être de classe E. |
| `int` | `size()` | Returns the number of elements in this list. |
| `boolean` | `add(E e)` | Appends the specified element to the end of this list. |
| `void` | `add(int index, E e)` | Inserts the specified element at the specified position in this list. |
| `E` | `get(int index)` | Returns the element at the specified position in this list. |
| `E` | `set(int index, E e)` | Replaces the element at the specified position in this list with e. |
| `boolean` | `contains(Object o)` | Returns true if this list contains the specified element. |
| `int` | `indexOf(Object o)` | Returns the index of the first occurrence of o, or -1 if it doesn't exist. |
| `void` | `clear()` | Removes all of the elements from this list. |
| `E` | `remove(int index)` | Removes the element at the specified position in this list. |
| `Object[]` | `toArray()` | Returns an array containing all of the elements in this list |

# Environnement Linux

Pour plus d'information, pensez à consulter le site de la PPTI : https://www-ppti.ufr-info-p6.jussieu.fr/

## Création et gestion de répertoires sous Linux

| | |
|---|---|
| `mkdir REPERTOIRE` | Création du répertoire de nom `REPERTOIRE` |
| `rmdir REPERTOIRE` | Destruction du répertoire de nom `REPERTOIRE` (qui doit être vide) |
| `cd REPERTOIRE` | Déplacement dans le répertoire de nom `REPERTOIRE` |
| `cd ..` | Déplacement vers le répertoire père. |
| `cd` | Déplacement vers le home répertoire |
| `ls` | Liste des fichiers et répertoires du répertoire courant |
| `pwd` | Affiche le nom (et le chemin) du répertoire courant |
| `cp SOURCE DESTINATION` | Copie du fichier `SOURCE` dans le fichier `DESTINATION` |
| `mv SOURCE DESTINATION` | Renomme ou déplace le fichier `SOURCE` en `DESTINATION` |

## Démarrage sous Linux

- Pour ouvrir une fenêtre de travail : cliquer sur l'icone "Terminal" dans le bandeau en haut de la fenêtre OU choisir menu Accessoires, option "Terminal".
- Lancer un éditeur de texte. Par exemple :
  - pour lancer l'éditeur `gedit`, tapez dans le terminal : `gedit &`
  - pour lancer l'éditeur `emacs`, tapez dans le terminal : `emacs &`

  *Attention* : si on oublie de taper le caractère "&" en fin de commande, on ne pourra plus rien exécuter dans la fenêtre de travail sauf en tapant CTRL Z pour interrompre la commande, puis en tapant la commande bg (background) pour relancer la commande sans perdre le contrôle de la fenêtre.
- Dans le terminal, pour reprendre une commande que vous avez déjà tapée dans le terminal : utilisez les flèches *haut* et *bas* pour se déplacer dans l'historique des commandes. Et utiliser les flèches *gauche* et *droite* pour se déplacer dans la commande que l'on peut alors modifier.

## Exécution de programmes

Soit un programme sauvegardé dans le fichier de nom `"Essai.java"` qui contient une classe appelée `"Essai"`.

- Pour compiler, taper dans le terminal la commande :
  `javac Essai.java`
  Si le programme comporte des erreurs, il apparaîtra des messages d'erreur avec l'indication de la ligne du programme correspondante, sinon un fichier `Essai.class` est créé dans le répertoire courant.
- Si la classe `Essai` contient la méthode `main` alors pour exécuter le programme, taper :
  `java Essai`
- Pour arrêter une exécution en cours (en cas de bouclage par ex.), taper : `[CTRL] C`

# Quelques bonnes pratiques pour écrire les programmes

### Indentation

L'indentation, c'est la disposition judicieuse des instructions les unes par rapport aux autres. L'indentation traduit visuellement la structure du programme, elle met en relief les alternatives, les répétitions, les classes, etc. C'est pourquoi, tout programme doit être rigoureusement indenté, sinon il devient rapidement illisible.

### Quelques conseils

— N'écrivez jamais plus de dix ou quinze lignes à la fois. Compilez et exécutez dès que possible. Corrigez tout de suite les erreurs en commençant impérativement par la première. Une erreur peut engendrer plusieurs messages. Si vous avez une erreur ligne 10, son origine est nécessairement située avant.
— Une règle de base : traduisez et comprenez les messages d'erreurs.

Les messages donnés par le compilateur ne sont qu'indicatifs. Si le compilateur vous indique : *ligne 30 ';' expected*, c'est-à-dire « point-virgule attendu », ne mettez pas un point-virgule à cette ligne. Recherchez l'origine exacte de l'erreur. Il est très rare que le compilateur vous donne la solution rigoureuse du problème diagnostiqué. C'est pour cela que vous devez connaître la syntaxe des instructions Java et bien comprendre ce que vous écrivez.

### Quelques erreurs fréquentes

— L'oubli d'une accolade est souvent très difficile à retrouver. Donc, chaque fois que vous tapez {, dans la foulée tapez } et ouvrez des lignes entre les deux en tapant simplement Entrée.
— Lorsque vous lancez une compilation javac Bonjour.java par exemple, si le système vous dit "*cannot read*", c'est qu'il n'a pu lire le fichier Bonjour.java. Autrement dit, vous n'êtes pas dans le bon répertoire. Changez de répertoire (commande cd).
— Autre erreur fréquente : vous écrivez une instruction en dehors d'une méthode. Un fichier Java est composé de classe(s). Une classe est constituée de déclarations de variables et de méthodes. Une méthode est composée de déclarations de variables (locales) et d'instructions. Une instruction est donc nécessairement à l'intérieur d'une méthode.
— Java impose de respecter la casse (c'est-à-dire les majuscules ou minuscules). L'identificateur `toto` est différent de `Toto` ; `setVisible` est différent de `setvisible` ; `Main` est différent de `main`, etc.