LU2IN002 : Éléments de programmation par objets avec JAVA

Licence de Sciences et Technologies
Mention Informatique

Fascicule de TD/TME

Année 2024-2025

TD 1 – Classe : définition, syntaxe

Exercice 1 – 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 1.1 Dans cette classe, quelles sont les variables qui sont (a) des variables d'instance ? (b) des paramètres ? (c) des variables locales à une méthode ?

Q 1.2 Où est le constructeur ? Comment le reconnaît-on ? Quel est le rôle des constructeurs en général ? Quand sont-ils appelés ?

Q 1.3 Quelles sont les méthodes de cette classe ?

Q 1.4 Écrire 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. Aide : pour la syntaxe, on peut se reporter à l'aide mémoire (Annexe A page 51).

Q 1.5 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 1.6 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 2 – Se présenter

Q 2.1 Une personne est représentée par son nom et son âge. Écrire 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 2.2 Écrire 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 2.3 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 2.4 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 2.5 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 2.6 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`). Aide : voir la syntaxe des boucles dans l'aide mémoire page 51

Exercice 3 – 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.

Q 3.1 Repartir de l'exercice 1 et ajouter un second constructeur qui prend en argument seulement le nom de la planète et fixe son rayon à 1000km (arbitrairement).

Q 3.2 Écrire un programme de test construisant deux planètes en utilisant les deux constructeurs pour vérifier le bon fonctionnement de cette approche.

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).

`abcDef()`	`abcDef`	`AbcDef`	`AbcDef()`
`String`	`true`	`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 (associatif à gauche), quelle est la valeur des expressions suivantes ?

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

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

QZ 3.5 Comment convertir un entier, un réel, un caractère... en chaîne de caractères ?

QZ 3.6 Quel est le type et la valeur de l'expression suivante ? `true || true == false`
Aide : voir la table de priorité des opérateurs dans l'aide mémoire page 52.

QZ 3.7 Qu'affiche les instructions suivantes ?

a) `System.out.println("Bonjour\nvous\ttous !");`

b) `System.out.print("Hello"); System.out.println(" world");`

Quizz 4 – Compilation et exécution

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

QZ 4.2 Une classe est composée de ... et de ...

QZ 4.3 Les instructions Java sont toujours situées à l'intérieur de ...

QZ 4.4 Les lignes composant une méthode sont soit des ... soit des ...

QZ 4.5 Quel est le nom de la méthode par lequel un programme Java commence son exécution ?

TME 1 – Introduction à Java – premiers pas

Exercice 4 – Préliminaires

Q 4.1 En salle de TME, si vous n'avez pas accès à Internet :
- dans les paramètres de votre navigateur, cherchez la partie concernant le proxy dans les paramètres réseaux
- passez en configuration manuelle en mettant : `proxy` et le port `3128` dans tous les champs

Q 4.2 Pour bien organiser vos fichiers, on veut créer le répertoire `LU2IN002` dans votre répertoire de travail, puis dans ce répertoire, on veut 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 (voir un rappel des commandes dans l'Annexe B) 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 été créé (commande `ls`),
5. se déplacer dans le répertoire `TME1`,
6. afficher le nom du répertoire courant (commande `pwd`).

Q 4.3 Pour écrire du code Java, il faut ouvrir un éditeur de texte. Il existe de nombreux éditeurs de texte : `gedit`, `vim`, `geany`, `emacs`... Par exemple, pour utiliser l'éditeur `gedit`, taper dans un terminal :

`gedit` *`MaClasse.java`* `&`

Si le fichier `MaClasse.java` existe dans le répertoire courant, le fichier sera ouvert, sinon cela le créera.
Remarques :
- *Attention* : ne pas oublier le '`&`' à la fin de la commande pour séparer le terminal et l'éditeur de texte.
- Vous ne devez PAS UTILISER d'IDE (c'est-à-dire pas d'eclipse, netbeans...) avant la semaine 5.
- Si vous avez sauvé votre fichier avec l'extension `".java"`, normalement l'éditeur de texte activera automatiquement la coloration syntaxique pour Java, sinon cherchez les options pour colorer la syntaxe, indenter les lignes et les numéroter dans les paramètres de votre éditeur.

Q 4.4 Lisez *rapidement* l'Annexe B et répondez *très brièvement* aux questions suivantes.
- Comment pouvez-vous gagner du temps quand vous tapez des commandes ou des noms de fichiers dans le terminal ?
- 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 ?

Remarque : pensez à mettre en application les suggestions de l'Annexe B à chaque séance de TME.

Exercice 5 – Premier programme

Q 5.1 Écrire la classe `Bonjour` (fichier `Bonjour.java`) dont la méthode main affiche le message `"Bonjour !"`.

Q 5.1.1 Quelle est la commande pour compiler cette classe ? Compiler la classe. Quel est le nom du fichier créé par la compilation ?

Q 5.1.2 Quelle est la commande pour exécuter ce programme ? Exécuter ce programme. Supprimer le fichier `Bonjour.class` et, sans recompiler, relancer la commande pour exécuter ce programme. Ce programme est-il exécuté ?

Q 5.2 Observer les erreurs de compilation :

Q 5.2.1 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 5.2.2 Rétablir le mot `static` correctement et supprimer le `"` terminant le mot `Bonjour`. Compiler et observer.

Q 5.2.3 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 5.2.4 Après avoir remis le bon nom à la fonction `main`, supprimer l'accolade `{` qui suit le `main`. Compiler et lire les messages.

Q 5.2.5 Après avoir remis l'accolade, supprimer le mot-clé `public`. La compilation réussit-elle ? Peut-on exécuter le programme ?

Q 5.2.6 Après avoir remis le mot clé `public`, supprimer le mot-clé `static`. La compilation réussit-elle ? Peut-on exécuter le programme ?

Exercice 6 – 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 6.1 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 6.2 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 7 – Solidarité villageoise

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

Q 7.1 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).

Q 7.2 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 page 3. Voir aussi la documentation de la classe `Math` page 53.

Q 7.3 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 7.4 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 (elle retourne par défaut le nom de la classe concaténé avec '@' puis avec un identifiant pour l'objet). Vérifiez-le en ajoutant dans le `main` les instructions pour afficher la méthode `toString` de chacun des villageois. Compilez et exécutez.

Q 7.5 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",123.456);` retourne la chaîne de caractères `"123.45"` (2 chiffres après la virgule).
Compilez et exécutez à nouveau votre programme. Comparez avec le résultat de la question précédente.

Q 7.6 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 7.7 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 7.8 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 7.9 Ajouter dans la classe `TestVillageois`, les instructions pour calculer le poids total que peuvent soulever les 4 villageois, et afficher un message pour indiquer s'ils réussissent à soulever le rocher ou pas.

Exercice 8 – Affichage avec passage à la ligne

Soit la classe `Lettre` suivante dont le but est de gérer un caractère.

```
public class Lettre {
        private char carac;
        public Lettre(char c) {
                carac=c;
        }
        public char getCarac() { return carac; }
        public int getCodeAscii() { return (int)carac; }
}
```

Q 8.1 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 8.2 On veut maintenant afficher l'alphabet comme sur la figure à droite ci-dessous.
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()`).

```
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
```

Remarque : si besoin des exercices de TME supplémentaires sont disponibles sur le site de l'UE.

TD 2 – Encapsulation, surcharge

Exercice 9 – 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␣=␣" + volume;
15        }
16 }
```

Q 9.1 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 9.2 Expliquer l'affectation de la ligne 5 : que représente `this.volume` ? `volume` ?

Q 9.3 Expliquer la ligne 8.

Q 9.4 Compléter la méthode d'instance `remplir(Bouteille b)` qui ajoute le contenu de `b` à la bouteille courante et vide la bouteille `b`.

Q 9.5 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 9.6 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 10 – 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`. Dans quelle classe faut-il écrire la méthode `addition` ?

Exercice 11 – Sélection de méthode

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

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

Q 11.1 Selon le principe de base de JAVA qui interdit deux signatures identiques pour des méthodes (pas de prise en compte du nom des paramètres et du type de retour), cette classe compile-t-elle ?

Q 11.2 Donner les affichages associés aux instructions suivantes (contenues dans une méthode `main`). Certaines lignes ne compilent pas : indiquer brièvement pourquoi.

```
20 Truc t = new Truc();
21 Truc t2 = new Truc(2);
22 double deux = 2;
23 int i = 2.5;
```

```
24 t.maMethode(2);
25 t.maMethode(deux);
26 t.maMethode(2.);
```

```
27 t.maMethode(1, 2);
28 t.maMethode(1, 2, 3);
29 t.maMethode(1., 2, 3);
```

Exercice 12 – Point : sur l'égalité

Soit les instructions suivantes (contenues dans une méthode `main`) :

```
1 Point p1 = new Point(1,2);
2 Point p2 = new Point(1,2);
3 Point p3 = new Point(2,3);
4 Point p4 = p1;
5 Point p5 = null;
6 Point p6 = p5;
7 p6 = new Point(3,4);
```

```
8 if(p1==p2) System.out.println("p1 egale p2");
9 if(p1==p3) System.out.println("p1 egale p3");
10 if(p1==p4) System.out.println("p1 egale p4");
11 if(p1==null) System.out.println("p1 egale null");
12
13 if(p1.equals(p2)) System.out.println("p1 egale p2 (2)");
14 if(p1.equals(p4)) System.out.println("p1 egale p4 (2)");
```

Q 12.1 Donner le nombre de variables et le nombre d'instances de la classe `Point` créées lors de l'exécution. Dessiner l'état de la mémoire après l'exécution de la première colonne de code.

Q 12.2 Quels sont les affichages à l'issue de l'exécution de la seconde colonne ?

Q 12.3 Donner les sorties associées aux commandes suivantes :

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

```
27 System.out.println(p6);
28 System.out.println(p6.toString());
```

Q 12.4 On ajoute encore les 2 instructions suivantes au programme principal. Quel est l'impact de chacune des deux lignes sur le nombre total d'instances présentes en mémoire ?

```
29 p3 = p1;
30 p1 = p4;
```

Quizz 5 – Fleur (constructeur, `this`)

Étudier 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,
6                    String couleur) {
7          nom = name;
8          this.couleur = couleur;
9      }
10     public Fleur (String nom) {
11         this(nom,"rouge");
12     }
13     public String toString() {
14         return nom + "␣de␣couleur␣" + couleur ;
15     }
16     public String getNom() { return nom; }
17 }
```

```
20 public class TestFleur {
21     public static void main (String[] args) {
22         Fleur tulipe=new Fleur("Tulipe","jaune");
23         System.out.println(tulipe.getNom());
24     }
25 }
```

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 22 et 23 ?

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 ci-contre. Quelle erreur est signalée à la compilation ?

```
30 public Fleur (String couleur) {
31         this("Marguerite",couleur);
32 }
```

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

```
40 public Fleur () {
41         couleur="jaune";
42         this("Jonquille");
43 }
```

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

```
50 public Fleur () {
51         this("Rose");
52         couleur="rouge";
53 }
```

Quizz 6 – Méthode `toString()`

QZ 6.1 `int k=3; System.out.println("k="+k.toString());` Ces instructions sont-elles correctes ?

QZ 6.2 Soit la classe suivante :

```
1  public 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 : "+f1)` ?

Quizz 7 – Egalité entre deux chaînes

On peut créer un objet `String` de deux façons : soit on utilise simplement des guillemets (*string literal*), soit on utilise le constructeur de la classe `String` (*string object*). Donner l'affichage obtenu par le code ci-dessous. En déduire la bonne façon, en général, de comparer deux chaînes de caractères.

```
1 String s1 =  "Bonjour" ; // c'est une "string literal"
2 String s2 =  "Bonjour" ;
3 String s3 =  new String("Bonjour") ; // c'est une "string object"
4
```

```
System.out.println("s1==s2 : "+(s1==s2)) ;
System.out.println("s1==s3 : "+(s1==s3)) ;
System.out.println("s1.equals(s3) : "+s1.equals(s3));
```

Quizz 8 – Encapsulation

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

```
public class Point {
        private int x;
        public int y;
        public void f1 () {}
        private void f2 () {}
}
```

```
public class TestPoint {
        public static void main(String[] args) {
                Point p1=new Point();
                System.out.println(p1.x);
                System.out.println(p1.y);
                p1.f1();
                p1.f2();
        }
}
```

TME 2 – Encapsulation, surcharge

Exercice 13 – 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 13.1 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. Écrivez ce constructeur le plus simplement possible.
- 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 13.2 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 14 – Course de relais 4 fois 100m

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

Q 14.1 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 14.2 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 14.3 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 le `main` compile.

Q 14.4 Ajouter dans la classe `Coureur` 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 mutateur : `void setPossedeTemoin(boolean possedeTemoin)` qui change la valeur de la variable d'instance `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`
  Rappel : `String.format("%.2f",123.4567);` retourne la chaîne `"123.45"` (deux chiffres après la virgule).

Q 14.5 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 14.6 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 15 – 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 dans l'intervalle [-2,2[.
  NB : utiliser obligatoirement `this(...)` dans ce second constructeur.

Q 15.1 Donner le code de la classe `Complexe`.

Q 15.2 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 : $i^2 = -1$ et $(1 + i) \times (2 + 2i) = 4i$

Q 15.3 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

Remarque : si besoin des exercices de TME supplémentaires sont disponibles sur le site de l'UE.

TD 3 – Composition, copie d'objets

Exercice 16 – Composition/agrégation et modélisation

Dessiner le **diagramme de classes** UML montrant seulement les relations de composition/agrégation pour les problèmes suivants.

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

Exercice 17 – Feu tricolore (composition)

Un feu tricolore contient 3 lampes : une verte, une orange et une rouge.

Q 17.1 Dessiner le **diagramme de classes** (sans les attributs ni les méthodes).
Soit la classe `Lampe` suivante :

```
public class Lampe {
    private boolean etat; // true allumée, false éteinte
    public Lampe() {
       etat = false;
    }
    public void allumer() {
       etat = true ;
    }
    public void eteindre() {
       etat = false ;
    }
}
```

Q 17.2 Écrire la classe `Feu` (feu tricolore) avec un constructeur à 3 paramètres. Donner les instructions pour créer un objet de la classe `Feu` (référencé par une variable appelée `ft1`) en utilisant ce constructeur et dessiner le **diagramme mémoire** correspondant. Quelle est la différence entre diagramme de classes et diagramme mémoire ?

Q 17.3 Peut-on créer des objets dans un constructeur ? La réponse est oui. Ajouter dans la classe `Feu` un constructeur sans paramètre. Soit l'instruction : `Feu ft2=new Feu();` Quelle différence entre le diagramme mémoire de l'objet référencé par `ft1` et celui de l'objet référencé par `ft2` ?

Q 17.4 Pourquoi le constructeur ci-contre est-il erroné ? Faire un schéma des objets en mémoire pour l'instruction suivante : `Feu ft3=new Feu(new Lampe());`

```
public Feu(Lampe lp) {
    verte=lp; orange=lp; rouge=lp;
}
```

Q 17.5 Trouver et expliquer les erreurs dans les instructions ci-après. Faire un schéma des objets en mémoire.

```
Lampe lp1=new Lampe(); Lampe lp2=lp1; Feu ft4=new Feu(lp1,lp2,lp1);
```

Q 17.6 Écrire le constructeur de copie de la classe `Feu`. Donner toutes les instructions nécessaires pour créer une copie de l'objet référencé par `ft1`.

Exercice 18 – Pion (copie d'objets et composition)

Soit le code suivant :

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

```
13 public class Pion {
14   private String nom ;
15   private Point position ;
16
17   public Pion(String n) {
18     nom = n;
19     position = new Point();
20   }
21   public void setNom(String n) { nom = n; }
22   public String getNom() { return nom; }
23   public void seDeplacer() {
24     position.bouger();
25   }
26 }
```

Q 18.1 On considère l'instruction suivante supposée être dans la méthode `main` d'une classe `TestPion`. Dessinez le diagramme mémoire correspondant. On suppose qu'au départ le pion a la position (0.234,0.567).

```
Pion p1=new Pion("Atchoum");
```

Q 18.2 Qu'affiche les instructions suivantes ? Quel est le problème ? Complétez le diagramme mémoire pour montrer le problème. Aide : combien y-a-t-il d'objets `Pion` créés ?

```
50 Pion p1 = new Pion("Atchoum");
```

```
51 Pion p2 = p1;
52 p2.setNom("Dormeur");
53 System.out.println(p1.getNom());
```

Q 18.3 Une solution possible est d'utiliser un constructeur de copie. Un étudiant propose le constructeur de copie suivant (colonne de gauche). **(a)** Qu'affiche les instructions de la colonne de droite ? **(b)** Ce constructeur est-il correcte ? Que se passe-t-il quand on fait se déplacer le pion référencé par `p3` (ligne 64) ? Donnez le diagramme mémoire pour montrer le problème. Aide : combien y-a-t-il d'objets `Point` créés ? **(c)** Proposez une solution utilisant la copie d'objets pour résoudre le problème. Donnez le diagramme mémoire correspondant.

```
25 // Dans la classe Pion
26 public Pion (Pion p) {
27   nom = p.nom;
28   position = p.position;
29 }
```

```
60 Pion p1 = new Pion("Atchoum");
61 Pion p3 = new Pion(p1);
62 p3.setNom("Dormeur");
63 System.out.println(p1.getNom());
64 p3.seDeplacer();
```

Quizz 9 – 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 9.1 La classe `A` contient-elle un constructeur ?

QZ 9.2 Combien y a-t-il de variables ? Combien y a-t-il d'objets créés ? Faire un diagramme mémoire.

QZ 9.3 Que se passe-t-il si on rajoute l'instruction `a3=null;` ? puis `a2=null;` et enfin `a1=null;` ?

TME 3 – Composition, copie d'objets

Exercice 19 – Tracteur (composition d'objets et copie d'objets)

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

Q 19.1 Donner le diagramme de classe correspondant.

Q 19.2 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 19.3 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 19.4 Écrire une classe `Cabine` qui a un volume (en mètres cubes (m3)) et une couleur de type `String`. Écrire aussi un constructeur avec paramètres, la méthode `toString()` qui rend une chaîne de caractères donnant le volume et la couleur, et le mutateur `setCouleur(String couleur)` pour pouvoir changer la couleur de la cabine.

Q 19.5 Ajouter dans la méthode `main` la création d'une cabine de 3 m^3 de couleur bleue.

Q 19.6 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 19.7 Créer un tracteur `t1` dans la méthode `main` avec les 4 roues et de la cabine bleue créées précédemment. Afficher ensuite ce tracteur.

Q 19.8 Ajouter l'instruction `Tracteur t2=t1;` puis peindre la couleur de la cabine du tracteur `t2` en rouge. Quelle est la couleur de la cabine de `t1` ? Expliquer pourquoi la couleur a changée. Que faut-il faire pour que lorsqu'on peint la cabine du tracteur `t2`, cela ne modifie pas la couleur de la cabine de `t1` ? Faites-le, puis *vérifiez bien* que votre solution fonctionne en affichant la couleur de la cabine de `t1`. Quel est le nom de l'une des techniques classiques vues en cours pour copier un objet ?

Exercice 20 – Classe `Triangle` (composition simple)

Q 20.1 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 à deux paramètres entiers : l'abscisse et l'ordonnée.
- Un constructeur sans paramètre qui initialise aléatoirement les valeurs d'abscisse et d'ordonnée entre 0 et 9 (compris).
- La méthode `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 `double distance(Point p)` qui retourne la distance entre le point courant et le point en paramètre. Aide : distance entre deux points (x_A, y_A) et (x_B, y_B) : $\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$ (voir la classe `Math` page 53 pour calculer la racine carrée).
- La méthode `deplaceToi(int newx, int newy)` qui déplace le point en changeant ses coordonnées.

Q 20.2 Tester cette classe en écrivant la méthode `main` (d'une classe `TestTriangle`) qui crée 2 points ($p1$ et $p2$), affiche leurs coordonnées et la distance entre les deux points.

Q 20.3 Écrire une classe `Triangle` à 3 variables d'instance prenant leur valeur dans la classe `Point`, avec :

- un constructeur à trois paramètres : les trois sommets du triangle.
- un constructeur sans paramètre.
- une méthode `toString()` qui retourne une chaîne de caractères décrivant le triangle sous la forme : `{(`x_A`,`y_A`);(`x_B`,`y_B`);(`x_C`,`y_C`)}`.
- une méthode `getPerimetre()` qui retourne le périmètre du triangle. Aide : la longueur d'un côté du triangle est la distance entre les deux points de ce côté.

Q 20.4 Ajouter dans la classe `TestTriangle` un troisième point ($p3$) , puis créer un triangle (référencé par une variable $t1$) en utilisant les trois points créés. Afficher le triangle et son périmètre.

Q 20.5 Écrire le constructeur de copie de la classe `Triangle`. Quelle méthode faut-il ajouter dans la classe `Point` et pourquoi ?

Q 20.6 Dans la méthode `main`, créer une copie du triangle référencé par $t1$. Afficher $t1$ et sa copie, puis déplacer le point $p1$, afficher à nouveau $t1$ et sa copie, et vérifier que le premier point de $t1$ a bien changé de coordonnées, mais que par contre le premier point de la copie n'a pas changé de coordonnées.

Q 20.7 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.

Exercice 21 – 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 d'une lettre choisie aléatoirement. Exemples de nom : `PersR`, `PersA`, `PersZ`, `PersU`...

Q 21.1 Une personne a un nom et peut avoir un conjoint. Au départ, une personne est célibataire. Écrire la classe `Personne` avec deux variables d'instance : `nom` (de type `String`) et `conjoint` (de type `Personne`). Ajouter un premier constructeur prenant en paramètre le nom de la personne (à quelle valeur initialiser la variable d'instance `conjoint` ?). Ajouter ensuite un deuxième constructeur sans paramètre qui initialise le nom de la personne à `"Pers"` suivi d'une lettre choisie aléatoirement entre `'A'` et `'Z'`. Écrire aussi la méthode `toString()` qui retourne le nom de la personne auquel est ajouté "célibataire" ou "marié(e)" selon le cas. Exemples : `"PersA, marié(e)"`, `"PersB, célibataire"`.

Q 21.2 Écrire la méthode `void epouser(Personne p)` qui marie cette personne et la personne `p`, et affiche un message de la forme `"PersX, célibataire se marie avec PersY, célibataire"`. Si l'une des 2 personnes est déjà mariée (ou si une personne essaye de se marier avec elle-même ou si `p` est null), le mariage est impossible, on affiche alors un message de la forme `"Le mariage de PersX, marié(e) avec PersY, célibataire est impossible"`.

Q 21.3 Écrire la méthode `void divorcer()` qui fait divorcer cette personne et affiche un message de la forme `"PersX, marié(e) divorce de PersY, marié(e)"`. Si la personne n'a pas de conjoint, alors un message est affiché pour indiquer que ce divorce est impossible .

Q 21.4 Écrire une méthode `main` qui crée trois célibataires `p1`, `p2` et `p3`, puis qui marie `p1` à `p2`, puis `p1` à `p3` (impossible), `p3` à `p1` (impossible), `p3` à lui-même (impossible), puis fait divorcer `p1`, puis fait divorcer `p3` (impossible, `p3` est célibataire). Après chaque groupe d'instructions, afficher les trois personnes pour vérifier que leur situation (marié(e) ou célibataire) est bien cohérente avec les affichages.

TD 4 – Tableaux

Exercice 22 – Base syntaxique

Q 22.1 Donner deux façons pour créer le tableau `tab` d'entiers suivant :

1	2	3

Q 22.2 Donner deux façons pour créer le tableau `mat` d'entiers à deux dimensions ci-contre. Comment obtenir la taille de la premier dimension ? de la deuxième dimension ?

4	5	6
7	8	9

Q 22.3 Soit le code suivant :

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

(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 la boucle `for` (lignes 2 et 3) en utilisant la variante de la boucle `for`. Obtient-on le même résultat ?

Q 22.4 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 22.5 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 23 – Tableau triangulaire

Q 23.1 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

Q 23.2 Afficher ce tableau avec la boucle `for` sans indice. Aide : utiliser `System.out.print(chaine)` pour afficher sans retourner à la ligne et `System.out.println()` pour afficher seulement un retour à la ligne.

Exercice 24 – 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 24.1 Écrire 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.

Questions :

(a) Peut-on allouer la mémoire pour le tableau lors de sa déclaration ?
(b) Peut-on déclarer le constructeur à un paramètre comme étant `private` ? Si oui, quelles sont les conséquences ?
(c) Pourquoi est-il inutile d'ajouter une variable d'instance pour stocker la taille du tableau ?

Q 24.2 On suppose que l'on est dans une méthode `main`, donnez l'instruction pour créer le n-uplet (7,8,9).

Q 24.3 Ajouter à la classe `NUplet` la 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 24.4 On ajoute dans la classe `NUplet` la méthode `toString()` ci-après (à gauche). Quelle est exactement la

chaîne de caractères retournée par cette méthode pour chacun des cas de la colonne de droite ?

```
1 public String toString() {
2         if (tab.length==0) return "()" ;
3         String s="(";
4         for(int i=0; i<(tab.length-1); i++) {
5                 s=s+tab[i]+",";
6         }
7         s=s+tab[tab.length-1];
8         return s+")";
9 }
```

(a) Le tableau `tab` ne contient aucune case.

(b) Le tableau `tab` contient une seule case.
Exemple : | 5 |

(c) Le tableau `tab` contient plusieurs cases.
Exemple : | 7 | 8 | 9 |

Q 24.5 On ajoute dans la classe `NUplet` le constructeur ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème (donnez le diagramme mémoire), puis proposez une solution.

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

```
11 int [] t123={1,2,3};
12 NUplet u1=new NUplet(t123);
13 t123[0]=50;
14 System.out.println(u1.toString());
```

Q 24.6 On ajoute dans la classe `NUplet` la méthode `getTab()` ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème (donnez le diagramme mémoire), puis proposez une solution.

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

```
21 NUplet u2=new NUplet(4,5,6);
22 int [] t456=u2.getTab();
23 t456[0]=70;
24 System.out.println(u2.toString());
```

Q 24.7 Écrire une méthode `boolean egal(NUplet n2)` qui rend vrai si `n2` est égal au n-uplet courant.

Exercice 25 – Représentation mémoire d'objets et de tableaux

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

Q 25.1 (a) Donner la représentation mémoire correspondant à l'exécution du code suivant. (b) Combien d'instances de `Truc` ont été créées à l'issue de l'exécution de ces lignes ?

```
1 Truc t1 = new Truc();
2 Truc t2 = t1;
3 Truc[] tabA = new Truc[3];
4 tabA[0] = new Truc(); tabA[1] = t1 ; tabA[2] = t2;
5 Truc[] tabB = tabA;
6 Truc t3 = tabB[2];
```

Q 25.2 Donner les instructions nécessaires pour dupliquer le tableau `tabA`. Le résultat est-il satisfaisant ?

Quizz 10 – Tableaux (révision)

QZ 10.1 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 10.2 Créer le tableau d'entiers suivant :

1	2	3	
1	2		
1	2	3	4

QZ 10.3 On considère la classe `Bouteille` vue dans l'exercice 9 page 7. Créer un tableau de 2 bouteilles, la première bouteille aura un volume de 3 litres et la deuxième de 1,5 litre.

Quizz 11 – Tableaux d'objets

Soient les déclarations : `int [] tabSimple=new int[5];` et `Integer [] tabObjet=new Integer[5];`.
Qu'affichent les instructions suivantes ?

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

Quizz 12 – Variable `final`

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

QZ 12.1 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 12.2 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) { this.x=x; this.y=y; }
6   public void setZ(double z) { this.z=z; }
7 }
```

TME 4 – Tableaux

Exercice 26 – Utilisation du tableau de la méthode `main` (tableau)

Le but de cet exercice est de comprendre comment utiliser le tableau en paramètre de la méthode ***`main`****.*
Dans la méthode `main` d'une classe `TestTableauMain`, écrire une boucle (sans indice) qui affiche le nombre d'arguments, puis chacun des arguments du programme. Voici 3 exemples d'exécution du même programme :

```
$ java TestTableauMain ab 123 cdef
   Il y a 3 arguments
   args[0]=ab
   args[1]=123
   args[2]=cdef
```

```
$ java TestTableauMain 10 20
   Il y a 2 arguments
   args[0]=10
   args[1]=20
```

```
$ java TestTableauMain
   Il y a 0 arguments
```

Exercice 27 – `VectN` : Vecteur d'entiers de taille n (tableau de type simple)

Q 27.1 On considère des vecteurs d'entiers de taille n, représentés sous la forme $[x_1, x_2, \cdots, x_n]$. Écrire une classe `VectN` qui comporte une (seule) variable d'instance `tab` de type tableau. Elle contient les constructeurs suivants :
- un constructeur `VectN(int n)` qui prend en paramètre le nombre d'éléments du vecteur,
- un constructeur `VectN(int n, int valMax)` qui initialise chaque élément du vecteur de taille n à une valeur aléatoire dans l'intervalle `[0,valMax]` (aide : si besoin, revoir les formules du Quiz 1 page 3),
- un constructeur sans paramètre qui créé un vecteur de 5 éléments initialisés aléatoirement entre 0 et 9,
- un constructeur `VectN(int a, int b, int c)` qui crée le vecteur `[a,b,c]`.

Aide : vérifier que dans chaque constructeur la réservation mémoire pour le tableau a bien été réalisée.
Bonne pratique de programmation : pour chaque constructeur où cela est possible, éviter de répéter des instructions, mais utiliser plutôt un appel à un autre constructeur.

Q 27.2 On veut pouvoir créer seulement des vecteurs initialisés avec des valeurs. (a) Comment faire pour que le constructeur de signature `VectN(int n)` (qui n'initialise pas les valeurs du tableau) ne puisse pas être utilisé pour créer des objets à l'extérieur de la classe ? (b) Dans la méthode `main` d'une classe de test, créer un vecteur avec chaque constructeur pour lequel cela est possible.

Solution (a) : On déclare le constructeur privé.

Q 27.3 Ajouter dans la classe `VectN` les méthodes suivantes, et les tester dans la méthode `main` :
- une méthode `somme` qui renvoie la somme des éléments du vecteur (utiliser la boucle sans indice).
- une méthode `toString` qui retourne une chaîne représentant les valeurs du vecteur sous la forme : `"[x1, x2, x3, ..., xn]"`. Aide : si besoin, voir la méthode `toString()` de la classe `NUplet` Q 24.4.

Q 27.4 Écrire dans la classe `VectN` l'accesseur `int [] getTab()` de la variable `tab`, puis tester dans la méthode `main` le code ci-après. D'après l'affichage obtenu, pensez-vous que c'est une bonne idée d'écrire une méthode qui retourne une variable d'instance de type tableau ? Proposer une solution, puis re-tester le code ci-après pour vérifier que la variable d'instance n'est plus modifiée.

```
VectN vect=new VectN(4,5,6);
int [] t=vect.getTab();
System.out.println("vect="+vect);
t[0]=100; // modif d'une valeur de t
System.out.println("vect="+vect);
```

Exercice 28 – Pile d'assiettes (classe contenant un attribut de type tableau d'objets)

Une pile est une structure de données classique en informatique, dont le principe est que le dernier élément inséré dans la pile doit être le premier élément retiré de la pile (principe LIFO pour "Last In, First Out").
On veut modéliser une pile d'assiettes telle que :

- on ne peut ajouter une assiette dans la pile que sur le sommet de la pile.
- on ne peut retirer de la pile que l'assiette qui est sur le sommet de la pile.
- une pile ne peut contenir qu'un nombre limité ($tailleMax$) d'assiettes.
- à tout moment, il y a nbA assiettes présentes dans la pile avec $0 \leq nbA \leq tailleMax$.

On veut écrire une classe `Pile` permettant de gérer une pile d'objets de type `Assiette` au moyen d'un tableau.

Q 28.1 Définir la classe `Assiette` qui contient une variable d'instance `diametre` de type `int`, un constructeur à un paramètre, un constructeur sans paramètre qui initalise le diamètre de l'assiette à 26 cm et une méthode `toString()` qui retourne pour une assiette de 30 cm : `"Assiette 30 cm"`.

Q 28.2 Écrire une classe `Pile` avec seulement 2 attributs (le tableau et le nombre nbA d'assiettes présentes dans la pile), un constructeur qui a comme paramètre la taille maximale de la pile (au départ, la pile est vide), ainsi que les opérations données ci-dessous.
Attention : ne pas confondre le nombre d'assiettes présentes dans la pile et la taille maximale de la pile.
Attention : vous ne devez pas avoir d'attributs pour la taille maximale de la pile, utilisez la taille du tableau.
La pile devra avoir les opérations suivantes :

- `boolean estVide()` qui indique si la pile est vide. Aide : la pile est vide si le nombre d'assiettes est 0.
- `boolean estPleine()` qui indique si la pile est pleine.
- `void empiler(Assiette a)` qui, si possible, ajoute l'élément en paramètre au sommet de la pile. Aide : inutile d'utiliser une boucle, nbA indique le sommet de la pile.
- `Assiette depiler()` qui, si possible, retire l'élément au sommet de la pile et le retourne.
- `String toString()` qui retourne une chaîne représentant le contenu de la pile, à raison d'une assiette par ligne, le sommet de pile étant la première valeur affichée. Attention : ne pas afficher les cases vides.

Q 28.3 Dans la méthode `main` d'une classe `TestPile`, créer une pile pouvant contenir au maximum 3 assiettes, puis empiler trois assiettes, dépiler une fois, puis empiler deux autres assiettes (la pile sera pleine), puis dépiler 4 fois (la pile sera vide). Afficher le contenu de la pile après chacune de ces opérations.

Exercice 29 – 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 :

$$C_j^i = \begin{cases} 1 & \text{si } j = 0 \text{ ou si } j = i \\ C_{j-1}^{i-1} + C_j^{i-1} & \text{sinon.} \end{cases}$$

Q 29.1 On veut écrire une classe `TrianglePascal` qui réserve uniquement la place mémoire nécessaire pour stocker le triangle. Écrire la classe `TrianglePascal` avec :

- un attribut de type tableau d'entiers à 2 dimensions
- un constructeur `TrianglePascal(int n)` qui prend en paramètre le nombre n de lignes du triangle et dont le but est de réserver uniquement la place mémoire nécessaire pour stocker le tableau triangulaire
- une méthode `remplirTriangle()` qui remplit les cases du tableau avec les valeurs du triangle de Pascal
- et une méthode `toString()` qui retourne la chaîne représentant le tableau sous la forme d'un triangle.

Q 29.2 Écrire une classe `TestTrianglePascal` qui crée et affiche plusieurs instances de la classe `TrianglePascal`.

TD 5 – Variables et méthodes de classes

Exercice 30 – Membres d'instance ou de classe

Q 30.1 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`
- `nbChenils`
- `getNbChiens()`
- `getNbChenils()`

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

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

Exercice 31 – Compter les trucs (compteur, variables d'instance et variables de classe)

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() {
13                return cpt;
14        }
15        public int getNum() {
16                return num;
17        }
18 }
```

Q 31.1 Quel est le nom de la variable de classe ? Comment la reconnaît-on ?

Q 31.2 Pourquoi la variable cpt a-t-elle été initialisée lors de sa déclaration (et pas dans le constructeur) ?

Q 31.3 On suppose que l'on est dans la méthode `main` d'une classe `Test`. **(a)** Peut-on écrire une instruction pour afficher la valeur de la variable `cpt` **sans utiliser d'objet** ? Si oui, écrire cette instruction. **(b)** Même question pour la variable `num`.

Q 31.4 On suppose toujours que l'on est dans la méthode `main` d'une classe `Test`. Quel est l'affichage obtenu par l'exécution du programme ci-dessous ? Faire un diagramme mémoire.

```
1 System.out.println(Truc.getCpt());
2 Truc n1=new Truc();
3 System.out.println(Truc.getCpt()+"␣"+n1.getNum());
4 Truc n2=new Truc(25);
5 System.out.println(Truc.getCpt()+"␣"+n1.getNum()+"␣"+n2.getNum());
6 Truc n3=new Truc();
7 System.out.println(Truc.getCpt()+"␣"+n1.getNum()+"␣"+n2.getNum()+"␣"+n3.getNum());
```

Exercice 32 – 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 32.1 A-t-on commis une *faute de conception* en déclarant plusieurs attributs comme public ? Justifier.

Q 32.2 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)
22       System.out.println("v1 egale v2");
23 if(v1.id == v2.id)
24       System.out.println("les points ont le même identifiant");
25 System.out.println("Compteur (1): "+v1.getCpt());
26 System.out.println("Compteur (2): "+Vecteur.getCpt());
27 System.out.println("Compteur (3): "+v1.cpt);
```

Exercice 33 – 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 `getIP()` retourne une chaîne de caractères correspondant à 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 33.1 Écrire la classe `GenerateurIP` qui contiendra les variables et méthodes suivantes :
- `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
- un constructeur privé et sans paramètre qui ne fait rien. A quoi cela sert-il de déclarer ce constructeur privé ?
- une méthode de classe `String getIP()` 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 33.2 Dans une méthode `main` d'une classe `TestGenerateurIP`, afficher 257 adresses IP pour vérifier que votre méthode fonctionne. Peut-on créer des instances de cette classe ?

Exercice 34 – 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 34.1 Ajouter à la classe `Vecteur` une méthode d'instance qui fait la somme de deux vecteurs.

Q 34.2 Ajouter à la classe `Vecteur` une méthode de classe `qui` fait la somme de deux vecteurs.

Q 34.3 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.

Quizz 13 – Compter les cercles (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 public class TestCercle {
15  public static void main(String [] args){
16      Cercle c=new Cercle(3);
17      System.out.println(EXPRESSION);
18  }
19 }
```

QZ 13.1 Barrer les réponses qui provoquent une erreur à la compilation, si dans la classe `TestCercle`, on remplace `EXPRESSION` par :

`c.PI`	`c.nbCercles`	`c.numero`
`c.rayon`	`c.surface()`	`c.getNbCercles()`

QZ 13.2 Barrer les réponses qui provoquent une erreur à la compilation, si dans la classe `TestCercle`, on remplace `EXPRESSION` par :

`Cercle.PI`	`Cercle.nbCercles`	`Cercle.numero`
`Cercle.rayon`	`Cercle.surface()`	`Cercle.getNbCercles()`

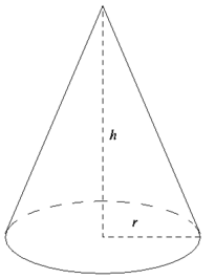
QZ 13.3 On suppose que l'on est dans le `main` d'une classe `Test2Cercle`. Quel est l'affichage obtenu ?

```
50 System.out.println(Cercle.getNbCercles());
51 Cercle c1=new Cercle(10);
52 Cercle c2=new Cercle(15);
53 Cercle c3=c2;
54 Cercle c4=new Cercle(30);
55 System.out.println(Cercle.getNbCercles());
56 System.out.println(c1.getNbCercles());
```

TME 5 – Variables et méthodes de classes

Exercice 35 – 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.



Q 35.1 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 35.2 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 du cône : $\frac{1}{3}\pi r^2 h$
- la méthode `String toString()` qui retourne une chaîne de caractères qui, pour un cône de rayon 5.4 et de hauteur 7.2, a le format : `"Cone r=5,4cm h=7,2cm de volume 219,9cm3"`. Aide : utiliser (une seule fois) la méthode `static` appelée `format` de la classe `String` (voir la partie Syntaxe de l'Annexe A) pour garder seulement 1 chiffre après la virgule pour tous les nombres réels.
- l'accesseur de la variable `nbCones` (aide : comment devrait être déclaré l'accesseur d'un attribut static ?)

Q 35.3 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 36 – Chaines aléatoires (méthodes de classe)

Q 36.1 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 : si besoin, revoir les formules du Quiz 1 page 3.
- 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 36.2 Pour quelle raison les méthodes `lettre()` et `chaine()` sont-elles des méthodes de classes ?

Q 36.3 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'instances de cette classe dans une autre classe.

Q 36.4 (a) Dans la méthode `main` de la classe `Alea`, afficher le résultat retourné par la méthode `chaine()`. Est-ce utile de préfixer l'appel de la méthode par `Alea` ? **(b)** Même question mais dans la méthode `main` d'une classe `TestAlea`. **(c)** Vérifier que l'on ne peut pas créer d'instance de `Alea` dans `TestAlea`.

Exercice 37 – Projet avec trio de personnes (static)

Q 37.1 Classe `Personne`

Q 37.1.1 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 créées depuis le début du programme,
- un constructeur sans paramètre qui initialise le nom de la personne à `Individu` suivi du nombre,
- l'accesseur `getNbPersonnes()`,
- une méthode `toString()` qui retourne le nom de la personne.

Q 37.1.2 Dans la méthode `main` d'une classe `TestProjet.java`, commencer par afficher le nombre de personnes créées depuis le début du programme (cela doit afficher 0), puis créer deux personnes et afficher-les (vérifier que

le nom de la première personne est `Individu1` et non pas `Individu0`), enfin, afficher à nouveau le nombre de personnes créées depuis le début du programme (cela doit afficher 2).

Q 37.1.3 On veut modifier la classe `Personne` 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`...

— Pour cela, ajouter dans votre classe `Personne` une variable `lettre` initialisée à `'A'` (qui devra prendre successivement les valeurs `'A'`, `'B'`, `'C'`...),

— puis modifier légèrement le constructeur pour générer les noms demandés.

Vérifier que l'affichage obtenu par votre méthode `main` (inutile de la modifier) ressemble à celui ci-contre.

```
nbPersonnes=0
IndividuA
IndividuB
nbPersonnes=2
```

Q 37.2 Un trio est composé de 3 personnes (tableau de 3 personnes), il contient également une variable `numero` déterminée 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 37.3 Un projet est composé d'un nom de projet et d'un objet `Trio`. Écrire 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 37.4 Ajouter dans le `main` de la classe `TestProjet`, les instructions nécessaires pour obtenir l'affichage :

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

Q 37.5 (optionnel) Si vous avez fait la classe `Alea` de l'exercice 36, ajouter dans la classe `Projet` un constructeur sans paramètre qui initialise le nom du projet avec une chaîne aléatoire en utilisant la méthode `chaine()` de la classe `Alea`. Puis tester ce constructeur en créant et en affichant un nouveau projet dans le `main`.

Q 37.6 Dans les classes `Personne`, `Trio` et `Projet`, ajouter (si nécessaire) des variables et méthodes pour afficher dans la méthode `main` le nombre de personnes créées, le nombre de trios créés et le nombre de projets créés.

TD 6 – Héritage et modélisation

Exercice 38 – 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 qui ont des roues, 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 39 – 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 39.1 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 39.2 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 39.3 (a) 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 ? **(b)** Pourquoi la méthode `getNbEnfants()` a-t-elle été déclarée `protected` dans la classe `Personne` ? Pourquoi l'attribut `nbEnfants` n'a-t-il pas été déclaré `protected` à la place de la méthode ?

Q 39.4 Écrire une méthode `modifierNumTel(String numTel)` qui permet de modifier le numéro de téléphone du salarié, et qui affiche, par exemple, pour le salarié `Albert` : `"Le salarié Albert a pour numéro 012345678"`.

Q 39.5 Trouver et expliquer les erreurs dans les instructions ci-dessous :

```
1 Personne p = new Personne("Albert");
2 p.ajouterEnfant();
3 p.prime();
4 p.estEnL2();
5
6 Etudiant e = new Etudiant("Ahmed",null,"L2");
7 e.ajouterEnfant();
8 e.prime();
9 e.estEnL2();
10
11 Salarie s1 = new Salarie("Amelle");
12 Salarie s2 = new Salarie("Pauline","0122334455");
```

Exercice 40 – Botanique (héritage et redéfinition de méthodes)

Q 40.1 Dessiner le diagramme de classes pour les classes suivantes. Précisez sur le diagramme les méthodes `toString()`.

```
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 { }
```

Q 40.2 (a) Est-ce que la classe `Plante` hérite d'une classe ? **(b)** Est-ce que la méthode `toString()` de `Plante` redéfinit une méthode ?

Q 40.3 Qu'affiche les instructions suivantes ? En tirer des conclusions sur l'héritage et la redéfinition de méthode.

```java
Plante p = new Plante(); System.out.println(p);
Arbre a = new Arbre(); System.out.println(a);
Fleur f = new Fleur(); System.out.println(f);
Marguerite m = new Marguerite(); System.out.println(m);
Chene c = new Chene(); System.out.println(c);
Rose r = new Rose(); System.out.println(r);
```

Q 40.4 Qu'affiche les instructions suivantes ? Rappel : le corps de méthode appelé est celui de l'objet, et non pas celui du type de la variable qui référence l'objet.

```java
    Plante pA = new Arbre(); System.out.println(pA);
    Plante pF = new Fleur(); System.out.println(pF);
    Plante pM = new Marguerite(); System.out.println(pM);
    Plante pR = new Rose(); System.out.println(pR);
    Plante pC = new Chene(); System.out.println(pC);
```

TME 6 – Héritage et modélisation

Exercice 41 – 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 41.1 Dessiner le diagramme de classes.

Q 41.2 Écrire une classe `Instrument` contenant 2 variables d'instance privées de type `int` pour stocker le poids et le prix de l'instrument. Munir la classe d'un constructeur à 2 paramètres pour initialiser les variables d'instance, ainsi que de la méthode `toString()` qui retourne par exemple : `"poids : 2 kg, prix : 500 euros"`.

Q 41.3 Un piano a un nombre de touches. Une guitare a un type (par exemple, `"classique"` pour une guitare classique). Écrire les classes `Piano` et `Guitare` avec une méthode `toString()` qui retournera par exemple : `"Piano 88 touches poids : 300 kg, prix : 700 euros"`. Ces classes comporteront également une méthode `public void jouer()` qui affichera par exemple pour `Guitare` : `"La guitare classique joue"`. Dans la méthode `main` d'une classe `TestOrchestre`, créer une guitare et un piano et les afficher.

Q 41.4 Un orchestre est composé d'un tableau d'instruments qui peut contenir au maximum `max` instruments. Écrire la classe `Orchestre` correspondante, avec aussi une variable pour stocker le nombre d'instruments courant et une méthode `ajouterInstrument(Instrument x)` qui, quand c'est possible, ajoute un instrument à l'orchestre.

Q 41.5 Ajouter à la classe `Orchestre` une méthode `jouer()` qui fait jouer l'ensemble de ses instruments.

a) Quel est le problème dans le code actuel ?

b) Pour essayer de résoudre le problème, écrire dans la classe `Instrument` une méthode `jouer()` qui affiche `"L'instrument joue"`. Dans le main, créez un orchestre, ajoutez-y la guitare et le piano, puis faire jouer l'orchestre. `"L'instrument joue"` n'est jamais affiché. Expliquez pourquoi. Aide : voir Q 40.4.

Solution a) : Pas de méthode `jouer` dans `Instrument` (même s'il y a une méthode `jouer` dans l'objet).

Q 41.6 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.* trompette, batterie...), quelle(s) classe(s) doit-on modifier ?

Exercice 42 – 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 42.1 Écrire le diagramme de classes en indiquant les attributs, puis ajouter au fur et à mesure de l'exercice les méthodes (pour simplifier le diagramme, ne pas préciser les constructeurs, ni les méthodes `toString`).

Q 42.2 Écrire le code java des classes `Vehicule`, `AMoteur`, et `SansMoteur` avec tous les constructeurs nécessaires et les méthodes `toString()`. La méthode `toString()` de `Vehicule` retournera par exemple pour le premier

véhicule créé : `"1 de marque MyVTT"`. Rappel : en général, la méthode `toString()` des classes filles contient l'appel à la méthode `toString()` de la classe mère : `super.toString()`.

Q 42.3 Écrire la classe `Velo` avec un constructeur, 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 42.4 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 42.5 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 42.6 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 42.7 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 42.8 Écrire la classe `Camion` avec un constructeur, une méthode `toString()` et aussi 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 42.9 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 42.10 Peut-on factoriser la déclaration de la méthode `transporter` ? Si oui, à quel niveau ?

TD 7 – Héritage et classe abstraite

Exercice 43 – 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.

On considère les classes `Mammifere` et `Chien` suivantes :

```
1  public class Mammifere { }
2  public class Chien extends Mammifere {
3      public void aboyer() { System.out.println("Ouaff"); }
4  }
```

Q 43.1 Pour chaque instruction ci-après, indiquez si il se produit une erreur à la compilation ? à l'exécution ? Faire un diagramme mémoire et expliquer chaque erreur.

```
11 Chien c1 = new Chien();
12 Mammifere m1 = c1;
13 Chien c2 = m1;
14 Chien c3 = (Chien) m1;
15 m1.aboyer();
16 ((Chien)m1).aboyer();
17 Mammifere m2 = new Mammifere();
18 Chien c4 = (Chien) m2;
```

Q 43.2 Est-ce que le code suivant peut produire une erreur à la compilation ? à l'exécution ? Si oui, quel est le problème et comment le corriger ? Aide : quel est le type de l'objet à la ligne 26 ?

```
21 Mammifere m3 ;
22 if (Math.random()<0.5)
23     m3=new Chien();
24 else
25     m3=new Mammifere();
26 ((Chien)m3).aboyer();
```

Exercice 44 – 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() {
6       return "Position : (" + x + "," + y + ") Surface : "+surface() ;
7     }
8     public abstract double surface() ;
9 }
10
11 public class Circle extends Shape {
12     private double radius ;
13     public Circle() {
14       super();   // pas obligatoire (appel implicite) mais très recommandé
15       radius = 1 ;
16     }
17     public Circle(double x, double y, double r) {
18       super(x,y) ;
19       radius = r ;
20     }
21     public String toString() {
22       return "Circle Radius : "+radius+" "+super.toString() ;
23     }
24 }
25
26 public class MainShape {
27     public static void main(String [] args) {
28       Circle c1 = new Circle(1,1,3) ;
29       Circle c2 = new Circle() ;
30       System.out.println(c1.toString() + "\n" + c2.toString());
31     }
32 }
```

Q 44.1 A quels membres (variables et méthodes) de `Shape` la classe `Circle` a-t-elle accès ?

Q 44.2 La compilation de la classe `Circle` échoue, expliquer pourquoi.

Q 44.3 (a) Pourquoi la méthode `surface()` a-t-elle était déclarée abstraite dans la classe `Shape` ? **(b)** La méthode `surface()` étant abstraite dans `Shape`, peut-on quand même l'utiliser dans `Shape` (voir ligne 6) ? **(c)** Pour que la classe compile, ajouter une méthode `surface()` dans `Circle`. **(d)** A la ligne 30, la surface des cercles est-elle affichée ?

Q 44.4 Créer une classe `Rectangle` qui hérite de `Shape`.

Q 44.5 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 45 – Une corbeille de fruits (`ArrayList`, `abstract` et `instanceof`)

Soient les classes suivantes :

```
1 public abstract class Fruit {
2         public abstract void afficher();
3 }
4 public class Kiwi extends Fruit {
5         public void afficher() { System.out.println("kiwi"); }
6         public void methKiwi() { System.out.println("Méthode de Kiwi"); }
7 }
8 public class Pomme extends Fruit {
9         public void afficher() { System.out.println("pomme"); }
10 }
```

On veut modéliser une corbeille de fruits comme une liste au sens `ArrayList` de fruits.

Q 45.1 Soit l'instruction : `import java.util.ArrayList;` A quoi sert cette instruction ? Dans quel(s) fichier(s) faut-il placer cette instruction ? A quel endroit dans le fichier ?

Q 45.2 Écrire la classe `Corbeille` qui possède une seule variable d'instance appelée `liste` qui est de type `ArrayList` de `Fruit` (voir la documentation de la classe `ArrayList` à la page 53). 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 `Kiwi` et 50% d'objets de type `Pomme`, et les ajoute à la liste.

Q 45.3 Ajoutez à la classe `Corbeille` une méthode `lister()` qui appelle la méthode `afficher()` de chacun des objets de la liste (utilisez une boucle avec indice). Est-ce qu'il est nécessaire de faire un cast ?

Q 45.4 Ajoutez à la classe `Corbeille` une méthode `methK()` qui pour chaque objet de la liste qui est de type `Kiwi` appelle la méthode `methKiwi()` (utilisez une boucle sans indice). Est-ce qu'il est nécessaire de faire un cast ?

Q 45.5 Ajoutez à la classe `Corbeille` une méthode `int compterPommes()` qui retourne le nombre de pommes.

Exercice 46 – Retro engineering

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

```
1 Expression v1=new Valeur(4.);
2 Expression v2=new Valeur(1.);
3 Expression v3=new Valeur(7.);
4 Expression v4=new Valeur(5.);
5 Expression v5=new Valeur(3.);
6 Expression v6=v5;
7 Operation p1=new Plus(v1,v2);
8 Operation m2=new Moins(v3,v4);
9 Operation mult=new Multiplie(p1,v5);
10 Operation p2=new Plus(v6,mult);
11 Operation d=new Divise(p2,m2);
12 System.out.println(d+"="+d.getVal());
```

Q 46.1 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 46.2 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 46.3 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 46.4 Donner le diagramme de l'état de la mémoire à la fin du programme (ligne 13).

Q 46.5 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 :

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

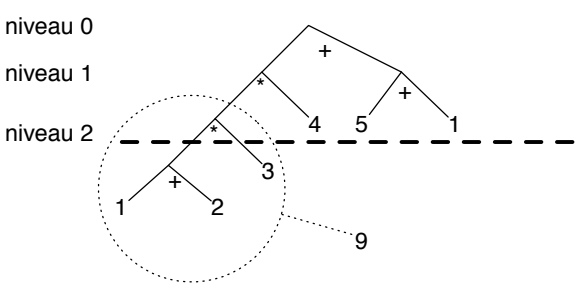
En l'état, le programme ne compile pas. Pourquoi ? Donner deux manières de remédier au problème. Discuter brièvement des avantages / inconvénients de ces deux manières de faire.

Q 46.6 Quel problème survient dans l'exécution du programme si l'on remplace la ligne 5 par :

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

Q 46.7 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 46.8 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 47 – `final` : les différentes utilisations

Q 47.1 Questions de cours

Q 47.1.1 A quoi sert un attribut `final` ? Où peut-il être initialisé ? Citer des cas d'utilisation.

Q 47.1.2 Dans quel cas déclarer une méthode comme `final` ?

Q 47.1.3 Dans quel cas déclarer une classe comme `final` ?

Q 47.1.4 Etant donné les usages répertoriés ci-dessus, à quoi sert le mot clé `final` en général ?

Q 47.2 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 47.2.1 Au niveau des attributs, serait-il intéressant d'ajouter le modifier `final` sur certains champs ? Pourquoi ?

Q 47.2.2 A quelle condition pourrait-on mettre `x` et `y` en mode `final` ? Proposer une solution pour conserver les fonctionnalités de la classe.

Q 47.2.3 Quelles fonctions pourraient être `final` ? Quel serait l'intérêt de la manipulation ?

Q 47.2.4 Quel serait l'intérêt de déclarer la classe `final` ? Cela empêche-t-il tout enrichissement futur ?

Q 47.2.5 Proposer un code pour la classe `PointNomme` (point ayant un attribut `nom`) après avoir déclaré Point en `final`.

Exercice 48 – Trousse de stylos (attribut de type `ArrayList`)

Une trousse peut contenir des stylos. On veut modéliser une trousse comme une liste au sens `ArrayList` de `Stylo`. On considère la classe `Stylo` suivante.

```
public class Stylo {
        private String couleur;
        public Stylo(String couleur) { this.couleur=couleur; }
        public String getCouleur() { return couleur; }
        public String toString() { return "Stylo "+couleur; }
}
```

Q 48.1 Soit l'instruction : `import java.util.ArrayList;` A quoi sert cette instruction ? Dans quel(s) fichier(s) faut-il placer cette instruction ? A quel endroit dans le fichier ?

Q 48.2 Écrire une classe `Trousse` avec un seul attribut de type `ArrayList` de `Stylo`, ainsi que les méthodes :
- méthode pour ajouter un stylo dans la trousse
- méthode qui retourne le nombre de stylos présents dans la trousse
- méthode `toString` (utiliser une boucle for SANS indice)

et tester ces méthodes dans une méthode `main` (une documentation de la classe `ArrayList` se trouve page 53).

Q 48.3 Ajouter dans la classe `Trousse` les méthodes suivantes, et les tester dans un `main` :
- `int compter(String coul)` qui retourne le nombre de stylos de couleur `coul` présents dans la trousse (utiliser une boucle AVEC indice)
- une méthode qui extrait le premier stylo de la trousse.

Quizz 14 – Classe et méthode abstraite

QZ 14.1 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 14.2 La classe suivante est-elle correcte ? Expliquez.

```
11 public class Z {
12         public abstract void f1();
13         public void f2() { }
14 }
```

QZ 14.3 La classe suivante est-elle correcte ? Expliquez.

```
21 public abstract class Z {
22         public abstract void g() { } ;
23         public void h();
24 }
```

QZ 14.4 Les instructions suivantes sont-elles correctes ? Expliquez et proposez deux solutions.

```
31 public abstract class A {
32             public abstract void f();
33 }
34 public class B extends A {}
```

Quizz 15 – Vocabulaire sur l'héritage

En utilisant quelques verbes de l'ensemble ci-après, écrire 3 courtes phrases caractérisant l'héritage : implémenter, instancier, importer, réemployer, ajouter, encapsuler, étendre, spécifier, redéfinir. L'héritage permet de ...

TME 7 – Héritage et classe abstraite

Exercice 49 – Ménagerie (héritage, classes et méthode abstract, tableaux)

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 49.1 Dessiner le diagramme de classes UML. Préciser les classes abstraites.

Q 49.2 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 49.3 On suppose qu'une vache a 4 pattes et qu'elle produit tous les jours la même quantité de lait (cette quantité est un nombre réel choisi aléatoirement entre 5 et 30 (non-compris) pour chaque vache). Écrire la classe `Vache` et sa classe mère avec notamment leur méthode `toString()`. Bien réfléchir à la position de chaque attribut.

Q 49.4 Écrire la classe `Boa` et sa classe mère.

Q 49.5 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 qui compte le nombre d'éléments actuellement présents dans le tableau.

Q 49.6 Ajouter une méthode `void midi()` qui fait crier tous les animaux de cette ménagerie. Ajouter aussi une méthode `vieillirTous()` qui fait vieillir d'un an tous les animaux de cette ménagerie.

Q 49.7 Écrire la méthode `main` qui crée une ménagerie, lui ajoute quelques animaux, les affiche avec leur âge, déclenche la méthode `midi()` et les fait vieillir d'un an.

Exercice 50 – 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 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(nombre)` qui retourne la racine carrée d'un nombre (voir annexe page 53).

Q 50.1 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 50.2 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 50.3 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 ?

TD 8 – Héritage et interface

Exercice 51 – 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 que l'on va représenter par une interface que l'on appelle `Submarine`. 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 51.1 Dessiner le diagramme de classe pour les classes `Animal`, `Chat`, `Dauphin`, `Mammifere`, `Merlu`, `Poisson`, `SousMarin` et l'interface `Submarine`.

Q 51.2 Écrire l'interface `Submarine` qui contient une méthode `seDeplacer()`. L'instruction suivante est-elle correcte ? `Submarine sub=new Submarine();`

Q 51.3 Les poissons forment une famille d'animaux qui peuvent se déplacer sous l'eau. Écrire la classe `Poisson`.

Q 51.4 Soit la classe : `public class Merlu extends Poisson {}` et l'instruction : `Merlu m=new Merlu();`
Un merlu a-t-il la propriété de nager sous l'eau ? Quelle est la valeur de l'expression : `m instanceof Submarine` ?

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

Q 51.6 Un sous-marin peut se déplacer sous l'eau et faire de l'écholocation. Écrire la classe `SousMarin`.

Q 51.7 Un océan peut contenir des `Submarine`. **(a)** Ajouter sur le diagramme de classe la classe `Ocean`. **(b)** Peut-on déclarer une `ArrayList` dont le type entre chevrons soit une interface ? La réponse est oui. Écrire une classe `Ocean` qui contient un attribut de type `ArrayList` de `Submarine`, qui contient une méthode pour ajouter un élément et une autre méthode pour déplacer tous les éléments. Voir la doc de `ArrayList` page 53.

Q 51.8 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 51.9 (a) Peut-on ajouter un chat dans l'océan ? (b) 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 52 – Interface `AvecReservoir`

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 52.1 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 52.2 Vérifier que le code suivant fonctionne avec la solution de la question précédente.

```
30       Vehicule [] tab={new Velo(), new Voiture()};
31       StationService station=new StationService();
32       for(Vehicule v : tab) {
33          v.rouler();
34          station.remplirReservoir(v);
35       }
```

Q 52.3 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 et modifiez la méthode `remplirReservoir` en conséquence (en particulier, quel doit être le type du paramètre de cette méthode ?).

Q 52.4 Créer un tableau avec un vélo, une voiture et une tondeuse. Quel doit être le type du tableau ? Puis faire le plein de tous les éléments du tableau à la station service.

Exercice 53 – Bols colorés (redéfinition de la méthode `equals`)

Rappel : la méthode `boolean equals(Object obj)` est définie dans `Object` (comme `toString()`) et réalise l'égalité référentielle. Cet exercice aborde quelques unes des difficultés rencontrées quand on redéfinit cette méthode.

```
1 public class Bol {
2         private int diametre;
3         public Bol(int diametre) {
4                 this.diametre=diametre;
5         }
6         public boolean equals(Bol b) {
7                 return diametre==b.diametre;
8         }
9 }
```

On considère la classe `Bol` précédente et les déclarations de variables suivantes :

```
20 Bol b1=new Bol(10);
21 Bol b2=new Bol(10);
22 Object objB2=b2;
```

Q 53.1 (a) Combien y a-t-il de méthodes `equals` accessibles à partir d'une **variable** de type `Bol` (par exemple, à partir de la variable `b1`) ? Expliquer pourquoi. (b) Quelle est la valeur de chacune des expressions ci-dessous ? Pour les expressions avec `equals`, préciser quelle est la méthode `equals` appelée et expliquer pourquoi.

- `b1==b2`
- `b1.equals(b2)`
- `b1.equals(objB2)`
- `objB2.equals(b1)`

Q 53.2 On veut que 2 **objets** de type `Bol` soient égaux s'ils ont le même diamètre, et même si les variables qui référencent ces objets ne sont pas de type `Bol`. Pour l'instant, est-ce le cas ? Comment résoudre ce problème ?

Q 53.3 On modifie la signature de la méthode `boolean equals(Bol b2)` de la classe `Bol` pour que ce soit une redéfinition de la méthode `equals` de `Object`. (a) Donner le code de cette nouvelle version de la méthode en utilisant `instanceof`. Quel doit être obligatoirement la signature ? (b) Refaire maintenant la question Q 53.1.

Héritage et `equals` On ajoute maintenant la classe suivante correspondante à un bol coloré :

```
30 public class BolColore extends Bol {
31         private String couleur;
32         public BolColore(int diam,String c) {
33                 super(diam);
34                 couleur=c;
35         }
36 }
```

et les déclarations de variables suivantes :

```
40 BolColore bc1=new BolColore(10,"gris");
41 BolColore bc2=new BolColore(10,"vert");
```

Q 53.4 Quelle est la valeur des expressions ci-après ? Est-ce que c'est ce qui est attendu ?

- `bc1.equals(bc2)`
- `bc1.equals(b1)`
- `b1.equals(bc1)`

Q 53.5 (a) Écrire la méthode `equals` de `BolColore` en utilisant `instanceof`. Quel doit être obligatoirement sa signature ? On supposera que la variable `couleur` ne peut pas être `null`. (b) Refaire maintenant question Q 53.4.

Q 53.6 On suppose maintenant que l'attribut `couleur` de `BolColore` peut être `null`. (a) Quel est le résultat de l'expression `couleur.equals(((BolColore)obj).couleur)` si `couleur` est `null` ? (b) Modifier la méthode `equals` de `BolColore` pour gérer le cas où `couleur` peut être `null`.

Q 53.7 (a) On veut maintenant que deux bols soient égaux si ils sont du même diamètre et du même type. Comment faire ? Modifier la méthode `equals` de `Bol` en conséquence. (b) Refaire maintenant la question Q 53.4.

Pour aller plus loin... En Java, la méthode `equals` doit suivre un certain nombre de spécifications pour être vraiment correcte. En particulier, il faut définir une relation d'équivalence (réflexivité, symétrie, transitivité) et aussi redéfinir la méthode `hashCode()`. Pour plus d'informations, voir la documentation de la méthode `equals` de la classe `Object` dans la Javadoc.

Exercice 54 – Roulant, Volant, Flottant (héritage d'interfaces)

Nous souhaitons gérer une grande liste de véhicules, 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 54.1 Donner la hiérarchie d'interface à créer.

Q 54.2 Donner la signature des classes `Voiture` et `Hydravion`, ainsi que les méthodes à coder impérativement.

TME 8 – Héritage et interface

Exercice 55 – 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 55.1 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         |
```

Classe Arbre On vous donne la classe `Arbre` suivante qui ne contient pas de difficultés :

```
public class Arbre {
    private String nom;
    public Arbre(String nom) { this.nom=nom; }
    public String toString() { return nom; }
}
```

Q 55.2 Écrire 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 55.3 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 55.4 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 55.5 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 doivent avoir une méthode qui retourne le poids de l'objet. Écrire les instructions nécessaires, dont la méthode `ramasserTout`.

Q 55.6 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 55.7 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 retourne le nombre d'objets dans le panier qui sont toxiques
- une méthode `toString()` qui retourne par exemple `"Panier contenant 5 objets, dont 2 toxiques (7,1kg sur 8,0kg)"`.

Q 55.8 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. Dans le main, créer un panier de 8kg, ramasser avec le panier des objets dans la forêt, puis afficher le panier.

TD/TME 9 – Héritage et liaison dynamique

Exercice 56 – Des fourmis à tous les étages

```
1 public class Fourmi {
2     protected String nom;
3     public Fourmi(String nom) {
4         this.nom = nom;
5     }
6 }
7 public class Ouvriere extends Fourmi {
8     public Ouvriere(String nom) {
9         super(nom);
10    }
11 }
```

```
12 public class Reine extends Fourmi {
13    private int cpt;
14    public Reine(String nom) {
15        super(nom);
16        cpt=0;
17    }
18    public Fourmi engendrer() {
19        cpt ++;
20        return new Ouvriere(nom+cpt);
21    }
22 }
```

Q 56.1 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.

```
23 Fourmi f1    = new Fourmi("f1");
24 Fourmi f2    = new Ouvriere("ouv1");
25 Ouvriere f3 = new Ouvriere("ouv2");
26 Fourmi f4    = new Reine("majeste1");
27 Ouvriere f5 = new Reine("majeste2");
28 Reine f6     = new Reine("majeste3");
```

```
29 Fourmi[] fourmilliere = new Fourmi[100];
30 fourmilliere[0]= f4.engendrer();
31 fourmilliere[1]= f5.engendrer();
32 fourmilliere[2]= f6.engendrer();
33 fourmilliere[3]= ((Reine) f2).engendrer();
34 fourmilliere[4]= ((Reine) f4).engendrer();
35 fourmilliere[5]= ((Reine) f6).engendrer();
```

Q 56.2 Sélection de méthodes. On souhaite maintenant nourrir nos fourmis... en fonction de leur hiérarchie.

```
36 public class Nourriture {
37     private String description;
38     public Nourriture(String description) {
39         this.description = description;
40     }
41     public String toString() {
42         return description;
43     }
44 }
45 public class GeleeRoyale extends Nourriture{
46     public GeleeRoyale(){
47         super("gelee");
48     }
49 }
```

```
50 public class Fourmi {
51     ...
52     public void manger(Nourriture n) {
53         System.out.println(nom+"␣mange␣"+n);
54     }
55 }
56 public class Reine extends Fourmi {
57     ...
58     public void manger(GeleeRoyale g) {
59         System.out.println(nom+ "␣(Reine)␣
               mange␣"+g);
60     }
61 }
```

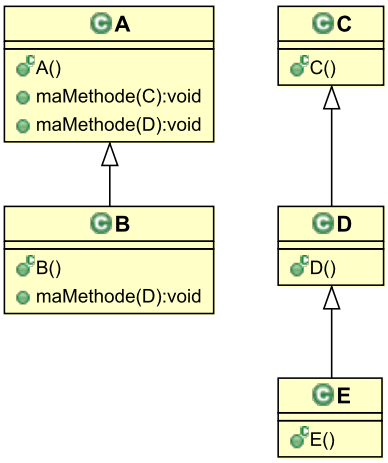
Donner les affichages lors de l'exécution du code suivant :

```
62 Reine r1 =  new Reine("majeste1");
63 r1.manger(new Nourriture("sucre"));
64 r1.manger(new GeleeRoyale());
```

```
65 Fourmi r2 =  new Reine("majeste2");
66 r2.manger(new Nourriture("viande"));
67 r2.manger(new GeleeRoyale());
```

Exercice 57 – Sélection de méthode

Q 57.1 Soit une hiérarchie de classes (figure ci-dessous). 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 57.2 Conversions implicites (ou pas). On envisage les 3 cas d'ajouts de méthodes suivants.

	Cas 1	**Cas 2**	**Cas 3**
dans `A`	`public void meth(double d)`	`public void meth(int i)`	`public void meth(int i)`
dans `B`	rien	`public void meth(double d)`	rien

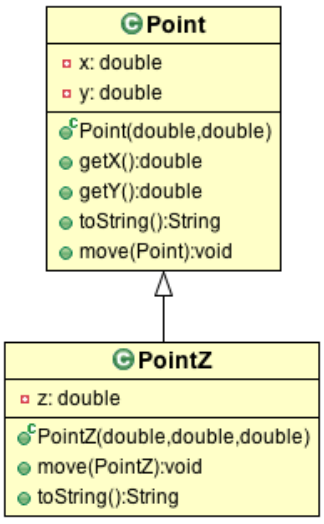
En envisageant chacun des 3 cas ci-dessus, quelles sont les lignes ci-dessous posant des problèmes de compilation ? Quelles sont les méthodes sélectionnées (pour le cas 2) ?

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

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

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

Exercice 58 – Redéfinition piégeuse




```
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 58.1 Pourquoi cette hiérarchie de classe est-elle discutable ?

Q 58.2 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 58.3 Que pensez-vous du programme suivant ? Qu'est-ce qui s'affiche ? Est-ce que ça vous semble logique ? Expliquer en détail ce qui s'est passé au niveau de la compilation et de l'exécution.

```
Point p = new Point(1,2);
Point p3d = new PointZ(1,2,3);
PointZ depl = new PointZ(1,1,1);  // déplacement à effectuer

System.out.println(p); // affichage avant modif
System.out.println(p3d);
p.move(depl); // modif
p3d.move(depl); // modif
System.out.println(p); // affichage après modif
System.out.println(p3d);
```

Exercice 59 – 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 :

```
package abc;
public class A {
        private int champPrive;
        int champSansModificateur;
        protected int champProtected;
        public int champPublique;
}
```

Q 59.1 Donner la déclaration des classes `B`, `C`, `D` et `E`, et faire un schéma.

Q 59.2 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`					
`champSansModificateur`					
`champProtege`					
`champPublic`					

Q 59.3 Si la classe `A` n'était pas déclarée `public`, est-ce que cela change la visibilité des variables ?

Quizz 16 – Héritage et liaison dynamique

Soient les 4 classes suivantes :

```
public class Animal {
        public void f() { }
        public String toString() {return "Animal";}
}
public class Poisson extends Animal {
        public void g() { }
        public String toString() {return "Poisson";}
}
public class Cheval extends Animal { }
public class Zoo { }
```

et les déclarations suivantes :

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

QZ 16.1 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.

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

QZ 16.2 Que retournent les instructions suivantes ?

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

QZ 16.3 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();`

TD 10 – Exceptions

Exercice 60 – 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 60.1 Expliquez brièvement ce que fait la ligne 10.

Q 60.2 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 60.3 En ajoutant seulement des instructions dans le `main`, proposez 2 solutions qui compilent et s'exécutent, et donnez l'affichage obtenu. Aide : le message d'erreur affiché par le compilateur indique les 2 solutions possibles : `error: unreported exception TropLourdException; must be caught or declared to be thrown`

Q 60.4 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 60.5 On modifie la signature de la classe `TropLourdException` en remplaçant `extends Exception` par `extends RuntimeException`. Qu'est-ce que cela change ?

Exercice 61 – Moyenne de notes valides (exceptions)

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 :

- Les arguments passés en ligne de commande sont récupérables par le tableau `args` du `main`. Par exemple : `java TestMoyenne 10 12 16 18` est équivalent à `String [] args={"10","12","16","18"};`
- 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.

Q 61.1 Est-ce que le programme compile ?

Q 61.2 Que donnent les exécutions suivantes ?

a) `java TestMoyenne 10 12 16 18`

b) `java TestMoyenne 11 1j 13`

c) `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 61.3 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 :

a) le programme n'affiche pas la moyenne, mais affiche seulement `"Erreur : note pas entière"` ?

b) 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 61.4 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 61.5 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.

a) 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.

b) A quel endroit faut-il lancer cette exception ? Quelles instructions et information faut-il rajouter ?

c) Capturer l'exception et afficher son message dans le main.

d) Quel est maintenant le résultat de l'exécution de :

- `java TestMoyenne` ?
- `java TestMoyenne 10a e5` ?

Q 61.6 On veut maintenant gérer les cas où la note est négative et où la note est strictement supérieure à 20.

a) 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"`.

b) 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 ?

c) Quel est le résultat de : `java TestMoyenne 14 25 7c 10 -3` ?

Exercice 62 – Gestion d'exceptions et finally

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("constructeur");
5         }
6 }
7 public class TestFinally {
8         // Exception déléguée à la méthode appelante (ici main)
9         public static void test1() throws MonException {
10                if (true) throw new MonException("lancée dans test1");
11                System.out.println("test1 : fin méthode");
12        }
13
14        // Exception capturée (et pas déléguée) dans la méthode test2
15        public static void test2() {
16                try {
17                        if (true) throw new MonException("lancée dans test2");
18                } catch (MonException e) {
19                        System.out.println("test2 : capture exception : "+e);
20                }
21                System.out.println("test2 : fin méthode");
22        }
23
24        // Exception capturée (et pas déléguée) dans la méthode test3 avec finally
25        public static void test3() {
26                try {
27                        if (true) throw new MonException("lancée dans test3");
28                } catch (MonException e) {
29                        System.out.println("test3 : capture exception : "+e);
30                } finally {
31                        System.out.println("test3 : finally effectué");
32                }
33                System.out.println("test3 : fin méthode");
34        }
35
36        // Exception déléguée à la méthode appelante (ici main) avec finally
37        public static void test4() throws MonException {
38                try {
39                        if (true) throw new MonException("lancée dans test4");
40                } finally {
41                        System.out.println("test4 : finally effectué");
42                }
43                System.out.println("test4 : fin méthode");
44        }
45
46        // Même cas que le test4, mais ici l'exception n'est pas levée
47        public static void test5() throws MonException {
48                try {
49                        if (false) throw new MonException("lancée dans test5");
50                } finally {
51                        System.out.println("test5 : finally effectué");
52                }
53                System.out.println("test5 : fin méthode");
54        }
```

```
55          // ========== Main ==========
56          public static void main(String [] args){
57                  try {
58                          test1();
59                  } catch (MonException e) {
60                          System.out.println("main : test1 : capture exception "+e);
61                  }
62                  test2();
63                  test3();
64                  try {
65                          test4();
66                  } catch (MonException e) {
67                          System.out.println("main : test4 : capture exception "+e);
68                  }
69                  try {
70                          test5();
71                  } catch (MonException e) {
72                          System.out.println("main : test5 : capture exception "+e);
73                  }
74                  System.out.println("Fin du programme");
75          }
76  }
```

Exercice 63 – Capture dans le main d'une exception prédéfinie (`try catch`)

Q 63.1 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 63.2 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.

TME 10 – Exceptions

Exercice 64 – Utilisation de `Scanner` et de `Thread.sleep(n)`

Q 64.1 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` (qui est une `RuntimeException`) 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 64.2 Si on ne veut pas capturer l'exception `InterruptedException`, que faut-il faire pour que le programme continue à compiler ?

Exercice 65 – EntierBorne (propagation d'exceptions)

Le but de l'exercice est de définir une classe `EntierBorne` qui représente un entier dont la valeur doit être comprise entre -100 et 100 et qui empêche le dépassement de ces bornes.

Q 65.1 Écrire la classe `HorsBornesException` avec un constructeur prenant en paramètre le message de l'exception.

Q 65.2 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. Cette classe contient trois attributs : une variable d'instance `valeur` de type `int`, ainsi que deux constantes représentant les bornes : `MIN` initialisée à -100 et `MAX` initialisée à 100. Écrire le constructeur à un paramètre de type `int` qui lève l'exception `HorsBornesException` si la valeur qui est passée en paramètre est plus grande que `MAX` ou plus petite que `MIN`, et la méthode `toString()`. Les messages des exceptions pourront être : `"Entier trop petit : -200"`, `"Entier trop grand : 300"`.

Q 65.3 Ajouter dans `EntierBorne` la méthode `EntierBorne somme(EntierBorne eb)` qui retourne un nouvel entier borné qui est la somme entre l'entier courant et `eb`. Cette méthode ne doit pas capturer l'exception.

Q 65.4 Dans le `main` d'une classe `TestEntierBorne`, créer un entier borné *ebX* avec une valeur aléatoirement choisie entre -150 et 150 et un autre entier borné *ebY* avec une valeur aléatoirement choisie entre 0 et 2, afficher ces entiers, puis afficher la somme des deux. Remarque : on demande à ce qu'il n'y ait qu'un seul `try`.

Q 65.5 Ajouter dans la classe `EntierBorne` une méthode `EntierBorne divPar(EntierBorne eb)` qui retourne un nouvel entier borné dont la valeur est le résultat de la division entière entre la valeur de l'objet courant et la valeur du paramètre. Si le paramètre `eb` a pour valeur 0, alors cette méthode doit lever l'exception `DivisionParZeroException`. Pour cela, on définira une nouvelle classe `DivisionParZeroException` avec le message `"division par zero"`. Remarque : la méthode `divPar` ne doit pas capturer les exceptions.

Q 65.6 Ajouter des instructions dans la méthode `main` pour afficher le résultat de la division de `ebX` par `ebY`. Remarque : on demande à ce qu'il n'y ait toujours qu'un seul `try`, mais il peut y avoir plusieurs `catch`.

Q 65.7 L'ordre des `catch` peut avoir de l'importance. Vérifiez-le en capturant `Exception` au lieu de capturer `HorsBornesException`. (a) Est-ce que mettre `catch(Exception e)` avant `catch(DivisionParZeroException e)` compile ? Pourquoi ? (b) Peut-on remplacer tous les `catch` par un seul bloc `catch(Exception e)` ? Si oui, pourquoi est-il cependant souvent nécessaire de mettre plusieurs `catch` ? (c) Peut-on remplacer toutes les instructions de gestion des exceptions dans le main par `throws Exception` dans la signature ? Si oui, pourquoi est-il préférable la plupart du temps de ne pas le faire ?

Exercice 66 – 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.

Q 66.1 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 66.2 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 66.3 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 Melissa 12 6 18 10
```

On supposera que *la première donnée est obligatoirement un nom* et que chaque donnée est correcte (pas de mélange entre lettres et chiffres). 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* : `int Integer.parseInt(String s)` rend l'entier représenté par la chaîne `s`, ou bien lève `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
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
```

Q 66.4 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 66.5 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 Melissa 12 6 18 10 12 6
le tableau de notes de l'étudiant Arthur est plein
le tableau de notes de l'étudiant Melissa est plein
les 5 étudiants :
[Anna 12 13 7 15, Tom, Arthur 9 12 15 0 13, Karim 15 8 11, Melissa 12 6 18 10 12]
```

TD/TME 11 – Patterns, manipulation de flux entrée / sortie

Exercice 67 (Examen 2016) – Gestion d'un système de tirage de boules de couleur

```
1 public class Boule {
2     private String couleur;
3     public Boule(String couleur) {
4         this.couleur = couleur;
5     }
6 }
```

```
7  // main
8  Boule b1 = new Boule("rouge");
9  Boule b2 = new Boule("jaune");
10 Boule b3 = new Boule("bleue");
11 Boule b4 = b1;
```

Q 67.1 A l'issue de l'exécution du `main` ci-dessus, combien y a-t-il de variables et d'instances de `Boule` ?

Q 67.2 Donner la ligne de code pour créer un tableau de `Boule` (nommé `urne`) contenant les 4 références précédentes.

Q 67.3 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 67.4 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.

```
20 public class BouleFactory {
21     public final String[] couleurs={"rouge","jaune","bleue"};
22     public static Boule build(){
23         return new Boule(couleurs[(int)Math.random()*couleurs.length]);
24     }
25 }
```

Q 67.5 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 67.6 Nous voulons faire des statistiques (comptages) sur les boules dans `gdeUrne`.

Q 67.6.1 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 67.6.2 Compter le nombre de boules de chaque couleur et afficher le résultat. Algorithme proposé :

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

```
30 public class BouleV2 {
31     private String couleur;
32     public final static BouleV2 ROUGE = new BouleV2("rouge");
33     public final static BouleV2 JAUNE = new BouleV2("jaune");
34     public final static BouleV2 BLEUE = new BouleV2("bleue");
35
36     private BouleV2(String couleur) { this.couleur = couleur; }
37 }
```

On donne le code suivant permettant de stocker 1000 `BouleV2` tirées aléatoirement dans une `ArrayList`.

```
38 BouleV2[] tab = {BouleV2.ROUGE, BouleV2.JAUNE, BouleV2.BLEUE};
39 ArrayList<BouleV2> gdeUrneV2 = new ArrayList<BouleV2>();
40 for (int i=0; i<1000; i++) gdeUrneV2.add(tab[(int)(Math.random() * tab.length)]);
```

Le développeur prétend que son architecture est : (a) tout autant sécurisée que la précédente, (b) plus rapide, (c) ne nécessite pas d'ajouter un code `equals`. Que penser de ces 3 affirmations ?

Exercice 68 (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     public final double frequence;
14
15     private Note(double frequence) { this.frequence = frequence; }
16
17     // pour les générer les demis tons
18     public Note diese(){ return new Note(frequence*demiTon); }
19     public Note bemol(){ return new Note(frequence/demiTon); }
20     // pour passer dans une gamme au dessus (facteur = nb de gamme au dessus)
21     public Note transpose(int facteur){return new Note(Math.pow(2,facteur)*frequence);}
22 }
```

Q 68.1 (a) Parmi les opérations suivantes, toutes effectuées en dehors de la classe `Note`, lesquelles posent problème ? Expliquer très brièvement. (b) Quels sont les points forts/faibles de l'architecture de cette classe ?

```
30 Note n1 = new Note(220);
31 Note n2 = Note.do_;
32 Note n3 = n2.re_;
33 Note si_bemol = Note.si_.bemol;
34 System.out.println(n2.frequence);
35 Note.mi_.frequence = 12;
36 Note do_aigu = Note.do_.transpose(1);
37 Note do_diese = Note.do_.diese();
38 Note do_diese2 = Note.do_ * Note.demiTon;
39 Note do_aigu2 = Note.do_.transpose(2.5);
```

Q 68.2 Pour ajouter une notion de rythme (noire, croche, blanche...), nous allons développer une nouvelle classe abstraite `Rythme` et des classes filles concrètes. (a) Est-il possible de faire hériter `Rythme` de `Note` ? (b) 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 68.3 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 68.4 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 :

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

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 68.5 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.

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 du `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 69 – Flux et fichiers

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

Q 69.1 Soit la classe `TestLitRepertoire` ci-dessous. 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.

```
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        }}}
```

Q 69.2 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 : voir la documentation de la classe `File`.

Exercice 70 – Manipulation de fichiers et d'arborescences

```
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 70.1 Dire ce qu'affiche l'exécution 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 70.2 Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier.

Q 70.3 Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la classe `File`.

Q 70.4 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 70.5 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 c'est la fin du fichier.
- `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 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 méthode `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 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 71 – Manipulations simples de flux (classe `Pixel`)

Cet exercice a 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 71.1 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 la classe `Pixel` 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 71.2 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 le `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         } catch (IOException e) {
9                 System.err.println("Erreur d'acces fichier: "+e);
10        }
11        try {
12                if (fOut != null)
13                        fOut.close();
14        } catch (IOException e) {
15                System.err.println("Erreur de fermeture du fichier: "+e);
16        }
```

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

Q 71.3 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 71.4 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 71.5 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.

```
public void charge(String f) throws [** COMPLETER **] {
        l = new ArrayList<Pixel>();
        int compte = 0;
        [** COMPLETER **] fIn = null;
        Pixel pLu = null;
        try {
            fIn = new  [** COMPLETER **];
            while (true) {
                pLu = new Pixel(fIn);
                [** COMPLETER **]
                compte++;
            }
        } catch ( [** COMPLETER **] e) {
            System.out.println("Fin de lecture: "+compte+" points chargés.");
        } finally {
            if (fIn != null)
                fIn.close();
        }
}
```

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

Q 71.6 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 72 – `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` :

```
public class TestStringBuilder {

    public static String testString(int nbIter,String chaine) {
        String s="";
        for(int i=0;i<nbIter;i++) {
            s+=chaine;
        }
        return s;
    }

    public static StringBuilder testStringBuilder(int nbIter,String chaine) {
        StringBuilder sb=new StringBuilder();
        for(int i=0;i<nbIter;i++) {
            sb.append(chaine);
        }
        return sb;
    }

    public static void main(String [] args) {
        int nbIter=100000; // Nombre d'itérations
        if (args.length==1) nbIter=Integer.parseInt(args[0]);

        String [] tab={"1","1234567890","12345678901234567890123456789012345678901234567890"};

        for(String chaine : tab) {
            System.out.println("### nbIter="+nbIter+" longueur="+chaine.length());

```

```
28              long debut1 = System.currentTimeMillis();
29              testString(nbIter,chaine);
30              long fin1 = System.currentTimeMillis();
31              System.out.println("String␣Durée␣:␣"+(fin1-debut1)+"␣ms");
32
33              long debut2 = System.currentTimeMillis();
34              testStringBuilder(nbIter,chaine);
35              long fin2 = System.currentTimeMillis();
36              System.out.println("StringBuilder␣Durée␣:␣"+(fin2-debut2)+"␣ms");
37      }
38 }
```

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

Q 72.1 Expliquez pourquoi il y a une si grande différence de temps de calcul entre `String` et `StringBuilder`.

Q 72.2 La longueur de la chaîne concaténée a-t-elle une forte influence pour `String` ? pour `StringBuilder` ?

Q 72.3 Quand il y a beaucoup de concaténations de chaînes de caractères, quelle classe faut-il mieux utiliser ?

Annexe A Aide mémoire Java

Convention d'écriture

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

Syntaxe

En-tête du main	`public static void main(String[] args)`
Afficher dans le terminal	`System.out.println(chaine);`
Affichage formaté	`System.out.format("%s=%.2f\n","prix",12.3456);`
Formatter une chaine	`String x=String.format("%s=%.2f","prix",12.3456);` // `x` vaut `"prix=12.35"`
Commentaires	`//` commentaire sur une ligne `/*` commentaire sur plusieurs lignes `*/`
Création de tableau	`type [] tabTypeSimple = new type [taille] ;` `MaClasse [] tabTypeObjet = new MaClasse [taille] ;`
Test du type de l'objet	*var* `instanceof` *NomClasse* : retourne `true` si *var* est de type *NomClasse*
Importation d'une classe	`import nompackage.NomClasse;`

Exceptions :

- Lever / lancer une exception : `throw new MonException();`
- Déléguer/transmettre/propager une exception : `public void maMethode() throws MonException {`
- Capturer une exception :

```
try {
    instructions qui peuvent lever une exception
} catch (MonException me) {
    System.out.println(me.getMessage());   // affiche le message de l'exception
} catch (AutreException ae) {
    System.out.println(ae.toString());    // affiche "NomClasse:"+getMessage()
} finally {
    instructions toujours executées
}
```

Principales instructions

Instruction

Expression ;
Instruction vide ;
{ instructions } *appelée bloc d'instruction*
Instruction de contrôle

Instructions de contrôle - Boucles

`for`

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

for (MaClasse mc : tableau) **{** // sans indice
  instructions
}

`while`

while (condition) **{**
  instructions
}

`do`

do {
  instructions
} while (condition) ;

Instructions de contrôle - Conditionnels

`if`

if (condition) **{**
  instructions
}

if (condition1) **{**
  instructions 1
} else if (condition2) **{**
  instructions 2
} else {
  instructions 3
}

`switch`

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

Grandes lignes de la structure d'une classe

```
1 public class MaClasse [extends ClasseMere] {
2     // --- Attributs (appelés aussi champs) ---
3     private int maVariable;       // Variable d'instance
4     private static int maVariableStatique=0; // Variable de classe (static)
5     private static final int CONSTANTE=3.1415;  // Constante (static final)
6     // --- Constructeurs ---
7     public MaClasse () {
8         ....
9     }
10    // --- Méthodes ---
11    public int getMaVariable() {// Accesseur (getter)
12        return maVariable;
13    }
14    public void setMaVariable(int v) {// Mutateur  (setter)
15        maVariable=v;
16    }
17    public String toString() {
18        ....
19        return chaine;
20    }
21 }
```

Tableau de codage des types simples

type	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	0L
`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 affectations.

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 (`static`). *Exemple :* pour calculer la surface d'un cercle de rayon 3.2cm, on peut calculer πr^2 ainsi : `double r=3.2; double s = Math.PI*Math.pow(r,2);`

`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
`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`	`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

Annexe B Environnement Linux

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

Démarrage sous Linux

- Pour ouvrir une fenêtre de travail : cliquer sur l'icone `"Terminal"` OU choisir dans le menu `Accessoires` l'option `"Terminal"`.
- Pour **gagner du temps** quand vous travaillez dans le terminal :
  - utilisez les flèches *haut* ↑ et *bas* ↓ pour se déplacer dans l'historique des commandes que vous avez déjà tapé
  - commencez à écrire le début d'une commande (ou le début du nom d'un fichier du répertoire courant), puis utilisez la touche `Tab` (tabulation) pour que la commande (ou le nom du fichier) soit complétée automatiquement
- Pour lancer un éditeur de texte, taper le nom de l'éditeur suivi de &. Par exemple pour lancer l'éditeur `gedit`, taper dans le terminal : `gedit &`
  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.

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 du répertoire courant
`cd`	Déplacement vers la racine de votre répertoire personnel
`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`

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 éviter et corriger rapidement les erreurs

- L'indentation traduit visuellement la structure du programme, elle met en relief les alternatives, les répétitions, les classes, etc. C'est pourquoi, indenter de manière lisible votre programme permet d'éviter des erreurs de programmation et de gagner du temps.
- 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 erreur.
- Une règle de base pour corriger rapidement les erreurs : cherchez à comprendre les messages d'erreurs.
  - Une erreur peut engendrer plusieurs messages.
  - Il arrive souvent que le compilateur ne vous indique pas l'erreur au bon endroit.
    - Si vous avez une erreur ligne 10, son origine est nécessairement située avant.
    - Si le compilateur vous indique : `"ligne 30 ';' expected"`, c'est-à-dire « point-virgule attendu », ne mettez pas un ';' à cette ligne. Recherchez l'origine exacte de l'erreur.
- 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.