



2i002 : Éléments de programmation par objets avec JAVA

Licence de Sciences et Technologies
Mention Informatique

Fascicule de TD/TME

Année 2017-2018



1 TD : classe : définition, syntaxe

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 48.

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
10    public String toString() {
11        String s="Planete_"+nom+"_de_rayon_"+rayon;
12        return s;
13    }
14
15    public double getRayon() {
16        return rayon;
17    }
18 }
```

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

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

Q 2.3 Quelles sont les méthodes de cette classe ?

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

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

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

Exercice 3 – Se présenter

Q 3.1 Une personne est représentée par son nom et son âge. Ecrire la classe `Personne` qui contient :

- les variables d'instance `nom` et `age`,
- un constructeur dont la signature est : `public Personne(String n, int a)`.

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

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

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

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

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

Q 3.6 Ajouter dans la classe `Personne`, la méthode `vieillir()` qui ajoute un an à la personne. Dans la classe `Presentation`, faites vieillir Paul de 20 ans (utiliser une boucle `for`), et Pierre de 10 ans (utiliser une boucle `while`), puis faites se présenter Paul et Pierre. Aide : voir la syntaxe des boucles page 49

Exercice 4 – Alphabet

Q 4.1 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.

Q 4.2 Recommencer en utilisant une boucle `while`.

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).

<code>abcDefg()</code>	<code>true</code>	<code>abcd</code>	<code>Abcd</code>
<code>Abc()</code>	<code>String</code>	<code>False</code>	<code>ABCD</code>

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).

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

Exercice 7 – Rétro-engineering

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

1 2 3 5 8 13 21 34 45

Quizz 1 – Compilation et exécution

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

QZ 1.2 Un fichier source Java contient ...

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

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

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

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` ?

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

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

Quizz 2 – Syntaxe des expressions

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

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

QZ 2.3 Sachant que le code ascii de `'1'` est 49, qu'affiche :

1. `System.out.println(1+"1")` ?
2. `System.out.println(1+'1')` ?
3. `System.out.println(true && 49 == '1')` ?

QZ 2.4 Qu'affiche : `System.out.println("Bonjour \nvous \ttous!")` ?

QZ 2.5 Pour générer un nombre aléatoire, on peut utiliser la méthode `Math.random()` qui rend un `double` entre 0 et 1 exclu. Comment faire pour générer un entier entre 10 et 30 compris ?

2 Encapsulation, surcharge

Exercice 8 – 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 completer
12    }
13    public String toString(){
14        return("Volume_du_liquide_dans_la_bouteille="+volume);
15    }
16 }
```

Q 8.1 Combien y-a-t-il de constructeurs dans cette classe ? Quel est la différence entre ces constructeurs ? Pour chaque constructeur, donner les instructions qui permettent de créer un objet utilisant ce constructeur.

Q 8.2 Expliquer l'affectation de la ligne 5 : que représente `this.volume` ? `volume` ?

Q 8.3 Expliquer la ligne 8.

Q 8.4 Compléter la méthode d'instance `remplir(Bouteille b)` qui ajoute le contenu de la bouteille `b` à la bouteille courante.

Q 8.5 Peut-on rajouter une méthode portant le même nom que la méthode précédente, mais prenant un paramètre de type `double` ? Si oui, écrire cette méthode.

Q 8.6 Quel va être le résultat de l'affichage des lignes 5, 6, 8 et 9 du programme ci-après ? Expliquer.

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

Exercice 9 – Salutation ! (appels aux constructeurs, surcharge)

Afin de comprendre le fonctionnement des appels aux constructeurs, on propose d'étudier les 2 classes suivantes à placer respectivement dans les fichiers `Salutation.java` et `TestSalutation.java`.

```
1 public class Salutation {
2     public Salutation() {
3         System.out.println("Appel_au_constructeur_sans_paramètre ");
4     }
5 }
```

```

6 public class TestSalutation {
7     public static void main(String [] args) {
8         Salutation s1=new Salutation();
9     }
10 }

```

Q 9.1 Quel est le résultat affiché par ce programme ?

Q 9.2 Dans la classe `Salutation`, ajouter un deuxième constructeur (surcharge de constructeurs). Ce constructeur prend en paramètre une chaîne de caractères et l'affiche.

Q 9.3 Dans la classe `TestSalutation`, ajouter plusieurs instances de la classe en appelant plusieurs fois chaque constructeur. Quel est le résultat affiché par ce programme ?

Exercice 10 – Gestion des complexes

Q 10.1 Classe Complexe

La classe `Complexe` possède :

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

Q 10.1.1 Donner le code de la classe `Complexe`

Q 10.1.2 Ajouter les méthodes suivantes (à vous de déterminer les signatures) :

- `toString` qui génère une chaîne de caractère de la forme : (*reelle* + *imag i*)
- `addition` de deux complexes,
- `multiplication` de deux complexes,
- `estReel` qui teste si le complexe est en fait réel (dans le cas où la partie imaginaire est nulle).

Q 10.1.3 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
NB : vous ajouterez en commentaire les résultats attendus

Q 10.1.4 Donner les instructions pour compiler ces deux classes et exécuter le test.

Exercice 11 – Pion

Soient les classes suivantes :

```

1 public class Pion {
2     private String nom ;
3     // position du pion
4     private double posx ;
5
6     public Pion(String n) {
7         nom=n;
8         posx=Math.random();
9     }
10    public void setNom(String n) { nom=n; }
11    public String getNom() { return nom; }
12 }
13 public class TestPion {
14     public static void main(String [] args) {
15         Pion unPion=new Pion("Atchoum");
16         Pion autrePion=unPion;
17         autrePion.setNom("Dormeur");
18         System.out.println(unPion.getNom());
19     }
20 }

```

Q 11.1 Que s'affiche-t-il ?

Q 11.2 Proposer une ou des solutions pour éviter le problème précédent.

Exercice 12 – Point : suite sur l'égalité

Soit le programme suivant :

```

1 // TestPoint.java
2 public class TestPoint {
3     public static void main(String[] args) {
4         Point p1 = new Point(1,2);
5         Point p2 = new Point(1,2);
6         Point p3 = new Point(2,3);
7         Point p4 = p1;
8         Point p5 = null;
9         Point p6 = p5;
10        p6 = new Point(3,4);
11        if (p1==p2)
12            System.out.println("p1_égale_p2");
13        if (p1==p3)
14            System.out.println("p1_égale_p3");
15        if (p1==p4)
16            System.out.println("p1_égale_p4");
17        if (p1==null)
18            System.out.println("p1_égale_null");
19        if (p1.equals(p2))
20            System.out.println("p1_égale_p2_(2)");
21        if (p1.equals(p4))
22            System.out.println("p1_égale_p4_(2)");
23    }
24 }

```

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

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

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

Q 12.3 Le comportement n'est pas celui attendu : proposer une modification de code pour améliorer l'objet `Point`.

Q 12.4 Donner les sorties associées aux commandes suivantes :

```

1 System.out.println(p5);   System.out.println(p6);
2 System.out.println(p5.toString());   System.out.println(p6.toString());

```

Exercice 13 – 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
4     public maMethode(int i) {
5         System.out.println("je_passe_dans:_maMethode(int_i)");
6     }
7     public maMethode(double d) {
8         System.out.println("je_passe_dans:_maMethode(double_d)");
9     }
10    public maMethode(double d1, double d2) {
11        System.out.println("je_passe_dans:_maMethode(double_d1,_double_d2)");
12    }
13    public maMethode(int i1, int i2, int i3) {
14        System.out.println("je_passe_dans:_maMethode(int_i1,_int_i2,_int_i3)");
15    }
16 }

```

Q 13.1 Selon le principe de base de JAVA qui interdit deux signatures identiques pour des méthodes (sans prise en compte du retour), cette classe compile-t-elle ?

Q 13.2 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 }

```

Quizz 3 – 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
10    public Fleur (String nom) {
11        this(nom, "rouge");
12    }
13
14    public String toString() {
15        return nom + " de couleur " + couleur ;
16    }
17
18    public String getNom() { return nom; }
19 }
20
21 public class Quizz {
22     public static void main (String[] args ) {
23         Fleur tulipe = new Fleur("Tulipe", "Jaune");
24         System.out.println(tulipe.getNom());
25     }
26 }

```

QZ 3.1 Donner les commandes pour compiler, puis exécuter ce programme.

QZ 3.2 Pourquoi a-t-on déclaré `private` les variables `nom` et `couleur` ?

QZ 3.3 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 23 et 24 ?

QZ 3.4 Dans la classe `Quizz`, quelle différence faites-vous entre `tulipe` et `"Tulipe"` ?

QZ 3.5 Quel est le rôle de la méthode `getNom()` ?

QZ 3.6 Dans le constructeur de la classe `Fleur`, aurait-on pu écrire `this.nom = name` ?

QZ 3.7 Si dans la méthode `main`, on rajoute l'instruction : `tulipe.toString()`; Quel est le résultat produit par cette instruction ?

QZ 3.8 Un étudiant rajoute le constructeur suivant. Quelle erreur est signalée à la compilation ?


```

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

```

QZ 3.9 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 }

```

Quizz 4 – Encapsulation

```

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

```

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

Quizz 5 – Méthode toString()

QZ 5.1 `int k=3; System.out.println("k="+k.toString());` Ces instructions sont-elles correctes ?

QZ 5.2 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)` ?

3 Composition, copie d'objets

Exercice 14 – Classe triangle

Q 14.1 Ecrire une classe `Point` à deux variables d'instance `posx` et `posy`, respectivement l'abscisse et l'ordonnée du point. Cette classe comprendra :

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

Q 14.2 Tester cette classe en écrivant la méthode `main` qui crée des points et affiche leurs coordonnées.

Q 14.3 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`).

Q 14.4 Ecrire une classe `TestTriangle`, contenant une méthode `main` dans laquelle on crée trois points, puis un triangle composé de ces trois points. On affichera ensuite les caractéristiques du triangle (les trois points, la longueur de ses côtés et son périmètre).

Q 14.5 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.

Exercice 15 – Feu tricolor

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

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

Q 15.1 Ecrire la classe `FeuTricolor` avec les constructeurs suivants :

- un constructeur sans paramètre qui crée un feu tricolor où toutes les lampes sont éteintes.
- un constructeur qui prend 3 lampes en paramètre. Donnez 2 façons de créer un objet de la classe `FeuTricolore` en utilisant ce constructeur.

Q 15.2 Pourquoi le constructeur suivant est-il erroné ? Faire un schéma des objets en mémoire.

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

Q 15.3 Trouvez et expliquez les erreurs dans les instructions ci-après. Faire un schéma des objets en mémoire.

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

Exercice 16 – Mariage (composition récursive)

On veut écrire un programme qui modélise le mariage et le divorce. Pour simplifier, on supposera que le mariage est possible entre deux personnes de même sexe, et que les personnes s'appellent "Individu" suivi de 3 lettres. Voici une autre possibilité pour écrire la classe **Personne** :

```

1 public class Personne {
2     private String nom;
3
4     public Personne() {
5         this("Individu");
6         nom = nom + tirageLettre()+ tirageLettre()+ tirageLettre();
7     }
8     public Personne(String nom) {
9         this.nom=nom;
10    }
11
12    private char tirageLettre(){
13        return (char) ((int) (Math.random()*26) + 'A');
14    }
15 }

```

Q 16.1 Compléter et modifier la classe **Personne** pour avoir le conjoint de cette personne (qui est une **Personne**). Par défaut une personne est célibataire. Ecrire aussi la méthode **toString()** qui retourne le nom de la personne auquel est ajouté "célibataire" ou "marié" suivant le cas. Par exemple : "IndividuA, marié".

Q 16.2 Ecrire la méthode **void epouser(Personne p)** qui marie cette personne et la personne **p**. Si le mariage est impossible, on affiche le message "Ce mariage est impossible!".

Q 16.3 Ecrire la méthode **void divorcer()** qui fait divorcer cette personne si c'est possible.

Q 16.4 Ecrire 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 :

IndividuA , celibataire

IndividuB , celibataire

IndividuC , celibataire

Mariage de IndividuA , celibataire et de IndividuB , celibataire :

IndividuA , celibataire se marie avec IndividuB , celibataire

Les personnes apres mariage :

IndividuA , marie(e)

IndividuB , marie(e)

Essai de mariage de IndividuA , marie(e) et de IndividuC , celibataire :

Ce mariage est impossible!

Divorce de IndividuA , marie(e) et de IndividuB , marie(e) :

IndividuA , marie(e) divorce de IndividuB , marie(e)

Les personnes apres divorce :

IndividuA , celibataire

IndividuB , celibataire

Exercice 17 – Tracteur (composition d'objets et copie d'objets)

On suppose qu'un tracteur est composé de 4 roues et d'une cabine.

Q 17.1 Écrire une classe `Roue` ayant un attribut privé de type `int` définissant son diamètre. Ecrire deux constructeurs, l'un avec un paramètre, et l'autre sans paramètre qui appellera le premier pour mettre le diamètre à 60 cm (petite roue). Ecrire aussi la méthode `toString()`.

Q 17.2 Créez une classe `TestTracteur` dans laquelle vous testerez la classe `Roue` dans une méthode `main` où vous créerez 2 grandes roues de 120 cm et 2 petites roues. Compiler et exécuter.

Q 17.3 Écrire une classe `Cabine` qui a un volume (en m3) et une couleur de type `String`. Ecrivez un constructeur avec paramètres et la méthode `toString()` qui rend une chaîne de caractères donnant le volume et la couleur. Ajouter le modifieur `setCouleur(String couleur)`.

Q 17.4 Ajouter dans la méthode `main` la création d'une cabine bleue.

Q 17.5 Ecrivez la classe `Tracteur` où celui-ci est constitué d'une cabine et de quatre roues, avec un constructeur avec 5 paramètres, d'une méthode `toString()`, d'une méthode `peindre(String couleur)` qui change la couleur de la cabine du tracteur.

Q 17.6 Créez un tracteur `t1` dans la méthode `main`. Il sera formé des 4 roues créées et de la cabine bleue créée précédemment. Affichez-ce tracteur.

Q 17.7 Ajoutez l'instruction `Tracteur t2=t1`; puis modifiez la couleur de la cabine du tracteur `t2`. Quelle est la couleur de la cabine de `t1`? Expliquez 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? Expliquez, puis faites-le.

Quiz 6 – Instanciation

Soient la classe suivante : `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 6.1 La classe `A` contient-elle un constructeur?

QZ 6.2 Combien y-a-t-il d'objets créés?

QZ 6.3 Combien y-a-t-il de références (appelées aussi *handles*) utilisés?

QZ 6.4 Que se passe-t-il si on rajoute l'instruction `a3=null`; ? `a2=null`; puis `a1=null`;

4 Tableaux

Exercice 18 – Base syntaxique

Q 18.1 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 18.2 En supposant une classe `Point` existante, déclarer, remplir et afficher un tableau contenant 10 instances de `Point`.

Q 18.3 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);
```

Exercice 19 – Tableau d'entiers

Q 19.1 Ecrire une classe `TableauInt` qui comporte une variable d'instance `tab` de type tableau de 10 entiers. Cette classe contient deux constructeurs :

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

Q 19.2 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 19.3 Tester ces méthodes au fur et à mesure dans la méthode `main` d'une classe `TestTableau`.

Q 19.4 Ajouter dans la classe `TableauInt` une méthode `boolean egal(TableauInt t)` qui teste si cet objet de type `TableauInt` a les mêmes entiers aux mêmes places que le tableau `t` passé en paramètre.

Exercice 20 – 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]` et l'affichage de l'histogramme sera :

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

Q 20.1 On souhaite écrire une classe `Histo` qui affiche un histogramme des notes. Pour cela, vous définirez :

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

— un constructeur qui prend en paramètre un tableau de notes et qui initialise l'histogramme à partir des notes.

Q 20.2 Ajouter à cette classe, une méthode `afficheHistogrammeTableau()` qui affiche l'ensemble des valeurs du tableau histogramme.

Q 20.3 Ajouter à cette classe, une méthode `afficheHistogramme()` qui affiche le résultat sous forme d'un histogramme, c'est-à-dire en associant à chaque élément du tableau une ligne comprenant autant de `*` que la valeur de cet élément. (comme dans l'exemple de l'énoncé)

Q 20.4 Ecrire une classe `TestHisto` dont la méthode `main` crée un tableau de notes aléatoires (150 étudiants), une instance de `Histo`, puis qui affiche le résultat sous les deux formes proposées.

Exercice 21 – Pile

Soit une pile de `Machin`, écrire une classe `Pile` permettant de gérer une pile de taille variable 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 21.1 Définir la classe `Machin` qui se caractérise par un nom et une valeur (*remarque* : `Machin` pourrait un objet plus sophistiqué, mais là n'est pas l'objet de l'exercice).

Q 21.2 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 21.3 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

Exercice 22 – Représentation mémoire d'objets et de tableaux

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

Q 22.1 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;
```

Q 22.2 Donner les instructions nécessaires pour dupliquer le tableau `tabO`. Etes-vous satisfait du résultat ?

Exercice 23 – Triangle de Pascal (tableau à 2 dimensions)

Le triangle de Pascal est une représentation des coefficients 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 23.1 Ecrire 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 23.2 Ajouter à la classe `TrianglePascal` une méthode `remplirTriangle()` qui calcule les valeurs du triangle de

Pascal et une méthode `toString()` qui retourne une chaîne de caractères représentant le tableau sous la forme d'un triangle.

Q 23.3 Ecrire une classe `TestTrianglePascal` qui crée plusieurs instances de la classe `TrianglePascal` et les affiche.

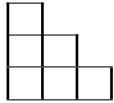
Quizz 7 – Tableaux

QZ 7.1 Créez un tableau `tabD` de `double` à une dimension contenant 8 cases.

QZ 7.2 Comment peut-on obtenir le nombre d'éléments du tableau `tabD` ?

QZ 7.3 Créez un tableau `tabI` d'entiers à deux dimensions de 4 lignes sur 3 colonnes.

QZ 7.4 Créez un tableau `tabTriangle` d'entiers à deux dimensions de 3 lignes mais dont la forme est celle d'un triangle (voir dessin ci-après).



QZ 7.5 On considère la classe `Bouteille` vue dans l'exercice 8 page 5. Créez un tableau de 2 bouteilles, la première bouteille aura un volume de 3 litres et la deuxième de 1,5 litre.

Quizz 8 – Tableaux d'objets

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());
```

5 Variables et méthodes de classes

Exercice 24 – Membres d'instance ou de classe

Q 24.1 On considère la classe Chien. Pour chacune des expressions suivantes, dire si ce sont des membres(*) d'instance (I) ou de classe (C) :

nom	chercherLivreSurChiens()
aboyer()	vidéothèqueSurChiens
siteWebSurChiens	metsPrefere
siteWebDuChien	bibliographieSurChiens
siteWebSPA	dateNaissance
couleurDuPoil	manger()
courir()	regarderDVD()

(*) On appelle membre, une variable (V) *ou* une méthode (M).

Exercice 25 – 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 25.1 Quel est le nom de la variable de classe ? Comment la reconnaît-on ?

Q 25.2 Pourquoi la variable cpt a-t-elle été initialisée ?

Q 25.3 Quel est l'affichage obtenu par l'exécution des lignes 4, 6, 8 et 9 du programme suivant :

```

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

Q 25.4 Peut-on ajouter une instruction à la fin du programme de la question précédente afin d'afficher la valeur de la variable `cpt` sans utiliser d'instance ? Même question pour la variable `num`.

Exercice 26 – 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
11     public static int getCpt(){return cpt;}
12 }

```

Q 26.1 A-t-on commis une *faute de conception* en déclarant plusieurs attributs comme public? (justifier en une phrase)

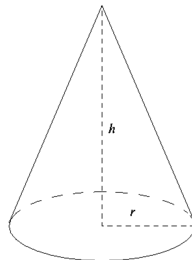
Q 26.2 Les propositions suivantes sont-elles correctes du point de vue syntaxique (compilation)? Donner les affichages pour les lignes correctes.

```

13 // dans la classe Vecteur
14 public int getCpt2(){return cpt;}
15 public static int getId(){return id;}
16 public static String format(Vecteur v){
17     return String.format("[%5.2f, %5.2f]", v.x, v.y);
18 }
19
20
21 // dans le main
22 if(v1.x == v2.x && v1.y == v2.y) System.out.println("v1_egale_v2");
23 if(v1.id == v2.id) System.out.println("les_points_ont_le_meme_identifiant");
24 System.out.println("Compteur: "+v1.getCpt());
25 System.out.println("Compteur(2): "+Vecteur.getCpt());
26 System.out.println("Compteur(3): "+v1.cpt);

```

Exercice 27 – 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 27.1 Ecrivez la classe **Cone** qui contiendra les variables ci-après. Attention : certaines de ces variables sont des variables de classe.

- r : le rayon du cône de type double,
- h : la hauteur du cône de type double,
- PI : une constante de type double dont la valeur est 3.14159,
- $nbCones$: le nombre de cônes créés depuis le début du programme.

Q 27.2 Ajoutez-y les méthodes ci-après. Attention : certaines de ces méthodes sont des méthodes de classe.

- le constructeur dont la signature est : `public Cone(double r, double h)`,
- le constructeur sans paramètre qui initialise le rayon et la hauteur du cône entre 0 et 10 (non compris). Ce constructeur doit appeler le premier constructeur. Aide : utiliser l'instruction `Math.random()` qui retourne un double entre 0 (compris) et 1 (non-compris).
- la méthode `double getVolume()` qui retourne le volume $V = \frac{1}{3}\pi r^2 h$ du cône,
- la méthode `String toString()` qui retourne une chaîne de caractère qui, pour un cône de rayon 5.4 et de hauteur 7.2, a le format : "Cone r=5.4 h=7.2 V=219.854736",
- l'accessor de la variable `nbCones`,

Q 27.3 Ecrire une classe `TestCone` qui contient une méthode `main` dans laquelle vous :

- créez deux instances de la classe `Cone` en appelant une fois chaque constructeur,
- puis, vous afficherez en une seule instruction les 2 cônes,
- enfin, vous afficherez de 3 manières différentes le nombre cônes créés depuis le début du programme.

Exercice 28 – Méthodes de classe

Q 28.1 Ecrire 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()`).

Q 28.2 Pour quelle raison les méthodes `lettre()` et `chaine()` sont-elles des méthodes de classes ?

Q 28.3 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`.

Exercice 29 – Adresse 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... Ecrire la classe `AdresseIP` qui contiendra les variables et méthodes suivantes :

- `tab` : une variable de classe de type tableau de 4 entiers où chaque case correspond à une partie de l'adresse IP. Ce tableau est initialisé à l'adresse IP : 192.168.0.0
- 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.

Exercice 30 – Somme de 2 vecteurs

On veut faire la somme de deux vecteurs dans l'espace, c'est-à-dire créer un nouveau vecteur résultant de la somme des deux vecteurs. Un vecteur est caractérisé par un triplet (x, y, z) de nombres réels, appelés coordonnées. Soient $AB = (x_1, y_1, z_1)$ et $BC = (x_2, y_2, z_2)$ deux vecteurs, alors le vecteur AC a pour coordonnées $(x_1+x_2, y_1+y_2, z_1+z_2)$. 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 30.1 Ajouter à la classe `Vecteur` une méthode d'instance qui fait la somme de deux vecteurs.

Q 30.2 Ajouter à la classe `Vecteur` une méthode de classe qui fait la somme de deux vecteurs.

Q 30.3 Dans une classe `TestVecteur`, écrire une méthode `main` qui initialise deux vecteurs, puis fait la somme des 2 vecteurs en utilisant la méthode d'instance et en utilisant la méthode de classe.

Exercice 31 – 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 31.1 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 31.2 Écrire une méthode de classe `boolean pair(int n)` qui vérifie que `n` est pair.

Q 31.3 Déclarer en variable `static` deux tableaux de `char`, que l'on appellera voyelles et consonnes. Initialiser lors de la déclaration le premier avec les consonnes et le second avec les voyelles.

Q 31.4 Écrire les méthodes `rendVoyelle()` et `rendConsonne()` qui rendent respectivement une voyelle et une consonne de façon aléatoire.

Q 31.5 Écrire une méthode `generNom()` qui rend un nom de longueur aléatoire comprise entre 3 et 6. On générera alternativement une consonne, une voyelle, une consonne, etc...

Q 31.6 Écrire une classe `TestNom` dont la méthode `main` génère et affiche, dans une boucle, une dizaine de nom.

Quizz 9 – 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() {
12        return PI*rayon*rayon;
13    }
14    public static int getNbCercles() {
15        return nbCercles;
16    }
17 }
18 //=====
19 public class TestCercle {
20     public static void main(String [] args) {
21         Cercle c=new Cercle(3);
22         System.out.println(EXPRESSION);
23     }

```

24 }

QZ 9.1 Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

<code>c.PI</code>	<code>c.nbCercles</code>	<code>c.numero</code>
<code>c.rayon</code>	<code>c.surface()</code> ;	<code>c.getNbCercles()</code> ;

QZ 9.2 Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

<code>Cercle.PI</code>	<code>Cercle.nbCercles</code>	<code>Cercle.numero</code>
<code>Cercle.rayon</code>	<code>Cercle.surface()</code> ;	<code>Cercle.getNbCercles()</code> ;

QZ 9.3 Soit la classe `Test2Cercle` suivante :

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

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

6 Héritage et modélisation

Exercice 32 – Personne (héritage)

Soient des personnes caractérisées par leurs nom, prénom, numéro de téléphone et le nombre d'enfants; des étudiants qui sont des personnes qui suivent un cursus; des salariés qui sont des personnes qui reçoivent un salaire.

Q 32.1 Donner la hiérarchie des classes.

Q 32.2 Quels sont les membres (variables et méthodes) hérités ?

Q 32.3 On donne ci-dessous les classes `Personne` et `Etudiant`. Écrire la classe `Salarie` qui hérite de `Personne` et qui possède un constructeur ayant comme paramètre le nom et le salaire.

```

1 public class Personne {
2     protected String nom;
3     protected String prenom;
4     protected String nuTel;
5     protected int nbEnfants;
6     public Personne(String n, String p, String t){
7         nom=n; prenom=p; nuTel=t; nbEnfants=0;
8     }
9     public Personne(String n){ nom=n; }
10    public String getNom() { return nom; }
11    public void ajouterEnfant() { nbEnfants++; }
12 }
13 //=====
14 public class Etudiant extends Personne {
15     private String cursus;
16     public Etudiant(String n, String p, String t, String c) {
17         super(n,p,t); cursus=c;
18     }
19     public boolean estEnL2 () { return cursus.equals("L2"); }
20 }

```

Q 32.4 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 ?

Q 32.5 Trouver 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(); p.ajouterEnfant();
5         double primP=p.prime();
6         System.out.println(p.getNom() + p.getSalaire());
7         Etudiant e = new Etudiant("Ahmed", "L2");
8         e.ajouterEnfant();
9         double primE=e.prime();
10        System.out.println(e.getNom());
11        boolean enL2 = e.estEnL2();
12        Salarie s = new Salarie("Pauline");
13        System.out.println(s.getNom());
14    }
15 }

```

Exercice 33 – Orchestre

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

Q 33.1 Dessiner l'arbre d'héritage du problème.

Q 33.2 Écrire une classe `Instrument` contenant deux variables d'instance de type double pour stocker le poids et le prix de l'instrument, respectivement. Munir la classe d'un constructeur à deux paramètres pour initialiser les variables d'instance, ainsi que de la méthode `toString()`. Quelle est la particularité de la méthode `toString()` d'un point de vue de l'héritage ?

Q 33.3 Écrire les classes `Piano`, `Guitare`, `Trompette`. Ces classes comporteront une méthode `jouer()` qui affichera, par exemple pour `Guitare`, "La guitare joue".

Q 33.4 Un orchestre sera composé d'un tableau d'instruments. Écrire la classe `Orchestre` correspondante, contenant une variable pour stocker le nombre maximal d'instruments, ainsi que le nombre d'instruments courant. Écrire une méthode `ajouterInstrument(Instrument i)` qui ajoute un instrument à l'orchestre lorsque ceci est possible.

Q 33.5 Dessiner le diagramme UML correspondant.

Q 33.6 Ajouter à la classe `Orchestre` une méthode `jouer()` qui fait jouer l'ensemble des instruments le constituant. Quel est le problème dans le code actuel et comment remédier à ce problème ?

Q 33.7 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) ?

Exercice 34 – Botanique

Q 34.1 Dessiner l'arbre d'héritage et dire ce qu'affiche le programme suivant :

```

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

Q 34.2 En tirer des conclusions sur l'héritage et la redéfinition de méthode.

Q 34.3 Qu'affiche le programme suivant :

```

1 Plante p2 = new Arbre();
2 System.out.println(p2);
```

```

3 Plante p3 = new Fleur();
4 System.out.println(p3);
5 Plante p4 = new Marguerite();
6 System.out.println(p4);
7 Plante p5 = new Rose();
8 System.out.println(p5);
9 Plante p6 = new Chene();
10 System.out.println(p6);

```

Exercice 35 – 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 35.1 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.

```

19 Fourmi f1 = new Fourmi("f1");
20 Fourmi f2 = new Ouvriere("ouv1");
21 Ouvriere f3 = new Ouvriere("ouv2");
22 Fourmi f4 = new Reine("majeste1");
23 Ouvriere f5 = new Reine("majeste2");
24 Reine f6 = new Reine("majeste3");
25 Fourmi[] fourmilliere = new Fourmi[100];
26 f2.manger(new Nourriture("sucre"));
27 fourmilliere[0] = f4.engendrer();
28 fourmilliere[1] = f5.engendrer();
29 fourmilliere[2] = f6.engendrer();
30 fourmilliere[3] = ((Reine) f2).engendrer();
31 fourmilliere[4] = ((Reine) f4).engendrer();
32 fourmilliere[5] = ((Reine) f6).engendrer();

```

Q 35.2 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 35.2.1 Est-il possible d'avoir une classe mère construite avec des arguments et une classe fille construite sans argument ?

Q 35.2.2 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_v viande"));
30 r2.manger(new GeleeRoyale());
```


7 Héritage et classe abstraite

Exercice 36 – Figures (héritage, constructeurs, méthode 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 36.1 De quels membres (variables d'instance et méthodes) de **Shape** hérite la classe **Circle** ?

Q 36.2 La compilation de la classe **Circle** échoue, expliquer pourquoi.

Q 36.3 Ajouter une méthode **surface()** à la classe **Circle** et modifier en conséquence la méthode **toString**.

Q 36.4 Créer une classe **Rectangle** qui hérite de **Shape**.

Q 36.5 Donner le code d'un **main** qui instancie un tableau de **Shape**, le remplit avec différents types de forme puis calcule l'aire totale de la figure composite (sans prendre en compte les recouvrements).

Exercice 37 – Ménagerie (tableaux, héritage, constructeur)

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

Q 37.1 Etablir graphiquement la hiérarchie des classes ci-dessus. Déterminer celles qui peuvent être déclarées abstraites ?

Q 37.2 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 ?

Q 37.3 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.

Q 37.4 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.

Q 37.5 Ajouter une méthode `void midi()` qui fait crier tous les animaux de cette ménagerie.

Q 37.6 Ecrire la méthode `vieillirTous()` qui fait vieillir d'un an tous les animaux de cette ménagerie.

Q 37.7 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.

Exercice 38 – Figure2D (Extrait de l'examen de janvier 2010)

On veut écrire les classes correspondant à la hiérarchie suivante (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()` et `toString()`.
- Toutes les instances d'objets de type 2D devront accepter la méthode `perimetre()`.

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

Rappel : dans la classe `Math`, il existe la constante `Math.PI` et la méthode `Math.sqrt()` qui retourne la racine carrée d'un nombre (voir annexe page 50).

Q 38.1 Quelles sont les particularités d'une méthode abstraite et les conséquences pour la classe et les classes dérivées ?

Q 38.2 Donner pour chacune des classes, en utilisant correctement les notions d'héritage et de classe abstraite :

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

Q 38.3 Écrire une classe appelée `TestFigure` qui contient une méthode `main`. Cette méthode créera un objet de chacun des types, et affichera sa surface et son périmètre.

Exercice 39 – 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 39.1 Donner la hiérarchie des classes (avec les **signatures** de méthodes abstraites et concrètes et la signature du constructeur lorsqu'il est nécessaire) à définir pour que ce programme puisse compiler et s'exécuter.

Attention : on veut que la dernière ligne du `main` affiche le calcul à effectuer dans le détail (cf question suivante)

Q 39.2 La hiérarchie de classes proposée définit une expression arithmétique qui peut être évaluée pour donner un résultat (méthode `getVal()`). Donner l'expression arithmétique (avec parenthèses) correspondant à l'objet `d` du programme donné ci-dessus.

Q 39.3 Donner le code des classes nécessaires pour que le programme s'exécute et affiche la formule évaluée et son résultat en ligne 13 (le code des classes `Plus`, `Moins`, `Multiplie` et `Divise` étant très proche, on ne donnera le code que de `Divise`).

Q 39.4 Donner le diagramme de l'état de la mémoire à la fin du programme (ligne 13).

Q 39.5 On souhaite maintenant pouvoir modifier l'attribut d'un objet `Valeur`. On ajoute alors la fonction `void setVal(double v)` à la classe `Valeur` qui fixe à `v` l'attribut de la classe. Soit la ligne de code suivante :

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

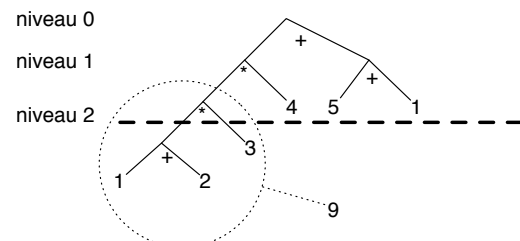
En l'état, le programme ne compile pas. Pourquoi? Donner deux manières de remédier au problème. Discuter brièvement des avantages / inconvénients de ces deux manières de faire.

Q 39.6 Quel problème survient dans l'exécution du programme si l'on remplace la ligne 5 par :

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

Q 39.7 Dire comment y remédier pour que le programme affiche le message d'erreur approprié et évite un arrêt brutal du programme. Décrire brièvement les méthodes à modifier et les éventuelles classes à créer.

Q 39.8 En voyant une expression comme un arbre, on souhaite développer une méthode `simplifie(int profondeur)` permettant la simplification d'une expression à partir d'une profondeur donnée, avec `profondeur` un entier supérieur à 0. Lorsque la profondeur désirée est atteinte, cette simplification consiste à remplacer l'expression concernée par un objet `Valeur` de valeur équivalente. Par exemple, l'objet `Expression` dont la formule est $((((1 + 2) * 3) * 4) + (5 + 1))$ se simplifie en $((9 * 4) + (5 + 1))$ par l'appel de `simplifie(2)`. Donner le code permettant cette simplification.



Exercice 40 – Final : les différentes utilisations

Q 40.1 Questions de cours

Q 40.1.1 A quoi sert un attribut **final** ? Où peut-il être initialisé ? Citer des cas d'utilisation.

Q 40.1.2 Dans quel cas déclarer une méthode comme **final** ?

Q 40.1.3 Dans quel cas déclarer une classe comme **final** ?

Q 40.1.4 Etant donné les usages répertoriés ci-dessus, à quoi sert le mot clé **final** en général ?

Q 40.2 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 + ", " + y + "]";
13    }
14    public void move(double dx, double dy){ x+=dx; y+=dy;}
```

Q 40.2.1 Au niveau des attributs, serait-il intéressant d'ajouter le modifier **final** sur certains champs ? Pourquoi ?

Q 40.2.2 A quelle condition pourrait-on mettre **x** et **y** en mode **final** ? Proposer une solution pour conserver les fonctionnalités de la classe.

Q 40.2.3 Quelles fonctions pourraient être **final** ? Quel serait l'intérêt de la manipulation ?

Q 40.2.4 Quel serait l'intérêt de déclarer la classe **final** ? Cela empêche-t-il tout enrichissement futur ?

Q 40.2.5 Proposer un code pour la classe **PointNomme** (point ayant un attribut **nom**) après avoir déclaré **Point** en **final**.

Quizz 10 – Classe et méthode abstraite

QZ 10.1 Les instructions suivantes sont-elles correctes ? Expliquez.

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

QZ 10.2 Les instructions suivantes sont-elles correctes ? Expliquez chaque erreur.

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

QZ 10.3 Les instructions suivantes sont-elles correctes ? Expliquez et proposez deux solutions.

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

Quizz 11 – Vocabulaire sur l'héritage

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

8 Héritage et liaison dynamique

Exercice 41 – 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 41.1 La méthode main suivante est-elle correcte ? Expliquez les erreurs.

```

1 public class Mammifere { ... }
2 public class Chien extends Mammifere {
3     public void aboyer() { System.out.println("Ouaff"); }
4     public static void main(String[] args) {
5         Chien c1 = new Chien();
6         Mammifere m1 = c1;
7         c1 = (Chien) m1;
8         c1 = m1;
9         Mammifere m2 = new Mammifere();
10        Chien c2 = (Chien) m2;
11    }
12 }
```

Exercice 42 – 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 p4 = new Point(1,1);
11        Point p3 = p1;
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 42.1 Qu'affiche l'exécution du main ?

Q 42.2 Redéfinir la méthode boolean equals(Object ob) de la classe Object dans la classe Point, de façon qu'elle teste l'égalité des coordonnées et non des références. Les instructions de test sont fournies dans la méthode main.

Q 42.3 Que se passe-t-il si dans la méthode main, on rajoute à la suite les instructions suivantes ? Comment résoudre le problème rencontré ?

```

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

Exercice 43 – Véhicules à moteurs

On considère un parc de véhicules. Chacun a un numéro d'identification (attribué automatiquement) 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 43.1 : Construire le graphe hiérarchique des classes décrites ci-dessus.

Q 43.2 Ecrire le code java des classes `Vehicule`, `AMoteur`, et `SansMoteur` avec tous les constructeurs nécessaires et les méthodes `toString()`.

Rappel de cours : Tout constructeur d'une sous-classe a implicitement comme première instruction un appel au constructeur sans paramètre de la super classe (s'il n'appelle pas lui-même un constructeur de la super classe explicitement).

Q 43.3 Ecrire une méthode `rouler(double distance)` qui fait avancer un véhicule. A quel niveau de la hiérarchie faut-il l'écrire ?

Q 43.4 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 ?

Q 43.5 Ecrire la classe `Velo` avec constructeur 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".

Q 43.6 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.

Q 43.7 : 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".

Q 43.8 Peut-on factoriser la déclaration de la méthode `transporter`, et si oui, à quel niveau ?

Q 43.9 On considère le main ci-dessous. Ce programme est-il correct ? Le corriger si nécessaire. Qu'affiche-t-il ?

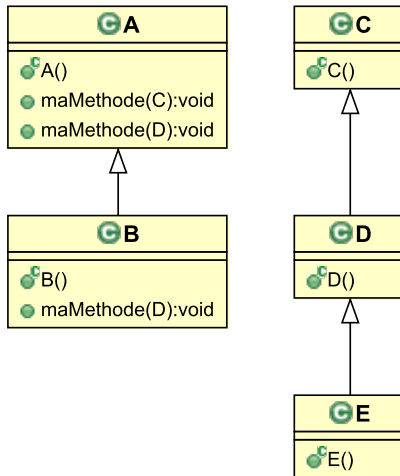
```

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 }
```

Exercice 44 – Sélection de méthode

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

Q 44.1 Pour chaque ligne de code appelant `maMethode`, dire quelle méthode est effectivement appelée.



```

1 A a = new A();
2 B b = new B();
3 A ab = new B();
4
5 C c = new C();
6 D d = new D();
7 E e = new E();
8 C cd = new D();
  
```

```

1 a.maMethode(c);
2 a.maMethode(d);
3 a.maMethode(cd);
4 a.maMethode((D) cd);
5 a.maMethode(e);
6
7 b.maMethode(c);
8 b.maMethode(d);
9 b.maMethode(cd);
10 b.maMethode((D) cd);
11 b.maMethode(e);
12
13 ab.maMethode(c);
14 ab.maMethode(d);
15 ab.maMethode(cd);
16 ab.maMethode((D) cd);
17 ab.maMethode(e);
  
```

Q 44.2 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 :

```

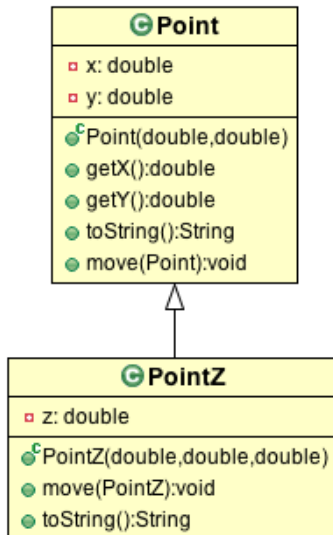
1 a.meth(2);
2 a.meth(2.);
3 b.meth(2);
4 b.meth(2.);
5 ab.meth(2);
6 ab.meth(2.);
  
```

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) ?

Exercice 45 – Redéfinition piègeuse

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 45.1 Pourquoi cette hiérarchie de classe est-elle discutable ?

Q 45.2 Syntaxe : les lignes 15, 19 et 23 sont-elles correctes ? Sinon, proposez des modifications. En ligne 23, peut-on utiliser directement `x` et `y` sans passer par les accesseurs ? Pourquoi ?

Q 45.3 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); // déplacement 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 45.3.1 Qu'est-ce qui s'affiche ?

Q 45.3.2 Est-ce que ça vous semble logique ?

Q 45.3.3 Expliquer en détail ce qui s'est passé au niveau de la compilation et de l'exécution.

Exercice 46 – Interfaces véhicules

Nous souhaitons gérer une grande liste de véhicule à moteur, chacun d'eux ayant comme propriété de pouvoir : **démarrer** et **s'arrêter**. 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 46.1 Donner la hiérarchie d'interface à créer.

Q 46.2 Donner la signature de la classe `Voiture` et les méthodes à coder impérativement.

Q 46.3 Donner la signature de la classe `Hydravion` et les méthodes à coder impérativement.

Exercice 47 – Interface réversible

Une interface correspond à une propriété. Nous envisageons dans cet exercice la propriété de réversibilité. Pour une chaîne 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.

Q 47.1 Donner le code de l'Interface `Reversible`

Q 47.2 Donner le code de la classe `StringReversible`

Q 47.3 Nous souhaitons maintenant créer une structure de type `ArrayList` réversible. Donner le code étendant l'`ArrayList<Object>`, ajoutant les méthodes nécessaires (dont `toString()` et surchargeant la méthode `get`).

NB : ajouter un attribut booléen indiquant si la structure est renversée ou pas.

Q 47.4 Ajouter quelques lignes de code pour rendre la réversibilité récursive quand c'est possible dans la structure précédente. Par exemple, quand la liste contient des `StringReversible`, nous souhaitons renverser la liste ET renverser les éléments de la liste si c'est possible.

Q 47.5 (Option) Proposer une seconde implémentation de la structure de données récursive basée sur la composition et non plus sur l'héritage (attribut `ArrayList` au lieu de `extends ArrayList`)

Exercice 48 – 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 48.1 Indiquer les modifications à effectuer dans la classe `Vecteur` pour utiliser `Comparable`

Q 48.2 Donner le code d'un main effectuant le tri d'une liste de `Vecteur` générée aléatoirement par rapport à leurs

normes.

Q 48.3 Donner la procédure et le code pour utiliser un `Comparator`. Quel est l'avantage de cette approche ?

Quizz 12 – 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 :

```
1 Animal a1=new Animal();  
2 Poisson p1=new Poisson();  
3 Cheval c1=new Cheval();  
4 Zoo z1=new Zoo();
```

QZ 12.1 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.

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

QZ 12.2 Que retournent les instructions suivantes ?

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

QZ 12.3 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.

- `Animal a2=p1;`
- `Animal a3=(Animal)p1;`
- `Poisson p2=a1;`
- `Poisson p3=(Poisson)a1;`

9 Package, documentation

Exercice 49 – Documentation Java, package

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

Q 49.2 Recherchez la classe `ArrayList`. D'après la documentation, combien a-t-elle de champs? Combien a-t-elle de constructeurs? Combien environ a-t-elle de méthodes? De quelles classes hérite-t-elle? A quel package appartient cette classe `ArrayList`?

Q 49.3 Il est possible de créer une documentation pour les classes que vous créez. Pour cela, il faut utiliser la commande `javadoc`. Récupérez sur le site web de l'UE le fichier `Clavier.java`. Placez ce fichier dans un répertoire vide, puis tapez la commande : `javadoc Clavier.java`, puis : `firefox index.html` Comparez les commentaires du fichier `Clavier.java` et la page web affichée.

Exercice 50 – Documentation Java, package

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

Q 50.2 Créez un objet de la classe A dans la classe B. Compilez les fichiers. Quelle instruction faut-il ajouter?

Exercice 51 – 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 51.1 Donner la déclaration des classes B, C, D et E, et faire un schéma.

Q 51.2 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
<code>champPrive</code>					
<code>champSansModifieur</code>					
<code>champProtege</code>					
<code>champPublic</code>					

Q 51.3 Si la classe A n'était pas déclarée `public`, est-ce que cela change la visibilité des variables ?

Exercice 52 – Compagnie de chemin de fer (ArrayList, instanceof)

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

Q 52.1 Dessiner la hiérarchie des classes `Train`, `ElemTrain`, `Motrice`, `Wagon`, `WVoyageur` et `WMarchandise`. **Q 52.2**

Ecrire les classes `ElemTrain` (abstraite), `Wagon` (abstraite), `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 ».

Q 52.3 Un `Train` possède une motrice et une suite de wagons (on gèrera cette suite obligatoirement par la classe `ArrayList` (voir la documentation page 51)). 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.

Q 52.4 Ajouter une méthode `void ajoute(Wagon w)` qui ajoute un wagon au vecteur de wagons du train.

Q 52.5 Redéfinir la méthode `public String toString()` qui retourne la composition de ce train.

Q 52.6 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 52.7 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.

Quizz 13 – 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.

QZ 13.1 Expliquez la ligne 1.

QZ 13.2 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 51). 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.

QZ 13.3 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`.

QZ 13.4 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`.

10 Exceptions

Rappel : Les exceptions sont un mécanisme de gestion des erreurs. Il existe 3 catégories d'exceptions : les exceptions qui étendent la classe `Exception` qui doivent obligatoirement être gérées par le programme, les exceptions qui étendent la classe `RuntimeException` qui peuvent être gérées par le programme, et les erreurs critiques qui étendent la classe `Error` qui ne sont pas censées être gérées en temps normal.

Toute instance de la classe `Exception` doit obligatoirement être capturée ou bien signalée comme étant propagée par toute méthode susceptible de la lever.

- Pour capturer une exception :

```

1 try {
2     instructions qui peuvent lever une exception
3 } catch (MonException me) {
4     System.out.println(me.toString());
5 } catch (AutreException ae) {
6     System.out.println(ae.getMessage());
7 } finally {
8     instructions toujours exécutées
9 }
```

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

Exercice 53 – Capture dans le main d'une exception prédéfinie (try catch)

Q 53.1 Soit classe `TestAttrapePas0` ci-dessous. Que se passe-t-il lors de l'exécution ?

```

1 public class TestAttrapePas0{
2     public static void main(String [] args){
3         int [] tab= {1,2,3,4,5};
4         for (int i=0; i<15; i++)
5             System.out.print(tab[i] + " ");
6     }
7 }
```

Q 53.2 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 a la position 5" quand l'exception se produit.

Exercice 54 – Try, catch, throw, throws, création d'une exception utilisateur

Q 54.1 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 transforme 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.

Q 54.2 Que donnent les exécutions suivantes :

1. `javaTestAttrapePas1 10 12 16 18`
2. `javaTestAttrapePas1 12 1j 10 13 15`
3. `javaTestAttrapePas1`

Q 54.3 Dans une classe `TestAttrape2`, réécrire 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 ?

Q 54.4 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`.

Q 54.5 Que donne l'exécution de la commande `javaTestAttrape3 mm reg 6r c5 mm` ?

Q 54.6 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 `javaTestIntervalle -10 -3 45 -78 -6 21` ?

Exercice 55 – 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 55.1 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.

Q 55.2 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`.

Q 55.3 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`.

Q 55.4 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.

Q 55.5 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`.

Q 55.6 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.

Q 55.7 Créer un jeu de tests pour ce programme, en réfléchissant aux différents cas possibles et les tester dans le `main`.

Exercice 56 – throw, throws, finally

Q 56.1 Donnez l’affichage produit par le programme ci-après. Expliquez les résultats.

```

1 public class MonException extends Exception {
2     public MonException(String s) {
3         super(s);
4         System.out.println("\nMonException: constructeur");
5     }
6 }
7
8 public class TestFinally {
9     /** Exception deleguee a la methode appelante (ici main). */
10    public static void test1() throws MonException {
11        if (true) throw new MonException("lancee dans test1");
12        System.out.println("test1: fin de la methode");
13    }
14
15    /** Exception capturee (et pas deleguee) dans la methode test2 */
16    public static void test2() {
17        try {
18            if (true) throw new MonException("lancee dans test2");
19        } catch (MonException e) {
20            System.out.println("test2: capture de l'exception: "+e);
21        }
22        System.out.println("test2: fin de la methode");
23    }
24
25    /** Exception capturee (et pas deleguee) dans la methode test3 avec finally */
26    public static void test3() {
27        try {
28            if (true) throw new MonException("lancee dans test3");
29        } catch (MonException e) {
30            System.out.println("test3: capture de l'exception: "+e);
31        } finally {
32            System.out.println("test3: finally est effectue");
33        }
34        System.out.println("test3: fin de la methode");
35    }
36
37    /** Exception deleguee a la methode appelante (ici main) avec finally */
38    public static void test4() throws MonException {
39        try {
40            if (true)
41                throw new MonException("lancee dans test4");
42        } finally {
43            System.out.println("test4: finally est effectue");
44        }
45        System.out.println("test4: fin de la methode");
46    }
47
48    /** Meme cas que le test4, mais ici l'exception n'est pas levee */
49    public static void test5() throws MonException {
50        try {
51            if (false) throw new MonException("lancee dans test5");
52        } finally {
53            System.out.println("test5: finally est effectue");
54        }
55        System.out.println("test5: fin de la methode");
56    }

```

```

57
58 public static void main(String [] args){
59     try {
60         test1();
61     } catch (MonException e) {
62         System.out.println("main:_test1:_capture_de_l'exception_"+e);
63     }
64     test2();
65     test3();
66     try {
67         test4();
68     } catch (MonException e) {
69         System.out.println("main:_test4:_capture_de_l'exception_"+e);
70     }
71     System.out.println();
72     try {
73         test5();
74     } catch (MonException e) {
75         System.out.println("main:_test5:_capture_de_l'exception_"+e);
76     }
77     System.out.println("Fin_du_programme");
78 }
79 }

```

Exercice 57 – 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 57.1 Définir une classe **MonTableau** qui possède les variables **tab** (tableau d'entiers) et **lgReelle** (entier) donnant le nombre de cases de **tab** réellement utilisées dans le tableau. Au départ, **lgReelle** vaut 0. Ecrire un constructeur prenant en paramètre la taille du tableau, et une méthode **ajouter(int n)** qui ajoute la valeur *n* à la suite du tableau sans vérifier s'il reste de la place.

Q 57.2 Ecrire la méthode **main** qui crée un objet **MonTableau** de 3 cases et y ajoute 10 entiers. Exécutez le programme. Que se passe-t-il ?

Q 57.3 Capturer dans la méthode **main** l'exception précédemment levée, et afficher le texte "**Dépassement des bornes a la position 3**" en utilisant la méthode **getMessage()** de l'exception levée.

Q 57.4 Définir un nouveau type d'exception appelée **TabPleinException**.

Q 57.5 Modifier la méthode **ajouter** pour lever cette exception quand le tableau est plein. Capturer cette exception dans la méthode **main**. Que retournent les méthodes **getMessage()** et **toString()** de cette exception ?

Exercice 58 – 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 58.1 Écrire la classe **Etudiant** correspondant à la description ci-dessus avec un constructeur à un paramètre, le nom. La méthode **toString()** rend le nom de l'étudiant suivi de ses notes.

Q 58.2 Ajouter la méthode **void entrerNote(int note)** qui entre la note dans la liste des notes de cet étudiant. Elle lèvera une exception **TabNotesPleinException** (à définir) dans le cas où le tableau de notes de cet étudiant serait plein. Cette exception sera capturée dans le **main**.

Q 58.3 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 représente pas un entier.

Ecrire le code du `main` qui récupère les données et affiche pour chacune "c'est une note" ou bien "c'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 c'est un nom,
12 c'est une note, 13 c'est une note, 7 c'est une note, 15 c'est une note,
Tom c'est un nom,
Arthur c'est un nom,
9 c'est une note, 12 c'est une note, 15 c'est une note, 0 c'est une note, 13 c'est une note, 12 c'est une note
```

Q 58.4 On souhaite gérer dans la classe `Etudiant` une liste au sens `ArrayList` d'étudiants. Une liste d'étudiants ne dépend pas d'un étudiant en particulier. Qu'en concluez-vous sur le type de variables que doit être la liste d'étudiants ?

Ajouter les instructions nécessaires dans la classe `Etudiant`. **Q 58.5** Enrichir/modifier le code précédent pour qu'il traite les données de la façon suivante :

- si c'est une chaîne de caractères, il crée une nouvelle instance d'étudiant portant ce nom.
- si c'est une note, il ajoute cette note à la liste des notes de l'étudiant créé précédemment, puis affiche la liste des étudiants. On pensera à traiter les différentes exceptions levées (on rappelle qu'un étudiant a au maximum 5 notes).

Voici une exécution possible :

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

11 Manipulation de flux entrée / sortie

La classe File

Le paquetage `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 la classe `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 les méthodes suivantes :

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

Exercice 59 – 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 59.1 Dire ce qu'affiche l'exécution suivante : `java TestFile "./2i002/TME11/Files/fichier1.txt"`

- Si le répertoire `./2i002/TME11/Files` existe
- Si le répertoire `./2i002/TME11/Files` n'existe pas

Q 59.2 Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier

Q 59.3 Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la

classe `File`

Q 59.4 Écrire une méthode `ls(File f)` permettant d'afficher tous les noms de fichiers contenus dans le répertoire passé en paramètre (ne pas afficher les répertoires)

Q 59.5 Écrire une méthode `lsRecuratif(File f)` permettant d'afficher tous les noms de fichiers contenus dans l'arborescence prenant sa racine au niveau du répertoire passé en paramètre (ne pas afficher les répertoires)

Les flux

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

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

Deux catégories de flux :

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

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

- la source pour les flux entrants (`FileInputStream`, `FileReader`, `InputStreamReader`, `StringReader`...)
- la destination pour les flux sortants (`FileOutputStream`, `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 caractères du tableau dans le flux.
- `void write(char[] cbuf, int debut, int nb)` Écrit `nb` des caractères du tableau dans le flux en commençant par celui d'index `debut`.
- `void write(String s)` Écrit la chaîne de caractères dans le flux.

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

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

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

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

Exercice 60 – 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 60.1 Écrire une méthode `String saisie()` qui demande à l'utilisateur de saisir une ligne de texte tant que la ligne entrée par l'utilisateur est différente de la chaîne "`_fin_`". Cette méthode retourne une chaîne de caractères contenant la concaténation de toutes les lignes saisies. Proposez une première solution utilisant des concaténations de `String`. Puis proposez une deuxième solution utilisant un seul objet `StringWriter`.

Q 60.2 Écrire une méthode `affiche(String fichier)` affichant le contenu du fichier dont le nom est passé en paramètre.

Q 60.3 Écrire une méthode `afficheLignes(String fichier)` affichant, en numérotant les lignes, le contenu du fichier passé en paramètre.

Q 60.4 Écrire une méthode `ecrireTexte(String fichier)` permettant de créer un nouveau fichier contenant un texte saisi par l'utilisateur.

Q 60.5 Écrire une méthode `ajouteTexte(String fichier)` permettant d'ajouter, en fin de fichier passé en paramètre, du texte saisi par l'utilisateur.

Q 60.6 Écrire une méthode `replace(int num, String newLigne, String fichier)` permettant de remplacer, dans le fichier passé en paramètre, la ligne numéro `num` par la nouvelle ligne `newLigne`.

Q 60.7 Écrire un programme proposant à l'utilisateur un menu lui permettant d'éditer un fichier dont le chemin est passé en argument. Exemple :

Fichier "Texte.txt"

1. Ajouter texte
2. Afficher fichier
3. Remplacer ligne
4. Quitter

Exercice 61 – Copie de fichiers binaires

Q 61.1 Écrire un programme permettant de copier un fichier binaire passé en premier argument sous le nom passé en second.

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 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 62 – Mise en mémoire tampon

Q 62.1 Sachant que la construction d'un `BufferedReader` se fait en passant un flux `Reader` en paramètre, écrivez l'ouverture d'un flux de lecture avec utilisation de la mémoire tampon sur un fichier "text.txt" du répertoire courant.

Q 62.2 Écrire une méthode `afficheLignesFichier(String fichier)` qui affiche ligne après ligne le texte du fichier dont le chemin est passé en paramètre.

Q 62.3 Sachant qu'il est également recommandé par soucis d'efficacité d'encapsuler tout flux en écriture dans un objet `BufferedWriter` (resp. `BufferedStream` pour l'écriture d'octets), écrire une classe `Ecrivain` ouvrant un flux en écriture sur un fichier à sa construction et disposant des méthodes données ci-dessus pour la classe `PrintWriter` (sauf méthode `flush()`). On pourra donner une version avec héritage et une version sans.

Exercice 63 – 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 (telle que vous devez l'avoir) prenant racine en un répertoire 2i002. 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 2i002 contenant les prenom, 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 63.1 Écrire un programme `RenduTMEProducer` prenant en argument le chemin du répertoire de TME concerné par le compte rendu et produisant en racine de ce répertoire un fichier "compteRenduTME.txt" de la forme de celui que vous avez l'habitude de rendre en fin de TME.

Entrée / Sortie standard

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

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

Pour la sortie, aucun problème, on sait déjà 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 64 – Classe Clavier

Q 64.1 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)`

Quizz 14 – String, classe immutable

QZ 14.1 Combien d'objets sont créés dans les instructions ci-après ?

String a="Bonjour"; a=a+" tous le monde";

QZ 14.2 Donnez une solution équivalente en utilisant un `StringWriter`.

Aide mémoire

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

```
1 class MaClasse [extends ClasseMere] {
2     private int maVariable;      // Variables (appelees aussi champs ou attributs)
3     private static int maVariableStatique=0; // Variables de classe
4     private static final int CONSTANCE=3.1415; // Constantes
5     public MaClasse () { // Constructeurs
6         ....
7     }
8     public int getMaVariable() { // Accesseurs (methodes get)
9         return maVariable;
10    }
11    public void setMaVariable(int v) { // Modificateurs (methodes set)
12        maVariable=v;
13    }
14    public String toString() {
15        ....
16        return chaine;
17    }
18    public void methode() {      // Autres methodes
19        ....
20    }
21 }
```

Commentaires

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

Divers

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

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 constant1 : 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 \geq 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 \leq 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 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 `<...>`.

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

La classe Clavier (non standard)

Pour une utilisation simplifiée des entrées/sorties, plusieurs méthodes ont été regroupées dans la classe `Clavier` (code disponible sur le site de l'UE). Récupérez la classe dans votre répertoire et compilez-la. Toutes les méthodes de cette classe sont des méthodes de classe (déclarées `static`).

Exemple : `int i = Clavier.saisirEntier("Taper un chiffre : ");`

<code>static int saisirEntier(String m)</code>	Affichage du message m et saisie d'un int rendu en valeur
<code>static double saisirDouble(String m)</code>	Affichage du message m et saisie d'un double rendu en valeur
<code>static String saisirLigne(String m)</code>	Affichage du message m et saisie d'une ligne de caractères rendu en valeur
<code>static void dormir(n)</code>	Arrêt de l'exécution du programme pendant n milli-secondes

Environnement Linux

Création et gestion de répertoires sous Linux

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

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 &` (voir le mode d'emploi d'emacs à la fin de l'annexe).

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.
- Pour imprimer un fichier sur l'imprimante : `a2ps nomfichier`
- Si l'icône de la clé USB ne s'affiche pas, taper `mount /mnt/media/usbkey` pour monter la clé. Pour démonter la clé : `umount /mnt/media/usbkey`.

Fin de session

Ne pas partir de la salle de TME sans quitter la session Linux : choisir menu "Bureau", option "Clôture la session...". NE PAS ETEINDRE LE PC.

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.

Emacs

L'éditeur emacs doit absolument être configuré pour pouvoir au minimum :

- coloriser le programme suivant la syntaxe : menu Options/syntax highlighting/ds ce buffer
- afficher le n° de ligne du curseur souris : menu Options/display/no de ligne
- signaler par une couleur différente la parenthèse correspondant à la parenthèse sur laquelle se trouve le curseur de la souris. Cette facilité est indispensable pour pouvoir détecter les parenthèses en trop, en moins, ce type d'erreur étant très difficile à déboguer : menu Options/display/parenhighlighting

Un **buffer** est une zone de travail en mémoire vive, qui est affichée à l'écran et dans laquelle vous travaillez (c'est l'équivalent de "fenêtre" lorsque vous travaillez sous Word). Lorsqu'on ouvre emacs, on se trouve dans le buffer nommé "scratch" (brouillon). **Ne pas travailler dans ce buffer** mais en ouvrir un nouveau (menu "ouvrir") pour travailler. L'éditeur de textes Emacs est un éditeur multibuffers, cela veut dire que vous pouvez travailler sur plusieurs fichiers à la fois (N'ouvrez pas un nouvel Emacs à chaque fichier!!!) en basculant de l'un à l'autre par le menu "Buffers". Pour fermer un fichier, choisissez l'item "fermer buffer courant" ("close current buffer").

La ligne tout en bas de la fenêtre s'appelle le minibuffer, consultez-le souvent.

Afin de disposer d'un environnement dédié à java, il faut que les noms de fichiers se terminent par .java. Il apparaît alors des items de menu "Java" et "JDE" qui permettent de compiler et d'exécuter sous Emacs (usage conseillé).

Création et sauvegarde de programmes

- Pour créer un nouveau fichier : fichier/ouvrir et entrez un nom pour votre fichier en bas de la fenêtre (ce nom ne doit pas déjà exister).
- Pour ouvrir un fichier existant : fichier/ouvrir et entrez le nom du fichier en bas de la fenêtre.
- Pour sauvegarder le buffer courant : fichier/enregistrer [sous]. Rappelons qu'il faut sauvegarder régulièrement un fichier afin d'éviter de perdre tout son travail au premier incident.

Saisie de programmes

Il suffit de taper le texte du programme dans le buffer. On dispose des flèches du clavier et de la souris pour se déplacer et se positionner n'importe où.

- Pour copier : sélectionner une zone avec le bouton gauche de la souris -> la zone sélectionnée se trouve dans le presse-papier.
- Pour coller : positionner la souris là où vous voulez coller, puis clic avec le bouton du milieu.
- Pour couper : Sélectionner la zone à couper, puis CTRL W.
Remarque : ne quitter l'éditeur Emacs qu'en fin de TP en cliquant dans la croix en haut à droite de la fenêtre. On peut aussi sélectionner l'item "Exit" via le menu fichier/exit emacs ou encore taper CTRL X CTRL C après avoir sauvé le fichier.

Compilation et exécution de programme

- La compilation et l'exécution se font en cliquant dans les menus "JDE/Compile" et "JDE/RunApp". Il s'ouvre alors une sous fenêtre "shell" dans laquelle sont affichés les messages du compilateur et de l'exécution.
- Cliquer sur l'erreur de compilation (qui s'affiche en vert) avec le bouton du milieu, le curseur se positionne alors automatiquement sur l'erreur dans le buffer source.
- On peut aussi lire directement le numéro de la ligne courante dans la ligne noire en bas de l'éditeur si l'éditeur est bien configuré.
- Pour ne conserver que la fenêtre courante et supprimer les autres, sélectionner unsplit windows dans le menu fichier.
- S'il apparaît des messages d'erreur indiquant une absence de bibliothèques, vérifier le JDE/Options/Project/Jde Global Classpath.
- L'interruption d'un programme se fait via le menu Signals/KILL lorsque le curseur se situe dans la sous fenêtre "shell".