LU2IN002 : Éléments de programmation par objets avec JAVA

Licence de Sciences et Technologies Mention Informatique

VERSION POUR LES ENSEIGNANTS

Fascicule de TD/TME

Année 2020-2021

Table des matières

1 Classe : définition, syntaxe (TD)

Objectif : syntaxe, compilation, exécution
Exercices à traiter : 1, 2, 3, 5, 6
Exercices de base pour revoir les boucles si nécessaire : 4 et 7
Parmi les 3 quizzes 1, 2 et 3 : le quizz 3 sur les nombres aléatoires sert beaucoup par la suite et est intéressant à corriger en TD, les autres peuvent être laissé à faire "à la maison".
Recommandations :

- On peut démarrer la séance 2 si les étudiants sont à l'aise sur la syntaxe... Sinon, il faut passer du temps à circuler dans les rangs pour les forcer à écrire du code individuellement, sans attendre la correction.
- On utilisera indifféremment les termes : variable d'instance, champ ou attribut.
- Sauf cas particulier, les variables sont déclarées `private` et les méthodes `public`.
- Utiliser une classe par fichier, une classe à part (et donc un fichier) pour le `main`.
- Faire attention à l'indentation des programmes.
- Demander aux étudiants si ils ont déjà fait du C ou du Python avant, pour vérifier que chaque étudiant a bien les bases de la programmation. Si ce n'est pas le cas, ils doivent se mettre à niveau rapidement là dessus car on ne fera pas d'apprentissage des bases dans ce cours.

Remarque : pour les 2 premières semaines, une partie "TME" (3) est fournie pour compléter cette partie "TD".

Exercice 1 (Cours) – Premier programme Java

Dans le fichier `Bonjour.java`, écrire une classe `Bonjour` qui affiche `"Bonjour Monde"`. Quel est le rôle de la méthode `main` ? Aide : pour la syntaxe, on se reportera à l'annexe page 158.

```
1 public class Bonjour {
2         public static void main(String[] args) {
3                 System.out.println("Bonjour␣!␣!␣!");
4         }
5 }
```

La méthode `main` est le point d'entrée du programme.

Exercice 2 – Planète

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

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

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

(a) Les variables d'instance sont : `nom` et `rayon` (ligne 2 et 3)
On les reconnaît car ces variables ne sont pas déclarées dans une méthode
(b) ligne 5 : `n` et `r` sont des paramètres de méthode
(c) ligne 10 : `s` est une variable locale à la méthode `toString()` quand la méthode `toString()` se termine, cette variable n'existe plus. Remarque : la variable `s` est inutile.

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

Le constructeur commence à la ligne 5 et termine à la ligne 8.
On reconnaît les constructeurs car ils :
- portent le même nom que la classe,
- seules méthodes qui par convention commencent par une majuscule
- n'ont pas de valeur de retour.

Le rôle des constructeurs est :
- d'initialiser les variables d'instance
- et d'effectuer les instructions nécessaires pour la création d'un objet de cette classe

Les constructeurs sont appelés quand on crée (mot-clé new) un objet de cette classe

Q 2.3 Quelles sont les méthodes de cette classe ?

Les méthodes sont la méthode standard `toString()` et l'accesseur : `getRayon()`

Q 2.4 Ecrire une nouvelle classe appelée `SystemeSolaire`. On souhaite que cette classe soit le point d'entrée du programme, que doit-elle contenir ? Créer un objet (ou instance) de la classe `Planete` pour la planète Mercure qui a un rayon de 2439.7 km et un autre objet pour la planète Terre qui a un rayon de 6378.137 km. Afficher la valeur de retour de la méthode `toString()` pour la planète Mercure, puis afficher le rayon de la planète Terre.

Elle doit contenir la méthode main.

```
1 public class SystemeSolaire {
2         public static void main(String [] args) {
3                 Planete m=new Planete("Mercure",2439.7);
4                 Planete t=new Planete("Terre",6378.137);
5                 System.out.println(m.toString());
6                 System.out.println("Le rayon de la Terre est "+t.getRayon());
7         }
8 }
```

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

Une seule classe par fichier. Le nom du fichier doit être `NomDeLaClasse.java`, c-à-d. : `SystemeSolaire.java`
`javac Planete.java` => création d'un fichier `Planete.class`
`javac SystemeSolaire.java` => création d'un fichier `SystemeSolaire.class`

`java SystemeSolaire` (SANS .java) (NON PAS : `java Planete`, car la méthode main se trouve dans la classe SystemeSolaire, et non pas dans Planete)
On peut aussi compiler par : `javac *.java` ou bien : `javac Planete.java SystemeSolaire.java`
En une ligne : `javac Planete.java SystemeSolaire.java && java SystemeSolaire`

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

Il est possible d'accéder en lecture au rayon mais pas au nom.
Il n'est pas possible de modifier les attributs.
Message : limiter les possibilité du client est un gage de sécurité ! (En général, une planète ne change pas de rayon)

Exercice 3 – Se présenter

Q 3.1 Une personne est représentée par son nom et son âge. Ecrire la classe `Personne` qui contient :
— les variables d'instance `nom` et `age`,

Discuter ici du choix du type pour chaque variable (char, int, float, double, boolean et String) (voir tableau de codage des types simples page 160)

— un constructeur dont la signature est : `public Personne(String n, int a)`.

```
1 public class Personne {
2         private String nom;
3         private int age;
4
5         public Personne(String n, int a) {
6                 nom=n;
7                 age=a;
8         }
9 }
```

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

L'idée c'est que les étudiants comprennent que la construction des objets et l'utilisation des méthodes (dans le main), c'est différent du schéma de la classe.

```
1 public class Presentation {
2         public static void main(String [] args) {
3                 Personne p1=new Personne("Paul",25);
4                 Personne p2=new Personne("Pierre",37);
5         }
6 }
```

Q 3.3 On souhaite maintenant avoir des méthodes qui nous permettent d'obtenir des informations sur les objets de la classe `Personne`. Ajouter dans la classe `Personne`, les méthodes suivantes :

— la méthode standard `public String toString()` dont le but est de *retourner* une chaîne de caractères au format suivant : "Je m'appelle <*nom*>, j'ai <*age*> ans" où <*nom*> et <*age*> doivent être remplacés par le nom et l'âge de la personne courante. Dans la classe `Presentation`, ajouter une instruction qui utilise cette méthode pour afficher le nom et l'âge de Pierre.
— la méthode `public void sePresenter()` dont le but est *d'afficher* la chaîne de caractères retournée par la méthode `toString()`. Dans la classe `Presentation`, ajouter une instruction qui utilise cette méthode pour afficher le nom et l'âge de Paul.
— Quelle différence y-a-t-il entre la méthode `toString()` et la méthode `sePresenter()` ?

Remarque : `toString()` est vue dans le cours 2 qui, pour certains groupes, a lieu après cette séance.

```
1 public String toString(){
2         return "Je␣m'appelle␣"+nom+",␣j'ai␣"+age+"␣ans";
3 }
4
5 public void sePresenter(){
6         System.out.println("Bonjour␣!␣"+toString());
7         //System.out.println("Bonjour ! Je m'appelle "+nom+", j'ai "+age+" ans");
8 }
```

Dans la classe Presentation :

```
1 System.out.println(p1.toString());
2 p2.sePresenter();
```

La méthode `toString()` (méthode standard de java dont on ne peut pas changer la signature) retourne une chaîne de caractères tandis que la méthode `sePresenter()` (non standard) affiche une chaîne de caractères sur le terminal. Retourner une chaîne et l'afficher ce n'est pas pareil!!!
Comparer les réponses aux questions a) et b) afin de faire comprendre la différence entre "afficher" et "retourner".

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

On ne peut plus utiliser cette méthode à l'extérieur de la classe. Donc on ne peut plus faire : `p2.sePresenter();`
Rappeler que sauf exception, les variables doivent être déclarées private et les méthodes public.

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

Non, car sauf exception les variables d'instance doivent être déclarées privées (sécurité, encapsulation). On ne peut donc pas écrire : `p1.age` dans la classe `Presentation`, `age` n'est pas accessible dans la classe `Presentation`.
Pour connaître la valeur de age, on peut ajouter un accesseur pour la variable age :
`public int getAge()` { return age; }
qui permet de récupérer la valeur à l'exterieur de la classe `Personne`, mais pas de modifier l'age de la personne (=> sécurité).
Le type de retour d'un accesseur doit être le même que le type de la variable qu'il retourne. Comme le type de age est int, le type de retour est int.

Q 3.6 Ajouter dans la classe `Personne`, la méthode `vieillir()` qui ajoute un an à la personne. Dans la classe `Presentation`, faites vieillir Paul de 20 ans (utiliser une boucle `for`), et Pierre de 10 ans (utiliser une boucle `while`), puis faites se présenter Paul et Pierre. Aide : voir la syntaxe des boucles page 160

```
public void vieillir() {
        age=age+1;     // ou age++;     ou age+=1;
}
```

ATTENTION : l'objectif est de faire des rappels sur for et while
A la suite, dans la classe `Presentation` :

```
for(int i=0;i<20;i++)
        p1.vieillir();
int i=0;  // pas d'ambiguité avec le i du for qui n'est visible que dans le for

while(i<10) {
        p2.vieillir();
        i++;
}
p1.sePresenter();  // 45 ans
p2.sePresenter();  // 47 ans
```

Exercice 4 – Alphabet

Q 4.1 Ecrire la classe `Alphabet` qui ne contient que la méthode `main` qui réalise le traitement suivant, en utilisant une boucle `for` :

Q 4.1.1 Afficher les chiffres de 0 à 9, ainsi que leur code ASCII.

Q 4.1.2 Afficher les lettres de l'alphabet de 'A' à 'Z', ainsi que leur code ASCII.

```
public class Alphabet {
        // Manipulation de char Unicode, affichage de caractères
        public static void main (String[] args) {

                // Affichage des chiffres
                for (char c = '0'; c <= '9'; c++)
                        System.out.print(c);
                System.out.println();
                System.out.println("-----------------");

                // Affichage des codes des chiffres
                for(int i = '0' ; i <= '9'; i++)
                        System.out.print(i + "␣") ;
                System.out.println();
                System.out.println("-----------------");

                // Affichage des lettres
                for (char c = 'A'; c <= 'Z'; c++)
                        System.out.print(c);
                System.out.println();
                System.out.println("-----------------");

                // Affichage des codes des lettres
                for (int i = 'A' ; i <= 'Z'; i++)
                        System.out.print(i + "␣") ;
        }

}
```

Q 4.2 Recommencer en utilisant une boucle while.

```
1 public class Alphabet {
2         // Manipulation de char Unicode, affichage de caractères
3         public static void main (String[] args) {
4         char c = '0';
5         // Affichage des chiffres
6         while (c <= '9'){
7                 System.out.print(c);
8                 c++;
9         }
10        System.out.println();
11        System.out.println("-----------------");
12
13        // Affichage des codes des chiffres
14        int i = '0';
15        while (i <= '9') {
16                System.out.print(i + "␣");
17                i++;
18        }
19        System.out.println();
20        System.out.println("-----------------");
21
22        // Affichage des lettres
23        char c='A';
24        while (c <= 'Z') {
25                System.out.print(c);
26                c++;
27        }
28
29        System.out.println();
30        System.out.println("-----------------");
31
32 }
```

Exercice 5 – Conventions de nommage

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

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

abcDefg()	AM	appel méthode : les méthodes commencent par convention par une minuscule mais peuvent contenir des majuscules, ce qui augmente la lisibilité
true	R	Mot-réservé
abcd	V	Variable d'instance, Variable locale
Abcd	NC	Nom classe
Abc()	AC	Appel constructeur sans paramètre
String	NC	Nom classe
False	NC	Nom classe
ABCD	CST	Constante car toute en majuscule. Par exemple, Math.PI est déclarée public static final double **PI**

Exercice 6 – Constructeurs multiples, méthodes multiples

Rappel : en JAVA, une méthode est identifiée par son nom ET ses arguments. Ainsi, deux méthodes avec le même nom et des arguments différents (nombre ou type des arguments) sont différentes. Le même principe prévaut avec les constructeurs.
En repartant de l'exercice 2 :
Q 6.1 Ajouter un second constructeur qui prend en argument seulement le nom de la planète et fixe son rayon à 1000km (arbitrairement).

```
1         public Planete(String n) {
2                 nom=n;
3                 rayon=1000;
4         }
```

Q 6.2 Écrire un programme de test construisant deux planètes en utilisant les deux constructeurs pour vérifier le bon fonctionnement de cette approche.

```
1 public class TestPlanete {
2         public static void main(String [] args) {
3                 Planete p1=new Planete("Mercure",2439.7);
4                 Planete p2=new Planete("Terre");
5                 System.out.println(p1.toString()); // ne pas oublier de tester l'
                      affichage, sinon on ne voit rien lors de l'exécution.
6                 System.out.println(p2.toString());
7         }
8 }
```

Hello there!

Exercice 7 – Rétro-engineering

Q 7.1 Écrire le programme permettant d'obtenir l'affichage suivant (en utilisant des boucles) :

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

```
1 public class TestSerie {
2         public static void main(String [] args) {
3                 int i = 1;
4                 int j = 1;
5                 for(int cpt=0; cpt<9; cpt++){
6                         System.out.print(j+"␣"); // pas de retour à la ligne + penser à
                              ajouter un espace
7                         int mem = j;
8                         j = i + j;
9                         i = mem;
10                }
11        }
12 }
```

Quizz 1 – Compilation et exécution

QZ 1.1 Un fichier source Java est sauvegardé avec l'extension ...

.java

QZ 1.2 Un fichier source Java contient ...

une classe (On peut en mettre plusieurs... Mais pas dans cette UE !)

QZ 1.3 Une classe est composée de ... et de ...

méthodes et variables

QZ 1.4 Les instructions Java sont toujours situées à l'intérieur de ...

méthodes

QZ 1.5 Les lignes composant une méthode sont soit des ... soit des ...

soit des instructions, soit des déclarations de variables locales à la méthode

QZ 1.6 Si vous n'utilisez pas l'environnement intégré, quelle commande tapez-vous pour compiler un fichier ? Pour exécuter un `.class` correspondant à un `main` ?

```
1 javac NomFichier.java
2 java NomFichier // sans le .class
```

QZ 1.7 Quel est le nom de la méthode par lequel un programme Java commence son exécution ?

main

QZ 1.8 Quel est l'en-tête de la méthode main ?

public static void main(String[] args)

Quizz 2 – Syntaxe des expressions

QZ 2.1 L'instruction suivante provoque-t-elle une erreur ? `float val=1.12;`

Oui, car en java, le type par défaut pour les réels est `double`. Message d'erreur à la compilation :
`Test.java:3: possible loss of precision found : double required: float`
Si l'on souhaite utiliser un float, en Java, on peut ajouter la lettre `f` au nombre : `float val=1.12f;`
(`1.12` est de type `double`, alors que `1.12f` est de type `float`)

QZ 2.2 Soient : `int x=2, y=5; double z=x/y;` Quelle est la valeur de z ?

La valeur de z est 0. Comme x et y sont des entiers, c'est la division entière qui est effectuée. Pour corriger, il suffit que l'une des 2 opérandes de la division soient de type `double`. Par exemple : `int z=x/(double)y;`

QZ 2.3 Sachant que le code ascii du caractère '1' est 49, quel est le type et la valeur des expressions suivantes :

a) `1+"1"` ?

type=`String` (`int+String=>String`), valeur= `"11"` (concaténation)

b) `1+'1'` ?

type=`int` (`int+char=>int`), valeur=50 (addition 1+49)

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

type=`String` (`char+String=>String`), valeur="11" (concaténation)

QZ 2.4 Sachant qu'en Java l'addition est évaluée de la gauche vers la droite, quelle est la valeur des expressions suivantes :

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

```
1+'1'+"1" => (1+'1')+"1" => 50+"1" => "501" (String)
```

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

```
"1"+'1'+1 => ("1"+'1')+1 => "11"+1 => "111" (String)
```

QZ 2.5 Quel est le type et la valeur de l'expression suivante ? `true && false == false`
Aide : voir la table de priorité des opérateurs page 160.

type=`boolean`, valeur=`true`
Pour répondre à la question, il faut savoir quel est l'opérateur le plus prioritaire. La table de priorité des opérateurs page 160 permet de voir que l'opérateur `==` est plus prioritaire que l'opérateur `&&`. L'expression est donc équivalente à : `true && (false == false)`

QZ 2.6 Qu'affiche : `System.out.println("Bonjour \nvous \ttous !")` ?

Cette question est pour introduire : $'\backslash n'$ et $'\backslash t'$.
Affiche : `"Bonjour"` puis nouvelle ligne, puis `"vous"`, puis espace tabulation, puis `"tous !"`
On peut aussi rappeler que `System.out.print(chaine);` (sans LN) affiche *chaine* sans retour à la ligne.

Quizz 3 – 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 [MAX,MIN[: `double val=Math.random()*(MAX-MIN)+MIN;`
- un entier dans [MAX,MIN] : `int val=(int)(Math.random()*(MAX-MIN+1)+MIN);`
- un booléen vrai dans `x` pour cent des cas : `boolean val=Math.random()<(x/100);`

Générer aléatoirement :

a) un réel dans [10,30[

```
double val=Math.random()*20+10;
```

b) un entier dans [50,150]

`int val=(int)(Math.random()*101+50);` △ ne pas oublier le cast
Remarque : si on ne se rappelle plus le calcul, on peut faire une étude aux bornes :
- si `Math.random()` = 0, alors : 0*x+y=MIN, donc y=MIN
- si `Math.random()` = 0.999..., alors : (int)(0.999*x+MIN)=MAX, donc x=MAX-MIN+1

c) un booléen vrai dans 25% des cas

```
boolean val=Math.random()<0.25;
```

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

On utilise la même formule que pour les entiers avec MIN='a' et MAX='z' :
```
char val=(char)(Math.random()*('z'-'a'+1)+'a');
```

TME 1 : Introduction à Java – premiers pas

Remarque : les intitulés des exercices de cette séance sont aussi disponibles en ligne sur le site Moodle de l'UE : https://moodle-sciences.upmc.fr/moodle-2020/course/view.php?id=2403

Objectifs :
- Prise en main de LINUX et de l'environnement
- Familiarisation avec la syntaxe Java
- Boucles for et while, instruction if
- Classe et instantiation

Exercice obligatoire : 8.
Exercices conseillés : 9, 10 et 11.
Ensuite, peut être un des exercices 12 ou 13 (ou les 2).
L'exercice 14 est long (et fastidieux), il peut être donné aux rapides ou laissé à faire "à la maison".

Exercice 8 – Préliminaires

Q 8.1 En salle de TME, si vous n'avez pas accès à internet :
- chercher l'outil de configuration d'accès dans Mozilla Firefox.

— passer en configuration manuelle en mettant proxy (port = 3128) sur tous les champs
Mettez ce site Moodle de l'UE dans les favoris de votre navigateur pour que vous puissiez y accéder rapidement à chaque séance de TME.

Q 8.2 Pour bien organiser vos fichiers, créer le répertoire `LU2IN002` dans votre répertoire de travail, puis dans ce répertoire, créer un répertoire `TME1`, Tous vos fichiers du `TME1` devront se trouver dans ce répertoire. Pour cela, ouvrir un nouveau terminal, puis taper les commandes qui permettent de réaliser les instructions suivantes :

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

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

```
mkdir LU2IN002 ; cd LU2IN002 ; mkdir TME1 ; ls ; cd TME1 ; pwd
```

Q 8.3 Pour ouvrir un éditeur de texte (par exemple, l'éditeur `gedit`) afin d'écrire du code Java ou du texte, il est possible d'utiliser la commande : `gedit &` ou bien `gedit` *`MaClasse.java`* `&` pour ouvrir le fichier *`MaClasse.java`*.
Attention : ne pas oublier le '`&`' à la fin de la commande pour séparer le terminal et l'éditeur de texte.
Il y a différents éditeurs de texte possibles : `gedit`, `vim`, `geany` ou `emacs`.
Remarque : vous ne devez PAS UTILISER d'IDE avant la semaine 5 (c'est-à-dire, PAS d'eclipse, PAS de netbean, ...)
Avec l'éditeur que vous avez choisi, chercher les options pour colorer la syntaxe, indenter les lignes et les numéroter.

Q 8.4
— Pourquoi est-il important d'indenter vos programmes ?
— Pourquoi est-il important de sauvegarder et de compiler régulièrement vos programmes sans attendre d'avoir écrit le programme en entier ?

Exercice 9 – Premier programme

Q 9.1 Écrire la classe `Bonjour` (fichier `Bonjour.java`) dont la méthode main affiche le message `"Bonjour !"`.

```
1 public class Bonjour {
2         public static void main(String[] args) {
3                 System.out.println("Bonjour␣!");
4         }
5 }
```

Q 9.1.1 Quelle est la commande pour compiler cette classe ? Compiler la classe. Quel est le nom du fichier créé par la compilation ?

```
javac Bonjour.java
Bonjour.class
```

Q 9.1.2 Quelle est la commande pour exécuter cette classe ?

java Bonjour

Q 9.1.3 Supprimer le fichier `Bonjour.class` et, sans recompiler, taper à nouveau la commande pour exécuter la classe. La classe est-elle exécutée ?

Non, car il faut que le fichier `Bonjour.class` se trouve dans le répertoire courant pour que la classe puisse être exécutée.

Q 9.2 Observer les erreurs de compilation :

Q 9.2.1 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.

Deux diagnostics d'erreurs de compilation. Donc, une erreur peut engendrer plusieurs diagnostics.
Le premier diagnostic `" ; expected"` ne correspond manifestement pas à la correction à effectuer. Le second : `"cannot resolve symbol"` est un peu plus significatif mais ne correspond tout de même pas à la correction à effectuer.
Conclusion : les diagnostics du compilateur ne doivent pas être suivis à la lettre. Ils indiquent seulement l'échec de l'analyse.

Q 9.2.2 Rétablir le mot `static` correctement et supprimer le `"` terminant le mot `Bonjour`. Compiler et observer.

Deux diagnostics : `"unclosed string litteral"` et `") expected"`

Q 9.2.3 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.

Le programme se compile correctement mais si on cherche à exécuter, la machine virtuelle Java ne trouve pas de méthode main par où commencer l'exécution. Java est sensible à la casse.

Q 9.2.4 Après avoir remis le bon nom à la fonction `main`, supprimer l'accolade { qui suit le `main`. Compiler et lire les messages.

Si vous oubliez une accolade, le compilateur diagnostiquera une erreur souvent très difficile à retrouver. Donc, chaque fois que vous tapez une {, tapez immédiatement l'accolade fermante } et ouvrez des lignes intermédiaires (en tapant Entrée) pour taper le reste du texte.

Q 9.2.5 Après avoir remis l'accolade, supprimer le mot-clé `public`. La compilation réussit-elle ? Peut-on exécuter le programme ?

La compilation et l'exécution se passent correctement.

Q 9.2.6 Après avoir remis le mot clé `public`, supprimer le mot-clé `static`. La compilation réussit-elle ? Peut-on exécuter le programme ?

La compilation réussie, mais on ne peut pas exécuter le programme.
`Exception in thread "main" java.lang.NoSuchMethodError: main`

La méthode `main` doit être obligatoirement `static`.

Exercice 10 – 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 10.1 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.

```
1 public class Segment {
2         private int x;
3         private int y;
4
5         public Segment(int extX,int extY) {
6                 x=extX;
7                 y=extY;
8         }
9         public int longueur() {
10                if (x<y)
11                        return y-x;
12                else
13                        return x-y;
14        }
15        public String toString() {
16                return "Segment␣["+x+","+y+"]";
17        }
18 }
```

Q 10.2 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.

```
1 public class TestSegment {
2         public static void main(String [] args) {
3                 Segment s1=new Segment(8,6);
4                 Segment s2=new Segment(12,5);
```

```
5                   System.out.println("longueur du"+s1+" = "+s1.longueur());
6                   System.out.println("longueur du"+s2+" = "+s2.longueur());
7
8                   if ( s1.longueur() < s2.longueur() )
9                           System.out.println("le "+s1+" est plus long");
10                  else
11                          System.out.println("le "+s2+" est plus long");
12          }
13  }
```

Exercice 11 – Solidarité villageoise

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

Q 11.1 Dans la classe `Villageois`, définir les variables suivantes :

- `nom` (le nom du villageois, de type `String`),
- `poids` (le poids (kg) du villageois, type `double`),
- `malade` (type `boolean`, sa valeur est `true` si le villageois est malade, `false` sinon).

Quel doit être le nom du fichier contenant cette classe ?

```
1 public class Villageois {
2         private String nom ;     // le nom du villageois
3         private double poids ;   // le poids du villageois
4         private boolean malade ;// le villageois est-il malade ?
5 }
```

Le fichier doit s'appeler : `Villageois.java`

Q 11.2 Ajouter dans la classe `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` qui vaut `true` dans 20% des cas et à `false` sinon.
  Aide : faire le quizz 3 avant de résoudre cette question. De plus, pour modéliser un évènement qui se produit dans 40% des cas, on regarde si `Math.random() < 0.40`. Voir aussi la documentation de la classe `Math` page 161.

```
1 public Villageois(String nomVillageois) {
2         nom = nomVillageois;
3         poids = Math.random()*100 + 50; // poids aléatoire entre 50 et 150
4         malade = Math.random() < 0.2;
5 }
```

Q 11.3 Dans une nouvelle classe `TestVillageois`, ajouter une méthode `main`, qui crée 4 instances de la classe `Villageois` et les affiche. Quel est le nom du fichier contenant cette classe `TestVillageois` ? Compiler et exécuter ce programme.

```
public static void main(String[] args) {
        Villageois v0 = new Villageois("A");
        Villageois v1 = new Villageois("B");
        Villageois v2 = new Villageois("C");
        Villageois v3 = new Villageois("D");

        System.out.println(v0.toString()+"\n"+v1.toString()+"\n"
                        +v2.toString()+"\n"+v3.toString());
}
```

Le fichier doit s'appeler : TestVillageois.java

Q 11.4 Ajouter dans la classe `Villageois` et utiliser dans la classe `TestVillageois` les méthodes suivantes :

- `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,

```
public String getNom() { return nom; }
public double getPoids() { return poids; }
public boolean getMalade() { return malade; }
```

- `public String toString()` qui retourne une chaîne de caractères décrivant les caractéristiques de ce villageois. Par exemple, `"villageois : Eustache, poids : 95 kg, malade : non"`

```
public String toString() {
        String maladeOuiNon = null;
        if (getMalade())
                maladeOuiNon = "oui";
        else
                maladeOuiNon = "non";
        String text ="villageois␣"+nom+",␣"
                +"poids␣:␣"+getPoids()+"Kg,␣"+"malade␣:␣"+maladeOuiNon
                + ",␣poids␣souleve␣:␣" +poidsSouleve();

        return text;
}
```

Q 11.5 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.

```
public double poidsSouleve() {
        if (getMalade()) return (getPoids() / 4.0);
        else return (getPoids() / 3.0);
}
```

Q 11.6 Ajouter dans la classe `TestVillageois`, les instructions pour calculer le poids total que peuvent soulever les 4 villageois définis dans le `main`, et afficher un message pour indiquer s'ils réussissent à soulever le rocher ou pas.

```
double poidsTotal=v0.poidsSouleve()+v1.poidsSouleve()
                +v2.poidsSouleve()+v3.poidsSouleve();
```

```
if ( poidsTotal > 200 )
        System.out.println("Le rocher a ete souleve !");
else
        System.out.println("Le rocher n'a pas ete souleve !");
```

Exercice 12 – Affichage avec passage à la ligne

Remarque : dans la pratique, on n'a pas besoin de la classe Lettre, mais le but est de manipuler des classes très simples

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

```
public class Lettre {
        private char carac;

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

Q 12.1 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`.

Remarque : rappeler aux étudiants que la classe peut être récupérer sur le Moodle.

```
for(char c='a';c<='z';c++) {
        Lettre l1=new Lettre(c);
        System.out.println("Le code ascii de "+c+" est :"+l1.getCodeAscii());
}
```

Q 12.2 On veut maintenant afficher l'alphabet comme ceci :

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

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

```
for(char c='a';c<='z';c++) {
        Lettre l=new Lettre(c);
        System.out.print(l.getCarac()+"␣");
        if ( (c-'a') % 5 == 4 ) {
                System.out.println();
        }
}
```

Exercice 13 – Formule de Newton

La suite de Newton définie ci-dessous converge vers la racine carrée du nombre `x` :
$U_0 = \frac{x}{2}$ et $U_i = \frac{1}{2}(U_{i-1} + \frac{x}{U_{i-1}})$ pour tout $i \geq 1$

Écrire une classe `SuiteNewton` qui étant donné un nombre x et un réel ε calcule la valeur de la racine de x avec une précision de ε en utilisant la suite de Newton.

```
public class SuiteNewton {
        // Calcul d'une racine carrée par la suite
        // de Newton Ui = (Ui-1 + x/Ui-1)/2
        //---------------- U0 = x / 2
        private double x;
        private double epsilon;
        public SuiteNewton(double x, double epsilon){
                this.x = x; this.epsilon = epsilon;
        }

        public double racine() {
                int i = 0; // Pour le calcul du nombre d'itérations
                double u = 0, v = x/2;
                if(x < 0) return Double.NaN;
                if (x != 0) {
                        do {
                                u = v;
                                v = (u + x / u)/ 2;
                                i++;
                        } while (Math.abs(u-v) > epsilon);
                }
                System.out.println("Nombre␣d'itérations = " + i);
                return v;
        }
}

//================

public class TestSuiteNewton {
        public static void main(String[] args) {
                System.out.println("Calcul␣de␣la␣racine␣carrée");
                double nombre = 5;
                System.out.println("Racine(" + nombre + ")=" + racine(nombre, 1e-4));
                System.out.println("Math.sqrt(" + nombre + ")=" + Math.sqrt(nombre));
        }
}
```

Exercice 14 – Libellé d'un chèque (*exercice long et fastidieux...*)

Écrire une classe `Libelle` qui traduit en toutes lettres un nombre entier inférieur à 1000 selon les règles usuelles de la langue française.
Par exemple : 123 s'écrit "cent vingt-trois" et 321 s'écrit "trois cent vingt et un".
Le programme doit prendre en compte les règles d'orthographe essentielles :

- les nombres composés inférieurs à cent prennent un trait d'union sauf vingt et un, trente et un, ..., soixante et onze.
- les adjectifs numéraux sont invariables sauf cent et vingt qui se mettent au pluriel quand ils sont multipliés et non suivis d'un autre nombre.

NB : toutes les méthodes *outils* pourraient être private, c'est une bonne illustration pour les étudiants

```
1 //class Libelle pour libeller le montant d'un chèque.
2 //la methode statique enLettres(nb) renvoie nb "en toutes lettres"
3 //nb doit etre inferieur a 1000
4 //la methode main teste un certain nombre de cas....
5
6 public class Libelle {
7
8 public String moinsDe16(int nb) {
9         switch (nb) {
10                case 0 : return "zero" ;
11                case 1 : return "un" ;
12                case 2 : return "deux" ;
13                case 3 : return "trois" ;
14                case 4 : return "quatre" ;
15                case 5 : return "cinq" ;
16                case 6 : return "six" ;
17                case 7 : return "sept" ;
18                case 8 : return "huit" ;
19                case 9 : return "neuf" ;
20                case 10 : return "dix" ;
21                case 11 : return "onze" ;
22                case 12 : return "douze" ;
23                case 13 : return "treize" ;
24                case 14 : return "quatorze" ;
25                case 15 : return "quinze" ;
26                default : return "seize" ;
27        }
28 }
29 public String chiffreDesDizaines (int nb){
30        switch (nb) {
31                case 0 : return "" ;
32                case 1 : return "dix" ;
33                case 2 : return "vingt";
34                case 3 : return "trente" ;
35                case 4 : return "quarante" ;
36                case 5 : return "cinquante" ;
37                case 6 : return "soixante" ;
38                case 7 : return "soixante-dix" ;
39                default : return "quatre-vingt" ;
40        }
41 }
42 public String moinsDe100(int nb) {
43        int dizaine = nb/10 ;
44        int unite = nb%10 ;
45        String lien = (unite == 1) && (dizaine<8) ? "␣et␣" : "-" ;
```

```
46          if (dizaine == 7 || dizaine == 9){
47                  dizaine--;
48                  unite = unite + 10 ;
49          }
50          return chiffreDesDizaines(dizaine) + lien + enLettres(unite) ;
51  }
52  public String moinsDe1000(int nb) {
53          int centaine = nb/100 ;
54          int suite = nb % 100 ;
55          String s = suite==0 ? "" : " " + enLettres(suite);
56          String c = centaine==1 ? "" : enLettres(centaine) + " ";
57          String cent = centaine>1 && suite == 0 ? "cents" : "cent" ;
58          return  c + cent + s ;
59  }
60  public String enLettres(int nb){
61          if (nb<=16) return moinsDe16(nb) ;
62          else if (nb == 80) return "quatre-vingts" ;
63          else if (nb<100) return moinsDe100(nb) ;
64          else if (nb<1000) return moinsDe1000(nb) ;
65          else return "mille ou plus ";
66  }
67  public static void main (String [] a) {
68          int [] tab={15,23,71,80,81,91,98,100,103,423,600,1000,1010};
69          System.out.println ("Quelques nombres en toutes lettres :\n ");
70          for(int i=0;i<tab.length;i++) {
71                  System.out.println (tab[i] + " : " + enLettres(tab[i]));
72          }
73  }
74  }
```

2 Encapsulation, surcharge

Séance 2 :
Objectif : syntaxe objet avancée (ie addition de vecteurs ou de complexes), surcharge de méthode, sélection de méthode, type des arguments, distinction instance/référence
Exercices conseillés : 15, 16, et 18.
Exercices pour les rapides : 17
L'exercice 17 est particulier : il fait appel à des choses que l'on verra en Semaine 6. Il peut être préparé en disant aux étudiants que certains points seront éclaircis par la suite.
Quizzes : 4, 5 et 6
Rappel : On utilise vraiment `static` seulement à partir du thème 5.

Exercice 15 – Classe Bouteille (surcharge de constructeurs, this)

Soit la classe `Bouteille` suivante :

```
1  public class Bouteille{
2          private double volume; // Volume du liquide dans la bouteille
3
4          public Bouteille(double volume){
5                  this.volume = volume;
6          }
7          public Bouteille() {
```

```
8                   this(1.5);
9           }
10          public void remplir (Bouteille b){
11                  // A compléter
12          }
13          public String toString(){
14                  return("Volume du liquide dans la bouteille ="+volume);
15          }
16  }
```

Q 15.1 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.

Il y a deux constructeurs : un avec un paramètre de type double et un sans paramètre.
Il peut y avoir plusieurs constructeurs (surchage), mais il faut que les paramètres ne soient pas les mêmes.

```
1 Bouteille x = new Bouteille();
2 Bouteille y = new Bouteille(10.2);
```

Q 15.2 Expliquer l'affectation de la ligne 5 : que représente `this.volume` ? `volume` ?

`this.volume` : variable d'instance de la classe bouteille
`volume` : variable locale passée en paramètre au constructeur

Q 15.3 Expliquer la ligne 8.

`this()` avec parenthèse correspond à un appel de constructeur, ici c'est le constructeur à 1 paramètre.
Rappeler l'usage de `this()` : il doit toujours être la première instruction dans le constructeur.

Q 15.4 Compléter la méthode d'instance `remplir(Bouteille b)` qui ajoute le contenu de `b` à la bouteille courante.

```
1 public void remplir (Bouteille b){
2           volume+=b.volume;
3           b.volume = 0; // OPT mais plus propre du point de vue physique
4 }
```

Remarque : bien que volume soit `private`, comme le paramètre `b` est de la même classe que la classe courante, on peut utiliser `b.volume` directement.
Faire remarquer que l'on peut avoir des objets en paramètres.

Q 15.5 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.

Oui, cela s'appelle la surcharge de méthode, il faut que le type des paramètres soit différent.

```
1 public void remplir (double v){
2           volume+=v;
3 }
```

Dans l'idéal, on vide aussi la bouteille v. C'est très bien pour leur montrer que l'on a aussi accès aux variables privées des autres instances.

Q 15.6 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 }
```

Ligne 5 : Volume du liquide dans la bouteille = 10
Ligne 6 : Volume du liquide dans la bouteille = 1.5
Ligne 8 : Volume du liquide dans la bouteille = 11.5
Ligne 9 : Volume du liquide dans la bouteille = 1.5

Exercice 16 – 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`.

On ajoute la méthode `addition` dans la classe `Couple` :

```
1   public Couple addition(Couple c2) {
2     return new Couple(this.x+c2.x,this.y+c2.y);
3   }
```

Dans le main `main` :

```
Couple cAPlusCB=cA.addition(cB);
```

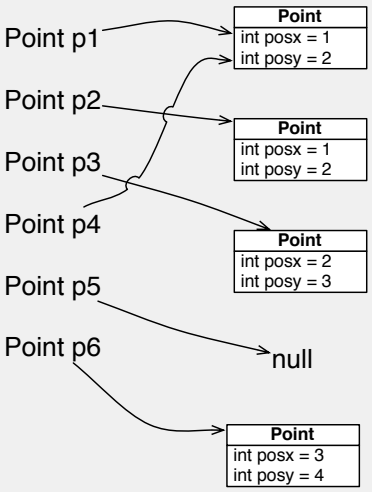
Exercice 17 – Point : sur l'égalité

Soit le programme suivant :

```java
// TestPoint.java
public class TestPoint {
   public static void main(String[] args){
     Point p1 = new Point(1,2);
     Point p2 = new Point(1,2);
     Point p3 = new Point(2,3);
     Point p4 = p1;
     Point p5 = null;
     Point p6 = p5;
     p6 = new Point(3,4);
   if(p1==p2)
     System.out.println("p1 egale p2");
   if(p1==p3)
     System.out.println("p1 egale p3");
   if(p1==p4)
     System.out.println("p1 egale p4");
   if(p1==null)
     System.out.println("p1 egale null");
   if(p1.equals(p2))
     System.out.println("p1 egale p2 (2)");
   if(p1.equals(p4))
     System.out.println("p1 egale p4 (2)");
   }
}
```

Dans cet exercice, on ne souhaite pas faire réécrire la méthode `equals()`, cela ne sera vu qu'en semaine 6 (on a besoin d'Object).
Cet exercice a pour but d'attirer leur attention que l'égalité entre objets n'est pas si simple car il faut regarder au niveau des variables d'instances... on y reviendra ensuite pour les objets composés.

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



Insister sur le vocabulaire : bien distinguer les variables, les instances et les références

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

Rappel : la méthode `equals` (comme `toString`) existe par défaut dans les objets. Son comportement initial est le même que `==`

```
p1 egale p4
p1 egale p4(2)
```

Q 17.3 Donner les sorties associées aux commandes suivantes :

```java
System.out.println(p5);   System.out.println(p6);
System.out.println(p5.toString());   System.out.println(p6.toString());
```

null
3,4
=> ERREUR NullPointerException
3,4

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

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

La première ligne supprime une instance (2, 3) qui n'est plus référencée
La seconde ligne est neutre concernant le nombre d'instances.

Exercice 18 – Sélection de méthode

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

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

Q 18.1 Selon le principe de base de JAVA qui interdit deux signatures identiques pour des méthodes (sans prise en compte du retour), cette classe compile-t-elle ?

Oui... Java très fortement typé

Q 18.2 Donner les affichages associés à l'exécution du programme suivant. Certaines lignes ne compilent pas : indiquer brièvement pourquoi.

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

```
13      }
14 }
```

3 erreurs :
l4 : constructeur avec argument inconnu
l6 : double vers int implicite interdit (perte de précision)
l12 : idem. 1. c'est très différent de 1!!
=======
Affichage :

```
1 maMethode(int i)
2 maMethode(double d) // c'est le type de la variable qui est important !
3 maMethode(double d)
4 maMethode(double d1,double d2) // il n'y a rien d'autre avec deux arguements...
5 // JAVA est tres type mais  tres doue pour trouver des signature compatible
6 maMethode(int, int, int)
```

Quizz 4 – Fleur (constructeur, `this`)

Etudier le programme ci-dessous puis répondre aux questions.

```
1 public class Fleur {
2         private String nom;
3         private String couleur;
4
5         public Fleur (String name, String couleur) {
6                 nom = name;
7                 this.couleur = couleur;
8         }
9         public Fleur (String nom) {
10                this(nom,"rouge");
11        }
12        public String toString() {
13                return nom + "␣de␣couleur␣" + couleur ;
14        }
15        public String getNom() { return nom; }
16 }
17
18 public class TestFleur {
19        public static void main (String[] args ) {
20                Fleur tulipe = new Fleur("Tulipe", "Jaune");
21                System.out.println(tulipe.getNom());
22        }
23 }
```

QZ 4.1 Donner les commandes pour compiler, puis exécuter ce programme.

Si on suppose une classe par fichier :
`javac Fleur.java TestFleur.java`
`java TestFleur` car la méthode main se trouve dans la classe `TestFleur`
Attention : Respecter les majuscules et minuscules.

QZ 4.2 Pourquoi a-t-on déclaré `private` les variables `nom` et `couleur` ?

En déclarant `private` ces variables on restreint leur domaine de visibilité, c'est-à-dire les endroits où l'on a accès à ces variables. Sinon, ces variables seraient modifiables de partout (même depuis la classe `TestFleur`).

QZ 4.3 La variable d'instance `nom` aurait-elle pu être déclarée après la variable `couleur` ? après la méthode `getNom()` ? Si oui, est-ce que cela aurait fait une différence ? Peut-on intervertir les lignes 20 et 21 ?

Oui. Oui. Cela ne fait aucune différence. L'ordre des déclarations des variables et des méthodes n'est pas important. On ne peut pas intervertir les lignes 20 et 21, car dans le corps d'une méthode l'ordre des déclarations est important.

QZ 4.4 Dans la classe `TestFleur`, quelle différence faites-vous entre `tulipe` et `"Tulipe"` ?

`tulipe` est un handle (référence ou "poignée") vers un objet, `"Tulipe"` est un objet de type `String`.

QZ 4.5 Quel est le rôle de la méthode `getNom()` ?

La méthode getNom() permet d'avoir accès au nom de la fleur (en lecture). On appelle ces méthodes « accesseurs » et l'usage est de préfixer leur nom par get (obtenir).

QZ 4.6 Dans le constructeur de la classe `Fleur`, aurait-on pu écrire `this.nom = name` ?

Rien ne s'oppose à ce que l'on emploie this mais cela n'a pas d'intérêt, car il n'y a pas d'ambiguïté à lever, contrairement à this.couleur=couleur ; où il faut lever l'ambiguïté.

QZ 4.7 Si dans la méthode main, on rajoute l'instruction : `tulipe.toString();` Quel est le résultat produit par cette instruction ?

La méthode standard `toString()` retourne une chaîne de caractères, mais ne l'affiche pas.
Donc l'instruction `tulipe.toString();` ne produit rien, car `toString()` retourne une valeur qui n'est ici pas utilisée
Il ne faut pas oublier d'écrire : `System.out.println(tulipe.toString());`

QZ 4.8 Un étudiant rajoute le constructeur suivant. Quelle erreur est signalée à la compilation ?

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

Dans la classe `Fleur`, il existe déjà un constructeur ayant la même signature (même type de paramètre, le nom du paramètre n'est pas important). Or il ne peut y avoir qu'un seul constructeur avec la même signature.

QZ 4.9 Un autre étudiant rajoute dans la classe `Fleur` le constructeur suivant. (a) Quelle erreur est signalée à la compilation ? (b) Quelle instruction l'étudiant a-t-il ajouté inutilement ? Expliquer.

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

(a) `TestFleur.java` : call to this must be first statement in constructor
L'appel au constructeur doit être la première instruction du constructeur.
(b) L'instruction : couleur="rouge" ; est inutile, car le constructeur qui est appelé réalise déjà cette instruction.

Quizz 5 – Encapsulation

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

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

p1.x : erreur x est déclarée private
p1.y : OK, mais il ne faut pas le faire, car afin de protéger les données, les variables doivent être déclarées private
p1.f1() ; : OK
p1.f2() ; : erreur, car f2() est une méthode privée, l'intérêt des méthodes privées est limité, car on ne peut pas les utiliser à l'extérieur de la classe, donc on évitera les méthodes privées.

Quizz 6 – Méthode `toString()`

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

Non, il y a une erreur à la compilation. k est un type simple, ce n'est pas un objet (une référence vers un objet), on ne peut pas appeler de méthodes à partir de k. Test.java :10 : int cannot be dereferenced

QZ 6.2 Soit la classe suivante :

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

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

Les instructions (a) et (b) affichent le `toString()` de Fleur. Dans `System.out.println(f1.toString());`, l'appel est explicite, dans `System.out.println(f1);` l'appel est fait automatiquement par le système. Il n'existe pas de méthode avec la signature :
System.out.println(Fleur f)
donc le système convertit l'objet Fleur en appelant automatiquement la méthode toString() de l'objet.
(c) "Affichage de :", puis retour à la ligne et tabulation, puis "Je suis une fleur".

TME 2 : Exercices simples et composition

Remarque : les intitulés des exercices de cette séance sont aussi disponibles en ligne sur le site Moodle de l'UE : https://moodle-sciences.upmc.fr/moodle-2020/course/view.php?id=2403.

Pour compléter : choisir des exercices parmi les exos de TD 2, et/ou aller sur le Moodle.
Exercices simples : 19 et 20

Exercice 19 – Course de relais 4 fois 100m

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

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

Q 19.2 Ajouter dans la classe `Coureur`, les constructeurs suivants :
- un constructeur prenant un seul paramètre correspondant au numéro du dossard, qui initialise `tempsAu100` avec un nombre aléatoire choisi dans l'intervalle $[12, 16[$, et `possedeTemoin` avec `false`,
- un constructeur sans paramètre qui appelle le constructeur à un paramètre et qui initialise `numDossard` avec un entier choisi aléatoirement entre 1 et 1000.

```
1 public class Coureur {
2         private int numDossard;
3         private double tempsAu100;
4         private boolean possedeTemoin;
5
6         public Coureur(int numD) {
7                 numDossard=numD;
8                 tempsAu100=Math.random()*4+12; // entre 12 et 16 exclu
9                 possedeTemoin=false;
10        }
11
12        public Coureur() {
13                this((int)(Math.random()*1000+1)); // entre 1 et 1000
14        }
15 }
```

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

```
1 public class TestCoureur {
2         public static void main(String [] args) {
3                 Coureur c1=new Coureur();
4                 Coureur c2=new Coureur();
5                 Coureur c3=new Coureur();
6                 Coureur c4=new Coureur();
7
8                 System.out.println("Presentation␣des␣coureurs␣:\n");
9                 System.out.println(c1.toString());
10                System.out.println(c2.toString());
11                System.out.println(c3.toString());
12                System.out.println(c4.toString());
```

Q 19.4 Ajouter et tester au fur et à mesure dans la méthode `main()` de `TestCoureur` les méthodes suivantes :
- les accesseurs : `int getNumDossard()`, `double getTempsAu100()`, `boolean getPossedeTemoin()`,
- le modifieur : `void setPossedeTemoin(boolean b)` qui modifie 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`.

```
1         public int getNumDossard() { return numDossard; }
2         public double getTempsAu100() {return tempsAu100; }
3         public void setPossedeTemoin(boolean b) { possedeTemoin=b; }
4
5         public String toString() {
6                 String s;
7                 if (aLeTemoin) s="oui";
8                 else s="non";
9                 return "Coureur␣"+ numDossard
10                                + "␣tempsAu100␣:␣"+ tempsAu100+ "s␣au␣100m"
11                                + "␣possedeTemoin␣:␣" + s ;
12        }
```

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

```
1         public void passeTemoin(Coureur c) {
2                 System.out.println("Moi,␣coureur␣"+ this.numDossard
3                         +",je␣passe␣le␣temoin␣au␣coureur␣"+c.numDossard + "\n");
4                 setPossedeTemoin(false);
5                 c.setPossedeTemoin(true);
6         }
7
8         public void courir() {
9                 System.out.println("Moi,␣coureur␣"+ numDossard + ",␣je␣cours\n");
10        }
```

Q 19.6 Ajouter dans la méthode main les instructions qui permettent de :

— faire courir en relais 4 fois 100m les quatre coureurs dans l'ordre c1, c2, c3, c4.
— calculer et afficher le temps total mis par les coureurs pour faire les 400m.

```
System.out.println("Attention, la course va commencer...\n");

System.out.println("Pan!\n");
c1.setPossedeTemoin(true);
c1.courir();
c1.passeTemoin(c2);
c2.courir();
c2.passeTemoin(c3);
c3.courir();
c3.passeTemoin(c4);
c4.courir();
double tempsTotal=c1.getTempsAu100()+c2.getTempsAu100()+
        c3.getTempsAu100()+c4.getTempsAu100();

System.out.println("Fin de la course\n");
System.out.println("Temps total : " + tempsTotal+ "\n");
```

Exercice 20 – Gestion des complexes

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

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

```
public class Complexe{
 private double reelle, imag;

 public Complexe(double reelle, double imag){
   this.reelle = reelle;
   this.imag = imag;
 }
 public Complexe(){
   this(Math.random()*4-2,Math.random()*4-2);
 }
```

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

```
public String toString(){
```

```
2     return "("+reelle+","+imag+"i)";
3  }
4  public boolean estReel(){
5     return imag == 0;
6  }
7  public Complexe addition(Complexe c){
8     return new Complexe(reelle+c.reelle , imag+c.imag);
9  }
10 public Complexe multiplication(Complexe c2) {
11    return new Complexe(reelle*c2.reelle-imag*c2.imag, reelle*c2.imag+imag*c2.reelle);
12 }
```

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

rem : en commentaire ci-dessous, les résultats de chaque print.

```
1  public class TestComplexe{
2   public static void main(String[] args){
3      Complexe c1=new Complexe();
4      Complexe c2=new Complexe(1,0);
5      Complexe c3=new Complexe(1,1);
6
7      System.out.println(c1); System.out.println(c2); System.out.println(c3);
8
9      System.out.println(c2.estReel()); // true
10     System.out.println(c3.estReel()); // false
11
12     System.out.println(c2.addition(c3)); // (2,1i)
13
14     Complexe c4=new Complexe(0,-1);
15     Complexe c5=new Complexe(1,1);
16     Complexe c6=new Complexe(2,2);
17     System.out.println("Mult␣="+c4.multiplication(c4)); // -1
18     System.out.println("Mult␣="+c5.multiplication(c6)); // 4i
19  }
20 }
```

3 Composition, copie d'objets

Objectif : Composition, représentation mémoire complexe.
Exercices conseillés :
TD : l'exercice 21 est un exercice simple de modélisation, il n'y a pas de classes à écrire, juste réfléchir sur les liens. Les exercices 22, 23 et 24.
TME : 25, 26.
Quizz : 7 sur l'instanciation.
Remarque : l'exercice 26 peut être traité en TME (pour les étudiants plus faibles) ou en TD (mais en général, le 24 prend trop de temps pour l'attaquer)

Exercice 21 – Composition/agrégation et modélisation

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

> Dans cette UE, on ne fait pas la différence entre composition et agrégation.
> Rappeler que plusieurs modélisations peuvent être possibles pour le même problème.

1. appartement / immeuble / pièce

```
Immeuble <o>---- Appartement <o>---- Pièce
```

2. camion poubelle / déchet / poubelle

```
Camion <>---- Dechet ----<> Poubelle
```
`Camion` n'a pas d'attribut de type `Poubelle`, car un camion ne stocke pas de poubelles.

3. aéroport / avion / piste

```
Avion ----<> Aéroport <o>--- Piste
```

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

Q 22.1 Soient les classes suivantes :

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

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

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

> Affiche `"Dormeur"`.
> Les variables `unPion` et `autrePion` référence le même objet (faire un schéma mémoire).
> Il y a un seul objet `Pion`.

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

```
1 public Pion (Pion r) {
2         nom=r.nom;     // this(r.nom); possible
3         posx=r.posx;     // ou this.posx=r.posx;
4 }
```

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

```
autrePion=new Pion (unPion);
```

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

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

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

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

Oui.

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

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

// Faire un diagramme mémoire au tableau
On déplace `autrePion` ... mais aussi `unPion`, car il y a un seul objet `Point` commun à deux objets `Pion`.

Q 22.2.3 Proposez une solution pour résoudre le problème.

Quand on copie un pion, il faut copier aussi son objet `Point`, pour que chaque pion ait son propre point.
On ajoute un constructeur par copie dans `Point` que l'on utilise dans le constructeur par copie de `Pion`.
Dans la classe `Point` :

```
1 public Point(Point aCopier) {
2    x=aCopier.x;
3    y=aCopier.y;
4 }
```

On modifie le constructeur par copie de `Pion` :
`posx=new Point(r.posx);`

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

Solution avec `posx` de type `double` :

```
public Pion clone() {
        Pion sosie=new Pion (this.nom);
        sosie.posx=this.posx; // sosie.posx autorisé car on est dans la même classe
        return sosie;
}
```

Solution avec `posx` de type `Point` :

```
public Pion clone() {
        Pion sosie=new Pion (this.nom);
        sosie.posx=this.posx.clone();  // méthode clone() de Point supposée existante
        return sosie;
}
```

Puis dans le main :

```
autrePion=unPion.clone();
```

Exercice 23 – Feu tricolore

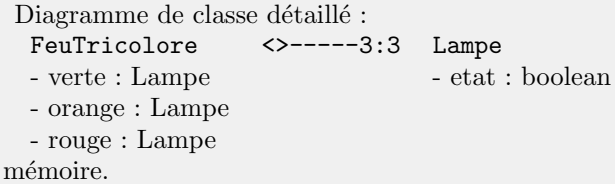
Un feu tricolore est composé de 3 lampes : une verte, une orange et une rouge. Soit la classe `Lampe` suivante :

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

Q 23.1 Dessiner le diagramme de classe.

Diagramme de classe simple :
`FeuTricolore<>----- Lampe`.
On peut ajouter la cardinalité `3:3`

Diagramme de classe détaillé :



Attention : à ne pas confondre avec le diagramme mémoire.

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

Le but de ces questions est que les étudiants comprennent les différentes façons d'utiliser un constructeur lorsque l'on a des objets en variable d'instance. Soit on crée les objets dans le constructeur (constructeur 1), soit on crée les objets avant, puis on les passe en paramètre (constructeur 2). Dans tous les cas, il faut créer une et une seule fois chaque objet.

```
class FeuTricolore {
    private Lampe verte,orange,rouge;
    public FeuTricolore() {
        verte=new Lampe();
        orange=new Lampe();
```

```
6           rouge=new Lampe();
7           // ou bien : this(new Lampe(),new Lampe(),new Lampe());
8       }
9       public FeuTricolore(Lampe v,Lampe o,Lampe r) {
10          verte=v;
11          orange=o;
12          rouge=r;
13      }
14  }
15  // Creation des objets avant, puis passage en parametre
16          Lampe l1=new Lampe();
17          Lampe l2=new Lampe();
18          Lampe l3=new Lampe();
19          FeuTricolore ft1=new FeuTricolore(l1,l2,l3);
20          FeuTricolore ft2=new FeuTricolore(new Lampe(),new Lampe(),new Lampe());
```

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

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

Il n'y a qu'une seule lampe. Les trois références : verte, orange, rouge pointent vers la même lampe.

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

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

lp1 et lp2 pointent vers le même objet Lampe. Lorsque l'on crée le feu tricolore, les références verte, orange et rouge pointent vers le même objet Lampe.

Exercice 24 – Mariage (composition récursive)

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

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

Une autre possibilité serait :

```
public Personne() {  // mais dans ce cas, il faut que tirageLettre() soit static...
                this("pers"+  tirageLettre()+ tirageLettre()+ tirageLettre());
        }
```

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

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

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

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

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

```
public class Personne {
        private String nom;
        private Personne conjoint;

        public Personne() {
                this("pers" + tirageLettre()+tirageLettre()+tirageLettre());
        }
        public Personne(String nom) {
                this.nom=nom;
                conjoint=null;
        }
        private char tirageLettre(){
                return (char) ((int) (Math.random()*26) + 'A');
        }

        public String toString() {
                String civilite;
                if (conjoint == null)
                        civilite= "␣celibataire";
                else
                        civilite= "␣marie(e)";
```

```
22                  return nom +"␣," + civilite + "␣";
23          }
24          public void epouser(Personne p) {
25                  if ((this == p) || (this.conjoint != null) ||
26                          (p.conjoint != null)) {
27                          System.out.println("Ce␣mariage␣est␣impossible!␣\n");
28                          return;
29                  }
30                  System.out.println(""+this+"␣se␣marie␣avec␣" + p + "\n");
31                  conjoint=p;
32                  p.conjoint=this;
33          }
34          public void divorcer() {
35                  if (conjoint == null) || (conjoint.conjoint == null)  {
36                          System.out.println("Divorce␣impossible!");
37                          return;
38                  }
39                  System.out.println(""+this+"␣divorce␣de␣" + conjoint + "\n");
40                  this.conjoint.conjoint=null;
41                  this.conjoint=null;
42          }
43  }
44  //====================
45  public class TestMariage {
46      public static void main(String[] args) {
47           Personne p1,p2,p3;
48           p1=new Personne();
49           p2=new Personne();
50           p3=new Personne();
51           System.out.println("Les␣personnes␣:\n");
52           System.out.println(""+ p1+"\n"+ p2 + "\n" + p3+"\n");
53
54           System.out.println("Mariage␣de␣"+p1+"␣et␣de␣"+p2+"␣:␣");
55           p1.epouser(p2);
56           System.out.println("Les␣personnes␣apres␣mariage␣:\n");
57           System.out.println(""+ p1+"\n"+ p2 +"\n");
58           System.out.println("Essai␣de␣mariage␣de␣"+p1+"␣et␣de␣" +p3+"␣:␣");
59           p1.epouser(p3);
60           System.out.println("Divorce␣de␣"+ p1 + "␣et␣de␣" +p2+"␣:␣");
61           p1.divorcer();
62           System.out.println("Les␣personnes␣apres␣divorce␣:\n");
63           System.out.println(""+ p1+"\n"+ p2 +"\n");
64      }
65  }
```

Exercice 25 – Classe triangle

Q 25.1 Écrire une classe `Point` à deux variables d'instance entières `posx` et `posy`, respectivement l'abscisse et l'ordonnée du point. Cette classe comprendra :

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

cet objet (c'est-à-dire l'objet sur lequel est invoquée cette méthode).
— La méthode `deplaceToi(int newx, int newy)` qui déplace le point en changeant ses coordonnées.

```
1 public class Point {
2         private int posx, posy;
3
4         public Point () {
5                 posx = 0;     // ou posx = (int)(Math.random() * 10)
6                 posy = 0;     // ou posy = (int)(Math.random() * 10)
7         }
8         public Point (int valx, int valy) {
9                 posx = valx;
10                posy = valy;
11        }
12
13        public void setPosx (int x) { posx = x; }
14        public void setPosy (int y) { posy = y; }
15        public int getPosx (int x) { return posx; }
16        public int getPosy (int y) { return posy; }
17
18        public String toString() { return "("+posx+","+posy+")"; }
19        public int carre(int nb) { return nb * nb; }
20
21        public float distance(Point p) {
22                float res = carre(posx - p.posx) + carre(posy-p.posy);
23                return (float)Math.sqrt(res);
24        }
25        public void deplaceToi(int newx, int newy) {
26                posx = newx;
27                posy = newy;
28        }
29        public void deplaceToiRel(int deltaX, int deltaY) {
30                posx += deltaX;
31                posy += deltaY;
32                // ou : deplaceToi(x+deltaX, y+deltaY)
33        }
34 }
```

Q 25.2 Tester cette classe en écrivant la méthode `main` qui crée des points et affiche leurs coordonnées.

```
1 public class TestPoint {
2     public static void main(String[] args) {
3          Point p1 = new Point(2, 5);
4          System.out.println("Coordonnee de p1 : "+p1.toString());
5          Point p2 = new Point(10, 5);
6          System.out.println("Coordonnee de p2 : "+p2.toString());
7
8          System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
9          p1.deplaceToi(0,0);
10         System.out.println ("Nouvelles coordonnees de p1 : "+p1.toString());
11         System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
12         p1.deplaceToiRel(2,5);
13         System.out.println ("Nouvelles coordonnees de p1 : "+p1.toString());
14         System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
15    }
16 }
```

Q 25.3 Ecrire une classe `Triangle` à trois variables d'instance prenant leur valeur dans la classe `Point`. Cette classe comprendra :

- Un constructeur par défaut.
- Un constructeur à trois paramètres : les trois sommets du triangle.
- Une méthode `getPerimetre()` qui retourne le périmètre du triangle.
- Redéfinir la méthode `public String toString()` qui retourne une chaîne de caractères décrivant le triangle (en utilisant la méthode `toString()` de la classe `Point`).

```
1 public class Triangle {
2         private Point A;
3         private Point B;
4         private Point C;
5
6         public Triangle () {
7                 A = new Point();
8                 B = new Point();
9                 C = new Point();
10        }
11        public Triangle (Point valA, Point valB, Point valC) {
12                A = valA;
13                B = valB;
14                C = valC;
15        }
16        public String toString() {
17                return "{"+A.toString()+";"+B.toString()+";"+C.toString()+"}";
18        }
19
20        // retourne la longueur du cote A-B
21        public float baseAB() {return A.distance(B);}
22
23        // retourne la longueur du cote B-C
24        public float baseBC() {return B.distance(C);}
25
26        // retourne la longueur du cote A-C
27        public float baseAC() {return A.distance(C);}
28
29        // retourne le perimetre d'un triangle
30        public float perimetre() {return baseAB()+baseAC()+baseBC();}
31 }
```

Q 25.4 Écrire une classe `TestTriangle`, contenant une méthode `main` dans laquelle on crée 3 points, puis un triangle composé de ces 3 points. On affichera ensuite les caractéristiques du triangle (les 3 points, la longueur de ses côtés et son périmètre).

```
1 public class TestTriangle {
2     public static void main(String[] args) {
3          Point p1 = new Point(-1, 0);
4          System.out.println ("Coordonnee de p1 : "+p1.toString());
5          Point p2 = new Point(1, 0);
6          System.out.println ("Coordonnee de p2 : "+p2.toString());
7          Point p3 = new Point(2, 1);
```

```
8           System.out.println ("Coordonnee␣de␣p3␣:␣"+p3.toString());
9           Triangle t = new Triangle(p1, p2, p3);
10          System.out.println ("Coordonnee␣du␣triangle␣:␣"+t.toString());
11          System.out.println ("Son␣cote␣A-B␣est␣de␣:␣"+t.baseAB());
12          System.out.println ("Son␣cote␣A-C␣est␣de␣:␣"+t.baseAC());
13          System.out.println ("Son␣cote␣B-C␣est␣de␣:␣"+t.baseBC());
14          System.out.println ("Son␣perimetre␣est␣de␣:␣"+t.perimetre());
15      }
16  }
```

Q 25.5 Comment tester l'égalité structurelle entre deux triangles ? Réfléchir à l'organisation du code et aux signatures des méthodes puis proposer une implémentation dans les différentes classes.

(1) signature : dans la classe `Triangle` : public boolean egalite(Triangle t)
Insister sur la signature (un seul argument)... Il est souvent nécessaire de donner un exemple d'usage depuis le main.
(2) Besoin d'un test dans la classe `Point` (surtout, on ne passe pas les attributs en public !) public boolean egalite(Point t)
(3) implémentation

```
1 // Dans la classe Triangle:
2 public boolean egalite(Triangle t){
3   return p1.egalite(t.p1) && p2.egalite(t.p2) && p3.egalite(t.p3);
4 }
5 // Dans la classe Point:
6 public boolean egalite(Point p){
7   return x.egalite(p.x) && y.egalite(p.y) ;
8 }
```

Exercice 26 – Tracteur (composition d'objets et copie d'objets)

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

Q 26.1 Donner le diagramme de classe correspondant.

```
Tracteur <>---- Roue
         <>--- Cabine
```

Q 26.2 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()`.

```
1 public class Roue{
2         private int diametre;
3
4         public Roue(int diam){ diametre = diam; }
5         public Roue(){ this(60); }
6         public String toString(){
7                 return "Roue␣de␣diametre:␣" + diametre + "\n";
8         }
9 }
```

Q 26.3 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.

Voir le **main** ci-dessous.

Q 26.4 Écrire une classe `Cabine` qui a un volume (en mètres cubes (m3)) et une couleur de type `String`. Écrire un constructeur avec paramètres et la méthode `toString()` qui rend une chaîne de caractères donnant le volume et la couleur. Ajouter le modifieur `setCouleur(String couleur)`.

```
1 public class Cabine{
2         private int volume;
3         private String couleur;
4 
5         public Cabine(int vol, String coul){
6                 volume = vol;
7                 couleur = coul;
8         }
9         public String toString(){
10                return "Cabine de Volume : " + volume + " et de Couleur : " + couleur;
11        }
12        public void setCouleur(String couleur) {
13                this.couleur=couleur;
14        }
15 }
```

Q 26.5 Ajouter dans la méthode `main` la création d'une cabine bleue.

```
1 Cabine cab = new Cabine(500, "bleu");
```

Q 26.6 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.

```
1 public class Tracteur{
2         private Cabine cab;
3         private Roue r1,r2,r3,r4; // On peut faire aussi avec un tableau de 4 roues
4         public Tracteur(Cabine c,Roue petite1,Roue petite2,Roue grande1,Roue grande2){
5                 cab = c;
6                 r1=petite1; r2=petite2; r3=grande1; r4=grande2;
7         }
8 
9         public Tracteur(){
10                this(new Cabine(400,"verte"),new Roue(),new Roue(),
11                        new Roue(150),new Roue(150));
12        }
13        public String toString(){
14                return "Tracteur :\n"+ cab + "\nLes roues :\n"+r1 + r2 + r3 + r4;
15        }
16 
17        public void peindre(String couleur) {
```

```
18                  cab.setCouleur(couleur);
19          }
20 }
```

Q 26.7 Créer un tracteur `t1` dans la méthode `main` avec les 4 roues et de la cabine bleue créées précédemment. Afficher ensuite ce tracteur.

Q 26.8 Ajouter l'instruction `Tracteur t2=t1;` puis modifier la couleur de la cabine du tracteur `t2`. Quelle est la couleur de la cabine de `t1` ? Expliquer pourquoi la couleur a changée. Que faut-il faire pour que `t1` et `t2` soient deux objets distincts qui ne contiennent pas les mêmes objets ? Expliquer et appliquer cette correction.

```
1 public class TestTracteur{
2         public static void main(String[] args){
3                 Roue r1 = new Roue(120);
4                 Roue r2 = new Roue(120);
5                 Roue r3 = new Roue();
6                 Roue r4 = new Roue();
7                 Cabine cab = new Cabine(500, "bleu");
8                 Tracteur t1 = new Tracteur(cab,r1,r2,r3,r4);
9                 System.out.println(t1);
10
11                Tracteur t2=t1;
12                t2.peindre("rouge");
13                System.out.println(t1);
14                Tracteur t3=new Tracteur(t1);
15                t3.peindre("jaune");
16                System.out.println(t1);
17        }
18 }
```

Il faut ajouter à chaque classe `des` constructeurs par copie ou une méthode de clonage. NB : le constructeur de copie c'est plutôt C++, le clonage plutôt JAVA.

```
1 public Roue(Roue r){ this(r.diametre); }
2 public Cabine(Cabine c) { this(c.volume,c.couleur); }
3 public Tracteur(Tracteur t) {
4         this(new Cabine(t.cab), new Roue(t.r1),new Roue(t.r2),new Roue(t.r3),new Roue(t
              .r4));
5 }
```

Quizz 7 – 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 7.1 La classe `A` contient-elle un constructeur ?

Si une classe ne contient pas de constructeurs, automatiquement java lui ajoute un constructeur (appelé constructeur par défaut) qui est un constructeur sans paramètre : `public A() {}`. Si par la suite, on rajoute un constructeur, ce constructeur est supprimé.

QZ 7.2 Combien d'objets sont-ils créés ?

Il y a deux objets créés, car il y a deux new (faire un schéma pour expliquer).

QZ 7.3 Combien de références (appelées aussi *handles*) utilisés ?

Il y a 4 handles (a1,a2,a3,a4). Expliquez null.

QZ 7.4 Que se passe-t-il si on rajoute l'instruction `a3=null;` ? `a2=null;` puis `a1=null;` ?

`a3=null;` : Plus aucune référence ne pointe vers le deuxième objet, le ramasse-miette (en anglais, garbage collector) libère l'espace mémoire pour cet objet.
`a2=null;` : a1 pointe toujours vers le premier objet, donc cet objet n'est pas détruit.
`a1=null;` : plus aucune référence ne pointe vers le premier objet, donc le ramasse-miettes libère l'espace mémoire pour cet objet.

4 Tableaux

Séance 4 :
Objectif : syntaxe et représentation mémoire des tableaux
Exercices conseillés :
TD : 27, 28, 32, 33.
TME : 30, 31 et implémentation de 33.
A faire si temps ou pour les plus faibles : 29
Quizzes : 8, 9, 10.

Exercice 27 – Base syntaxique

Q 27.1 Donner les lignes de commande pour déclarer et remplir un tableau de 10 `double` (tirés aléatoirement). Afficher le tableau dans la console (en utilisant obligatoirement l'accesseur à la dimension du tableau).

Q 27.2 En supposant une classe `Point` existante, déclarer, remplir et afficher un tableau contenant 10 instances de `Point`.

```
1  double[] tabd = new double[10];
2  for(int i=0; i<tabd.length; i++){ // on ne doit pas voir de constante (10) dans la
       boucle
3    tabd[i] = Math.random();
4    System.out.println(tabd[i]); // on peut faire dans une autre boucle pour plus de
         clarté
5  }
6
7  Point[] tabp = new Point[10];
8  for(int i=0; i<tabp.length; i++){ // on ne doit pas voir de constante (10) dans la
       boucle
9    tabp[i] = new Point(Math.random(), Math.random());
10   System.out.println(tabp[i]); // on peut ajouter .toString() ou pas
11 }
```

Q 27.3 Quel affichage correspond aux lignes suivantes ?

```
1 int [] t1 = {1,2,3}; // syntaxe reduite correcte (a connaitre)
2 int [] t2 = {1,2,3};
3 int [] t3 = t1;
4 System.out.println(t1 == t2); System.out.println(t1 == t3);
```

false true
Ce sont des objets (presque comme les autres)

Exercice 28 – N-uplets

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 28.1 Ecrire la classe `NUplet` qui contient pour seul attribut un tableau `tab` d'entiers et les constructeurs :

- un constructeur : `NUplet(int n)` qui réserve `n` cases mémoires pour le tableau référencé par `tab`
- un constructeur : `NUplet(int n, int x)` qui crée un tableau de n cases mémoires et initialise toutes les cases du tableau à la même valeur x (exemple : le 5-uplet (3,3,3,3,3)). Attention : on demande que vous appeliez le constructeur à un paramètre
- un constructeur : `NUplet(int a, int b, int c)` qui crée le triplet `(a,b,c)`. Attention : on demande que vous appeliez le constructeur à un paramètre

```
1 public class NUplet {
2         private int [] tab;
3
4         public NUplet(int n) {
5                 tab=new int [n];
6         }
7         public NUplet(int n,int x) {
8                 this(n);
9                 for(int i=0;i<tab.length;i++) {
10                        tab[i]=x;
11                }
12        }
13        public NUplet(int a, int b, int c) {
14                this(3);
15                tab[0]=a; tab[1]=b; tab[2]=c;
16        }
```

Remarques :

- comme le nombre de cases mémoires du tableau dépend du constructeur appelé, on est obligé de réserver l'espace mémoire pour le tableau dans les constructeurs, et non pas lors de la déclaration.
- le premier constructeur pourrait être privé

Q 28.2 On suppose que l'on est dans une méthode `main`, donnez les instructions pour créer le 5-uplet (3,3,3,3,3) et le triplet (7,8,9).

```
1 NUplet u1=new NUplet(5,3);
2 NUplet u2=new NUplet(7,8,9);
```

Q 28.3 Ajouter à la classe `NUplet` les méthodes suivantes :

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

```
1         public String toString() {
2                 if (tab.length==0) {
3                         return "()";
4                 }
5                 String s="(";
6                 for(int i=0; i<(tab.length-1); i++) {
7                         s=s+tab[i]+",";
8                 }
9                 s=s+tab[tab.length-1];
10                return s+")";
11        }
12        public int somme() {
13                int s=0;
14                for(int x : tab ) { // for sans modification du tableau
15                        s+=x;
16                }
17                return s;
18        }
```

Remarque : dans `toString()`, ce n'est pas adapté d'utiliser la boucle `for` pour les tableaux, car il y a (n-1) virgules à afficher.

Q 28.4 On ajoute dans la classe `NUplet` le constructeur ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème, puis proposez une solution.

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

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

Cela affiche `(50,2,3)`. Car il y a un seul tableau référencé par deux variables : `t123` dans le main et l'attribut `tab` de l'objet référencé par `u3`.
Solution : il faut copier (ou dupliquer) le tableau dans le constructeur

```
public NUplet(int [] tab) {
  this(tab.length);
  for(int i=0;i<tab.length;i++)
     this.tab[i]=tab[i];
}
```

Conclusion : faire attention quand on passe un tableau en paramètre à ce genre de problème

Q 28.5 On ajoute dans la classe `NUplet` la méthode `getTab` ci-dessous à gauche. Qu'affiche les instructions à droite ? Expliquez le problème, puis proposez une solution.

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

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

Cela affiche `(70,5,6)`. Car il y a un seul tableau référencé par deux variables : l'attribut `tab` de l'objet référencé par `u4` et `t456` dans le main.
Solution : il faut copier (ou dupliquer) le tableau dans la méthode `getTab()`

```
public int [] getTab() {
  int [] tab2=new int [tab.length];
  for(int i=0;i<tab.length;i++)
    tab2[i]=tab[i];
  return tab2;
}
```

Conclusion : faire attention quand on retourne un tableau à ce genre de problème

Q 28.6 Écrire une méthode `public boolean egal(NUplet n2)` qui rend vrai si `n2` est égal au n-uplet courant.

```
public boolean egal(NUplet n2) {
  if (n2==null) return false;
  if (this == n2) return true;
  if(tab.length != n2.tab.length)
      return false;
  for(int i=0;i<tab.length;i++) {
      if (tab[i]!=n2.tab[i])
          return false;
  }
  return true;
}
```

Syntaxe d'appel : `u1.egal(u2)`

Exercice 29 – Tableau d'entiers

Q 29.1 Écrire une classe `TableauInt` qui comporte une variable d'instance `tab` de type tableau de 10 entiers. Cette classe contient deux constructeurs :

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

```
1  public class TableauInt{
2          private int[] tab;
3
4          public TableauInt() {
5                  tab = new int[10];
6                  for (int i=0; i<tab.length; i++)
7                          tab[i]=(int)(Math.random()*101);  // de 0 à 100
8          }
9          public TableauInt(int n) {
10                 tab = new int[10];
11                 for (int i=0; i< tab.length; i++)
12                         tab[i]=n+i;     // de n à n+tailleMax
13         }
14 }
```

Q 29.2 Ajouter dans cette classe les trois méthodes suivantes :
- une méthode `public String toString()` qui rend une chaîne représentant les valeurs du tableau sous la forme : `"[a0, a1, a2, ...]"`.
- une méthode `rangMax` qui renvoie le rang du maximum du tableau.
- une méthode `somme` qui renvoie la somme des éléments du tableau.

Q 29.3 Tester ces méthodes au fur et à mesure dans la méthode `main` d'une classe `TestTableau`.

```
public String toString(){
        if (tab.length==0) return "[]";
        String ch = "[";
        for (int i=0; i<tab.length-1; i++) {
                ch += tab[i];
                ch += ",";
        }
        ch += tab[tab.length-1];
        ch += "]";
        return ch;
}

// Calcul le rang (indice) de l'element maximum du tableau
public int rangMax() {
        int rang = 0;
        for (int i=1; i<tab.length; i++) {
                if (tab[i] > tab[rang])
                rang = i;
        }
        return rang;
}

// calcule la somme des elements du tableau
public int somme() {
        int som = 0;
        for (int i=0; i<tab.length; i++) {
                som += tab[i];
        }
        return som;
}
```

Q 29.4 Ajouter dans la classe `TableauInt` une méthode `boolean egal(TableauInt t)` qui teste si cet objet de type `TableauInt` a les mêmes entiers aux mêmes places que le tableau `t` passé en paramètre.

```
// retourne l'egalite de deux tableaux (memes elements)
public boolean egal(TableauInt ref) {
        if (ref == null) return false;
        if (this == ref) return true;
        // les tableaux font la même taille par définition de la classe...
        for (int i=0; i<tab.length; i++) {
                if (tab[i] != ref.tab[i])
                        return false;
        }
        return true;
}
```

Insister sur le fait qu'à partir de la séance 6 on n'utilisera plus que la forme standard de equals

Exercice 30 – Histogramme de notes

Dans cet exercice, il s'agit d'écrire un programme qui permet de représenter un histogramme de notes entières comprises entre 0 et 20 (c'est-à-dire il y a 21 notes possibles).

Par exemple, si dans une classe, il y a 10 étudiants qui ont obtenus les notes suivantes : 2, 3, 4, 3, 0, 0, 2, 3, 3, 2 (c'est-à-dire 2 étudiants ont obtenu la note 0, 0 étudiant ont obtenu la note 1, 3 étudiants la note 2, 4 étudiants la note 3, 1 étudiant la note 4), le tableau représentant l'histogramme sera : [2, 0, 3, 4, 1].

L'affichage de l'histogramme correspondant donnera :

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

Q 30.1 On souhaite écrire une classe `Histo` qui affiche un histogramme des notes. Pour cela, vous définirez :

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

La déclaration de NBNOTES est facultative... Mais c'est plus propre. On peut aussi en profiter pour introduire les constantes qui ne seront abordées que plus tard en cours. On corrige en aveugle (ils admettent) et on leur dit que les explications viendront dans le cours 5.

```
1 public class Histo {
2         private final static int NBNOTES=21;
3         private int [] hist;
4
5         public Histo(){
6                 hist=new int[NBNOTES];
7                 for (int i=0;i<hist.length;i++)
8                         hist[i]=0;
9         }
10        public Histo(double[] notes){
11                this(); // creation du tableau
12                int x;
13                for (int i=0; i<notes.length; i++) {
14                        x= (int) notes[i]; // conversion en entier
15                            // on ne teste pas la validite de la note... On fait l'
                                  hypothese que c'est OK
16                        hist[x]+=1;
17                }
18        }
19 }
```

Discussion facultative sur la déclaration de Histo() en `private` : il ne sert a rien au client, autant le cacher

Q 30.2 Ajouter à cette classe, une méthode `afficheHistogrammeTableau()` qui affiche l'ensemble des valeurs du tableau histogramme.

Q 30.3 Ajouter à cette classe, une méthode `afficheHistogramme()` qui affiche le résultat sous forme d'un histogramme, c'est-à-dire en associant à chaque élément du tableau une ligne comprenant autant de `*` que la valeur de cet élément. (comme dans l'exemple de l'énoncé)

```
1 public void afficheHistogrammeTableau() {
2         for (int i=0;i<hist.length;i++)
3                 System.out.print(hist[i]+"␣");
4         System.out.println();
5 }
6 public void afficheHistogrammeTableau(String mes){     // exemple facultatif
7         System.out.print(mes+"␣:␣");
8         afficheHistogrammeTableau();
9 }
10 public void afficheHistogramme(){
11         for (int i=0;i<hist.length;i++){
12                 System.out.print(i+"␣|␣");
13                 for (int j=1;j<=hist[i];j++)
14                         System.out.print("*");
15                 System.out.println();
16         }
17 }
18 public String toString() {
19         String s=new String();  // ou String s = "";
20         for (int i=0;i<hist.length;i++)
21                 s=s.concat(hist[i]+"␣");
22         return s;
23 }
```

Il existe 2 méthodes afficheHistogrammeTableau(), elles ont des signatures différentes. Il s'agit d'une ***surcharge***. La méthode **toString()** a la même signature que celle de Object. Il s'agit d'une ***redéfinition*** (mais ce n'a pas encore été vu en cours).

Q 30.4 Écrire une classe `TestHisto` dont la méthode `main` crée un tableau de notes aléatoires (150 étudiants), une instance de `Histo`, puis qui affiche le résultat sous les deux formes proposées.

```
1 public class TestHisto {
2         public static void main(String [] args){
3                 int n = 150;
4                 double[] notes = new notes[n];
5                 for(int i=0; i<n;i++) notes[i] = Math.random()*20;
6                 Histo physique=new Histo(notes);
7                 physique.afficheHistogrammeTableau();
8                 physique.afficheHistogrammeTableau("Voici␣l'histogramme");
9                 System.out.println(physique);   // toString() implicite
10         }
11 }
```

Exercice 31 – Pile

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

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

Q 31.1 Définir la classe `Machin` qui se caractérise par un nom et une valeur (*remarque* : `Machin` pourrait un objet plus sophistiqué, mais là n'est pas l'objet de l'exercice).

> Si on veut faire court, ne pas hésiter à faire une pile d'entiers (ou de `Point` que l'on a bcp vu en cours)

Q 31.2 Définir la classe `Pile` avec ses variables d'instance ou de classe, un constructeur qui a comme paramètre la taille maximale de la pile, ainsi que les méthodes données ci-dessus.

Q 31.3 Tester cette classe en écrivant une méthode `main` qui empile trois `Machin` précédemment initialisés, dépile deux fois, puis empile un autre `Machin`. Afficher le contenu de la pile après chacune de ces opérations

```
1 public class Machin {
2         private String nom;
3         private int valeur;
4 
5         public Machin(String nom,int valeur){
6                 this.nom=nom;
7                 this. valeur = valeur;
8         }
9         public String getNom() {return nom;}
10        public int getValeur() {return valeur;}
11 
12        public String toString() {
13                return "(" + nom + "," + valeur + ")";
14        }
15 }
16 
17 public class Pile {
18        private Machin[] tab;
19        private int indSommet;  // indice du sommet de pile
20 
21        public Pile(int tailleMax) {
22                tab = new Machin[tailleMax];
23                indSommet = -1;
24        }
25 
26        public void empiler(Machin m){  // empiler en queue
27                if (estPleine()) {
28                        System.out.println("la pile est pleine");
29                        return;
30                } else {
31                        indSommet++;
32                        tab[indSommet] = m;
33                }
34        }
35        public Machin depiler(){        // depiler si possible
36                if (estVide()){
37                        System.out.println("tableau est vide");
38                        return null;
39                } else {
40                        indSommet =  indSommet - 1;
41                        return tab[indSommet + 1];
42                }
43        }
44 
45        public boolean estVide(){return (indSommet < 0);}
46        public boolean estPleine(){return (indSommet >= tab.length);}
```

```
47
48          public String toString(){
49                  String s = "";
50                  if (estVide()) return "la pile est vide!\n";
51                  for (int i=0;i<=indSommet;i++)
52                          s += (" "+tab[i]);
53                  return s+"\n";
54          }
55  }
56
57  public class TestPileTableau{
58          public static void main(String[] args) {
59                  // creation des differents elements de type Machin
60                  Machin a = new Machin("a",1);
61                  Machin b = new Machin("b",2);
62                  Machin c = new Machin("c",3);
63                  Machin d = new Machin("d",4);
64                  Machin e = new Machin("e",5);
65                  Machin f = new Machin("f",6);
66                  Pile t = new Pile(10);
67                  System.out.println(t.toString());
68                  t.empiler(a);
69                  System.out.println("apres empilement de (a,1) :\n");
70                  System.out.println(T.toString()); // pile => 1 elements
71                  t.empiler(b);
72                  System.out.println("apres empilement de (b,2) :\n");
73                  System.out.println(T.toString()); // pile => 2 elements
74                  t.empiler(c);
75                  System.out.println("apres empilement de (c,3) :\n");
76                  System.out.println(T.toString()); // pile => 3 elements
77                  t.depiler();
78                  System.out.println("apres depilement de (c,3) :\n");
79                  System.out.println(T.toString()); // pile => 2 elements
80                  t.empiler(b);
81                  System.out.println("apres empilement de (b,2) :\n");
82                  System.out.println(T.toString()); // pile => 3 elements
83                  t.empiler(e);
84                  System.out.println("apres empilement de (e,5) :\n");
85                  System.out.println(T.toString()); // pile => 4 elements
86                  t.depiler();
87                  System.out.println("apres depilement :\n");
88                  System.out.println(T.toString()); // pile => 3 elements
89          }
90  }
```

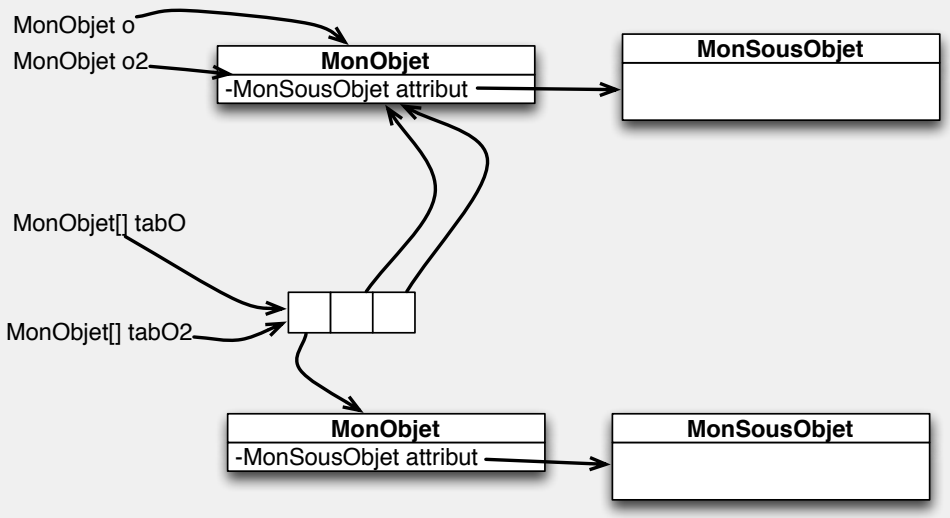
Exercice 32 – Représentation mémoire d'objets et de tableaux

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

Q 32.1 Donner la représentation mémoire correspondant à l'exécution du code suivant. Combien y a-t-il d'instances de `Truc` ?

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

Ca donne une belle toile d'araignée... (on se fiche du sous-objet (trop compliqué sinon)



Q 32.2 Donner les instructions nécessaires pour dupliquer le tableau `tabO`. Etes-vous satisfait du résultat ?

```
Truc[] tabO2 = new Truc[tabO.length];
for(int i=0; i<tabO.length; i++)
  tabO2[i] = tabO[i];
```

Le résultat n'est pas totalement satisfaisant : le tableau est dupliqué mais pas les éléments de ce tableau... Ne pas hésiter à faire un schéma de l'état de la mémoire au tableau.

Exercice 33 – Triangle de Pascal (tableau à 2 dimensions)

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

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

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

Q 33.1 Écrire une classe `TrianglePascal` qui réserve uniquement la place mémoire nécessaire pour stocker le triangle de Pascal dont le nombre de lignes est passé en paramètre du constructeur.

Q 33.2 Ajouter à la classe `TrianglePascal` une méthode `remplirTriangle()` qui calcule les valeurs du triangle de Pascal et une méthode `toString()` qui rend la chaîne de caractères représentant le tableau sous la forme d'un triangle.

Q 33.3 Écrire une classe `TestTrianglePascal` qui crée plusieurs instances de la classe `TrianglePascal` et les affiche.

Ce qui est intéressant ici, c'est la déclaration du tableau en plusieurs étapes.

```
public class TrianglePascal {
    private int [][] triangle;

    public TrianglePascal(int n) {
        if (n > 0) {
```

```
6                 triangle = new int [n][];
7           } else {
8                 System.out.println("anomalie dans la taille (par defaut : 5)");
9                 triangle = new int [5][];
10          }
11          for (int i = 0 ; i < triangle.length ; i++){
12                triangle[i] = new int[i+1];
13          }
14    }
15    public void remplirTriangle(){
16          for (int i = 0 ; i < triangle.length ; i++){
17                for (int j = 0 ; j < (i+1) ; j++){
18                      if ((j == 0) || (i == j))
19                            triangle[i][j] = 1;
20                      else
21                            triangle[i][j] =triangle[i-1][j-1] + triangle[i-1][j];
22                }
23          }
24    }
25    public String toString(){
26          String s = "";
27          for(int i=0 ; i<triangle.length ; i++){
28                for(int j = 0 ; j<triangle[i].length ; j++)
29                      s += triangle[i][j]+" ";
30                s += "\n";
31          }
32          return s;
33    }
34 }
35 //-----------------------------------------------
36 public class TestTrianglePascal{
37    public static void main (String[] args) {
38          TrianglePascal t1 = new TrianglePascal (6);
39          t1.remplirTriangle();
40          System.out.println(t1.toString());
41          // test cas d'anomalie
42          TrianglePascal t2 = new TrianglePascal (0);
43          t2.remplirTriangle();
44          System.out.println(t2.toString());
45    }
46 }
```

Quizz 8 – Tableaux

QZ 8.1 Créer un tableau `tabD` de `double` à une dimension contenant 3 cases.

```
double [] tabD=new double [3];
double [] tabD2={5,6.1,9.3}; // autre solution : déclaration avec initialisation
double [] tabD3=new double[]{5,6.1,9.3};
```

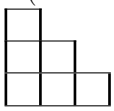
QZ 8.2 Comment peut-on obtenir le nombre d'éléments du tableau tabD ?

```
tabD.length
```

QZ 8.3 Créer un tableau `tabI` d'entiers à deux dimensions de 4 lignes sur 3 colonnes.

```
int [][] tabI=new int [4][3];
int [][] tabI2={{1,1,1},{2,2,2},{3,3,3},{4,4,4}};
```

QZ 8.4 Créer un tableau `tabTriangle` d'entiers à deux dimensions de 3 lignes mais dont la forme est celle d'un triangle (voir dessin ci-après).



La declaration s'effectue en 2 étapes :
- d'abord, on déclare un tableau de 3 lignes où chaque ligne est un tableau d'entiers
- puis, pour chaque ligne, on déclare un tableau d'entiers à la bonne taille

```
int [][] tabTriangle=new int [3][];
for(int i=0;i<tabTriangle.length;i++) tabTriangle[i]=new int [i+1];
// Remarque : tabTriangle.length est la taille de la deuxieme dimension
// Autres possibilites
int [][] tabTriangle2={new int[1],new int [2], new int [3]};
int [][] tabTriangle3={{1},{2,2}, {3,3,3}};
```

QZ 8.5 On considère la classe `Bouteille` vue dans l'exercice 15 page 24. Créer un tableau de 2 bouteilles, la première bouteille aura un volume de 3 litres et la deuxième de 1,5 litre.

```
Bouteille [] tabB=new Bouteille[2];
tabB[0]=new Bouteille(3);
tabB[1]=new Bouteille();
```

Ou bien directement :

```
Bouteille [] tabB2={new Bouteille(3),
                    new Bouteille()};
```

Quizz 9 – Tableaux d'objets

Trop difficile à faire pour les étudiants en semaine 3, à faire après l'héritage semaine 6

Qu'affichent les instructions suivantes ?

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

`System.out.println(tabSimple[3]);` affiche 0.
`System.out.println(tabObject[3]);` affiche null, car le handle `tabObject[3]` ne fait référence à aucun objet.
Le 1er : `System.out.println(tabObject[3].toString());` provoque une erreur à l'exécution, car le handle `tabObject[3]` ne fait référence à aucun objet, il n'y a donc pas de méthode `toString()` (Exception `NullPointeurException`).
Le 2ieme : `System.out.println(tabObject[3].toString());` affiche le `toString()`

Quizz 10 – `final`

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

QZ 10.1 Pour chaque instruction ci-dessous, indiquez si l'instruction compile ou pas.

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

`final int a=25;` // OK, une variable locale peut être déclarée `final`, par exemple pour éviter qu'on modifie accidentellement sa valeur dans le code (évite bug de programmation)
`a=17;`// ne compile pas, car `a` est final, ne peut pas être modifiée après initialisation
`final int b;` // OK, on n'est pas obligé d'initialiser lors de la déclaration
`b=10;` // OK, car `b` n'a pas été initialisée lors de la déclaration
`b=20;` // ne compile pas, car `b` a déjà été initialisée

QZ 10.2 Soit la classe ci-dessous, indiquez les lignes qui ne compilent pas et expliquez.

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

`this.y=y;` ne compile pas, car `y` a déjà été initialisée lors de la déclaration
`this.z=z;` ne compile pas, même si la variable `z` n'a pas été initialisée avant, cette instruction ne compile pas, car un attribut `final` ne peut être initialisé que que lors de la déclaration (comme `y`) ou dans le constructeur (comme `x`)

5 Variables et méthodes de classes

Séance 5 :
Objectif : concept static, constantes
Exercices conseillés :
TD : 34 (rapidement à l'oral), 35, 36, 38, 40
TME : 37, 39, 41
Quizz 11

Exercice 34 – Membres d'instance ou de classe

Q 34.1 On rappelle qu'un membre d'une classe est soit une variable (V) soit une méthode (M). On considère les classes ci-dessous. Pour chacune des expressions sous la classe, dire si ce sont des membres d'instance (I) ou de classe (C) de cette classe :

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

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

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

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

Corriger la moitié de l'exercice et laisser le reste pour que les étudiants s'entraînent.

Classe `Chien`

`nom`	VI
`nbChiots`	VI
`nbChiens`	VC
`getNbChiens()`	MC (accesseur variable statique)
`siteWebDuChien`	VI
`siteWebSPA`	VC
`aboyer()`	MI
`chercherLivreSurChiens()`	MC
`regarderDVD()`	cela dépend si c'est le chien qui regarde (MI) ou qqn d'autres (MC)

Classe `Chenil`

`nbChiots`	VI
`nbChiens`	VI
`getNbChiens()`	MI

On peut écrire un début de classe `Chien` pour qu'ils voient le rapport entre l'exercice et la syntaxe java.

```
1 public class Chien {
2         private String nom;
3         private int nbChiots;
4         private static int nbChiens=0;
5         public static int getNbChiens() { return nbChiens; }
6         ...
7 }
```

Classe `Maison`

`nbPièces`	VI
`prix`	VI
`prixMoyenEnFrance`	VC
`listeClassesEnergetiques`	VC
`classeEnergetique`	VI
`cptVentesDeCetteMaison`	VI
`cptVentesEnFrance`	VC
`getCptVentesEnFrance()`	MC (accesseur variable statique)

Classe `Stylo`

`taille`	VI
`TAILLE_STANDARD`	VC (constante)
`cptStylosProduits`	VC

Exercice 35 – Variables d'instance et variables de classes

Soit la classe `Truc` suivante :

```
1 public class Truc {
2         private static int cpt=0;
3         private int num;
4
5         public Truc() {
6                 cpt++;
7                 num=cpt;
8         }
9         public Truc(int x) {
10                num=x;
11        }
12        public static int getCpt() { return cpt; }
13        public int getNum() { return num; }
14 }
```

Q 35.1 Quel est le nom de la variable de classe ? Comment la reconnaît-on ?

cpt. On la reconnaît car elle est précédée du mot clé `static`.

Q 35.2 Pourquoi la variable cpt a-t-elle été initialisée ?

La variable de classe `cpt` est un compteur qui sera incrémenté à chaque appel du constructeur sans paramètre, il faut donc lui donner une valeur initiale.
Cette variable compte le nombre d'objets `Truc` créés avec le constructeur sans paramètre. Comme au départ, il y a 0 objets créés, on initialise cette variable à 0.

Q 35.3 Quel est l'affichage obtenu par l'exécution des lignes 4, 6, 8 et 9 du programme suivant :

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

Ligne 4 : 1
Ligne 6 : 1 1
Ligne 8 : 1 25 2
Ligne 9 : 2 2 2
Ici, le but est de montrer que la valeur de cpt change,mais pas la valeur de la variable num de n1

Q 35.4 Peut-on ajouter une instruction à la fin du programme de la question précédente afin d'afficher la valeur de la variable `cpt` sans utiliser d'instance ? Même question pour la variable `num`.

Oui, car cpt est static et getCpt() est aussi static : System.out.println(Numero.getCpt()) ;
Non, pour num, car num est une variable d'instance, qui dépend donc de l'instance à laquelle elle appartient.

Exercice 36 – 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 36.1 A-t-on commis une *faute de conception* en déclarant plusieurs attributs comme public ? Justifier brièvement.

Tous les attributs public sont final : ils sont donc protégés (non modifiables depuis le client). Pas de problème.

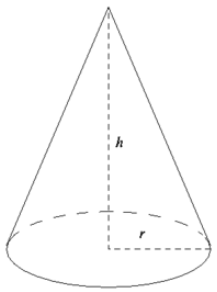
Q 36.2 Les propositions suivantes sont-elles correctes du point de vue syntaxique (compilation) ? Donner les affichages pour les lignes correctes.

```
12 // dans la classe Vecteur
13 public int getCpt2(){return cpt;}
14 public static int getId(){return id;}
15 public static String format(Vecteur v){
16         return String.format("[%5.2f, %5.2f]", v.x, v.y);
17 }
18
19 // dans le main
20 Vecteur v1 = new Vecteur(1, 2); v2 = new Vecteur(1, 2);
21 if(v1.x == v2.x && v1.y == v2.y) System.out.println("v1 egale v2");
22 if(v1.id == v2.id) System.out.println("les points ont le meme identifiant");
23 System.out.println("Compteur: "+v1.getCpt());
24 System.out.println("Compteur (2): "+Vecteur.getCpt());
25 System.out.println("Compteur (3): "+v1.cpt);
```

```
11 // dans la classe Vecteur
12 public int getCpt2(){return cpt;} // OK
13 public static int getId(){return id;} // NON: static => pas d'acces aux attributs
14 public static String format(Vecteur v){ // OK
15         return String.format("[%5.2f, %5.2f]", v.x, v.y);
16 }
17
18 // dans le main
19 if(v1.x == v2.x && v1.y == v2.y) System.out.println("v1 egale v2");    // OK: v1 egale
       v2
20 if(v1.id == v2.id) System.out.println("les points ont le meme identifiant"); // OK
21 System.out.println("Compteur: "+v1.getCpt()); // OK : Compteur: 3
22 System.out.println("Compteur (2): "+Vecteur.getCpt()); // Compteur (2): 3
23 System.out.println("Compteur (3): "+v1.cpt);    // NON: champs prive
```

Exercice 37 – Cône de révolution

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



Q 37.1 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.

```
1 public class Cone {
2         private double r;
3         private double h;
4         public static final double PI=3.1415;    // NB: normalement, on utilise Math.PI
5         private static int nbCones=0;
```

Q 37.2 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)`,

```
1 public Cone(double r, double h) {
2         this.r=r;
3         this.h=h;
4         nbCones++;
5 }
```

- 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()`.

```
1 public Cone() {
2         this(Math.random()*10,Math.random()*10);
3 }
```

rappel : pas besoin de faire nbCones++, car le 1er constructeur le fait déjà
rappel : pas d'instruction avant this()
Attention : surtout ne pas écrire : this(r,h); ce r et ce h étant les variables d'instances qui ne sont pas encore initialisées

- la méthode `double getVolume()` qui retourne le volume $V = \frac{1}{3}\pi r^2 h$ du cône,

```
1 public double getVolume() {
2         return 1.0/3*PI*r*r*h;
3 }
```

// Attention : 1/3=0 car la division entre 2 nombres de types int est la division entière

- la méthode `String toString()` qui retourne une chaine de caractère qui, pour un cône de rayon 5.4 et de hauteur 7.2, a le format : `"Cone r=5.4 h=7.2 V=219.854736"`,

```
1 public String toString() {
2         return "Cone␣r="+r+"␣h="+h+"␣V="+getVolume();
```

```
3 }
```

— l'accesseur de la variable `nbCones` (aide : comment devrait être déclaré l'accesseur d'un attribut static ?)

L'accesseur d'un attribut static devrait être static, comme cela, on peut faire : `Cone.getNbCones()` sans utiliser d'instance.

```
public static int getNbCones() { return nbCones;  }
```

Q 37.3 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).

```
1 public static void main(String [] args) {
2         System.out.println("Nombre␣de␣cones␣crees␣:␣"+ Cone.getNbCones() );  // 0
3         Cone c1=new Cone(); System.out.println(c1);
4         Cone c2=new Cone(5.4,7.2);  System.out.println(c2);
5         System.out.println("Nombre␣de␣cones␣crees␣:␣"+ Cone.getNbCones() );  // 2
6 }
```

Exercice 38 – Méthodes de classe

Q 38.1 Écrire la classe `Alea` qui contient les deux méthodes de classe suivantes :

— la méthode de classe `lettre()` qui retourne un caractère choisi aléatoirement parmi les 26 lettres de l'alphabet (c'est-à-dire entre 'a' et 'z').
Aide : utiliser `Math.random()` qui retourne un double entre 0 et 1 (non compris).

— 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()`).

```
1 public class Alea {
2         public static char lettre() {
3                 return (char)(((char)(Math.random()*('z'-'a'+1)))+'a');
4         }
5         public static String chaine() {
6                 String s="";
7                 for(int i=0;i<10;i++) {
8                         s+=lettre();
9                 }
10                return s;
11        }
12 }
```

Q 38.2 Pour quelle raison les méthodes `lettre()` et `chaine()` sont-elles des méthodes de classes ?

Retourner aléatoirement un caractère ne dépend pas d'une instance de la classe.

Q 38.3 Dans la méthode `main()` de la classe `Alea`, afficher le résultat retourné par la méthode `chaine()`. Même question pour la méthode `main()` d'une classe `TestAlea`.

Dans la classe `Alea` : `System.out.println(chaine());`
Dans la classe `TestAlea` : `System.out.println(Alea.chaine());`

Q 38.4 La classe `Alea` est une boite à outils : il n'y a pas besoin de créer d'instance pour l'utiliser. Afin d'ôter toute ambiguïté, proposer une solution pour interdire la création d'instance de cette classe.

ajouter un constructeur privé

```
1 private Alea(){}
```

Exercice 39 – Génération d'adresses IP

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

Q 39.1 Ecrire la classe `GenerateurIP` qui contiendra les variables et méthodes suivantes :
- un constructeur `private`, car on ne veut pas créer d'instance de cette classe. NB : ce construceur ne fait rien
- `tab` : une variable de classe de type tableau de 4 entiers où chaque case correspond à une partie de l'adresse IP. Ce tableau est initialisé à l'adresse IP : 192.168.0.0
- une méthode de classe `String getAdresseIP()` qui retourne la prochaine adresse IP. Cette méthode incrémente d'abord le 4ième nombre de l'adresse IP. Si ce nombre est supérieur à 255 alors le 3ième nombre est incrémenté, et le 4ième est remis à 0. Remarque : cette méthode s'occupe seulement des 3ième et 4ième chiffres de l'adresse IP, elle ne s'occupe pas du cas où la prochaine IP est celle après 192.168.255.255.

```
1 public class GenerateurIP {
2         private static int [] tab={192,168,0,0};
3         private GenerateurIP() {}
4         public static String getAdresseIP() {
5                 tab[3]+=1;
6                 if (tab[3]>255) {
7                         tab[3]=0;
8                         tab[2]+=1;
9                 }
10                return tab[0]+"."+tab[1]+"."+tab[2]+"."+tab[3];
11        }
12 }
```

Q 39.2 Dans une méthode `main` d'une classe `TestGenerateurIP`, afficher 257 adresses IP pour vérifier que votre méthode fonctionne.

```
1 public class TestGenerateurIP {
2         public static void main(String [] args) {
3                 for(int i=0;i<257;i++) {
4                         System.out.println(GenerateurIP.getAdresseIP());
5                 }
```

```
6         }
7 }
```

Exercice 40 – Somme de 2 vecteurs

Remarque : les méthodes de classes sont utiles dans certains problèmes où on n'a pas besoin d'instance de la classe ou pour éviter un groupe d'instruction répétitive. Par exemple, `Math.random()`, cela évite de créer un objet `Math` ou `java.util.Random`.

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

Q 40.2 Ajouter à la classe `Vecteur` une méthode de classe `qui` fait la somme de deux vecteurs.

```
1 public Vecteur sommeVec (Vecteur v) {
2         return new Vecteur(this.x + v.x, this.y + v.y, this.z + v.z);
3 }
4 public static Vecteur sommeVec (Vecteur v1, Vecteur v2) {
5         return new Vecteur(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);
6 }
```

Q 40.3 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.

```
1 public class TestVecteur {
2         public static void  main (String[] args) {
3                 Vecteur v1 = new Vecteur();
4                 Vecteur v2 = new Vecteur();
5                 Vecteur v3= v1.sommeVec(v2);
6                 Vecteur v4= Vecteur.sommeVec(v1, v2);
7         }
8 }
```

Exercice 41 – Génération de noms (tableau de caractères, méthode de classe)

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

Q 41.1 Écrire une méthode de classe `rendAlea(int inf, int sup)` qui rend un entier naturel aléatoire entre inf et sup compris. Aide : lisez la documentation Java (voir site web de l'UE) de la classe `Random`.

Q 41.2 Écrire une méthode de classe `boolean estPair(int n)` qui vérifie que `n` est pair.

Q 41.3 Déclarer en variable `static` deux tableaux de `char` de noms voyelles et consonnes. Initialiser lors de la déclaration le premier avec les consonnes et le second avec les voyelles.

Q 41.4 Écrire les méthodes `rendVoyelle()` et `rendConsonne()` qui rendent respectivement une voyelle et une consonne de façon aléatoire.

Q 41.5 Écrire une méthode `genereNom()` qui rend un nom de longueur aléatoire comprise entre 3 et 6 caractères en générant alternativement une consonne et une voyelle.

Q 41.6 Écrire une classe `TestNom` dont la méthode `main` génère et affiche, dans une boucle, une dizaine de noms générés.

```
1 public class Nom {
2         private static char[] consonnes= {'B','C','D','F','G','H','J','K',
3                 'L','M','N','P','Q','R','S','T','V','W','X','Z'};
4         private static char[] voyelles={'A','E','I','O','U','Y'};
5
6         public static int rendAlea(int inf, int sup) {
7                 // rend un nat entre inf et sup, inf<=sup
8                 return (int)(inf + Math.random()*(sup-inf+1));
9                 // OU => Random r=new Random();return r.nextInt(sup-inf+1);
10        }
11        public static char rendVoyelle() {
12                return voyelles[rendAlea(0,voyelles.length-1)];
13        }
14        public static char rendConsonne() {
15                return consonnes[rendAlea(0,consonnes.length-1)];
16        }
17        public static boolean estPair(int n) {
18                return ((n\%2) == 0);
19        }
20        public static String genereNom() {
21                int lg;
22                String s="";
23                // tirage de la longueur entre 3 et 6 :
24                lg=rendAlea(3,6);
25                for (int i=0; i<lg; i++) {
26                        if (pair(i)) s+= rendConsonne();
27                        else s+=rendVoyelle();
28                }
29                return s;
30        }
31 }
32 //=============================================
33 public class TestNom {
34        public static void main(String [] args) {
35                int nbGeneres=10;
36                for (int i=0; i<nbGeneres; i++) {
37                        System.out.println("Nom␣gener␣:␣"+ Nom.generNom() + "\n");
```

```
38                      }
39              }
40 }
```

Quizz 11 – Variables et méthodes de classes

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

```
1 public class Cercle {
2     public static final double PI=3.14159;
3     private static int nbCercles=0;
4     public final int numero;
5     private int rayon;
6     public Cercle(int r) {
7         rayon=r;
8         nbCercles++;
9         numero=nbCercles;
10    }
11    public double surface() { return PI*rayon*rayon;  }
12    public static int getNbCercles() { return nbCercles; }
13 }
14 //==================
15 public class TestCercle {
16    public static void main(String [] args) {
17        Cercle c=new Cercle(3);
18        System.out.println(EXPRESSION);
19    }
20 }
```

QZ 11.1 Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

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

c.nbCercles est fausse, car nbCercles est privée
c.rayon est fausse, car rayon est privée
Remarque : c.numero (correcte car numero est public)

QZ 11.2 Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

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

Cercle.nbCercles est fausse, car nbCercles est privée
Cercle.numero est fausse, car numero n'est pas statique
Cercle.rayon est fausse, car rayon est privée et n'est pas statique
Cercle.surface(); est fausse, car surface() est une méthode d'instance (elle n'est pas statique) et doit donc être utilisée à partir d'une instance
Remarques :
Cercle.PI (correcte car cette variable est statique)
Cercle.getNbCercles(); (correcte car cette méthode est statique)

QZ 11.3 Soit la classe `Test2Cercle` suivante :

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

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

Pour les deux questions, cela affiche 3 car getNbCercles() est une méthode statique et il y a 3 objets Cercle créés (et 4 références (handles)).

6 Héritage et modélisation

Les semaines 6 à 9 sont consacrées à l'héritage.
Séance 6 :
Objectifs : base de l'héritage, `protected`, redéfinition de méthode dans les cas simples...
Exercices conseillés :
TD : 42 (modélisation) 43 (Etudiant/Salarie) 44 (botanique)
TME : 45 (Orchestre), 46 (Véhicules à moteur) ou exercices site web
Note : le cours sur les classes abstraites est normalement déjà fait. Ne pas hésiter à interpeller les étudiants sur le sujet.

Exercice 42 – Héritage et modélisation

Rappeler que plusieurs modélisations peuvent être possibles pour le même problème.

Dessiner le diagramme de classes correspondant aux problèmes suivants.

1. Une voiture est un véhicule qui contient 4 roues

   `Voiture` hérite de `Vehicule`, `Voiture` composée de `Roue`

2. Les vélos, voitures et camions sont des véhicules roulants, tandis qu'un char d'assaut est un véhicule à chenille

   Modélisation possible :
   `Velo`, `Voiture`, `Camion` héritent de `VehiculeRoulant`
   `VehiculeRoulant` composé de `Roue` (`VehiculeRoulant` contient un attribut tableau de `Roue`, le nombre de roues est un paramètre du constructeur)
   `Char` héritent de `VehiculeChenille`
   `VehiculeRoulant` et `VehiculeAChenille` héritent de `Vehicule`

3. Un cartable contient des fournitures (trousses, stylos....). Une trousse peut contenir des stylos.

`Cartable <>---- Fourniture <---- Trousse <>---- Stylo` (ajouter héritage de `Stylo` vers `Fourniture`)

4. Les animaux (renard, lièvre...) d'une forêt sont soit herbivores, soit carnivores.

```
Forêt <>---- Animal <---- Herbivore <---- Lièvre
                    <---- Carnivore <---- Renard
```

Exercice 43 – Personne (héritage)

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

```java
public class Personne {
  protected final String nom;
  protected String numTel;
  private int nbEnfants;
  public Personne(String nom, String numTel){
      this.nom=nom; this.numTel=numTel; nbEnfants=0;
  }
  public Personne(String nom){
      this(nom,null);
  }
  public String getNom() { return nom; }
  public String getNumTel() { return numTel; }
  protected int getNbEnfants() { return nbEnfants; }
  public void ajouterEnfant() { nbEnfants++; }
}


public class Etudiant extends Personne {
  private String cursus;
  public Etudiant(String n, String t, String c) {
      super(n,t);
      cursus=c;
  }
  public boolean estEnL2 () { return cursus.equals("L2"); }
}
```

Faire un diagramme des classes au tableau peut aider.

Q 43.1 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.

```java
public void afficherInfo() {
  System.out.println("Nom :"+nom);
  System.out.println("NumTel :"+numTel);
  System.out.println("NbEnfants :"+nbEnfants);
  System.out.println("Cursus :"+cursus);
}
```

```java
public void modifierInfo() {
  nom="toto";
  numTel="0102030405";
  nbEnfants=-1;
  cursus="L0";
}
```

```
1  public void afficherInfo() {
2    Sop("Nom␣:"+nom);
3    Sop("NumTel␣:"+numTel);
4    // Sop("NbEnfants :"+nbEnfants); private
5    Sop("Cursus␣:"+cursus);
6  }
```

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

Q 43.2 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.

```
1 public class Salarie extends Personne {
2     private double salaire;
3     public Salarie(String n, double s) {
4       super(n);
5       salaire=s;
6     }
7     public double getSalaire() { return salaire; }
8 }
```

Q 43.3 Ecrire 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 ?

La méthode se trouve dans `Salarié`, car seuls les salariés ont un salaire.
`nbEnfants` variable `private` avec accesseur `protected` et "setter" limité à faire +1. Conséquences :

- classe `Personne peut modifier`, car la variable n'est pas final (comparer avec l'attribut `nom`)
- classes filles (ou classes du même package) peuvent connaître la valeur, mais pas modifier
- autres classes ne peuvent ni modifier ni connaître la valeur
- toutes les classes peuvent augmenter le nombre d'enfants, mais seulement de 1 (évite de pouvoir entrer des valeurs aberrantes comme -1)

```
1     public double prime() {
2         return 5*salaire*getNbEnfants()/100;
3     }
```

Q 43.4 Ecrire une méthode `modifierNumTel(String numTel)` qui permet de modifier le numéro de téléphone de l'employé, et qui affiche, par exemple, pour l'employé `Albert` : `"Le salarié Albert a pour numéro 012345678"`.

Du fait des choix de modélisation qui ont été avec `final` et `protected` : `nom` est accessible, mais non modifiable ; `numTel` accessible et modifiable (uniquement par `super.numTel` car le paramètre de même nom cache la visibilité).

```
1     public void modifierNumTel(String numTel) {
2         super.numTel=numTel;
3         System.out.println("Le␣salarié␣"+nom+"␣a␣pour␣numéro␣"+super.numTel);
4     }
```

Q 43.5 Trouver et expliquer les erreurs dans la méthode `main` ci-dessous :

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

```
3       Personne p = new Personne("Albert");
4       p.ajouterEnfant();
5       p.prime();
6       p.estEnL2();
7
8       Etudiant e = new Etudiant("Ahmed",null,"L2");
9       e.ajouterEnfant();
10      e.prime();
11      e.estEnL2();
12
13      Salarie s1 = new Salarie("Amelle");
14      Salarie s2 = new Salarie("Pauline","0122334455");
15      Salarie s3 = new Salarie("Yves","0123401234",2000);
16    }
17 }
```

- `p.ajouterEnfant();` // OK
- `p.prime();` // Faux car prime() pas dans la classe `Personne`
- `p.estEnL2();` // Faux car estEnL2() pas dans la classe `Personne`
- `e.ajouterEnfant();` // OK par héritage
- `e.prime();` // Faux car prime() pas dans la classe `Etudiant`
- `e.estEnL2();` // OK
- `Salarie s1 = new Salarie("Amelle");` // Faux, car le constructeur de `Salarie` a deux paramètres (pas d'héritage des constructeurs de la classe `Personne`)
- `Salarie s2 = new Salarie("Pauline","0122334455");` // Faux, car le deuxième constructeur de `Salarie` devrait être un double, et non pas `String`
- `Salarie s3 = new Salarie("Yves","0123401234",2000);` // Faux, car pas de construceur avec trois paramètres dans `Salarie`

Remarque : si `TestPersonne` est dans le même package les instructions suivantes fonctionnent, sinon elles sont fausses :

- `System.out.println(p.nom);` // OK si même package ; faux sinon
- `p.getNbEnfants();` // OK si même package ; faux sinon

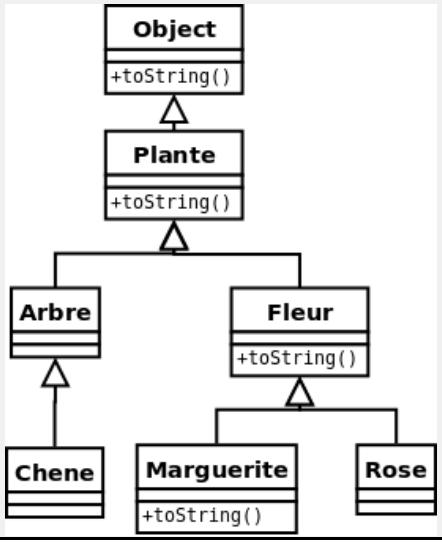
Exercice 44 – Botanique (héritage et redéfinition de méthodes)

Q 44.1 Dessiner l'arbre d'héritage et dire ce qu'affiche le programme suivant :

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

```
20        Rose r = new Rose();System.out.println(r);
21      }
22 }
```

p– >Je suis une Plante
a– >Je suis une Plante
f– >Je suis une Fleur
m– >Je suis une Marguerite
c– >Je suis une Plante
r– >Je suis une Fleur



Q 44.2 En tirer des conclusions sur l'héritage et la redéfinition de méthode.

Remarque : dans cet exemple, la classe de la référence des objets est toujours égale à leur classe réelle.
Par héritage ou par rédéfinition, toutes les classes ci-dessus possèdent une méthode toString().La recherche du corps de la méthode se fait en remontant la hiérarchie d'un cran tant que la méthode n'y figure pas. Arrêt lorsque la méthode est trouvée.

Q 44.3 Qu'affiche le programme suivant ? Rappel : le corps de méthode appelé est celui de l'objet, et non pas celui du type de la variable qui référence l'objet.

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

Faire un diagramme mémoire pour un des objets. Par exemple pour la marguerite.

```
p2->Plante
p3->Fleur
p4->Marguerite
p5->Fleur
p6->Plante
```

La méthode `toString()` appelée est celle de l'objet réelle, et non pas celle de Plante.

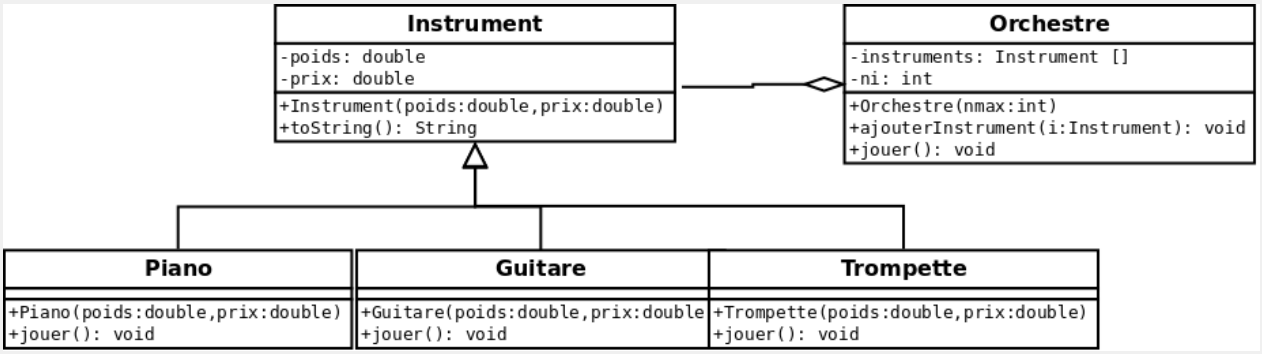
Exercice 45 – Orchestre

Exercice de base sur l'héritage pour introduire l'utilité des classes abstraites et de la redéfinition de méthodes.

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

Q 45.1 Dessiner le diagramme de classes.

Ne pas hésiter à donner la version UML light (boite avec le nom des classes + flèche d'héritage tête triangle vide.



Q 45.2 Écrire une classe `Instrument` contenant deux variables d'instance de type double pour stocker le poids et le prix de l'instrument, respectivement. Munir la classe d'un constructeur à deux paramètres pour initialiser les variables d'instance, ainsi que de la méthode toString(). Quelle est la particularité de la méthode toString() d'un point de vue de l'héritage ?

Premier exo pour montrer la redéfinition de toString() de Object. Parler du @Override pour éviter les typos dans les signatures.
@Override : Indicates that a method declaration is intended to override a method declaration in a superclass. If a method is annotated with this annotation type but does not override a superclass method, compilers are required to generate an error message.

```
1 public class Instrument {
2         private double poids;
3         private double prix;
4         public Instrument(double poids, double prix) {
5                 this.poids = poids;
6                 this.prix = prix;
7         }
8         @Override
9         public String toString() {
10                return "Instrument [poids=" + poids + ", prix=" + prix + "]";
11        }
12 }
```

Si le groupe est bon, on peut se permettre une digression sur le @Override qui permet de vérifier d'éventuelle faute de frappe (ie tostring ou toSring) à la compilation.

Q 45.3 Écrire les classes `Piano`, `Guitare`, `Trompette`. Ces classes comporteront une méthode `jouer()` qui affichera, par exemple pour `Guitare` : `"La guitare joue"`.

```java
public class Guitare extends Instrument {
        public Guitare(double poids, double prix) {
                super(poids, prix);
        }
        public void jouer(){
                System.out.println("La guitare joue");
        }
}

public class Piano extends Instrument {
        public Piano(double poids, double prix) {
                super(poids, prix);
        }
        public void jouer(){
                System.out.println("Le piano joue");
        }
}

public class Trompette extends Instrument {
        public Trompette(double poids, double prix) {
                super(poids, prix);
        }
        public void jouer(){
                System.out.println("La trompette joue");
        }
}
```

Q 45.4 Un orchestre sera composé d'un tableau d'instruments. Écrire la classe `Orchestre` correspondante, contenant une variable pour stocker le nombre d'instruments courant. Écrire une méthode `ajouterInstrument(Instrument i)` qui ajoute un instrument à l'orchestre lorque ceci est possible.

```java
public class Orchestre {
        private Instrument[] instruments;
        private int ni;
        public Orchestre(int nmax) {
                ni = 0;
                instruments = new Instrument[nmax];
        }
        public void ajouterInstrument(Instrument i){
                if(ni<instruments.length){
                        instruments[ni] = i;
                        ni++;
                } else {
                        System.err.println("Le tableau est plein!");
                }
        }
        public void jouer(){
                for(int i=0;i<ni;i++){
                        instruments[i].jouer();
                }
        }
}
```

Q 45.5 Ajouter à la classe `Orchestre` une méthode `jouer()` qui fait jouer l'ensemble des intruments le constituant. Quel est le problème dans le code actuel et comment remédier à ce problème ?

Pas de méthode jouer dans la classe mère instrument. Solution : on en ajoute une (qui ne fait rien). Le mieux serait de rendre la méthode `jouer()` (et la classe `Instrument`) abstraite puisqu'on ne peut pas spécifier le comportement de la méthode pour un instrument générique.

Q 45.6 Écrire une classe `TestOrchestre` avec la méthode `main()` qui créer un orchestre composé d'une guitare, d'un piano et d'une trompette, et fait jouer cet orchestre. Comment faire évoluer le code pour ajouter un nouvel instrument (*e.g.* batterie) ?

```
1 public class TestOrchestre {
2         public static void main(String[] args) {
3                 Orchestre e = new Orchestre(4);
4
5                 Piano p = new Piano(500,1000);
6                 Guitare g = new Guitare(10,200);
7                 Trompette t = new Trompette(2,500);
8
9                 e.ajouterInstrument(p);
10                e.ajouterInstrument(g);
11                e.ajouterInstrument(t);
12                e.jouer();
13        }
14 }
```

Pour ajouter un nouvel instrument il suffit de créer la classe correspondante (*e.g* Batterie) avec la redéfinition de jouer().

Exercice 46 – Véhicules à moteurs

On considère un parc de véhicules. Chacun a un numéro d'identification (attribué automatiquement à l'aide d'un compteur statique) et une distance parcourue (initialisée à 0). Parmi eux on distingue les véhicules à moteurs 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 46.1 : Construire le graphe hiérarchique des classes décrites ci-dessus.

```
Vehicule(id, distParcourue)
  |
  |_______AMoteur (capacitéReservoir, niveauEssence)
              |_____Voiture (nbPlaces)
              |_____Camion (Volume)
  |
  |______ SansMoteur ()
              |_____Vélo (nbVitesses)
```

Q 46.2 Ecrire le code java des classes `Vehicule`, `AMoteur`, et `SansMoteur` avec tous les constructeurs nécessaires et les méthodes `toString()`.

```
1 public class Vehicule { // la classe pourraît être asbtract
2   private static int nbVehicules=0;
3   private int id;
4   private double distParcourue=0;
5   public Vehicule() {
6     nbVehicules++;
7     id=nbVehicules;
8   }
9   public String toString() {
10    return "" + id +"␣";
11  }
12 }
13 public abstract class AMoteur extends Vehicule {
14  private double capaciteReservoir;
15  private double niveauEssence=0;
16  public AMoteur(double capa) {
17    super();
18    capaciteReservoir=capa;
19  }
20  public String toString() {
21    return super.toString() + "\n␣Vehicule␣a␣moteur,␣reservoir␣:␣" + capaciteReservoir
          +"␣litres.␣\n␣Niveau␣actuel␣a␣:␣" + niveauEssence + "␣litres";
22  }
23 }
24 public abstract class SansMoteur extends Vehicule {
25  public SansMoteur() {
26    super();
27  }
28  public String toString() {
29    return super.toString() + "␣\n␣Vehicule␣sans␣moteur";
30  }
31 }
```

Q 46.3 Ecrire une méthode `rouler(double distance)` qui fait avancer un véhicule. A quel niveau de la hiérarchie faut-il l'écrire ?

Il faut l'écrire dans la classe Vehicule.
Note : dans le `println()`, l'appel à `toString` est implicite.

```
1 public void rouler(double distance) {
2     distParcourue+=distance;
3     System.out.println("vehicule␣"+ this + "a␣fait"+ distance+ "km\n");
4 }
```

Q 46.4 Ecrire 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 ?

Il faut les écrire dans la classe `AMoteur` :

```
1   public void approvisionner(double nbLitres) {
2     if (niveauEssence+nbLitres > capaciteReservoir)
3        System.out.println("ca␣deborde..!");
4     else {
5        niveauEssence +=nbLitres;
```

```
6         System.out.println("ajoute␣"+ nbLitres + "␣litres␣dans␣" + this);
7       }
8     }
9     public boolean enPanne() {
10      if (niveauEssence==0) System.out.println("plus␣d'essence!\n");
11      return (niveauEssence==0);
12    }
```

Q 46.5 Ecrire la classe `Velo` avec consructeur et méthode `toString()` et une méthode `void transporter(String depart, String arrivee)` qui affiche par exemple `"le vélo n°2 a roulé de Dijon à Châlon"`.

```
1   public class Velo extends SansMoteur {
2     private int nbVitesses;
3     public Velo(int n) {
4       super();
5       nbVitesses=n;
6     }
7     public String toString() {
8       return "␣Velo␣"+super.toString();
9     }
10    public void transporter(String depart, String arrivee) {
11      System.out.println(this + "␣roule␣de␣"+ depart + "␣a␣"+ arrivee +"\n");
12    }
13  }
```

Q 46.6 Dans la méthode `main` d'une classe `TestVehicule`. Créez un vélo, et testez les méthodes de la classe `Vélo`.

```
1 public class TestVehicule {}
2   public static void main(String[] args) {
3     Velo v1=new Velo(17); // nb de vitesses
4     System.out.println(v1);
5     v1.rouler();
6     v1.transporter("Paris","Lyon");
7   }
8 }
```

Q 46.7 Ecrire la classe `Voiture` avec constructeur et méthode `toString()` et une méthode `void transporter(int n, int km)` qui affiche par exemple `"la voiture n°3 a transporté 5 personnes sur 200 km"` ou bien `"plus d'essence!"` suivant les cas.

```
1 public class Voiture extends AMoteur {
2     private int nbPlaces;
3     public Voiture(double capa,int n) {
4       super(capa);
5       nbPlaces=n;
6     }
7     public String toString() {
8       return "␣Voiture␣"+super.toString();
9     }
```

```
10    public void transporter(int n, int km) {
11      if (enPanne()) {
12        return;
13      }
14      System.out.println(this + "␣transporte␣"+ n +"␣personnes␣sur␣"+ km + "␣km");
15    }
16  }
```

Q 46.8 : Ecrire la classe `Camion` avec constructeur, la méthode `toString()` et une méthode `void transporter(String materiau, int km)` qui affiche par exemple "plus d'essence !" ou bien "le camion n°4 a transporté des tuiles sur 500 km".

```
1 public class Camion extends AMoteur {
2     private double volume;
3     public Camion(double capa,double vol) {
4       super(capa);
5       volume=vol;
6     }
7     public String toString() {
8       return "␣Camion␣"+super.toString();
9     }
10    public void transporter(String materiau, int km) {
11      if (enPanne()) {
12        return;
13      }
14      System.out.println(this + "transporte␣des" + materiau +"␣sur␣"+ km + "km");
15    }
16  }
```

Q 46.9 Peut-on factoriser la déclaration de la méthode transporter, et si oui, à quel niveau ?

Non, les signatures sont toutes différentes

SUPPLEMENT On considère le main ci-dessous. Ce programme est-il correct ? Le corriger si nécessaire. Qu'affiche-t-il ?

```
1 public static void main(String[] args) {
2   Vehicule v1=new Velo(17); // nb de vitesses
3   Vehicule v2=new Voiture(40.5,5); // capacite reservoir, nb de places
4   Vehicule v3=new Camion(100.0,100.0); // capacite reservoir, volume
5   System.out.println("Vehicules␣:␣"+v1+v2+v3);
6   System.out.println();
7   v2.approvisionner(35.0); // litres d'essence
8   v3.approvisionner(70.0);
9   System.out.println();
10  v1.transporter("Dijon","Valence");
11  v2.transporter(5,300);
12  v3.transporter("tuiles",1000);
13 }
```

Non, erreur de compilation. Pour pouvoir envoyer les messages approvisionner et transporter à un objet d'une sous-classe `de` Véhicule, il faut caster explicitement ce véhicule dans sa sous-classe réelle.
Cela donne, après correction :

```java
public class MainVehicule {
    public static void main(String[] args) {
      Vehicule v1=new Velo(17); // nb de vitesses
      Vehicule v2=new Voiture(40.5,5); // km,nb de Places
      Vehicule v3=new Camion(100.0,100.0); // km,volume
      System.out.println("Vehicules : "+v1+v2+v3);
      System.out.println();
      ((AMoteur)v2).approvisionner(35.0); // litres d'essence
      ((AMoteur)v3).approvisionner(70.0);
      System.out.println();
      ((Velo)v1).transporter("Dijon","Valence");
      ((Voiture)v2).transporter(5,300);
      ((Camion)v3).transporter("tuiles",1000);
    }
  }
```

L'exécution donne :

```
Vehicules :  Velo 1  Voiture 2  Camion 3
ajoute 35.0 dans  Voiture 2
ca deborde..!
ajoute 70.0 dans  Camion 3
Velo 1  roule de Dijon a Valence
Voiture 2  transporte 5 personnes sur 300 km
Camion 3 transporte tuiles sur 1000km
Press any key to continue...
```

7 Héritage et classe abstraite

Séance 7 :
Objectifs : cast, classe/méthode abstraite
Exercices conseillés :
TD : 47 (cast `Chien`) 48 (`Shape`)
TME : 49 (Ménagerie) 50 (Figure2D) ou exercices du site web
En fonction du temps restant :
- Quizz possibles : Quizz 12 (abstract) Quizz 13 (vocabulaire héritage)
- Autres exercices possibles : 51 (RetroEngineering) 52 (methodes et classes final)

Exercice 47 – 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.

Q 47.1 La méthode main suivante est-elle correcte ? Expliquez les erreurs.

```java
public class Mammifere { ... }
public class Chien extends Mammifere {
  public void aboyer() { System.out.println("Ouaff"); }
  public static void main(String[] args) {
    Chien c1 = new Chien();
    Mammifere m1 = c1;
```

```
7       c1 = (Chien) m1;
8       c1 = m1;
9       Mammifere m2 = new Mammifere();
10      Chien c2 = (Chien) m2;
11    }
12  }
```

```
Erreur de compilation pour la ligne : c1=m1;
Chien.java:7: incompatible types:
     found  : Mammifere
     required: Chien
c1=m1; => m1, variable de la classe Chien, ne peut référencer un objet de la
super-classe Mammifère. Il faut faire un cast explicite.

Pas d'erreur à la compilation pour la ligne : Chien c2=(Chien)m2;
Par contre,  erreur à l'exécution :
     Exception in thread "main" java.lang.ClassCastException: Mammifere
     cannot be cast to Chien at Chien.main(Chien.java:9)

Atttention : ce n'est pas parce que le cast passe à la compilation,
qu'il n'y a pas une erreur.
```

Exercice 48 – Figures (méthode et classe abstraite)

Soit le programme Java constitué des classes suivantes :

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

```
30        System.out.println(c1.toString() + "\n" + c2.toString());
31      }
32 }
```

Q 48.1 De quels membres (variables d'instance et méthodes) de `Shape` hérite la classe `Circle` ?

`Circle` hérite de x, y de la classe `Shape`,où il y a rédéfinition de la méthode `toString()`-héritée de la classe `Object`. La méthode `toString` est encore redéfinie dans la classe `Circle`. La méthode abstraite `surface()` de `Shape` devra être définie dans les sous-classes.

Q 48.2 La compilation de la classe `Circle` échoue, expliquer pourquoi.

La méthode abstraite `surface()` n'est pas redéfinie dans la sous-classe `Circle`, donc soit la classe `Circle` devrait être déclarée abstraite, soit la méthode `surface()` devrait être redéfinie.
Rappel : Toute classe ayant une méthode abstraite est forcément abstraite.

Q 48.3 Ajouter une méthode `surface()` à la classe `Circle` et modifier en conséquence la méthode `toString`.

```
1   public double surface() { return Math.PI*radius*radius; }
2   public String toString() {
3     return super.toString() + "␣Rayon␣:␣" + radius + "␣surface␣:␣" + surface();
4   }
```

Q 48.4 Créer une classe `Rectangle` qui hérite de `Shape`.

```
1 public class Rectangle extends Shape {
2   private double h,l ;
3   public Rectangle() {
4     super();
5     l=1;h=1;
6   }
7   public Rectangle(double x, double y, double l,double h) {
8     super(x,y) ;
9     this.h=h ;
10    this.l=l;
11  }
12  public String toString() {
13    return super.toString() + "␣Rectangle␣de␣longueur␣" + l+ ",␣de␣hauteur␣"+ h + "␣et␣
          de␣surface␣:␣" +surface();
14  }
15  public double surface() {
16    return l*h;
17  }
18 }
```

Q 48.5 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).

```
Shape[] tab = new Shape[3];
tab[0] = new Circle();
tab[1] = new Rectangle();
tab[2] = new Rectangle(1, 1, 3, 5);
double aire = 0;
for(Shape s:tab) aire += s.surface();
```

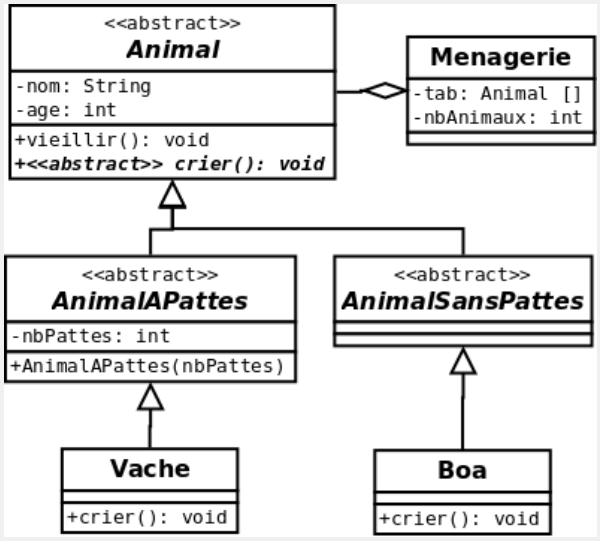
Exercice 49 – Ménagerie (tableaux, héritage, constructeur)

Exercice sur redéfinition de méthode et classe `abstraite`

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 Etablir graphiquement la hiérarchie des classes ci-dessus. Déterminer celles qui peuvent être déclarées abstraites ?

Abtraites :
Animal, Animal à pattes, Animal sans pattes
Concrètes :
Vache, Boa, Saumon, Canard, Mille-Pattes



Q 49.2 Ecrire 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 ?

```
// Cette classe est abstraite car elle contient une methode abstraite
public abstract class Animal {
  private String nom;
  private int age;
  public Animal(String nom, int age) {
    this.nom=nom;
    this.age=age;
  }
```

```
8     public Animal(String nom) {
9       this(nom,1);
10    }
11    public void vieillir() {
12      age++;
13    }
14    public abstract void crier(); /** Methode abstraite */
15    public String toString() {
16      return "Je␣m'appelle␣"+nom+",␣j'ai␣"+age +"ans";
17    }
18  }
```

Q 49.3 Ecrire toutes les sous-classes de la classe `Animal` en définissant les méthodes `toString()` et les méthodes `crier()` qui affichent le cri de l'animal.

```
1 // Cette classe est abstraite, car elle herite de la methode abstraite crier()
2
3   public abstract class AnimalAPattes extends Animal {
4     private int nbPattes;
5     public AnimalAPattes(String nom, int n) {
6       super(nom);
7       nbPattes=n;
8     }
9     public String toString() {
10      return super.toString()+",␣j'ai␣"+nbPattes+"␣pattes";
11    }
12  }
13
14 // Cette classe est abstraite, car elle herite de la methode abstraite crier()
15  public abstract class AnimalSansPattes extends Animal{
16    public AnimalSansPattes(String nom){
17      super(nom);
18    }
19  }
20  public class Vache extends AnimalAPattes {
21    public Vache(String nom) { super(nom,4); }
22    /** On donne ici le corps de la methode crier()*/
23    public void crier() { System.out.println("␣Meuuuh␣!␣"); }
24    public String toString() {
25      return super.toString()+"␣et␣je␣suis␣une␣vache␣";
26    }
27  }
28  public class Boa extends AnimalSansPattes {
29    public Boa(String nom) {
30      super(nom);
31    }
32    public void crier(){
33      System.out.println("␣SSSSSSSS!␣");
34    }
35    public String toString() {
36      return super.toString()+"␣et␣je␣suis␣un␣boa";
37    }
38  }
```

De même, pour `Saumon` et `Canard`

Q 49.4 Ecrire 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.

```
1 public class Menagerie {
2     private Animal [] tab;
3     private int nbAnimaux=0;
4     public Menagerie(int taille) {
5         tab=new Animal[taille];
6     }
7     public void ajouter(Animal a) {
8         if (nbAnimaux==tab.length) {
9             System.out.println("La menagerie est pleine !\n");
10            return; // Fin de la methode
11        }
12        tab[nbAnimaux]=a;
13        nbAnimaux++;
14    }
15    public Animal enlever()} {
16        if (nbAnimaux==0) {
17            System.out.println("La menagerie est vide !\n");
18            return null; // Fin de la methode
19        }
20        nbAnimaux--;
21        Animal a=tab[nbAnimaux];
22        tab[nbAnimaux]=null;
23        return a;
24    }
25    public String toString() {
26        String s="";
27        for (int i=0; i<nbAnimaux; i++)
28            s += tab[i]+"\n";
29        return s;
30    }
31 }
```

Q 49.5 Ajouter une méthode `void midi()` qui fait crier tous les animaux de cette ménagerie.

Q 49.6 Ecrire la méthode `vieillirTous()` qui fait vieillir d'un an tous les animaux de cette ménagerie.

Comme tous les animaux contiennent la méthode `crier`, on peut parcourir le tableau sans faire attention au type de l'animal.

```
1  public void midi() {
2      for (int i=0; i<nbAnimaux; i++) {
3          tab[i].crier();
4      }
5  }
6  public void vieillirTous() {
7      for (int i=0; i<nbAnimaux; i++)
8          tab[i].vieillir();
9  }
```

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

```java
public class TestMenagerie {
  public static void main(String [] args) {
    Menagerie m=new Menagerie(20);
    Animal b1= new Boa("Beatrice");
    Animal b2= new Boa("Bernard");
    Animal v1= new Vache("Marguerite");
    Animal v2= new Vache("Blanchette");
    m.ajouter(b1);
    m.ajouter(b2);
    m.ajouter(v1);
    m.ajouter(v2);
    System.out.println("Il␣est␣midi␣:");
    m.midi();
    System.out.println(m);
    m.vieillirTous();
    System.out.println(m);
  }
}
```

```
---- EXECUTION ----
Il est midi :
SSSSSSSS!
SSSSSSSS!
Meuuuh !
Meuuuh !
Je m'appelle Beatrice, j'ai 3 ans et je suis un boa
Je m'appelle Bernard, j'ai 3 ans et je suis un boa
Je m'appelle Marguerite, j'ai 3 ans, j'ai 4 pattes et je suis une vache
Je m'appelle Blanchette, j'ai 3 ans, j'ai 4 pattes et je suis une vache
Je m'appelle Beatrice, j'ai 4 ans et je suis un boa
Je m'appelle Bernard, j'ai 4 ans et je suis un boa
Je m'appelle Marguerite, j'ai 4 ans, j'ai 4 pattes et je suis une vache
Je m'appelle Blanchette, j'ai 4 ans, j'ai 4 pattes et je suis une vache
```

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 d'objets de type 2D devront accepter la méthode `perimetre()`.

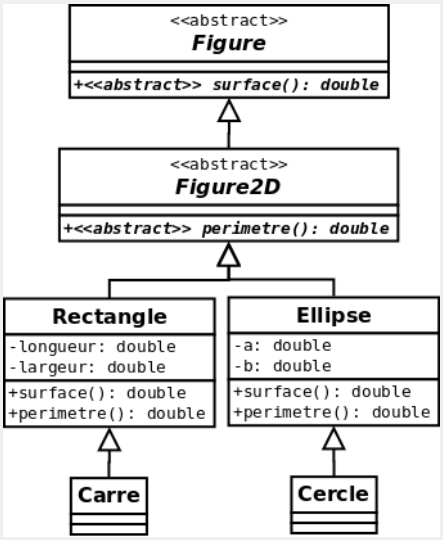
Rappel sur les ellipses : une ellipse est caractérisée par la longeur a du demi-grand axe et la longueur b du demi-petit axe. Sa surface est $\pi * a * b$ et son périmètre est $2\pi\sqrt{\frac{(a^2+b^2)}{2}}$.

Rappel : dans la classe `Math`, il existe la constante `Math.PI` et la méthode `Math.sqrt()` qui retourne la racine carrée

d'un nombre (voir annexe page 161).

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 ?

Une méthode abstraite est une méthode dont on ne précise que l'en-tête (signature) et qui sera implémenter dans les classes concrètes.
Toute classe `contenant` une méthode abstraite est forcément abstraite.
Une classe `concrète` se doit d'implémenter les méthodes abstraites de la super-classe.



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.

```
1 public abstract class Figure {
2     public abstract double surface();
3     public String toString() { return "c'est une figure";}
4 }
5 public  abstract class Figure2D extends Figure {
6     public abstract double perimetre();
7 }
8 public  class Rectangle extends Figure2D {
9     private double longueur, largeur;
10    public Rectangle (double l1, double l2) {
11      super();
12      longueur = l1; largeur = l2;
13    }
14    public String toString() { return "Rectangle "+longueur+"x"+largeur; }
15    public double surface() { return longueur*largeur; }
16    public double perimetre() { return 2*(longueur+largeur) ;  }
17 }
18 public class Carre extends Rectangle {
19    public Carre(double cote) {
20      super(cote, cote);
21    }
22 }
23 public class Ellipse extends Figure2D {
24    private double a, b;
25    public Ellipse (double a, double b) {
26      super();
```

```
27        this.a = a; this.b = b;
28      }
29      public String toString() { return "Ellipse "+a+"x"+b; }
30      public double surface() { return Math.PI*a*b; }
31      public double perimetre() { return 2*Math.PI*Math.sqrt(a*a+b*b)/2; }
32    }
33  public class Cercle extends Ellipse {
34      public Cercle(double rayon) {
35        super(rayon, rayon);
36      }
37    }
```

NB : l'héritage entre le cercle et l'ellipse constitue un débat sans fin... Il est intéressant d'expliquer les deux solutions et de justifier l'usage de cette correction : le cercle étend ellipse car un Cercle EST UNE Ellipse du point de vue géométrique (vérité terrain). Dans l'idéal, on choisit toujours de faire l'héritage qui correspond à la vérité terrain.

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 ?

Le tableau doit être de type `Figure2D` pour pouvoir afficher le périmètre.

```
1 public  class TestFigure {
2    public static void main (String [] args) {
3       Figure2D [] tabFig2D={new Rectangle( 10,4), new Carre(25), new Ellipse(24, 12),
            new Cercle(15)};
4       for(Figure2D fig : tabFig2D) {
5          System.out.println(fig.toString()+" s="+fig.surface()+" p="+fig.perimetre());
6       }
7    }
8 }
```

Exercice 51 – Retro engineering

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

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

Q 51.1 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)

```
1 Expression (ABS)
2  + ABS double getVal()
3         ->  Valeur
4             + double getVal()
5             +String toString()
6         -> Operation (ABS)
7                 -> Plus
8                   + Plus(Expression, Expression)
9                   + double getVal()
10                  +String toString()
11                -> Moins
12                  + Moins(Expression, Expression)
13                  + double getVal()
14                  +String toString()
15                -> Multiplie
16                  + Multiplie(Expression, Expression)
17                  + double getVal()
18                  +String toString()
19                -> Divise
20                  + Divise(Expression, Expression)
21                  + double getVal()
22                  +String toString()
```

Q 51.2 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.

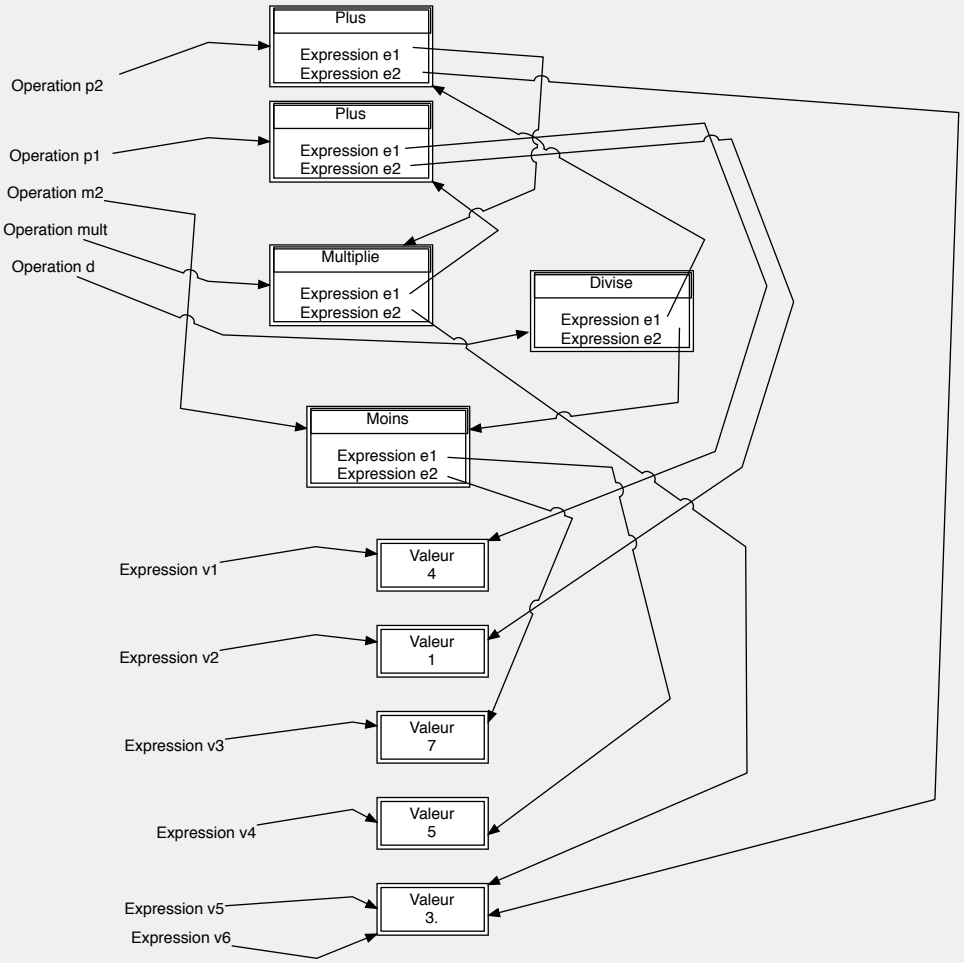
$$(((4+1)*3)+3)/(7-5)$$

Q 51.3 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).

```
1 public abstract class Expression {
2     public abstract double getVal();
3 }
4
5 public class Valeur extends Expression{
6     private double val;
7     public Valeur(double val) {
8         super(); //OPT
9         this.val = val;
10    }
11
12    public double getVal() {
13        return val;
14    }
15     public String toString(){
16        return ""+val; // 0.25 de penalite pour l'oubli de ""+
17    }
```

```
18 }
19
20 public abstract class Operation extends Expression{
21 }
22
23 public class Plus extends Operation{
24     private Expression e1, e2;
25
26     public Plus(Expression e1, Expression e2) {
27         super();
28         this.e1 = e1;
29         this.e2 = e2;
30     }
31
32     public double getVal() {
33             return e1.getVal()+e2.getVal();
34     }
35     public String toString(){
36         return "("+e1.toString()+"+"+e2.toString()+")";
37     }
38 }
```

Q 51.4 Donner le diagramme de l'état de la mémoire à la fin du programme (ligne 13).



Q 51.5 On souhaite maintenant pouvoir modifier l'attribut d'un objet `Valeur`. On ajoute alors la fonction `void setVal(double v)` à la classe `Valeur` qui fixe à `v` l'attribut de la classe. Soit la ligne de code suivante :

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

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

setValeur pas définie pour Expression
- ((Valeur)v6).setValeur(5)
- ou déclarer une fonction abstraite setValeur dans la classe Expression (mais çà implique de la définir partout en dessous donc pas terrible)

Q 51.6 Quel problème survient dans l'exécution du programme si l'on remplace la ligne 5 par :

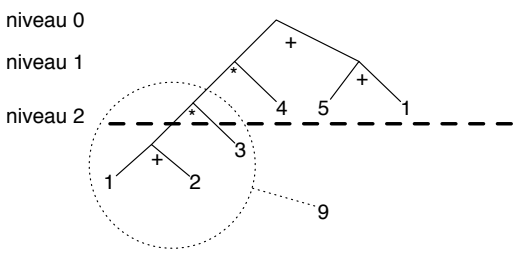
```
1 Expression v4=new Valeur(7);
```

Division par 0

Q 51.7 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.

ATTENTION : copier coller d'examen. Si vous n'en êtes pas là en TD, on saute
(1) Définir une classe Nan qui hérite de Expression et qui se propage dans toutes les opérations
(2) Ajouter un try/catch au niveau des getVal qui retourne Nan + un message en cas de problème

Q 51.8 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.




```java
// Expression
public abstract Expression simplifie(int level);

// Valeur
public Expression simplifie(int level){
  return this;
}

// Plus
public Expression simplifie(int level){
  if(level >0)
    return new Plus(e1.simplifie(level-1), e2.simplifie(level-1));
  return new Valeur(getVal());
}
```

Exercice 52 – `final` : les différentes utilisations

Q 52.1 Questions de cours

Q 52.1.1 A quoi sert un attribut `final` ? Où peut-il être initialisé ? Citer des cas d'utilisation.

Un attribut dont la valeur ne peut jamais être changée

```java
// Usage 1
public class Truc{
  private final int i = 1;
// Usage 2
public class Truc{
  private final int i;
  public Truc(){
    i = 1; // init possible dans le constructeur
  }
```

Pour les constantes (comme π...) : pour être sur que la valeur ne change pas
Pour les choses qui doivent être fixées une fois pour toutes (identifiants...)

Q 52.1.2 Dans quel cas déclarer une méthode comme `final` ?

si on ne veut pas qu'elle soit redéfinie dans une classe fille : on estime que le comportement ne peut pas changer dans le futur
cf exemple dans la question suivante
NB : apparemment d'après le web, aucun gain de performance à attendre...

Q 52.1.3 Dans quel cas déclarer une classe comme `final` ?

Si on ne veut pas qu'elle soit redéfinie : une classe pour une tache simple
Exemple : Integer, Double...

Q 52.1.4 Etant donné les usages répertoriés ci-dessus, à quoi sert le mot clé `final` en général ?

A améliorer la sécurité du programme en évitant des commandes "interdites"

Q 52.2 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 52.2.1 Au niveau des attributs, serait-il intéressant d'ajouter le modifier `final` sur certains champs ? Pourquoi ?

Q 52.2.2 A quelle condition pourrait-on mettre `x` et `y` en mode `final` ? Proposer une solution pour conserver les fonctionnalités de la classe.

— sur `id` : une fois l'identifier régler, il ne change plus... L'init est faite dans le constructeur, c'est OK.

```
1 private final int id;
```

— sur x,y : incompatible avec la méthode move, on ne peut pas passer en final.
S'il n'y avait pas la méthode `move`, on pourrait mettre `x,y` en final... On aurait alors un comportement à la String : pour modifier il faut créer une nouvelle instance. Il n'y a plus aucun problème de clonage ou de références... C'est très sécurisé

```
1 public class Point {
2     private final double x,y;
3
4     public Point createTranslatedPoint(double dx, double dy){
5       return new Point(x+dx, y+=dy);
6     }
7 }
```

Une telle modification est intéressante : si on considère l'addition de deux Point, il y a toujours une ambiguïté pour savoir si l'addition modifie le point courant ou génère une nouvelle instance... Avec final, plus de doute !

Q 52.2.3 Quelles fonctions pourraient être `final` ? Quel serait l'intérêt de la manipulation ?

Toutes les fonctions pourraient être final... sauf toString(). Les méthodes sont très simples, on garantit la fonctionnalité pour toutes les classes filles en déclarant les méthodes final : il est impossible qu'à un niveau donné le comportement de ces méthodes soit changé. Encore une fois : sécurisation

Q 52.2.4 Quel serait l'intérêt de déclarer la classe `final` ? Cela empêche-t-il tout enrichissement futur ?

Q 52.2.5 Proposer un code pour la classe `PointNomme` (point ayant un attribut `nom`) après avoir déclaré Point en `final`.

Classe final = impossible de créer une classe fille... Mais il est possible d'enrichir une classe par composition/délégation

```
1 public class PointNomme{
2   private Point p;
3   private String nom;
4   public PointNomme(double x, double y, String nom){
5     this.nom = nom; p = new Point(x,y);
6   }
7
8   public double getX(){return p.getX();} // delegation
```

Avec ce système, une chose fondamentale change : un PointNomme N'EST PAS un Point... Si une méthode est définie comme : `void maMethode(Point p)`, on ne peut pas lui donner un PointNomme...
Securisation (mais aussi limitation) de l'environnement de la classe déclarée final.

Quizz 12 – Classe et méthode abstraite

QZ 12.1 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 }
```

Rappel de cours : Si une classe est déclarée `abstraite` alors on ne peut pas créer d'objet de cette classe (pas de `new ClasseAbstraite()`).
Z est une classe abstraite. On ne peut pas créer d'instance de cette classe.
TestQuizzAbstract.java :4 : Z is abstract ; cannot be instantiated
Remarque : une classe abstraite ne possède pas forcemment une méthode abstraite.

QZ 12.2 Les instructions suivantes sont-elles correctes ? Expliquez chaque erreur.

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

Rappel de cours : une méthode abstraite est une méthode sans corps, elle n'a qu'une en-tête. Si une classe possède une méthode abstraite, cette classe doit être déclarée abstraite.
f() est une méthode abstraite, donc la classe Z doit être déclarée abstraite.
g() est une méthode abstraite, donc elle ne doit pas contenir de corps entre accolades ou bien elle ne doit pas contenir abstract, c'est ou l'un ou l'autre.
h() ne possède pas de corps (pas d'accolades), elle doit donc soit être déclarée abstraite, soit avoir des accolades.

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

QZ 12.3 Les instructions suivantes sont-elles correctes ? Expliquez et proposez deux solutions.

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

Rappel de cours : Si une classe hérite d'une méthode abstraite, elle doit soit être déclarée abstraite, soit définir le corps de la méthode abstraite.
B hérite d'une classe abstraite, il faut soit la déclarée abstraite, soit définir le corps de la méthode abstraite.
Solution 1 : on déclare B abstraite.

```
1
2 public abstract class B extends A {}
```

Solution 2 : on définit le corps de la méthode f()

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

Quizz 13 – Vocabulaire sur l'héritage

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

Quelques phrases possibles :
L'héritage permet de réemployer les champs et méthodes d'une classe ancêtre
L'héritage permet d'ajouter des éléments spécifiques, champs et/ou méthodes,
L'héritage permet de redéfinir le comportement de certaines méthodes.

8 Héritage et interface

Séance 8 :
Objectifs : `interface`, `equals` standard, `ArrayList`
Exercices conseillés :
TD : 53 (equals), 54 (interface Submarine), 55 (interface Motorise)
TME : 57 (interface Reversible), 60 (Chemin de fer (ArrayList, instanceof)) ou exercices site web
Autres exercices possibles :
— Exos sur interface : 56 (héritage d'interfaces), 58 (interface Comparable)
— Exos sur ArrayList : 59 (Attribut ArrayList)

Exercice 53 – Redéfinition de la méthode `equals`

Soit la classe `Point` ci-dessous :

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

Q 53.1 Qu'affiche l'exécution du main ?

```
p1=p2 : false (égalité référentielle)
p1=p3 : true
p1=p4 : false
```

Q 53.2 Rédéfinir la méthode `boolean equals(Object ob)` de la classe `Object` dans la classe `Point`, de façon qu'elle teste l'égalité des coordonnées et non des références. Les instructions de test sont fournies dans la méthode `main`.

Il faut bien penser à caster o en Point sinon o.x provoque une erreur à la compilation !

```
1 public boolean equals(Object o) {
2   Point p = (Point)o;
3   return (this.x == p.x) && (this.y == p.y);
4 }
```

Affiche alors :

```
p1=p2 : true
p1=p3 : true
p1=p4 : false
```

Q 53.3 Que se passe-t-il si dans la méthode `main`, on rajoute à la suite les instructions suivantes ? Comment résoudre le problème rencontré ?

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

Le programme affiche :

```
p1=p2 : true
p1=p3 : true
p1=p4 : false
Exception in thread "main" java.lang.ClassCastException: String cannot be cast to Point
        at Point.equals(TestMethodeEquals.java:8)
        at TestMethodeEquals.main(TestMethodeEquals.java:29)
```

Le problème est qu'on ne peut pas transtyper un String en un Point. Pour résoudre le problème, on doit d'abord vérifier que l'objet passé en paramètre est bien un objet de type Point, pour cela, on utilise l'opérateur `instanceof`.

```
1  public boolean equals(Object o) {
2     if (o instanceof Point) {
3        Point p = (Point)o;
4        return (this.x == p.x) && (this.y == p.y);
5     }
6     return false;
7  }
```

NB : la correction ci dessus est (un peu) fausse car si `o` est une instance d'une sous classe de Point, on peut répondre `true` alors que c'est faux... Bonne correction :

```
1  public boolean equals(Object o) {
2     if (o==null) return false;
3     if (getClass() == o.getClass()) {
4        Point p = (Point)o;
5        return (this.x == p.x) && (this.y == p.y);
6     }
7     return false;
8  }
```

Exercice 54 – Interface `Submarine`

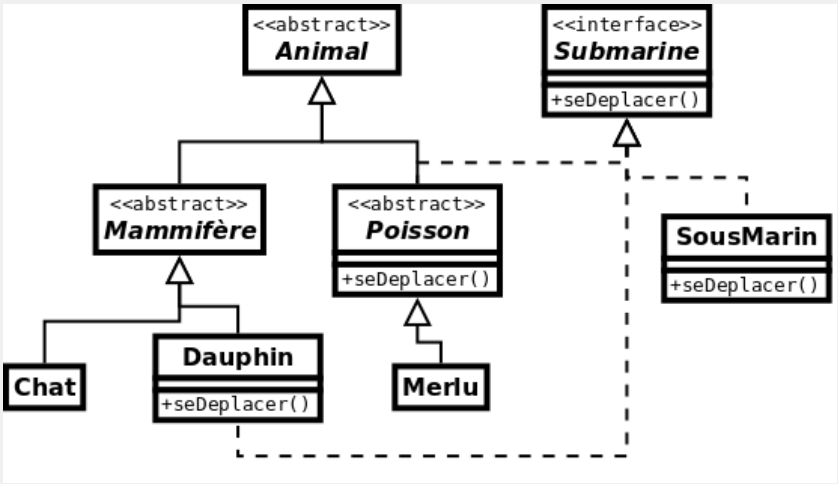
Soient les trois classes suivantes :

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

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

Q 54.1 Dessiner le diagramme de classe pour les classes `Animal`, `Chat`, `Dauphin`, `Mammifere`, `Merlu`, `Poisson`, `SousMarin` et l'interface `Submarine`.

Interface flèche en pointillé.



Q 54.2 Ecrire l'interface `Submarine` qui contient une méthode `seDeplacer`.

```
public interface Submarine {
  public void seDeplacer();
}
```

Q 54.3 Les poissons forment une famille d'animaux qui peuvent se déplacer sous l'eau. Ecrire la classe `Poisson`.

```
public abstract class Poisson extends Animal implements Submarine { // abstract
  public void seDeplacer() {System.out.println("Le poisson nage");}
}
```

Remarque : dans cet exercice, `Poisson` peut être abstract, car c'est un concept abstrait par rapport à `Merlu` qui est concret.

Q 54.4 Soit la classe suivante : `public class Merlu extends Poisson {}`. Un merlu a-t-il la propriété de nager sous l'eau ?

Oui par héritage de la classe `Poisson`.

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

Bien faire remarquer qu'une classe hérite directement au plus d'une classe, mais qu'une classe peut implémenter plusieurs interfaces.
Au tableau, inutile de tout écrire, juste donner la signature de la classe et le nom des méthodes présentes.

```
public class Dauphin extends Mammifere implements Submarine, Echolocation {
  public void seDeplacer() {System.out.println("Le dauphin nage");}
  public void envoyerSon() {System.out.println("Envoyer son dauphin");}
  public void ecouterSon() {System.out.println("Ecouter son dauphin");}
}
```

Q 54.6 Un sous-marin peut se déplacer sous l'eau et faire de l'écholocation. Ecrire la classe `SousMarin`.

Au tableau, inutile de tout écrire, juste donner la signature de la classe et le nom des méthodes présentes.

```
public class SousMarin implements Submarine, Echolocation {
  public void seDeplacer() { System.out.println("Le sous-marin se déplace");}
  public void envoyerSon() {System.out.println("Envoyer son sous-marin");}
  public void ecouterSon() {System.out.println("Ecouter son sous-marin");}
}
```

Q 54.7 Ecrire une classe `Mer` qui contient un attribut de type `ArrayList` de `Submarine`, qui contient une méthode pour ajouter un élément dans la liste et une autre méthode pour déplacer tous les éléments.

```
1 import java.util.ArrayList;
2 public class Mer {
3         private ArrayList<Submarine> als=new ArrayList<Submarine>();
4         public void ajouter(Submarine s) {
5                 als.add(s);
6         }
7         public void deplacer() {
8                 for(Submarine sub : als) {
9                         sub.seDeplacer();
10                }
11        }
12 }
```

Q 54.8 On suppose que l'on est dans une méthode `main` d'une classe `TestSubmarine`, créer une mer, y ajouter un merlu, un dauphin et un sous-marin, puis déplacer-les dans la mer.

```
1 public class TestSubmarine {
2         public static void main(String [] args) {
3                 Mer m=new Mer();
4                 m.ajouter(new Merlu());
5                 m.ajouter(new Dauphin());
6                 m.ajouter(new SousMarin());
7                 m.deplacer();
8         }
9 }
```

Q 54.9 Peut-on ajouter un chat dans la mer ?

Non, la classe `Chat` n'hérite pas de `Submarine`.
`m.ajouter(new Chat(); => error: incompatible types: Chat cannot be converted to Submarine`

Q 54.10 On suppose qu'une certaine espèce de chats très particuliers aime nager sous-l'eau dans la mer. Peut-on ajouter une classe `ChatSub` correspondante à cette espèce ? Faut-il modifier la classe `Mer` pour cela ?

Oui, on crée une classe `ChatSub` qui hérite de `Chat` et qui implémente `Submarine`, et il n'y aucune classe existante à modifier.

```
1 public class ChatSub extends Chat implements Submarine {
2   public void seDeplacer() {System.out.println("Le chat nage");}
3 }
```

`m.ajouter(new ChatSub(); => OK`

Exercice 55 – Interface `Motorise`

On suppose que tous les véhicules motorisés (voiture, moto...) doivent faire le plein d'essence 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 }
22 public class TestVehicules {
23    public static void main(String [] args) {
24       Vehicule [] tab={new Velo(), new Voiture(), new Moto()};
25       StationService station=new StationService();
26       for(int i=0;i<tab.length;i++) {
27          tab[i].rouler();
28          station.remplirReservoir(tab[i]);
29       }
30    } }
```

Q 55.1 Expliquez pourquoi la méthode `remplirReservoir(Vehicule v)` de la classe `StationService` 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.

— Cette méthode répète presque le même code (test if) pour `Voiture` et `Moto`. Si il y avait beaucoup de véhicules avec un réservoir, on ne va pas faire un `if` pour chaque cas.
— De plus, si on ajoute un camion, la méthode est fausse, car le plein du réservoir du camion ne sera pas fait. Plus généralement, si on ajoute des véhicules, on ne devrait pas avoir à modifier les autres classes.

Solution sans interface :
On ajoute une classe :

```
1 public abstract class VehiculeAMoteur extends Vehicule {
2    public abstract void faireLePlein();
3 }
```

et on fait hériter `Voiture`, `Moto` et `Camion` de cette classe : `extends VehiculeAMoteur`.
Ensuite, on réécrit la méthode `remplirReservoir(Vehicule v)` pour qu'elle marche pour tous les véhicules.

```
1 public class StationService {
2    public void remplirReservoir(Vehicule v) {
3       if ( v instanceof VehiculeAMoteur ) ((VehiculeAMoteur)v).faireLePlein();
4       else System.out.println("Inutile pas de réservoir");
5    }
6 }
```

Q 55.2 Créer un tableau avec un vélo, une voiture et une moto, et faire le plein de tous éléments du tableau à la station service.

```
1 public class TestVehiculeMotoriseV2 {
2    public static void main(String [] args) {
3       Vehicule [] tab={new Velo(), new Voiture(), new Moto()};
4       StationService station=new StationService();
5       for(int i=0;i<tab.length;i++) {
6          tab[i].rouler();
7          station.remplirReservoir(tab[i]);
8       }
9    } }
```

Q 55.3 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.

Non, la solution précédente ne fonctionne pas, car une tondeuse ne peut pas hériter de `Vehicule`.
Mettre une classe `Engin` en classe mère de `Vehicule` et `Tondeuse` ne fonctionne pas non-plus, car on ne veut pas faire le plein de certains véhicules (par exemple, vélo). C'est dans ce genre de cas, qu'utiliser une interface est vraiment très intéressant.
Une solution est d'ajouter une interface `Motorise` :

```
1 public interface Motorise {
2    public void faireLePlein() ;
3 }
4 public class Voiture/Camion/Moto extends Vehicule implements Motorise {
5    ...
6    public void faireLePlein() {System.out.println("Le plein est fait");};
7 }
8 public class Tondeuse implements Motorise {
9    public void faireLePlein() {System.out.println("Plein tondeuse fait");};
10 }
```

```
11 public class StationService {
12    public void remplirReservoir(Object o) {
13       if ( o instanceof Motorise ) ((Motorise)o).faireLePlein();
14       else System.out.println("Inutile pas de réservoir");
15    }
16 }
```

Une autre solution (Solution B) est de faire une méthode :

```
1    public void remplirReservoir(Motorise m) {
2       m.faireLePlein();
3    }
```

mais dans ce cas, on sera obligé de faire le `instanceof+cast` dans le main (voir question suivante).

Q 55.4 Créer un tableau avec un vélo, une voiture et une tondeuse et faire le plein de tous éléments du tableau à la station service.

Le type du tableau ne peut pas être :
- `Vehicule`, car une tondeuse n'est pas un véhicule
- `Motorise`, car un vélo n'hérite pas de l'interface `Motorise`

mais on peut utiliser le type `Object`.

```
1 public class TestVehiculeMotoriseV3 {
2    public static void main(String [] args) {
3       Object [] tab={new Velo(), new Voiture(), new Tondeuse()};
4       StationService station=new StationService();
5       for(Object o : tab) {
6          station.remplirReservoir(o);
7       }
8    }
9 }
```

Solution B : dans le for

```
1          if ( o instanceof Motorise) {
2             station.remplirReservoir(((Motorise)o));
3          }
```

Exercice 56 – Héritage d'interfaces pour les véhicules

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

Q 56.1 Donner la hiérarchie d'interface à créer.

```
1 Vehicule
2   + void demarrer()
3   + void arreter()
4
5 |__ Roulant extends Vehicule
6           + void rouler()
7 |__ Volant extends Vehicule
8           + void voler()
```

```
|__ Flottant extends Vehicule
        + void naviguer()
```

Q 56.2 Donner la signature de la classe `Voiture` et les méthodes à coder impérativement.

```
public class Voiture implements Roulant{
  public void demarrer(){..}
  public void arreter(){..}
  public void rouler(){..}
}
```

Q 56.3 Donner la signature de la classe `Hydravion` et les méthodes à coder impérativement.

```
public class Hydravion implements Volant,Roulant{
  public void demarrer(){..}
  public void arreter(){..}
  public void voler(){..}
  public void naviguer(){..}
}
```

Exercice 57 – Interface `Reversible`

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

Q 57.1 Donner le code de l'interface `Reversible`.

```
public interface Reversible {
   public void reverse();
}
```

Q 57.2 Donner le code de la classe `StringReversible`. Nous rappelons que la classe String est immutable et `final`. Vous aurez besoin des méthodes de `String` suivantes : `int length()` qui renvoie la longueur de la chaine et `char charAt(int i)` qui renvoie le caractère à la position `i`.

```
public class StringReversible implements Reversible{
    private String str;

    public StringReversible(String str) {
        this.str = str;
    }

    public void reverse() { // stocker la chaine renversee simplifie le toString
```

```
9           String str2 = "";
10          for(int i=str.length()-1; i>=0; i--)
11              str2 += str.charAt(i);
12          str = str2;
13      }
14
15      public String toString(){
16          return str;
17      }
18  }
```

Q 57.3 Donner deux exemples d'utilisation dans un `main`. Est-il possible de déclarer une variable de type `Reversible` ?

Oui, on peut déclarer une variable `Reversible`

```
1 StringReversible sr = new StringReversible("toto");
2 System.out.println(sr);
3 sr.reverse();
4 System.out.println(sr);
5 // second exemple
6 Reversible sr2 = new StringReversible("titi");
```

Q 57.4 Nous souhaitons maintenant créer une structure de type `ArrayList` réversible. Donner le code étendant l'`ArrayList<Object>`, ajoutant les méthodes nécessaire (dont `toString()` et surchargeant la méthode `get`.
NB : ajouter un attribut booléen indiquant si la structure est renversée ou pas.

```
1 import java.util.ArrayList;
2
3 public class ArrayRev extends ArrayList<Object> implements Reversible{
4     private boolean rev;
5
6     public ArrayRev() {
7         super();
8         rev = false;
9     }
10
11    public void reverse() {
12       rev = !rev;
13    }
14
15    public Object get(int i){
16        return rev?super.get(size()-1-i):super.get(i);
17    }
18
19    public String toString(){
20        String str="";
21        for(int i=0; i<size(); i++){
22            str += get(i).toString();
23        }
24        return str;
25    }
26 }
```

Q 57.5 Ajouter quelques lignes de code pour rendre la réversibilité récursive quand c'est possible dans la structure précédente. Par exemple, quand la liste contient des `StringReversible`, nous souhaitons renverser la liste ET renverser les éléments de la liste si c'est possible.

```
1     public void reverse() {
2         for(Object o: this)
3             if(o instanceof Reversible)
4                 ((Reversible) o).reverse();
5         rev = !rev;
6     }
```

Q 57.6 (Option) Proposer une seconde implémentation de la structure de données récursive basée sur la composition et non plus sur l'héritage (attribut `ArrayList` au lieu de `extends ArrayList`)

```
1 public class ArrayRev2 implements Reversible{
2     private ArrayList<Object> li;
3
4     public ArrayRev2() {
5         li = new ArrayList<Object>();
6     }
7
8     public void add(Object o){
9         li.add(o);
10    }
11
12    public void reverse() {// recursif
13        ArrayList<Object> tmp = new ArrayList<Object>();
14        for(int i=li.size()-1; i>=0; i--){
15            if(li.get(i) instanceof Reversible)
16                ((Reversible) li.get(i)).reverse();
17            tmp.add(li.get(i));
18        }
19        li = tmp;
20    }
21
22    public String toString(){
23        String str="";
24        for(int i=0; i<li.size(); i++){
25            str += li.get(i).toString();
26        }
27        return str;
28    }
29 }
```

Exercice 58 – Interface `Comparable`

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

```
1 public class Vecteur {
2     private double x,y;
3     public Vecteur(double x, double y){
4         this.x = x;
5         this.y = y;
```

```
6     }
7     public double norme(){return Math.sqrt(x*x+y*y);}
8 }
```

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

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

(2) Interface Comparable

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

(3) Interface Comparator

Q 58.1 Indiquer les modifications à effectuer dans la classe `Vecteur` pour utiliser `Comparable`

Q 58.2 Donner le code d'un main effectuant le tri d'une liste de `Vecteur` générée aléatoirement par rapport à leurs normes.

Le but n'est pas d'insister sur la syntaxe générique... Mais on est obligé de l'effleurer.

```
1 public class Vecteur implements Comparable<Vecteur> {
2 ...
3     public int compareTo(Vecteur o) {
4         double d1 = this.norme(); double d2 = o.norme();
5         if(d1 < d2) return -1;
6         if(d1 == d2) return 0;
7         return 1;
8     }
```

```
1 public class Test {
2     public static void main(String[] args) {
3         ArrayList<Vecteur> arr = new ArrayList<Vecteur>();
4         for(int i = 0; i<3; i++){
5             arr.add(new Vecteur(Math.random()*10, Math.random()*10));
6             System.out.println(arr.get(arr.size()-1));
7         }
8         Collections.sort(arr);
9         for(Vecteur v:arr){
10            System.out.println(v);
11        }
12    }
13 }
```

Q 58.3 Donner la procédure et le code pour utiliser un `Comparator`. Quel est l'avantage de cette approche ?

Avantage : pas besoin de modifier les classes existantes !

```
1 import java.util.Comparator;
2
3 public class VComp implements Comparator<Vecteur>{
4   // constructeur implicite
5
```

```
6     public int compare(Vecteur o1, Vecteur o2) {
7         double d1 = o1.norme(); double d2 = o2.norme();
8         if(d1 < d2) return -1;
9         if(d1 == d2) return 0;
10        return 1;
11    }
12 }
13 // Dans le main
14
15 public class Test {
16    public static void main(String[] args) {
17        ...
18        Collections.sort(arr, new VComp());
19        ...
20    }
21 }
```

Exercice 59 – Attribut de type `ArrayList`

Soient les classes suivantes :

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

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

Q 59.1 Expliquez la ligne 1.

> Java est fourni avec un ensemble de classes. Par exemple, les classes String, Math, System. Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles appelés packages. L'instruction `import` permet d'utiliser les classes d'un certain package. L'instruction `import java.util.ArrayList;` permet d'utiliser la classe `ArrayList` du package `java.util`.

Q 59.2 Ecrire la classe `ListeDeA` qui possède une seule variable d'instance appelée `liste` qui est de type `ArrayList` de `A` (voir la documentation de la classe `ArrayList` à la page 162). Ajoutez-y un constructeur qui prend en paramètre le nombre `n` d'objets à créer à l'initialisation de la liste. Ce constructeur crée aléatoirement 50% d'objets de type `B` et 50% d'objets de type `C` et les ajoute à la liste.

```
1 public class ListeDeA {
2         private ArrayList<A> liste;
3         public ListeDeA (int n) {
4                 liste=new ArrayList<A>();
5                 for(int i=0;i<n;i++) {
6                         if (Math.random()<0.5) {
7                                 liste.add(new B());
8                         } else {
9                                 liste.add(new C());
10                        }
11                }
12        }
13
14
15 public class TestArrayList {
16        public static void main(String [] args) {
17                ListeDeA l=new ListeDeA(8);
18                l.afficherListe();
19                l.afficherMethode();
20        }
21 }
```

Q 59.3 Ajoutez à la classe `ListeDeA` une méthode `afficherListe()` qui appelle la méthode `afficher()` de chacun des objets de la liste. Utilisez cette méthode dans une méthode `main`.

```
1         public void afficherListe() {
2                 for(A a: liste) {
3                         a.afficher();
4                 }
5         }
```

Q 59.4 Ajoutez à la classe `ListeDeA` une méthode `afficherMethode()` qui pour chaque objet de la liste appelle la méthode `methodeB()` si cet objet est un objet de type `B`, et appelle la méthode `methodeC()` si cet objet est un objet de type `C`. Utilisez cette méthode dans une méthode `main`.

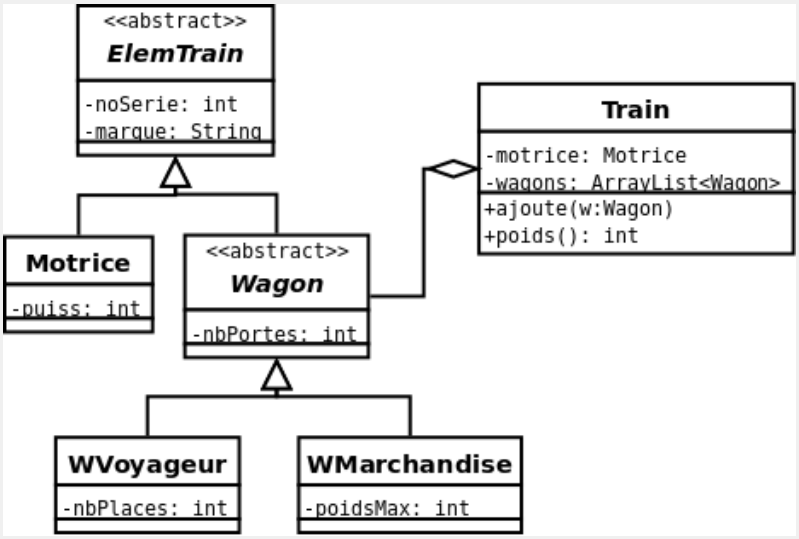
```
1         public void afficherMethode() {
2                 for(int i=0;i<liste.size();i++) {
3                         A a=liste.get(i);
4                         if (a instanceof B) {
5                                 B b=(B)a;
6                                 b.methodeDeB();
7                         } else if (a instanceof C) {
8                                 C c=(C)a;
9                                 c.methodeDeC();
10                        } else {
11                                System.out.println("erreur␣ni␣B␣n␣C");
12                        }
13                }
14        }
```

Exercice 60 – Compagnie de chemin de fer (ArrayList, instanceof)

Remarque : Exercice utilisant la classe **ArrayList, instanceof**

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

Q 60.1 Dessiner la hiérarchie des classes `Train`, `ElemTrain`, `Motrice`, `Wagon`, `WVoyageur` et `WMarchandise`.



Q 60.2 Ecrire les classes `ElemTrain` (abstraite), `Wagon` (abstraite), `Motrice`, `WVoyageur` et `WMarchandise` avec au moins un constructeur avec paramètres et une redéfinition de la méthode `public String toString()` qui retourne pour un élément son type et son numéro de série, par exemple : « Wagon Marchandise 10236 (Alstom)». Pour plus de propreté, vous utiliserez un compteur `static` pour générer les numéros de série commençant à 10000.

```
1 public abstract class ElemTrain {
2     private static int cpt=10000;
3     private int noSerie;
4     private String marque;
5     public ElemTrain(String m) {
6         noSerie=cpt++;
7         marque=m;
8     }
9     public String toString() { return noSerie+ "␣("+marque+")"; }
10 }
11 public class Motrice extends ElemTrain {
12     private int puiss;
13     public Motrice(int no, String m, int p) {
14         super(no, m);
15         puiss=p;
16     }
```

```
17      public String toString() {
18          return "Motrice␣"+super.toString();
19      }
20  }
21  public abstract class Wagon extends ElemTrain{
22      private int nbPortes;
23      public Wagon(int no, String m, int nbPo) {
24          super(no, m);
25          nbPortes=nbPo;
26      }
27  }
28
29  public class WVoyageur extends Wagon {
30    private int nbPlaces;
31    public WVoyageur(int no, String m, int nbPo, int nbPl) {
32        super(no,m,nbPo);
33        nbPlaces=nbPl;
34    }
35    public String toString() {
36        return "Wagon␣Voyageur␣"+super.toString();
37    }
38  }
39
40  class WMarchandise extends Wagon {
41    private int pdsMax;
42    public WMarchandise(int no, String m, int nbPo, int pds) {
43        super(no, m, nbPo);
44        pdsMax=pds;
45    }
46    public int getPdsMax() {
47        return pdsMax;
48    }
49  }
```

Q 60.3 Un Train possède une motrice et une une suite de wagons (on gèrera cette suite obligatoirement par la classe `ArrayList` (voir la documentation page 162). Ecrire la classe `Train` avec au minimum un constructeur a un paramètre de type `Motrice` qui construit un train réduit à cette motrice, et ayant donc un ensemble vide de wagons.

```
1 import java.util.ArrayList;
2 public class Train {
3   public Motrice motrice;
4   public ArrayList <Wagon> wagons;
5   public Train(Motrice m) {
6       motrice=m;
7       wagons=new ArrayList<Wagon>();
8   }
9   public void ajoute(Wagon w) {
10      wagons.add(w);
11  }
12
13  public String toString() {
14      return motrice+"␣"+wagons;
15  }
16  public int poids() {
17      Object o;
18      int p=0; // accumulateur
```

```
19        int c=wagons.size();
20        for (int i=0; i<c; i++) {
21            o=wagons.get(i);
22            if (o instanceof WMarchandise)
23                p+=((WMarchandise)o).getPdsMax();
24        }
25        return p;
26    }
27 }
```

Après avoir expliquer le fonctionnement de `instanceof`, il est utile d'insister sur la laideur du code produit :) En général, on préfère des fonctions abstraite de haut niveau pour gérer ce type de problème. Avec instanceof, le code n'est pas évolutif, si on ajoute un nouveau type de wagon, il faudra faire une modification ici aussi : c'est la caractéristique d'un *mauvais code*.

Q 60.4 Ajouter une méthode `void ajoute(Wagon w)` qui ajoute un wagon au vecteur de wagons du train.

Q 60.5 Redéfinir la méthode `public String toString()` qui retourne la composition de ce train.

Q 60.6 Ecrire une méthode `poids()` qui retourne le poids maximum de marchandise que peut transporter le train. *Indication* : On peut utiliser l'opérateur `instanceof` qui rend vrai si et seulement si un objet est instance d'une classe. Exemple d'utilisation : `if (a instanceof A)...`

Q 60.7 Ecrire la méthode principale `public static void main(String[] args)` dans une classe `MainTrain`. Cette méthode crée une motrice, des wagons de voyageur et des wagons de marchandise, crée un train formé de ces éléments, affiche la composition de ce train ainsi que le poids transporté.

```
1 class MainTrain {
2   public static void main(String[] args) {
3     Motrice m=new Motrice(5634,"MMM",2000); // no serie, marque, puiss
4     // no serie, marque, nbPortes, nb voyageurs :
5     WVoyageur wv=new WVoyageur(7845,"WWW",6,100);
6     // no serie, marque, nbPortes, poids max :
7     WMarchandise wm=new WMarchandise(9997, "WWW",2,1000);
8     WMarchandise wm2=new WMarchandise(3087, "WWW2",3,2000);
9     WMarchandise wm3=new WMarchandise(3114, "WWW3",3,1500);
10    Train t=new Train(m);
11    t.ajoute(wv);
12    t.ajoute(wm);
13    t.ajoute(wm2);
14    t.ajoute(wm3);
15    System.out.println("composition du train : ");
16    System.out.println(t.toString());
17    System.out.println("poids maximum charge par le train : "+t.poids());
18  }
19 }
20
21 /* EXECUTION :
22 composition du train :
23 Motrice 5634 [Wagon Voyageur 7845, Wagon Marchandise 9997,
24 Wagon Marchandise 3087, Wagon Marchandise 3114]
25 poids maximum charge par le train : 4500
26 Press any key to continue...
27 */
```

9 Héritage et liaison dynamique

Séance 9 :
Objectif : liaison dynamique, redéfinition complexe, package, éventuellement documentation
Exercices conseillés :
TD :
- faire un exercice parmi : 61 (Fourmi) 62 (sélection de méthodes) 63 (redéfinition piégeuse) Quizz 14 (liaison dynamique)
- possibilité de faire en fonction des besoins des exercices qu'on n'a pas eu le temps de faire avant. Par exemple : 52 (méthodes et classes final), 59 (attribut de type `ArrayList`)
- autre possibilité : prendre un peu d'avance sur les exceptions en faisant l'exercice 67 (try-catch)

TME :
normalement la semaine du TME SOLO, sinon prendre des exercices dans les semaines précédentes ou sur le site web
Exercices optionnels pour les groupes bien avancés : 64 (doc Java) 65 (package) 66 (visibilité et package)

Exercice 61 – Des fourmis à tous les étages

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

Q 61.1 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.

```
18 Fourmi f1    = new Fourmi("f1");
19 Fourmi f2    = new Ouvriere("ouv1");
20 Ouvriere f3 = new Ouvriere("ouv2");
21 Fourmi f4    = new Reine("majeste1");
22 Ouvriere f5 = new Reine("majeste2");
23 Reine f6     = new Reine("majeste3");
24 Fourmi[] fourmilliere = new Fourmi[100];
25 f2.manger(new Nourriture("sucre"));
26 fourmilliere[0]= f4.engendrer();
27 fourmilliere[1]= f5.engendrer();
28 fourmilliere[2]= f6.engendrer();
29 fourmilliere[3]= ((Reine) f2).engendrer();
30 fourmilliere[4]= ((Reine) f4).engendrer();
31 fourmilliere[5]= ((Reine) f6).engendrer();
```

-0.5 par faute (non détection d'err, ajout d'err, faute dans le nom de la fourmi)

```
33 Fourmi f1   = new Fourmi("f1");
34 Fourmi f2   = new Ouvriere("ouv1");
35 Ouvriere f3 = new Ouvriere("ouv2");
```

```
36 Fourmi f4 = new Reine("majeste1");
37 Ouvriere f5 = new Reine("majeste2"); // KO compil: incoherence dans la subsomption
38 Reine f6 = new Reine("majeste3");
39 Fourmi[] fourmilliere = new Fourmi[100];
40 f2.manger(new Nourriture("sucre"));
41 fourmilliere[0] = f4.engendrer(); // KO compil: methode non visible sur une variable
       Fourmi
42 fourmilliere[1] = f5.engendrer(); // KO : variable non creee // pas trop de
       penalisation: seulement si l'etudiant est manifestement incoherent par rapport a la
       declaration.
43 fourmilliere[2] = f6.engendrer(); // OK : majeste31
44 fourmilliere[3]=((Reine) f2).engendrer(); // KO execution : ClassCastException
45 fourmilliere[4]=((Reine) f4).engendrer(); // OK : majeste11
46 fourmilliere[5]=((Reine) f6).engendrer(); // OK : majeste32
```

Q 61.2 On souhaite maintenant nourrir nos Fourmis... En fonction de leur hiérarchie. Nous introduisons à cet effet les classes suivantes :

```
1 public class Nourriture{
2    private String description;
3
4    public Nourriture(String description){   this.description = description;  }
5    public String toString(){   return description;  }
6 }
7
8 public class GeleeRoyale extends Nourriture{
9    public GeleeRoyale(){  super("gelee pour la reine"); }
10 }
11
12 // et modification des classes existantes:
13 public class Fourmi{
14 ...
15    public void manger(Nourriture n){   System.out.println(nom+" mange "+n);  }
16 }
17 public class Reine extends Fourmi{
18 ...
19    public void manger(GeleeRoyale g){
20       System.out.println(nom+ " (Reine) mange de "+g);
21    }
22 }
```

Q 61.2.1 Est-il possible d'avoir une classe mère construite avec des arguments et une classe fille construite sans argument ?

Evidemment, l'exemple est donné dans l'énoncé (et en cours)

Q 61.2.2 Sélection de méthodes. Donner les affichages lors de l'exécution du code suivant :

```
25 Reine r1 = new Reine("majeste1");
26 Fourmi r2 = new Reine("majeste2");
27 r1.manger(new Nourriture("un peu de sucre"));
28 r1.manger(new GeleeRoyale());
29 r2.manger(new Nourriture("un peu de viande"));
30 r2.manger(new GeleeRoyale());
```

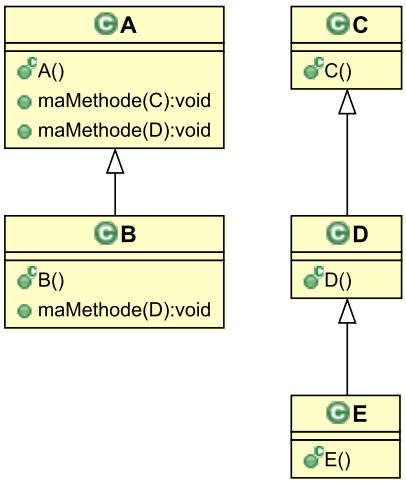
Piege sur r2 sur la gelée royale :

```
majeste1 mange un peu de sucre
majeste1 (Reine) mange de gelee royale pour la reine
majeste2 mange un peu de viande
majeste2 mange gelee pour la reine
```

Exercice 62 – Sélection de méthode

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

Q 62.1 Pour chaque ligne de code appelant `maMethode`, dire quelle méthode est effectivement appelée.




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

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

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

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

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

Je numérote les 3 solutions de haut en bas
sur l'instance `a` :

- 1 : pas de piège
- 2 : pas de piège
- 1 : PIEGE : on est dynamique sur les instances qui appellent les méthodes mais pas sur les arguments ! `cd` est de type C : sélection de la méthode correspondante
- 2 : pas de piège (mais gare à l'exécution si on fait mal le cast)
- 2 : pas de signature exacte correspondant à l'appel ⇒ recherche de la signature compatible la plus proche (E est un D), D est plus proche de E (par rapport à C→E)

```
je suis dans A (arg C)
je suis dans A (arg D)
je suis dans A (arg C) // PIEGE: on est dynamique sur les instances qui appellent les methodes mais pas sur les arguments ! cd est de type C: selection de la methode correspondante
je suis dans A (arg D)
je suis dans A (arg D) // pas de signature exacte correspondant a l'appel ⇒ recherche de la signature compatible la plus proche (E est un D), D est plus proche de E (par rapport a C→E)

je suis dans A (arg C)
je suis dans B (arg D)
je suis dans A (arg C)
je suis dans B (arg D)
je suis dans B (arg D)
```

```
12
13 je suis dans A (arg C)
14 je suis dans B (arg D)
15 je suis dans A (arg C)
16 je suis dans B (arg D)
17 je suis dans B (arg D)
```

Même comportements sur les instances `b` et `ab`
Remarque sur les priorités : entre une signature exacte dans la classe mère et une signature approchée dans la classe de l'instance, on prend la signature exacte (l9 et l15)

Q 62.2 Conversions implicites (ou pas)

On envisage 3 ajouts de méthodes :

Cas 1 :

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

Cas 2 :

```
1 // dans A
2 public void meth(int i)
3 // dans B
4 public void meth(double d)
```

Cas 3 :

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

Le code à exécuter est maintenant le suivant :

```
1        a.meth(2);
2        a.meth(2.);
3        b.meth(2);
4        b.meth(2.);
5        ab.meth(2);
6        ab.meth(2.);
```

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

Cas 1 :
Tout est OK à la compilation + exécution : conversion implicite int⇒double

Cas 2 :

```
1 a.meth(2);   // OK 1
2 a.meth(2.);   // KO compil
3 b.meth(2); // OK 1
4 b.meth(2.); // OK 2
5 ab.meth(2); // OK 1
6 ab.meth(2.); // KO compil
```

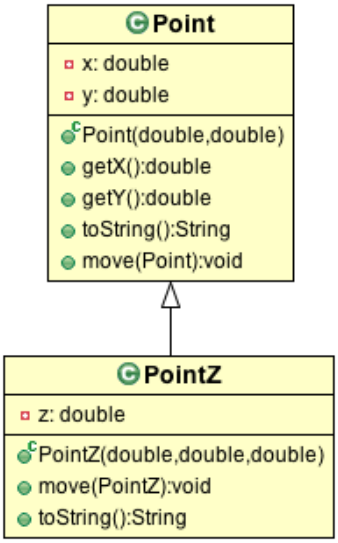
Pas de conversion double ⇒ int (trop dangereux)

Cas 3 :

```
1 a.meth(2);
2 a.meth(2.);   // KO compil
3 b.meth(2);
4 b.meth(2.); // KO compil
5 ab.meth(2);
6 ab.meth(2.); // KO compil
```

Exercice 63 – Redéfinition piégeuse

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




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

Q 63.1 Pourquoi cette hiérarchie de classe est-elle discutable ?

Il faut savoir si un PointZ est une spécialisation (cas particulier) de Point ou l'inverse.
- Si on considère qu'il s'agit d'un cas particulier (c'est un Point qui a pour particularité d'avoir une composante en Z) ⇒ OK
  un PointZ **EST UN** Point
- Si on considère que PointZ = point 3D... Il y a un soucis. En effet, du point de vue logique, un Point2D EST UN Point3D et pas l'inverse...

Q 63.2 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 ?

Le prog est correct.
Le super.get... en ligne 23 est évidemment optionnel.
On ne peut pas utiliser directement `x` et `y` car ce sont des attributs privés de la classe mère... Il faudrait qu'ils soient `protected`.

Q 63.3 Que pensez-vous du programme suivant ?

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

Q 63.3.1 Qu'est-ce qui s'affiche ?

Q 63.3.2 Est-ce que ça vous semble logique ?

Q 63.3.3 Expliquer en détail ce qui s'est passé au niveau de la compilation et de l'exécution.

Affichage :

```
1 [1.0, 2.0]
2 [1.0 2.0 3.0]
3 [2.0, 3.0]
4 [2.0 3.0 3.0] // composante en Z non modifiee!!
```

1. Compilation : présélection sur le type des variables

   ```
   1 p3d.move(depl); // -> move(Point p)
   ```

2. Execution :
   (a) Recherche de `move(Point p)`
   (b) La méthode existe dans la super-classe, elle est exécutée
   (c) PointZ peut toujours remplacer un Point, il rentre dans la méthode

Exercice 64 – Documentation Java

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

Q 64.1 Recherchez la classe `Random`. Combien a-t-elle de constructeurs ? Combien a-t-elle de méthodes ? A quel package appartient cette classe `Random` ? La classe `Math` appartient-elle au même package que la classe `Random` ? Aide : les packages sont écrits tout en minuscule.

2 constructeurs, 10 méthodes + 11 méthodes héritées de la classe Object
java.util
Non, la classe `Math` appartient au package java.lang.

Q 64.2 Recherchez la classe `ArrayList`. D'après la documentation, combien a-t-elle de champs ? Combien a-t-elle de constructeurs ? Combien environ a-t-elle de méthodes ? De quelles classes hérite-t-elle ? A quel package appartient cette classe `ArrayList` ?

1 champs hérité
3 constructeurs
20 méthodes+26 méthodes héritées
Voici l'arbre d'héritage :

```
java.lang.Object
    extended by java.util.AbstractCollection<E>
          extended by java.util.AbstractList<E>
              extended by java.util.ArrayList<E>java.lang.Object

->java.util.AbstractCollection<E>
```

```
->java.util.AbstractList<E>

->java.util.Vector<E>
```

Elle appartient au package : java.util

Q 64.3 Il est possible de créer une documentation pour les classes que vous créez. Pour cela, il faut utiliser la commande javadoc. Récupérez sur le site web de l'UE le fichier `Clavier.java`. Placez ce fichier dans un répertoire vide, puis tapez la commande : javadoc `Clavier.java`, puis : `firefox index.html` Comparez les commentaires du fichier `Clavier.java` et la page web affichée.

L'idée c'est qu'ils comprennent que les commentaires du fichier java sont ajoutés dans la doc HTML.

Exercice 65 – Package Java

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

Q 65.1 Créez 3 classes A, B et C chacune dans un fichier différent. Déclarez ces classes public. Mettez la classe `A` dans le package pack1 et les classes B et C dans le package pack2. Ajoutez rapidement une méthode avec des commentaires à chaque classe (pour cela, il faut mettre les commentaires entre `/** ... */` avant le nom de la méthode ou de la classe). Générez une (et une seule) documentation pour ces 3 classes.

Il faut mettre en haut de chaque fichier :

```
package pack1; // pour A
package pack2; // pour B et C
javadoc A.java B.java C.java
```

Attention : seules les classes déclarées public sont dans la doc HTML

Q 65.2 Créez un objet de la classe `A` dans la classe B. Compilez les fichiers. Quelle instruction faut-il ajouter ?

Il faut importer la classe `A` dans la classe B. Au début de la classe B, on ajoute l'instruction : `import pack1.A;`
Il faut compiler les deux fichiers en même temps : `javac A.java B.java`

Exercice 66 – Visibilité et package

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

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

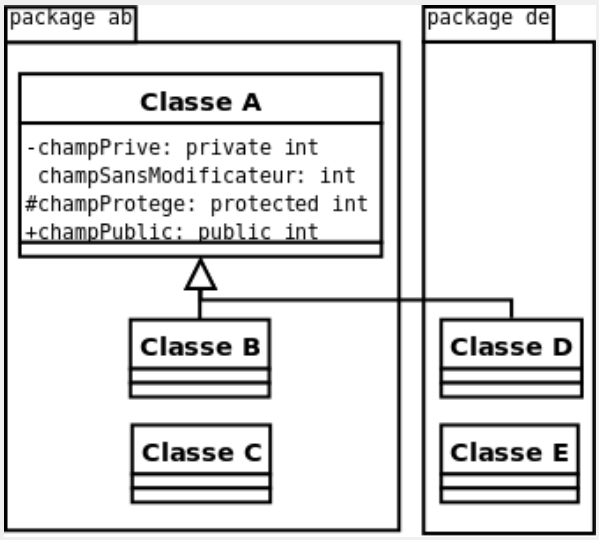
Si un champ d'une classe `A` :

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

— est `public`, il est accessible de partout dans le paquetage de `A` et, si `A` est publique, de partout ailleurs.
On considère les classes `A`, `B`, `C` qui sont dans le package `abc`, et les classes `D` et `E` qui sont dans le package `de`. Les classes `B` et `D` héritent de la classe `A`. On donne la classe `A` suivante :

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

Q 66.1 Donner la déclaration des classes `B`, `C`, `D` et `E`, et faire un schéma.



Obligatoirement chaque classe dans un fichier différent.

```
1 package abc;
2 public class B extends A { }
3
4 package abc;
5 public class C { }
6
7 package de;
8 public class D extends A { }
9
10 package de;
11 public class E { }
```

Q 66.2 Compléter le tableau ci-dessous en cochant les cases pour lesquelles les variables d'instance de la classe `A` sont visibles.

	Classe `A`	Classe `B`	Classe `C`	Classe `D`	Classe `E`
`champPrive`					
`champSansModifieur`					
`champProtege`					
`champPublic`					

	Classe A	Classe B	Classe C	Classe D	Classe E
champPrive	X				
champSansModifieur	X	X	X		
champProtege	X	X	X	X	
champPublic	X	X	X	X	X

Q 66.3 Si la classe A n'était pas déclarée public, est-ce que cela change la visibilité des variables ?

Oui, les variables ne sont visibles que dans le paquetage. C'est un cas que l'on rencontre rarement.

Quizz 14 – Héritage et liaison dynamique

Soient les 4 classes suivantes :

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

et les déclarations suivantes :

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

QZ 14.1 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.

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

a1.f() ;p1.f() ;p1.g() ; sont correctes.
a1.g() ; est incorrecte car la méthode g() n'existe pas dans la classe Animal.
Pour vous en convaincre, changez le nom de la méthode f() par vieillir() et le nom de la méthode g() par nager().
Les animaux et les poissons vieillissent, mais seuls les poissons nagent

QZ 14.2 Que retournent les instructions suivantes ?

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

- `a1.toString()` retourne "Animal"
- `p1.toString()` retourne "Poisson"
- `c1.toString()` retourne "Animal" (comme toString() n'est pas définie dans la classe Cheval, c'est la méthode toString() de Animal qui est utilisée)
- `z1.toString()` retourne Zoo@1bc4459 (comme toString() n'est pas définie dans la classe Zoo, c'est la méthode toString() de Object qui est utilisée)

QZ 14.3 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();`

- `Poisson p2=a1;` Erreur à la compilation, car un animal ne peut pas être converti automatiquement en poisson
- `Poisson p3=(Poisson)a1;` OK compilation, mais erreur à l'exécution car l'objet référencé par a1 est réellement un animal et non pas un poisson
- `Poisson p4=a2;` Erreur à la compilation, car un animal ne peut pas être converti automatiquement en poisson
- `a2.g()` Erreur à la compilation, car la méthode `g()` n'est pas visible dans `Animal`, même si l'objet référencé est à l'origine un poisson

10 Exceptions

Séance 10
Objectif : exceptions
Exercices conseillés :
TD : 67 (try-catch) 68 (création d'exception)
TME : 71 (MonTableau) 72 (Etudiant)
Autres exercices possibles : 69 (EntierBorne) 70 (Révision+`finally`)

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

— Pour capturer une exception :

```
try {
    instructions qui peuvent lever une exception
} catch (MonException me) {
    System.out.println(me.toString());
} catch (AutreException ae) {
    System.out.println(ae.getMessage());
} finally {
    instructions toujours executees
}
```

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

Exercice 67 – Capture dans le main d'une exception prédéfinie (`try catch`)

Q 67.1 Soit classe `TestAttrapePas0` ci-dessous. Que se passe-t-il lors de l'exécution ?

```
public class TestAttrapePas0{
    public static void main(String[] args){
        int [] tab= {1,2,3,4,5};
        for (int i=0; i<15; i++)
            System.out.print(tab[i] + "␣");
```

```
6           System.out.print("Fin");
7       }
8   }
```

Une exception de dépassement des bornes du tableau n'est pas attrapée par le programmeur et est donc levée directement par java après l'affichage des 5 valeurs du tableau.

```
// EXECUTION : java TestAttrapePas0
1 2 3 4 5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at TestAttrapePas0.main(TestAttrapePas0.java:8)
```

Q 67.2 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.

```
1 public class TestAttrapePas0 {
2     public static void main(String[] args){
3         int[] tab= {1,2,3,4,5};
4         try {
5             for (int i=0; i<15; i++)
6                 System.out.print(tab[i] + "␣");
7         } catch(ArrayIndexOutOfBoundsException e) {
8             System.out.println("Exception␣:␣depassement␣des␣bornes␣position␣"+e.
                  getMessage());
9         }
10        System.out.print("Fin");
11    }
12 }

// EXECUTION : java TestAttrapePas0
1 2 3 4 5
Exception : depassement des bornes position 5
Fin
```

Si le `try... catch` est de part et d'autre du `System.out`, on aura 10 fois le message

Exercice 68 – `Try, catch, throw, throws`, création d'une exception utilisateur

Q 68.1 Écrire une classe `TestAttrapePas1` dans laquelle on définira une méthode de classe `moyenne(String[] tab)` qui, étant donné un tableau de chaînes de caractères représentant des notes (entiers entre 0 et 20), rend la moyenne entière de ces notes. Testez cette méthode dans un main, en affichant la moyenne des notes passées en argument sur la ligne de commande, sans capturer l'exception éventuellement levée.
Indications :

- Utiliser la méthode `Integer.parseInt` qui transforrme une chaîne de caractères en entier et lève une exception `NumberFormatException` si la chaîne n'est pas un entier.
- Les arguments qui sont passés en ligne de commande sont récupérables par le tableau `String[] args` passé en paramètre de la méthode main.

```java
/** Levee des exceptions predefinies NumberFormatException ou  ArithmeticException,
    qui ne sont pas capturees */
public class TestAttrapePas1 {
  public static int moyenne(String [] tab) {
    int note,somme=0,n=0;
    for (int i=0;i<tab.length; i++) {
      note=Integer.parseInt(tab[i]);   //exception non capturee
      somme=somme+note;
      n=n+1;
    }
    return (somme/n);     // exception non capturee
  }
  public static void main(String[] args) {
    System.out.println("La moyenne est : "+moyenne1(args));
  }
}
```

Q 68.2 Que donnent les exécutions suivantes :

1. `javaTestAttrapePas1 10 12 16 18`
2. `javaTestAttrapePas1 12 1j 10 13 15`
3. `javaTestAttrapePas1`

```
1)  EXECUTION QUAND TOUT VA BIEN :
    java AttrapePas1 10 12 16 18
    La moyenne est : 14
2) EXECUTION QUAND TOUT VA MAL :
    java TestAttrapePas1 12 1j 10 13 15
    Exception in thread "main" java.lang.NumberFormatException:
     For input string: "1j"
       at java.lang.NumberFormatException.forInputString(Unknown Source)
         at java.lang.Integer.parseInt(Unknown Source)
           at java.lang.Integer.parseInt(Unknown Source)
            at TestAttrapePas1.moyenne1(TestAttrapePas1.java:10)
             at TestAttrapePas1.main(TestAttrapePas1.java:18)

3) DEUXIEME EXECUTION QUAND TOUT VA MAL, MAIS AUTREMENT :
ON A OUBLIE DE PASSER LA LISTE DES NOTES
   >java TestAttrapePas1
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at TestAttrapePas1.moyenne1(TestAttrapePas1.java:14)
         at TestAttrapePas1.main(TestAttrapePas1.java:18)
```

Q 68.3 Dans une classe `TestAttrape2`, reécrire une méthode `moyenne(String[] tab)`qui calcule la moyenne des notes de tab, mais capture cette fois l'exception levée si une note n'est pas entière et la traite en affichant le message « la note n'est pas entière ».

1. Où peut-on attraper l'exception `NumberFormatException` ?
2. Que se passe-t-il si aucune des notes n'est pas entière ou s'il n'y a aucune note ?

1. 1ere version en attrapant l'exception dans le main :

```
/**   TestAttrape2.java :  l'exception est levee dans la boucle,
* mais attrapee dans le main, ce qui interrompt
*  le programme des qu'une note n'est pas entiere  */
public class TestAttrape2 {
  // inchange par raport a la question 1
  public static int moyenne(String [] liste) {
    int note,somme=0,n=0;
    for (int i=0;i<liste.length; i++) {
      note=Integer.parseInt(liste[i]);  // levee de l'exception
      somme=somme+note;
      n=n+1;
    }
    return (somme/n);   // exception non capturee
  }
  public static void main(String [] args) {
    try {
      System.out.println("La moyenne est :"+moyenne(args));
    } catch(NumberFormatException e) {
      System.out.println("la note n'est pas entiere");
    }
  }
}
```

```
EXECUTION :
java TestAttrape2_0 10  12 lm 63 54 pp
la note n'est pas entiere
```

2. 2eme version en attrapant l'exception dans la boucle :

```
/* TestAttrape2.java : capture, A L'INTERIEUR de la BOUCLE,
* d'une exception predefinie. Le programme n'est ainsi
* donc pas interrompue.*/
public class TestAttrape2 {
  public static int moyenne(String [] liste) {
    int note,somme=0,n=0;
    for (int i=0;i<liste.length; i++) {
      try {
        note=Integer.parseInt(liste[i]);
        somme=somme+note;
        n=n+1;
      } catch(NumberFormatException e) {
        System.out.println("la "+(i+1)+" eme note n'est pas entiere\textbackslashn
            ");
      }
    }
    return (somme/n);  // exception non capturee
  }
  // inchange par rapport a la qestion1
  public static void main(String [] args) {
    System.out.println("La moyenne est :"+moyenne1(args));
  }
}
```

```
EXECUTION :
  java TestAttrape2 11 lm 16 1e pp 18
  la 2 eme note n'est pas entiere
  la 4 eme note n'est pas entiere
```

```
    la 5 eme note n'est pas entiere
    La moyenne est : 15
  Execution :
    java TestAttrape2 mm reg 6r c5 mm
    la 1 eme note n'est pas entiere
    la 2 eme note n'est pas entiere
    la 3 eme note n'est pas entiere
    la 4 eme note n'est pas entiere
    la 5 eme note n'est pas entiere
    Exception in thread "main" java.lang.ArithmeticException:  by zero
  Il y a levée de l'exception "ArithmeticException" car division par 0
```

Q 68.4 Écrire une classe `AucuneNoteEntiereException` dérivée de la classe `Exception`. Dans une classe `TestAttrape3` réécrire la méthode moyenne qui lancera une instance de la classe `AucuneNoteEntiereException` lorsque ce cas se présentera. Cette exception sera capturée dans le main.

```
1  public class AucuneNoteEntiereException extends Exception {
2    public AucuneNoteEntiereException (String s) {
3      super(s);
4    }
5    public String toString() {
6      return "aucune note n'est valide\n";
7    }
8  }

1 /* Lancement et capture d'une exception creee par l'utilisateur */
2    public class TestAttrape3 {
3      public static int moyenne(String [] liste)
4      throws AucuneNoteEntiereException {
5        int note,somme=0,n=0;
6        for (int i=0;i<liste.length; i++) {
7          try {
8            note=Integer.parseInt(liste[i]);
9            somme=somme+note;
10           n=n+1;
11         } catch(NumberFormatException e) {
12           System.out.println("la "+(i+1)+" eme note n'est pas entiere\n");
13         }
14       }
15       if (n==0) throw new AucuneNoteEntiereException("Lancement de
             AucuneNoteEntiereException:");
16       return (somme/n);
17     }
18     public static void main(String [] args) {
19       try {
20         System.out.println("La moyenne est : "+moyenne(args));
21       }   catch(AucuneNoteEntiereException e) {
22         System.out.println(""+ e.getMessage()+ e);
23       }
24     }
25   }
```

Q 68.5 Que donne l'exécution de la commande `javaTestAttrape3 mm reg 6r c5 mm` ?

```
java TestAttrape3 mm reg 6r c5 mm
  la 1 eme note n'est pas entiere
  la 2 eme note n'est pas entiere
  la 3 eme note n'est pas entiere
  la 4 eme note n'est pas entiere
  la 5 eme note n'est pas entiere
Lancement de AucuneNoteEntiereException : aucune note n'est valide
```

Q 68.6 Créer de même une classe `PasEntre0et20Exception` qui servira à traiter les cas où une note serait négative ou strictement supérieure à 20. Où faut-il capturer cette nouvelle exception ? Modifier le programme dans une classe `TestIntervalle` pour qu'il lève et capture aussi cette exception. Que donne l'exécution de la commande `java`TestIntervalle -10 -3 45 -78 -6 21 ?

```
1  /* On capture plusieurs type d'exceptions d'ou
2  * plusieurs blocs catch */
3  public class PasEntre0et20Exception extends Exception {
4    public PasEntre0et20Exception(String s){super(s);}
5  }
6  public class TestIntervalle {
7    public static int moyenne(String [] liste) throws AucuneNoteEntiereException  {
8      int note,somme=0,n=0;
9      for (int i=0;i<liste.length; i++) {
10       try {
11         note=Integer.parseInt(liste[i]);
12         if (note < 0)
13               throw new PasEntre0et20Exception("negative");
14         if (note > 20)
15               throw new PasEntre0et20Exception("superieure a 20");
16         somme=somme+note;
17         n=n+1;
18       } catch(PasEntre0et20Exception e) {
19         System.out.println("la " + (i+1)+" eme note est "+e.getMessage());
20       } catch(NumberFormatException e) {
21         System.out.println("la "+(i+1)+ "eme note n'est pas entiere\n");
22       }
23     }
24     if (n==0)
25       throw new AucuneNoteEntiereException("Lancement de AucuneNoteEntiereException :
           ");
26     return (somme/n);
27   }
28   public static void main(String [] args) {
29     try {
30       System.out.println("La moyenne est :"+moyenne(args));
31     } catch(AucuneNoteEntiereException e) {
32       System.out.println(""+ e.getMessage()+ e);
33     }
34   }
35 }
```

```
java TestIntervalle -10 -3 45 -78 -6 21
```

```
la 1 eme note est negative
la 2 eme note est negative
la 3 eme note est superieure a 20
la 4 eme note est negative
la 5 eme note est negative
la 6 eme note est superieure a 20
Lancement de AucuneNoteEntiereException : aucune note n'est valide
```

Exercice 69 – EntierBorne (throw,throws)

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

Q 69.1 Écrire dans une classe `TestEntierBorne` la méthode `main` qui saisit une valeur entière. On utilisera obligatoirement la méthode `saisirLigne` de la classe `Clavier` non standard qui affiche un message et lit un `String`, puis la méthode `parseInt` de la classe `Integer` (voir la documentation en ligne pour cette méthode) pour transformer la chaîne saisie en entier. Dans le cas où la saisie n'est pas un entier, cette méthode peut lever l'exception `NumberFormatException`.
Que se passe-t-il à l'exécution si la saisie n'est pas entière ? Expliquez.

Une erreur `NumberFormatException` est levée

Q 69.2 Traiter maintenant l'exception levée dans le main. Ajouter les instructions pour que le main s'endorme pendant n secondes en utilisant la méthode `sleep`de la classe `Thread` qui lève une exception de type `InterruptedException`.

```
1 try {
2   n=Integer.parseInt(Clavier.lireLigne("Entrez un nb secondes (entier) : "));
3   System.out.println("entier lu : "+n);
4   n1000=1000*n;
5   System.out.println("debut de l'arret de "+n+" secondes");
6   Thread.sleep(n1000);
7 } catch(NumberFormatException e) {
8   System.out.println("la saisie n'est pas un entier");
9 }catch(InterruptedException e) {
10  System.out.println("pb ds sleep(n)");
11 }
```

Q 69.3 Écrire la classe `EntierBorne` qui est une classe « enveloppe » du type simple int, i.e. qui "enveloppe" une variable d'instance de type int dans un objet de cette classe. Écrire le constructeur à un paramètre de type `int`qui peut lever l'exception `HorsBornesException` si la valeur qui est passée en paramètre est plus grande que 10000 ou plus petite que -10000, et la méthode `toString()`. On définira pour cela la classe `HorsBornesException`.

```
1     class HorsBornesException extends Exception {
2       public HorsBornesException(String s) {super(s);}
3     }
4     class EntierBorne { //les bornes pour les el de la classe :
5       static int maxEntier=10000,minEntier=-10000;
```

```
6       int valeur;// la valeur de l'Entier
7       public EntierBorne(int i) throws HorsBornesException {
8         if (i<minEntier)
9            throw new HorsBornesException("Constructeur:tu es trop petit");
10        if (i>maxEntier)
11           throw new HorsBornesException("Constructeur:tu es trop grand");
12        valeur=i;
13      }
14      public int getValeur() {return valeur;}
15      public String toString() { return ""+valeur+" "; }
16    }
```

Q 69.4 Définir la méthode `EntierBorne somme(EntierBorne i)` qui rend un objet `EntierBorne` dont la valeur est la somme de cet élément et du paramètre. Elle pourra lever sans la capturer l'exception `HorsBornesException` si la somme est trop grande.

```
1   public EntierBorne somme(EntierBorne i) throws HorsBornesException {
2     // rend un objet EntierBorne somme de cet el et de i
3     int val=this.valeur+i.valeur;
4     if (val<minEntier)
5        throw new HorsBornesException(" somme:je suis trop petit ");
6     if (val>maxEntier)
7        throw new HorsBornesException(" somme:je suis trop grand ");
8     return new EntierBorne(val);
9   }
```

Q 69.5 Définir la méthode `EntierBorne divPar(EntierBorne i)` qui rend un objet EntierBorne dont la valeur est la division entière de cet élément par le paramètre i. Elle pourra lever l'exception `HorsBornesException` ou l'exception `DivisionParZeroException`.

```
1   public EntierBorne divPar(EntierBorne i) throws ArithmeticException,
2   HorsBornesException {
3     // rend un objet EntierBorne division de cet el par i
4     if (i.valeur == 0) throw new ArithmeticException("division par zero");
5     int val=(int)(this.valeur/i.valeur);
6     return new EntierBorne(val);
7   }
```

Q 69.6 On définira ensuite la méthode `EntierBorne factorielle()` qui calcule la factorielle de cet élément. Elle pourra, en plus de l'exception `HorsBornesException`, lever l'exception `IllegalArgumentException` dans le cas où n serait négatif.

```
1   public EntierBorne factorielle() throws Exception {
2     if (this.getValeur() <0)
3        throw new IllegalArgumentException("x doit etre >= 0");
4     int fact=1;
5     for (int i=2; i<=getValeur(); i++)
6        fact *= i;
7     return new EntierBorne(fact);
8   }
```

Q 69.7 Créer un jeu de tests pour ce programme, en réfléchissant aux différents cas possibles et les tester dans le main.

```
1  public class TestEntierBorne {
2    public static void main(String [] args) {
3      int n=0; // nbbre saisi de secondes
4      int n1000=0; // n en milisecondes
5      int m=0; //
6      EntierBorne eb=null,eb1=null,eb2=null;
7      try {
8        n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier (nb de secondes d'
             arret): "));
9        System.out.println("entier lu : "+n);
10       n1000=1000*n;
11       System.out.println("debut de l'arret de "+n+" secondes");
12       Thread.sleep(n1000);
13       throw new Exception("Tout va bien");
14     } catch(NumberFormatException e) {
15       System.out.println("la saisie n'est pas un entier");
16     }catch(InterruptedException e) {
17       System.out.println("pb ds sleep(n)");
18     }
19     catch(Exception e) {
20       System.out.println("Fin de l'arret de "+n+" secondes");
21     }
22     System.out.println();
23     System.out.println("TEST SOMME : ");
24     try {
25       n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
26       System.out.println(" "+n);
27       eb=new EntierBorne(n); // peut lever exception HorsBornes
28       m=Integer.parseInt(Clavier.saisirLigne("Entrez un autre entier : "));
29       System.out.println(" "+m);
30       eb1=new EntierBorne(m); // peut lever exception HorsBornes
31       eb2=eb.somme(eb1); //leve exception HorsBornes si somme trop grande
32       throw new Exception("Tout va bien");
33     } catch(HorsBornesException e) {
34       System.out.println(e.getMessage());
35     } catch(NumberFormatException e) {
36       System.out.println("la saisie n'est pas entiere");
37     } catch(Exception e) {
38       System.out.println("EntierBornes : "+eb+" "+eb1);
39       System.out.println("somme des EntierBorne : "+eb2);
40     }
41     /*-------------------------------------------------*/
42     System.out.println();
43     System.out.println("TEST DIVISION : ");
44     try {
45       n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
46       System.out.println(" "+n);
47       eb=new EntierBorne(n);
48       m=Integer.parseInt(Clavier.saisirLigne("Entrez un autre entier : "));
49       System.out.println(" "+m);
50       eb1=new EntierBorne(m); // peut lever HorsBornesException
51       eb2=eb.divPar(eb1);
```

```
52          throw new Exception("Tout va bien");
53        } catch(NumberFormatException e) {
54          System.out.println("la saisie n'est pas entiere");
55        } catch(HorsBornesException e) {
56          System.out.println(e.getMessage());
57        } catch(ArithmeticException e){
58          System.out.println("Entiers Bornes : " + eb + eb1);
59          System.out.println(e.getMessage());
60        } catch(Exception e){
61          System.out.println("EntierBornes : " + eb + eb1);
62          System.out.println("division des EntierBorne : "+eb2);
63        }
64        //-------------------------------------------------
65        System.out.println();
66        System.out.println("TEST FACTORIELLE : ");
67        try {
68          n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
69          System.out.println(" "+n);
70          eb=new EntierBorne(n); // peut lever exception HorsBornes
71          //leve HorsBornes si factorielle trop grande
72          // ou IllegalArgument si negatif
73          eb2=eb.factorielle();
74          throw new Exception("Tout va bien");
75        } catch(HorsBornesException e) {
76          System.out.println(e.getMessage());
77        } catch(NumberFormatException e) {
78          System.out.println("la saisie n'est pas entiere");
79        } catch(Exception e) {
80          System.out.println("EntierBorne : "+eb);
81          System.out.println("factorielle de EntierBorne : "+eb2);
82        }
83      }
84    }
```

```
EXECUTION 1:
Entrez un entier(nb de secondes d'arret) :
3
3
debut de l'arret de 3 secondes
Fin de l'arret de 3 secondes
TEST SOMME :
Entrez un entier :
5
5
Entrez un autre entier :
10000
10000
somme:je suis trop grand
TEST DIVISION :
Entrez un entier :
5000
5000
Entrez un autre entier :
10
10
EntierBornes : 5000 10
division des EntierBorne : 500
TEST FACTORIELLE :
Entrez un entier :
9
9
Constructeur:tu es trop grand
```

```
EXECUTION 2 :
Entrez un entier(nb de secondes d'arret) :
2
2
debut de l'arret de 2 secondes
Fin de l'arret de 2 secondes
TEST SOMME :
Entrez un entier :
78
78
Entrez un autre entier :
-789
-789
EntierBornes : 78  -789
somme des EntierBorne : -711
TEST DIVISION :
Entrez un entier :
987
987
Entrez un autre entier :
0
0
Entiers Bornes : 987 0
division par zero
TEST FACTORIELLE :
Entrez un entier :
6
6
EntierBorne : 6
factorielle de EntierBorne : 720
```

Exercice 70 – throw, throws, finally

Q 70.1 Donnez l'affichage produit par le programme ci-après. Expliquez les résultats.

Cet exercice a pour but d'expliquer le fonctionnement de finally. Regarder l'affichage obtenue. Chaque méthode test() permet de comprendre un peu mieux le fonctionnement des exceptions et de finally.

```
1 public class MonException extends Exception {
2    public MonException(String s) {
3       super(s);
4       System.out.println("\n␣MonException␣:␣constructeur");
5    }
6 }
7 public class TestFinally {
8    /** Exception deleguee a la methode appelante (ici main).*/
9    public static void test1() throws MonException {
10      if (true) throw new MonException("lancee␣dans␣test1");
11      System.out.println("test1␣:␣fin␣de␣la␣methode");
12   }
13 /** Exception capturee (et pas deleguee) dans la methode test2 */
14 public static void test2() {
15   try {
16      if (true) throw new MonException("lancee␣dans␣test2");
```

```
17    } catch (MonException e) {
18       System.out.println("test2 : capture de l'exception : "+e);
19    }
20    System.out.println("test2 : fin de la methode");
21 }
22 /** Exception capturee (et pas deleguee) dans la methode test3 avec finally */
23 public static void test3() {
24    try {
25       if (true) throw new MonException("lancee dans test3");
26    } catch (MonException e) {
27       System.out.println("test3 : capture de l'exception : "+e);
28    } finally {
29       System.out.println("test3 : finally est effectue");
30    }
31    System.out.println("test3 : fin de la methode");
32 }
33
34 /** Exception deleguee a la methode appelante (ici main) avec finally */
35 public static void test4() throws MonException {
36    try {
37       if (true)
38          throw new MonException("lancee dans test4");
39    } finally {
40       System.out.println("test4 : finally est effectue");
41    }
42    System.out.println("test4 : fin de la methode");
43 }
44
45 /** Meme cas que le test4, mais ici l'exception n'est pas levee */
46 public static void test5() throws MonException {
47    try {
48       if (false) throw new MonException("lancee dans test5");
49    } finally {
50       System.out.println("test5 : finally est effectue");
51    }
52    System.out.println("test5 : fin de la methode");
53 }
54
55 public static void main(String [] args){
56    try {
57       test1();
58    } catch (MonException e) {
59       System.out.println("main : test1 : capture de l'exception "+e);
60    }
61    test2();
62    test3();
63    try {
64       test4();
65    } catch (MonException e) {
66       System.out.println("main : test4 : capture de l'exception "+e);
67    }
68    System.out.println();
69    try {
70       test5();
71    } catch (MonException e) {
72       System.out.println("main : test5 : capture de l'exception "+e);
73    }
74    System.out.println("Fin du programme");
75 }
76 }
```

```
Attention : finally est TOUJOURS EFFECTUE qu'il y ait une exception ou non (voir test5).

Attention : comme le montre le test4,  finally ne capture pas l'exception.

Mais si l'exception est capturée, alors le programme continue avec l'instruction
suivant le finally, sinon le bloc finally est exécutée, puis l'exception est
déléguée à la méthode appelante.


 VOICI L'AFFICHAGE OBTENUE
MonException : constructeur
main : test1 : capture de l'exception MonException: lancee dans test1
MonException : constructeur
test2 : capture de l'exception : MonException: lancee dans test2
test2 : fin de la methode
MonException : constructeur
test3 : capture de l'exception : MonException: lancee dans test3
test3 : finally est effectue
test3 : fin de la methode
MonException : constructeur
test4 : finally est effectue
main : test4 : capture de l'exception MonException: lancee dans test4
test5 : finally est effectue
test5 : fin de la methode
Fin du programme
```

Exercice 71 – MonTableau

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

Q 71.1 Définir une classe `MonTableau` qui possède les variables `tab` (tableau d'entiers) et `lgReelle` (entier) donnant le nombre de cases de `tab` réellement utilisées dans le tableau. Au départ, `lgReelle` vaut 0. Ecrire un constructeur prenant en paramètre la taille du tableau, et une méthode `ajouter(int n)` qui ajoute la valeur n à la suite du tableau sans vérifier s'il reste de la place.

```
1 public class MonTableau {
2     private int [] tab;
3     private int lgReelle;
4
5     public MonTableau(int max) {
6         tab=new int[max];
7         lgReelle=0;
8     }
9
10    public void ajouter(int i) {
11        tab[lgReelle]= i;
12        lgReelle++;
13    }
14 }
```

Q 71.2 Ecrire la méthode `main` qui crée un objet `MonTableau` de 3 cases et y ajoute 10 entiers. Exécutez le programme. Que se passe-t-il ?

```
1 public static void main(String[] args){
2     MonTableau t=new MonTableau(3);
3     for(int i=1;i<=10;i++) {
4         t.ajouter(i);
5     }
6 }
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at MonTableau.ajouter(MonTableau.java:11)
        at MonTableau.main(MonTableau.java:19)
```

Q 71.3 Capturer dans la méthode `main` l'exception précédemment levée, et afficher le texte `"Depassement des bornes a la position 3"` en utilisant la méthode `getMessage()` de l'exception levée.

```
1 public static void main(String[] args){
2     MonTableau t=new MonTableau(3);
3     try {
4         for(int i=1;i<=10;i++) {
5             t.ajouter(i);
6         }
7     } catch (ArrayIndexOutOfBoundsException e) {
8         System.out.println("Depassement␣des␣bornes␣position␣:␣"+e.getMessage());
9     }
10 }
```

Q 71.4 Définir un nouveau type d'exception appelée `TabPleinException`.

```
1 public class TabPleinException extends Exception {
2     public TabPleinException(String s) {
3         super(s);
4     }
5 }
```

Q 71.5 Modifier la méthode `ajouter` pour lever cette exception quand le tableau est plein. Capturer cette exception dans la méthode `main`. Que retourne les méthodes `getMessage()` et `toString()` de cette exception ?

```
1     public void ajouter(int i) throws TabPleinException {
2         if (lgReelle < tab.length) {
3             tab[lgReelle]= i;
4             lgReelle++;
5         } else {
6             throw new TabPleinException("Le␣tableau␣est␣plein");
7         }
8     }
```

```
9
10 public static void main(String[] args){
11     MonTableau t=new MonTableau(3);
12     try {
13         for(int i=1;i<=10;i++) {
14             t.ajouter2(i);
15         }
16     } catch (ArrayIndexOutOfBoundsException e) {
17         System.out.println("Depassement␣des␣bornes␣position␣:␣"+e.getMessage());
18     } catch (TabPleinException tpe) {
19         System.out.println("Depassement␣:␣"+tpe.getMessage());
20         System.out.println("Depassement␣:␣"+tpe.toString());
21     }
22 }
```

La méthode `getMessage()` retourne le texte passée dans le constructeur. La méthode `toString()` de `Exception` retourne NomDeLaClasse+le texte passée dans le constructeur.

Exercice 72 – Extrait de l'examen de 2007-2008 S1

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

Q 72.1 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.

```
1 public class Etudiant {
2     private String nom;
3     private int [] tabNotes;
4     private int nbNotes;
5     public Etudiant(String n) {
6         nom=n;
7         tabNotes=new int[5];
8         nbNotes=0;
9     }
10    public String toString() { // rend le nom et les notes
11        String s= "";
12        for (int i=0; i<nbNotes; i++) {
13            s += tabNotes[i] + "␣";
14        }
15        return nom + "␣" + s ;
16    }
17 }
```

Q 72.2 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`.

```
1 public class TabNotesPleinException extends Exception {
2     public TabNotesPleinException(String s) {
```

```
3         super(s);
4     }
5 }
```

```
1     public void entrerNote(int note) throws TabNotesPleinException {
2         if (nbNotes<tabNotes.length) {
3             tabNotes[nbNotes]=note;
4             nbNotes++;
5         } else {
6             throw new TabNotesPleinException("le tableau de notes de l'etudiant " +
                  this.nom+ " est plein");
7         }
8     }
```

Q 72.3 En supposant que la classe qui contient le `main` s'appelle `TestEtudiants`, on veut passer sur la ligne de commande une liste d'étudiants avec leurs notes, par exemple :

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

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

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

```
1 public class TestEtudiants {
2     public static void main(String [] args) {
3       int note;
4       // lecture des donnees sur ligne de commande :
5       for (int i=0;i<args.length; i++) {
6         try {
7           note=Integer.parseInt(args[i]);
8           System.out.print(args[i]+" c'est une note, ");
9         }
10        catch(NumberFormatException e) {
11          System.out.println("\n" + args[i]+" c'est un nom, ");
12        }
13      }
14    }
15 }
```

Q 72.4 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`.

La liste d'étudiants doit être statique.

```
class Etudiant {
    private static ArrayList<Etudiant> vEtu=new ArrayList<Etudiant>();
    public Etudiant(String n) {
        ...
        vEtu.add(this);
    }
    public static int getNbEtudiants() {
        return vEtu.size();
    }
    public static String listeEtudiants() {
        return vEtu.toString();
    }
}
```

Q 72.5 Enrichir/modifier le code précédent pour qu'il traite les données de la façon suivante :

- si c'est une chaîne de caractères, il crée une nouvelle instance d'étudiant portant ce nom.
- si c'est une note, il ajoute cette note à la liste des notes de l'étudiant créé précédemment, puis affiche la liste des étudiants. On pensera à traiter les différentes exceptions levées (on rappelle qu'un étudiant a au maximum 5 notes).

Voici une exécution possible :

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

```
public class TestEtudiants {
  public static void main(String [] args) {
    int note;
    Etudiant eCourant=null;
    // lecture et stockage des donnees de la ligne de commande :
    for (int i=0;i<args.length; i++) {
      try {
          note=Integer.parseInt(args[i]);
          eCourant.entrerNote(note);
      } catch(TabNotesPleinException e) {
          System.out.println(e.getMessage());
      } catch(NumberFormatException e) {
          eCourant = new Etudiant(args[i]);
      }
    }
    //affichage des etudiants :
    System.out.println("les "+ Etudiant.getNbEtudiants() + "etudiants :\n" +
        Etudiant.listeEtudiants());
```

```
18      }
19 }
```

11 Patterns, manipulation de flux entrée / sortie

Séance 11
Objectifs (secondaires) : patterns, flux
TD :
- Cela peut être utile de réviser les exceptions : par exemple 70 (Révision+`finally`)
- Un exercice sur les patterns : 73 (Singleton `Boule`)

TME : normalement semaine de soutenance des projets, sinon exercice 74 (Singleton `Note`) ou exercice sur le site web des annales
Exercices sur les flux : 75 (classe File) 76 (traitement de texte) 77 (copie fichiers binaires) 78 (tampon) 79 (compte-rendu TME) 80 (classe `Clavier`)
Quizz 15 (`String` immutable)

Exercice 73 (Examen 2016) – Gestion d'un système de tirage de boules de couleur

```
1 public class Boule {
2     private String couleur;
3
4     public Boule(String couleur) {
5         this.couleur = couleur;
6     }
7 }
```

```
1 public static void main(String[] args){
2     Boule b1 = new Boule("rouge");
3     Boule b2 = new Boule("jaune");
4     Boule b3 = new Boule("bleue");
5     Boule b4 = b1;
6 }
```

Q 73.1 A l'issue de l'exécution du code ci-dessus, combien y a-t-il de variables et d'instances de `Boule` ?

4 variables, 3 instances

Q 73.2 Donner la ligne de code pour créer un tableau de `Boule` (nommé `urne`) contenant les 4 références précédentes.

```
1 Boule[] urne = {b1, b2, b3, b4};
2 // ou
3 Boule[] urne = new Boule[4];
4 urne[0] = ...
```

Q 73.3 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 ?

rem : penser à utiliser urne.length
Il faut peut être rappeler d'utiliser `Math.random()` qui rend un réel entre 0 et 1 (exclu).

```
1 Boule choix = urne[(int)(Math.random()*urne.length)];
```

Probabilité de choisir une boule rouge : 1 chance sur 2.

Q 73.4 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.

```
1 public class BouleRandomFactory {
2     public final String[] couleurs = {"rouge","jaune","bleue","verte","orange","violette"};
3     public static Boule build(){
4         return new Boule(couleurs[(int) Math.random()*couleurs.length]);
5     }
6 }
```

- Attribut static
- parenthèses manquantes après le (int) pour encadrer toute l'expression mathématique
Version correcte :

```
1 public class BouleRandomFactory {
2     public final static String[] couleurs = {"rouge","jaune","bleue","verte","orange","
          violette"};
3
4     public static Boule build(){
5         return new Boule(couleurs[(int)(Math.random()*couleurs.length)]);
6     }
7 }
```

Q 73.5 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`).

```
1     ArrayList<Boule> gdeUrne = new ArrayList<Boule>();
2     for (int i=0; i<1000; i++)
3         gdeUrne.add(BouleFactory.build());
```

Q 73.6 Nous voulons faire des statistiques (comptages) sur les boules dans `gdeUrne`.

Q 73.6.1 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).

```
1     public boolean equals(Object obj) {
2         if (this == obj)      return true;
3         if (obj == null)      return false;
4         if (getClass() != obj.getClass())   return false;
5         Boule other = (Boule) obj;
6         if (! couleur.equals(other.couleur))
7             return false;
8         return true;
9     }
```

Q 73.6.2 Compter le nombre de boules de chaque couleur et afficher le résultat dans la console.
Algorithme proposé (mais pas imposé) :

1. Pour toutes les couleurs du tableau `couleurs` (de la classe BouleFactory)
   (a) Créer une boule de la couleur courante

(b) Initialiser un compteur à 0

(c) Pour toutes les boules de `gdeUrne`

i. Tester l'égalité avec la boule courante et incrémenter le compteur en cas de correspondance

(d) Afficher la couleur courante et le nombre de boules trouvées

```
1     for(String c:BouleFactory.couleurs){
2         Boule courante = new Boule(c);
3         int cpt = 0;
4         for(Boule b: gdeUrne)
5             if(b.equals(courante))
6                 cpt++;
7         System.out.println(c+" : "+cpt);
8     }
```

Q 73.7 Un second développeur propose une nouvelle architecture (correcte) pour remplacer la classe `Boule` et la factory associée (sans les aspects aléatoires) :

```
1 public class BouleV2 {
2     private String couleur;
3
4     public final static BouleV2 ROUGE = new BouleV2("rouge");
5     public final static BouleV2 JAUNE = new BouleV2("jaune");
6     public final static BouleV2 BLEUE = new BouleV2("bleue");
7     public final static BouleV2 VERTE = new BouleV2("verte");
8     public final static BouleV2 ORANGE = new BouleV2("orange");
9     public final static BouleV2 VIOLETTE = new BouleV2("violette");
10
11    private BouleV2(String couleur) {
12        this.couleur = couleur;
13    }
14 }
```

Q 73.7.1 Donner le code permettant de stocker 1000 `BouleV2` tirées aléatoirement dans une `ArrayList`.

```
1 public static void main(String[] args){
2     BouleV2[] tab = {BouleV2.ROUGE, BouleV2.JAUNE, BouleV2.BLEUE, BouleV2.VERTE,
          BouleV2.ORANGE, BouleV2.VIOLETTE};
3     ArrayList<BouleV2> gdeUrneV2 = new ArrayList<BouleV2>();
4     for (int i=0; i<1000; i++) gdeUrneV2.add(tab[(int)(Math.random() * tab.length)]);
5 }
```

Q 73.7.2 Le développeur prétend que son architecture est
— tout autant sécurisée que la précédente,
— plus rapide,
— ne nécessite pas d'ajouter un code `equals`
Que penser de ces 3 affirmations ?

Il s'agit d'une variante du Singleton (vu en Cours)
— le code est vraiment sécurisé : on ne peut pas créer d'instances, seulement utiliser les existantes ;
— il est (bcp) plus rapide car on n'a beaucoup moins d'instance à gérer (moins de mémoire à allouer), notamment quand il y a bcp de boules comme dans l'exemple ci-dessus ;

— le `==` marche très bien car il s'agit de tester des égalités référentielles.
Remarque : on ne peut pas créer d'autres boules que celles prévues au départ... Plus de sécurité, mais moins de possibilités. [non demandé aux étudiants]

Exercice 74 (Examen 2017) – **Quelques notes de musique**

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

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

Q 74.1 Parmi les opérations suivantes, toutes effectuées en dehors de la classe `Note`, lesquelles posent problème ? Expliquer très brièvement.

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

```
1 Note n1 = new Note(220); // impossible (constr private)
2 Note n2 = Note.do_; // OK
3 Note n3 = n2.re_; // OK
4 Note si_bemol = Note.si_.bemol; // NON: appel de méthode sans parenthèses
5 System.out.println(n2.frequence); // OK
6 Note.mi_.frequence = 12; // NON: attribut final
7 Note do_aigu = Note.do_.transpose(1); // OK
```

```
Note do_diese = Note.do_.diese(); // OK
Note do_diese2 = Note.do_ * Note.demiTon; // NON : attribut privé
Note do_aigu2 = Note.do_.transpose(2.5); // NON : int attendu, pas double
```

Q 74.2 Quels sont les points forts/faibles de l'architecture de cette classe ?

+ Bien sécurisée : pas de possibilité de créer des notes absurdes
+ accès direct à la frequence + sécurisation car cst
- pas vraiment de point faible... Obligation de créer une nouvelle instance pour une nouvelle note... Mais ce n'est pas vraiment un point faible.

Q 74.3 Pour ajouter une notion de rythme (noire, croche, blanche...), nous allons développer une nouvelle classe abstraite `Rythme` et des classes filles concrètes. Est-il possible de faire hériter `Rythme` de `Note` ?

Non (classe final) - Obligation de mentionner `final`

Q 74.4 Donner le code de la classe `Rythme`, qui gère une note et sa durée (`double`) et possède deux accesseurs vers la durée et la fréquence. Donner aussi le code de la classe `Noire` qui dure `1.0` temps.

```
public abstract class Rythme {
    private double duree;
    private Note n;

    public Rythme(double duree, Note n) {
        this.duree = duree;
        this.n = n;
    }
    public double getDuree(){
        return duree;
    }
    public double getFreq(){  (accès direct à la frequence obligatoire)
        return n.frequence;
    }
}
public class Noire extends Rythme {
    public Noire( Note n) { // 1 constructeur
        super(1., n);
    }
}
```

Q 74.5 Donner le code de la classe `Partition` qui étend la classe `ArrayList<Rythme>` et qui gère un attribut `double tempo` (pour indiquer à quelle vitesse jouer cette partition). Cette classe doit (évidemment) gérer l'ajout et l'accès à la i^e note du tableau. Elle possède aussi un accesseur pour le `tempo`.

```
public class Partition extends ArrayList<Rythme>{
    private double tempo;

    public Partition(double tempo) {
```

```
5           super(); // facultatif
6           this.tempo = tempo;
7       }
8
9       public double getTempo() {
10          return tempo;
11      }
12  }
```

Q 74.6 En cherchant sur internet, nous avons trouvé une classe permettant de générer des sons : `Player`... Cette classe n'est pas compatible avec notre architecture : elle attend pour sa construction un argument de type `Iterator<double[]>`. Un itérateur est une interface qui impose l'implémentation des méthodes suivantes :

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

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

```
1 public class Traducteur implements Iterator<double[]>{
2     private Partition p;
3     private int index;
4     private double t;
5     public Traducteur(Partition p) {
6         this.p = p;
7         index = 0;
8         t = 0;
9     }
10
11    public boolean hasNext() {
12        return index < p.size();
13    }
14    public double[] next() {
15        Rythme r = p.get(index);
16        index ++;
17        double[] retour = new double[]{t, r.getDuree()* p.getTempo()/60., r.getFreq()};
18        t += r.getDuree() * p.getTempo()/60.;
19        return retour;
20    }
21 }
```

Q 74.7 Proposer une classe de test qui construit une partition de 3 notes, la donne à un traducteur puis vérifie le bon fonctionnement de celui-ci en faisant défiler les triplés et en les affichant dans la console.

```
1     public static void main(String[] args)  {
2         Partition p = new Partition(60);
```

```
3         p.add(new Noire(Note.do_));
4         p.add(new Noire(Note.re_));
5         p.add(new Noire(Note.mi_));
6
7         Traducteur t = new Traducteur(p);
8         while(t.hasNext()){
9             double[] triplet = t.next();
10            System.out.println(triplet[0]+" "+triplet[1]+" "+triplet[2] );
11        }
12    }
```

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`
- `boolean delete()`	détruit le fichier ou le répertoire
- `boolean exists()`	indique si le fichier existe physiquement
- `String getAbsolutePath()`	renvoie le chemin absolu du fichier
- `File getParentFile()`	renvoie un objet `File` pointant sur le chemin parent de celui de l'objet `File` courant
- `boolean isDirectory()`	indique si l'objet `File` pointe sur un répertoire
- `boolean isFile()`	indique si l'objet `File` pointe sur un fichier
- `File[] listFiles()`	si l'objet `File` est un répertoire, renvoie la liste des fichiers qu'il contient
- `boolean mkdir()`	création du répertoire
- `boolean mkdirs()`	création de toute l'arborescence du chemin
- `boolean renameTo(File f)`	renomme le fichier

Exercice 75 – Manipulation de fichiers et d'arborescences

Soit la classe `TestFile` suivante :

```
1 import java.io.File;
2 import java.io.IOException;
3
4 public class TestFile{
5     public static void main(String[] args){
6         try {
7             File f=new File(args[0]);
8             f.delete();
9             System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
10            f.createNewFile();
11            System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
12            System.out.println(f.getAbsolutePath());
13            System.out.println(f.getPath());
14        } catch(IOException e){
15            System.out.println(e);
16        }
17    }
18 }
```

Q 75.1 Dire ce qu'affiche l'execution suivante : `java TestFile "./lu2in002/TME11/Files/fichier1.txt"`
- Si le répertoire `"./lu2in002/TME11/Files"` existe
- Si le répertoire `"./lu2in002/TME11/Files"` n'existe pas

```
// Si le répertoire existe :
Le fichier existe : non
Le fichier existe : oui
/home/.../LU2IN002/TME11/Files/fichier1.txt
./fichier1.txt

// Si le répertoire n'existe pas
Le fichier existe : non
java.io.IOException: Le chemin d'acces specifie est introuvable

// L'exception est envoyee a la creation du fichier
```

Q 75.2 Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier

```
// Ajouter les deux lignes suivantes juste apres la construction de f :

File r=f.getParentFile();
if (r!=null){
    r.mkdirs();
}
```

Q 75.3 Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la classe `File`

```
public static void pwd(){
        File f=new File(".");
        System.out.println(f.getAbsolutePath());
}
```

Q 75.4 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)

```
public static void ls(File f){
        File[] childs=f.listFiles();
        for(int i=0;i<childs.length;i++){
            File fic=childs[i];
            if (fic.isFile()){
                System.out.println(fic.getAbsolutePath());
            }
        }
}
```

Q 75.5 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)

```
1 public static void lsRecursif(File f){
2         File[] childs=f.listFiles();
3         for(int i=0;i<childs.length;i++){
4             File fic=childs[i];
5             if (fic.isFile()){
6                 System.out.println(fic.getAbsolutePath());
7             } else {
8                 lsRecursif(fic);
9             }
10        }
11 }
```

Les flux

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

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

Deux catégories de flux :

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

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

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

La classe `Reader` définit principalement les méthodes suivantes :

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

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

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

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

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

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

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

Exercice 76 – Traitement de texte

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

Q 76.1 Écrire une méthode `String saisie()` qui demande à l'utilisateur de saisir une ligne de texte tant que la ligne entrée par l'utilisateur est différente de la chaîne "_fin_". Cette méthode retourne une chaîne de caractères contenant la concaténation de toutes les lignes saisies. Proposez une première solution utilisant des concaténations de `String`. Puis proposez une deuxième solution utilisant un seul objet `StringWriter`.

Il existe 4 façons de concaténer des chaines de caractères : l'opérateur "+", la méthode `String.concat()`, et les classes `StringBuffer` et `StringBuilder`. Mais avant de les comparer, il faut comprendre la base du problème : **l'immutabilité** de la classe `String`. `String` étant une classe immutable, une variable de type `String` ne peut pas être modifiée. Lorsque l'on croit modifier une `String`, en vérité, on crée une nouvelle `String` à partir de l'ancienne qui sera soumise au garbage collector.
Exemple : `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour");a=new String(a+" tous le monde");`
Le premier objet est détruit lorsqu'on crée le deuxième.

```
1 public static String saisie(){
2         String texte="";
3         String ligne="";
4         while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
5             texte+=ligne+"\n";
6         }
7         return(texte);
8 }
9
10 // On peut faire remarquer qu'il serait plus efficace d'utiliser un StringWriter
11 public static String saisie(){
```

```
12          StringWriter texte=new StringWriter();
13          String ligne="";
14          while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
15              texte.write(ligne+"\n");
16          }
17          return(texte.toString());
18  }
19
20  // ou un java.lang.StringBuilder
21  public static String saisie(){
22          StringBuilder texte=new StringBuilder();
23          String ligne="";
24          while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
25              texte.append(ligne+"\n");
26          }
27          return(texte.toString());
28      }
```

Q 76.2 Écrire une méthode `affiche(String fichier)` affichant le contenu du fichier dont le nom est passé en paramètre.

```
1  public static void affiche(String fichier) throws IOException{
2          File file=new File(fichier);
3          FileReader lecteur=new FileReader(file);
4          try {
5              int c;
6              StringBuilder texte=new StringBuilder();
7              while((c=lecteur.read())!=-1){
8                  texte.append((char)c);
9              }
10             System.out.println(texte.toString());
11         }
12         finally{
13             lecteur.close();
14         }
15 }
16
17 // --- Ou avec un tableau de char ---
18 public static void affiche(String fichier) throws IOException{
19         File file=new File(fichier);
20         FileReader lecteur=new FileReader(file);
21         try {
22             int nb=0;
23             char[] buf=new char[100];
24             StringBuilder texte=new StringBuilder();
25             while((nb=lecteur.read(buf))>0){
26                 texte.append(buf);
27                 buf=new char[100];
28             }
29             System.out.println(texte.toString());
30         }
31         finally{
32             lecteur.close();
33         }
34 }
```

Q 76.3 Écrire une méthode `afficheLignes(String fichier)` affichant, en numérotant les lignes, le contenu du fichier passé en paramètre.

```
public static void afficheLignes(String fichier) throws IOException{
        File file=new File(fichier);
        FileReader lecteur=new FileReader(file);
        try {
            int c;
            int nbLines=1;
            StringBuilder texte=new StringBuilder();
            boolean nligne=true;
            while((c=lecteur.read())!=-1){
                if (nligne){
                    texte.append(nbLines+" : ");
                    nbLines++;
                    nligne=false;
                }
                char car=(char) c;
                texte.append(car);
                if (car=='\n'){
                    nligne=true;
                }
            }
            System.out.println(texte.toString());
        }
        finally{
            lecteur.close();
        }
}
```

Q 76.4 Écrire une méthode `ecrireTexte(String fichier)` permettant de créer un nouveau fichier contenant un texte saisi par l'utilisateur.

```
public static void ecrireTexte(String fichier) throws IOException{

        File file=new File(fichier);
        FileWriter ecrivain=new FileWriter(file);
        try {
            String saisie=saisie();
            ecrivain.write(saisie);
            ecrivain.flush(); // ecriture dans le fichier maintenant
        }
        finally{
            ecrivain.close();
        }
}
```

Q 76.5 Écrire une méthode `ajouteTexte(String fichier)` permettant d'ajouter, en fin de fichier passé en paramètre, du texte saisi par l'utilisateur.

```
public static void ajouteTexte(String fichier) throws IOException{

```

```
3         File file=new File(fichier);
4         File tmp=new File("tmp_file.txt");
5
6         FileWriter ecrivain=new FileWriter(tmp);
7         FileReader lecteur=new FileReader(file);
8         try {
9                 int nb=0;
10                char[] buf=new char[100];
11                while((nb=lecteur.read(buf))>0){
12                    ecrivain.write(buf,0,nb-1);
13                    ecrivain.write("\n");
14                    ecrivain.flush();
15                    buf=new char[100];
16
17                }
18
19                String saisie=saisie();
20                ecrivain.write(saisie);
21                ecrivain.flush();
22        }
23        finally{
24                ecrivain.close();
25                lecteur.close();
26                if (!file.delete()){
27                    throw new IOException("Probleme suppression fichier");
28                }
29                if (!tmp.renameTo(file)){
30                    throw new IOException("Probleme renommage du fichier");
31                }
32        }
33}
34
35// Ou avec le constructeur de FileWriter qui accepte en second argument un boolean
      append
36
37public static void ajouteTexte(String fichier) throws IOException{
38
39        File file=new File(fichier);
40        FileWriter ecrivain=new FileWriter(file,true);
41
42        try {
43                String saisie=saisie();
44                ecrivain.write(saisie);
45                ecrivain.flush();
46        }
47        finally{
48                ecrivain.close();
49        }
50}
```

Q 76.6 Écrire une méthode `replace(int num, String newLigne, String fichier)` permettant de remplacer, dans le fichier passé en paramètre, la ligne numéro `num` par la nouvelle ligne `newLigne`.

```
1public static void replace(int num, String newLigne, String fichier) throws IOException
      {
2         File file=new File(fichier);
```

```
3         File tmp=new File("tmp_file.txt");
4
5         if (file.exists()){
6             FileWriter ecrivain=new FileWriter(tmp);
7             FileReader lecteur=new FileReader(file);
8             try {
9                 int c;
10                int nbLines=0;
11                StringBuilder ligne=new StringBuilder();
12
13                while((c=lecteur.read())!=-1){
14                    char car=(char) c;
15                    ligne.append(car);
16                    if (car=='\n'){
17                        nbLines++;
18                        if (nbLines==num){
19                            ecrivain.write(newLigne+"\n");
20                        } else {
21                            ecrivain.write(ligne.toString());
22                        }
23                        ligne=new StringBuilder();
24                    }
25                }
26                if (nbLines<num) {
27                    System.out.println("Pas de ligne "+num);
28                }
29            }
30            finally{
31                ecrivain.close();
32                lecteur.close();
33                if (!file.delete()){
34                    throw new IOException("Probleme suppression fichier");
35                }
36                if (!tmp.renameTo(file)){
37                    throw new IOException("Probleme renommage du fichier");
38                }
39            }
40        } else {
41            System.out.println("Pas de ligne "+num);
42        }
43
44    }
```

Q 76.7 Écrire un programme proposant à l'utilisateur un menu lui permettant d'éditer un fichier dont le chemin est passé en argument. Exemple :

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

```
1 public static int menu(String fichier){
2         System.out.println("Fichier "+fichier);
3         System.out.println("1. Ajouter texte");
4         System.out.println("2. Afficher fichier");
```

```
5         System.out.println("3. Remplacer ligne");
6         System.out.println("4. Quitter");
7         int choix=Clavier.saisirEntier("Votre choix ?");
8         return(choix);
9 }
10
11 public static void main(String[] args){
12         int choix;
13         String fichier=args[0];
14         try {
15             File file=new File(fichier);
16             if (!file.exists()){
17                 File parent=file.getParentFile();
18                 if (parent!=null){
19                     parent.mkdirs();
20                 }
21                 file.createNewFile();
22             }
23
24             while((choix=menu(fichier))!=4){
25                 if ((choix<1) || (choix>4)){
26                     System.out.println("Choix invalide");
27                     continue;
28                 } else if (choix==1){
29                     ajouteTexte(fichier);
30                 } else if (choix==2){
31                     afficheLignes(fichier);
32                 } else if (choix==3){
33                     int num=Clavier.saisirEntier("Quele ligne a remplacer ?");
34                     String newLigne=Clavier.saisirLigne("Entrez la nouvelle ligne "+num
                           );
35                     replace(num,newLigne,fichier);
36                 }
37             }
38         } catch(IOException e) {
39             System.out.println(e);
40         }
41 }
```

Exercice 77 – Copie de fichiers binaires

Q 77.1 Écrire un programme permettant de copier un fichier binaire dont le nom est donné en premier argument sous le nom donné en second argument.

```
1 import java.io.*;
2
3 public class CopyFiles {
4    public static void copyFile(File srcFile, File destFile) throws IOException {
5       InputStream oInStream = new FileInputStream(srcFile);
6       OutputStream oOutStream = new FileOutputStream(destFile);
7
8       // Transfer bytes from in to out
9       byte[] oBytes = new byte[1024];
10      int nLength;
```

```
11
12        while ((nLength = oInStream.read(oBytes)) > 0) {
13           oOutStream.write(oBytes, 0, nLength);
14           oOutStream.flush();
15        }
16        oInStream.close();
17        oOutStream.close();
18     }
19     public static void main(String[] args){
20        try {
21           copyFile(new File(args[0]),new File(args[1]));
22        } catch(IOException e) {
23           System.out.println(e);
24        }
25     }
26 }
```

La mise en mémoire tampon

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

Exercice 78 – Mise en mémoire tampon

Q 78.1 Sachant que la construction d'un `BufferedReader` se fait en passant un flux `Reader` en paramètre, écrivez l'ouverture d'un flux de lecture avec utilisation de la mémoire tampon sur un fichier "text.txt" du répertoire courant.

```
1 BufferedReader in= new BufferedReader(new FileReader(new File("text.txt")));
```

Q 78.2 Écrire une méthode `afficheLignesFichier(String fichier)` qui affiche ligne après ligne le texte du fichier dont le chemin est passé en paramètre.

```
1 public static void afficheLignesFichier(String fichier) throws IOException,
     FileNotFoundException{
2         BufferedReader lecteur=new BufferedReader(new FileReader(new File(fichier)));
3         try {
4             String ligne="";
5             while((ligne=lecteur.readLine())!=null){
6                 System.out.println(ligne);
7             }
8         }
9         finally{
10                lecteur.close();
11        }
12 }
```

Q 78.3 Sachant qu'il est également recommandé par souci d'efficacité d'encapsuler tout flux en écriture dans un objet `BufferedWriter` (resp. `BufferedStream` pour l'écriture d'octets), écrire une classe `Ecrivain` ouvrant un flux en écriture sur un fichier à sa construction et disposant des méthodes données ci-dessus pour la classe `PrintWriter` (sauf méthode `flush()`). On pourra donner une version avec héritage et une version sans.

```
// --- Avec heritage ---
import java.io.*;
public class Ecrivain extends PrintWriter {
    public Ecrivain(String fichier) throws IOException{
        super(new BufferedWriter(new FileWriter(new File(fichier))));
    }
    public void finalize(){
        close();
    }
}

// --- Sans heritage ---
import java.io.*;
public class Ecrivain {
    private PrintWriter pw=null;
    public Ecrivain(String fichier) throws IOException{
        pw=new PrintWriter(new BufferedWriter(new FileWriter(new File(fichier))));
    }
    public void close(){
        if (pw!=null){
            pw.close();
            pw=null;
        }
    }
    public void finalize(){
        close();
    }
    public void println(String s){
        if (pw!=null){
            pw.println(s);
        } else {
            System.out.println("Flux ferme");
        }
    }
    public void print(String s){
        if (pw!=null){
            pw.print(s);
        } else {
            System.out.println("Flux ferme");
        }
    }
}
```

Exercice 79 – Production automatique de compte rendu TME

L'objectif de cet exercice est d'utiliser les connaissances acquises sur la lecture et l'écriture de fichier pour programmer un outil de production automatique de compte rendu de TME.
On considère que l'utilisateur dispose d'une arborescence de fichiers (telle que celle de votre répertoire) prenant racine en un répertoire LU2IN002. Ce répertoire contient un répertoire par TME (numérotés de TME1 à TME11), chacun d'entre eux contenant eux mêmes un répertoire par exercice (Exo1, Exo2, ... ExoN). On considère également que l'on dispose d'un fichier "etudiants.txt" dans le répertoire LU2IN002 contenant les prenoms, noms et numeros d'etudiants

des utilisateurs du programme (une ligne par étudiant). Le fichier doit se terminer par une ligne "Groupe : <numero du groupe>". Enfin, chaque répertoire d'exercice contient deux fichiers "intitule.txt" et "executions.txt", le premier contenant l'énoncé de l'exercice, le second contenant les résultats d'exécution des programmes ainsi que les observations qui ont pu avoir été faites.

Q 79.1 Écrire un programme `RenduTMEProducer` prenant en argument le chemin du répertoire de TME concerné par le compte rendu et produisant en racine de ce répertoire un fichier "compteRenduTME.txt" de la forme de celui que vous avez l'habitude de rendre en fin de TME.

```
1 import java.io.*;
2 import java.util.ArrayList;
3 public class RenduTMEProducer {
4
5     public static ArrayList<File> getAllJavaFiles(File f){
6         ArrayList<File> ret=new ArrayList<File>();
7         File[] childs=f.listFiles();
8         for(int i=0;i<childs.length;i++){
9             File fic=childs[i];
10            if (fic.isFile()){
11                String path=fic.getAbsolutePath();
12                int iext=path.lastIndexOf(".");
13
14                if ((iext>0) && (iext<path.length()-4)){
15                    String ext=path.substring(iext+1,iext+5);
16                    if (ext.compareTo("java")==0){
17                        ret.add(fic);
18                    }
19                }
20            } else {
21                ret.addAll(getAllJavaFiles(fic));
22            }
23        }
24        return(ret);
25    }
26
27    public static String getTexteFromFile(File f) throws IOException{
28        StringBuilder sb=new StringBuilder();
29        BufferedReader lecteur=new BufferedReader(new FileReader(f));
30        try {
31            String ligne="";
32            while((ligne=lecteur.readLine())!=null){
33                    sb.append(ligne+"\n");
34            }
35        }
36        finally{
37            lecteur.close();
38        }
39        return(sb.toString());
40    }
41    public static void produceCompteRendu(String rep) throws IOException{
42        File tme=new File(rep);
43        if (!tme.exists()){
44            throw new IOException("Le repertoire de TME donne n existe pas");
45        }
46        tme=new File(tme.getAbsolutePath());
47        if (!tme.isDirectory()){
48            throw new IOException("Le chemin donne n est pas un repertoire");
49        }
50
```

```
51          File lu2in002=tme.getParentFile();
52          if (lu2in002.getName().compareTo("lu2in002")!=0){
53              throw new IOException("Le repertoire parent du repertoire donne n'est pas
                    lu2in002");
54          }
55
56          File etudiants=new File(lu2in002.getAbsolutePath()+"/etudiants.txt");
57          if (!etudiants.exists()){
58              throw new IOException("Le repertoire lu2in002 ne contient pas de fichier
                    nomme etudiants.txt");
59          }
60
61          File rendu=new File(tme.getAbsolutePath()+"/compteRenduTME.txt");
62          System.out.println(rendu.getAbsolutePath());
63          boolean ok=true;
64          if (rendu.exists()){
65              ok=false;
66              String ecrase=Clavier.saisirLigne("Le repertoire "+rep+" contient deja un
                    compte rendu de TME. Voulez vous l'ecraser ?(Oui/Non)");
67              ecrase=ecrase.toLowerCase();
68              if (ecrase.compareTo("oui")==0){
69                  ok=true;
70              }
71          }
72          if (ok){
73              PrintWriter ecrivain=new PrintWriter(new BufferedWriter(new FileWriter(
                    rendu)));
74              try {
75                  ecrivain.println("Compte Rendu lu2in002 \n");
76                  String texteEtudiants=getTexteFromFile(etudiants);
77                  ecrivain.println(texteEtudiants+"\n");
78                  ecrivain.println("Numero du TME : "+tme.getName()+"\n");
79
80                  File[] exos=tme.listFiles();
81                  for(int i=0;i<exos.length;i++){
82                      File exo=exos[i];
83                      if (exo.isDirectory()){
84                          System.out.println("exo : "+exo.getAbsolutePath());
85                          ecrivain.println("// ----------------------");
86                          ecrivain.println("Intitule de l'exercice :");
87                          File intitule=new File(exo.getAbsolutePath()+"/intitule.txt");
88                          if (intitule.exists()){
89                              ecrivain.println(getTexteFromFile(intitule));
90                          }
91                          ecrivain.println("");
92                          ecrivain.println("Classes : \n");
93                          ArrayList<File> classes=getAllJavaFiles(exo);
94                          for(File classe:classes){
95                              ecrivain.println(getTexteFromFile(classe)+"\n");
96                          }
97                          ecrivain.println("Executions et observations :");
98                          File execution=new File(exo.getAbsolutePath()+"/executions.txt"
                                );
99                          if (execution.exists()){
100                             ecrivain.println(getTexteFromFile(execution));
101                         }
102                         ecrivain.println("");
103                     }
104                 }
```

```
105             } catch(IOException e) {
106                 ecrivain.close();
107                 rendu.delete();
108                 throw(e);
109             }
110             finally{
111                 ecrivain.close();
112             }
113         }
114     }
115
116     public static void main(String[] args){
117         try {
118             produceCompteRendu("ExempleArborescence/lu2in002/TME11");
119         } catch(IOException e) {
120             System.out.println(e+"\n");
121             e.printStackTrace();
122         }
123
124     }
125 }
```

Entrée / Sortie standard

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

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

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

Exercice 80 – Classe Clavier

Q 80.1 Sachant que le paquetage `java.io` contient une classe de flux `InputStreamReader` permettant de lire des caractères à partir d'un flux entrant d'octets, réécrire le code de la classe `Clavier`, notamment :

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

```
1 import java.io.* ;
2
3 /**Cette classe implante des saisies au clavier par lecture d'une ligne. */
4 public class Clavier {
5
6     private static final BufferedReader in =
7         new BufferedReader (new InputStreamReader (System.in)) ;
8
9     /** Affiche le message et retourne un int lu au clavier. */
```

```
10      public static int saisirEntier (String mess){
11          while(true){
12              try {
13                  return Integer.parseInt (saisirLigne ( mess)) ;
14              } catch (NumberFormatException e) {
15                  mess = "Recommencez␣:␣" ;
16              }
17          }
18      }
19
20      /** Affiche le message et retourne une ligne lue au clavier. */
21      public static String saisirLigne (String mess) {
22          System.out.println (mess) ;
23          try {
24              return in.readLine () ;
25          } catch (IOException e){
26              return null; // provisoire !!
27          }
28      }
29  } // Clavier
```

Quizz 15 – `String`, classe immutable

QZ 15.1 Combien d'objets sont créés dans les instructions ci-après ?
`String a="Bonjour"; a=a+" tous le monde";`

Il y a trois objets créés. `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour"); b=new String("tout le monde"); a=new String(a+b); b=null;`
Le compilateur, s'il est bien optimisé peut réussir à réduire le nombre à 2 [je ne sais pas ce qu'il fait].

QZ 15.2 Donnez une solution équivalente en utilisant un `StringWriter`.

Un seul objet nécessaire.

```
1 import java.io.StringWriter;
2
3 StringWriter sw=new StringWriter();
4 sw.write("Bonjour");
5 sw.write("␣tous␣le␣monde");
6 System.out.println(sw.toString());
```

Aide mémoire

Site web de l'UE

Le site web annuel de l'UE :

https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2020/ue/LU2IN002-2020oct/

Convention d'écriture

— Le nom des classes (et des constructeurs) commence par une majuscule ;

— Le nom des méthodes, des variables ou des instances commence par une minuscule ;
— Les mots réservés sont obligatoirement en minuscules ;
— Les constantes sont généralement en majuscules.

En-tête du main

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

Grandes lignes de la structure d'une classe

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

Commentaires

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

Divers

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

Principales instructions

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

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

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

Instruction de contrôle - Boucles

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

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

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

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

Tableau de codage des types simples

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

Table de priorité des opérateurs

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

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

La classe `Math` (standard)

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

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

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

La classe `String` (standard)

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

La classe `Arraylist` (standard)

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

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

Environnement Linux

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

Création et gestion de répertoires sous Linux

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

Démarrage sous Linux

— Pour ouvrir une fenêtre de travail : cliquer sur l'icone "Terminal" dans le bandeau en haut de la fenêtre OU choisir menu Accessoires, option "Terminal".
— Lancer un éditeur de texte. Par exemple :
  — pour lancer l'éditeur `gedit`, tapez dans le terminal : `gedit &`
  — pour lancer l'éditeur `emacs`, tapez dans le terminal : `emacs &`
  Attention : si on oublie de taper le caractère "&" en fin de commande, on ne pourra plus rien exécuter dans la fenêtre de travail sauf en tapant CTRL Z pour interrompre la commande, puis en tapant la commande bg (background) pour relancer la commande sans perdre le contrôle de la fenêtre.
— Dans le terminal, pour reprendre une commande que vous avez déjà tapée dans le terminal : utilisez les flèches *haut* et *bas* pour se déplacer dans l'historique des commandes. Et utiliser les flèches *gauche* et *droite* pour se déplacer dans la commande que l'on peut alors modifier.

Exécution de programmes

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

Quelques bonnes pratiques pour écrire les programmes

Indentation

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

Quelques conseils

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

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

Quelques erreurs fréquentes

— L'oubli d'une accolade est souvent très difficile à retrouver. Donc, chaque fois que vous tapez {, dans la foulée tapez } et ouvrez des lignes entre les deux en tapant simplement Entrée. C'est ce qu'on pourrait appeler la mise en place de la structure d'un programme avant d'écrire le corps du programme.
— Lorsque vous lancez une compilation javac Bonjour.java par exemple, si le système vous dit "*cannot read*", c'est qu'il n'a pu lire le fichier Bonjour.java. Autrement dit, vous n'êtes pas dans le bon répertoire. Changez de répertoire (commande cd).
— Autre erreur fréquente : vous écrivez une instruction en dehors d'une méthode. Un fichier Java est composé de classe(s). Une classe est constituée de déclarations de variables et de méthodes. Une méthode est composée de déclarations de variables (locales) et d'instructions. Une instruction est donc nécessairement à l'intérieur d'une méthode.

— Java impose de respecter la casse (c'est-à-dire les majuscules ou minuscules). L'identificateur `toto` est différent de `Toto` ; `setVisible` est différent de `setvisible` ; `Main` est différent de `main`, etc.