2i002 (ex LI230) : Éléments de programmation par objets avec JAVA

Licence de Sciences et Technologies Mention Informatique

VERSION POUR LES ENSEIGNANTS

Fascicule de TD/TME

Année 2014-2015

Table des matières

1 TD : classe : définition, syntaxe

On utilisera indifféremment les termes : variable d'instance, champ ou attribut.
Sauf cas particulier, les variables sont déclarées `private` et les méthodes `public`.
Utiliser une classe par fichier, une classe à part (et donc un fichier) pour le `main`.
Faire attention à l'indentation des programmes.

Site de l'UE

http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2014/ue/objin-2014oct/

Documentation Java

http://www-ari.ufr-info-p6.jussieu.fr/OUTILS/documentation/doc/Java/jdk1.6/docs/api

Objectifs

- Syntaxe Java
- Etude des types primitifs
- Instructions de base : if, for, while...
- Evaluation d'expressions
- Classe simple
- Création d'objets

Exercices

La présentation des types simples et des priorités des opérateurs est en ANNEXE page 193 à la fin du poly.
Propositions : En TD, faire les exercices 1,2, 3, puis choisir en fonction du niveau des étudiants.

Exercice 1 – 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 191.

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

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

Exercice 2 – Planète

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

```
1 public class Planete {
2         private String nom;
3         private double rayon; // en kilometre
4
5         public Planete(String n,double r) {
6                 nom=n;
7                 rayon=r;
8         }
9
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 ?

(a) Les variables d'instance sont : nom et rayon (ligne 2 et 3)
On les reconnaît car ces variables ne sont pas déclarées dans une méthode
(b) ligne 5 : n et r sont des paramètres de méthode
(c) ligne 11 : s est une variable locale à la méthode toString() quand la méthode toString() se termine, cette variable n'existe plus. Remarque : la variable s est inutile.

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

Le constructeur commence à la ligne 5 et termine à la ligne 8.
On reconnaît les constructeurs car ils :
– portent le même nom que la classe,
– seules méthodes qui par convention commencent par une majuscule
– n'ont pas de valeur de retour.
Le rôle des constructeurs est :
– d'initialiser les variables d'instance
– et d'effectuer les instructions nécessaires pour la création d'un objet de cette classe
Les constructeurs sont appelés quand on crée (mot-clé new) un objet de cette classe

Q 2.3 Quelles sont les méthodes de cette classe ?

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

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

Elle doit contenir la méthode main.

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

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

Une seule classe par fichier. Le nom du fichier doit être `NomDeLaClasse.java`, c-à-d. : `SystemeSolaire.java`
`javac Planete.java` => création d'un fichier `Planete.class`
`javac SystemeSolaire.java` => création d'un fichier `SystemeSolaire.class`
`java SystemeSolaire` (SANS .java) (NON PAS : `java Planete`, car la méthode main se trouve dans la classe SystemeSolaire, et non pas dans Planete)
On peut aussi compiler par : `javac *.java` ou bien : `javac Planete.java SystemeSolaire.java`

Exercice 3 – Se présenter

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

Discuter ici du choix du type pour chaque variable (char, int, float, double, boolean et String) (voir tableau de codage des types simples page 192)

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

```
public class Personne {
        private String nom;
        private int age;

        public Personne(String n, int a) {
                nom=n;
                age=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.

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

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

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()` ?

```
public String toString(){
        return "Je␣m'appelle␣"+nom+",␣j'ai␣"+age+"␣ans";
}

public void sePresenter(){
        System.out.println("Bonjour␣!␣"+toString());
        //System.out.println("Bonjour ! Je m'appelle "+nom+", j'ai "+age+" ans");
}
```

Dans la classe Presentation :

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

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

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

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

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

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

Q 3.6 Ajouter dans la classe `Personne`, la méthode `vieillir()` qui ajoute un an à la personne. Dans la classe `Presentation`, faites vieillir Paul de 20 ans (utiliser une boucle `for`), et Pierre de 10 ans (utiliser une boucle `while`), puis faites se présenter Paul et Pierre. Aide : voir la syntaxe des boucles page 192

```
1 public void vieillir() {
2         age=age+1;     // ou age++;     ou age+=1;
3 }
```

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

```
1 for(int i=0;i<20;i++)
2         p1.vieillir();
3 int i=0;  // pas d'ambiguité avec le i du for qui n'est visible que dans le for
4
5 while(i<10) {
6         p2.vieillir();
7         i++;
8 }
9 p1.sePresenter();  // 45 ans
10 p2.sePresenter();  // 47 ans
```

Exercice 4 – Evaluation d'expression

Qu'affiche le morceau de programme suivant ?

```
1 boolean b;
2 char c = 'd';
3 int x = 8;
4 b = false;
5
6 if (!(b && x >= 0) || c >= 'a' && c <= 'z' && x > 20) {
7         if (x == 99 && c == 'N') System.out.println("Bonjour");
8         else System.out.println("Good Morning");
9 } else {
10         if (x == 0 ||  c == 'B') System.out.println("Ciao");
11         else System.out.println("Kenavo");
12 }
```

Ce programme affiche "Good Morning"
En utilisant une évaluation abrégée, on a :
- pour le 1er if : non (faux et vrai) ou (vrai et vrai et faux), donc vrai
  (priorité des opérateurs : d'abord les >= et <=, puis les && et enfin le ||)
- pour le 2eme if : (faux et faux), donc faux

Exercice 5 – Evaluation et affichage

L'affichage dans une fenêtre Console se fait grâce à l'appel de la méthode `println` :
`System.out.println(argument);`
Cette méthode admet un seul argument de type *chaine de caractères*. Si ce n'est pas le cas, cet argument est évalué et converti en chaîne de caractères avant affichage. Aide : voir les informations au début de l'exercice 7.
Qu'affichent les instructions suivantes ? (Il peut y avoir des erreurs)

```
System.out.println(34);
```

34

```
System.out.println(1+2);
```

3 Evaluation (addition)

```
System.out.println("1"+"2");
```

12 Evaluation (concatenation)

```
System.out.println(1+1.0);
```

2.0 Evaluation (conversion en double)

```
System.out.println('1'+"2");
```

12 Evaluation (concatenation)

```
System.out.println(1+"2");
```

12 Evaluation (concatenation)

```
System.out.println('1'+'2');
```

99 char convertis en int('1' code 49 et '2' code 50)

```
System.out.println('1'+'c'+"bon");
```

148bon addition ('1' code 49 et 'c' code 99), puis concaténation

```
System.out.println(true);
```

true

```
System.out.println(true && false);
```

false

```
System.out.println(false && 1 == 0);
```

false

```
System.out.println(false || 1 == 0);
```

false

```
System.out.println(1+1 == 2);
```

true

```
System.out.println('a' < 'B');
```

false Les majuscules sont avant les minuscules (code 'a' : 97 ; code 'B' : 66)

```
System.out.println("true" > "false");
```

ERREUR l'opérateur > ne peut pas s'appliquer sur les chaînes de caractères

```
System.out.println(true + false);
```

ERREUR l'opérateur + ne peut pas s'appliquer sur des booléens

```
System.out.println("true" + "false");
```

truefalse

```
System.out.println(true > false);
```

ERREUR l'opérateur > ne peut pas s'appliquer sur des booléens

Exercice 6 – Alphabet

Q 6.1 En utilisant une boucle `for` :

Q 6.1.1 Afficher les chiffres de 0 à 9, ainsi que leur code ASCII.

Q 6.1.2 Afficher les lettres de l'alphabet de 'A' à 'Z', ainsi que leur code ASCII.

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

Q 6.2 Recommencer en utilisant une boucle while.

```
1 public class Alphabet {
2         // Manipulation de char Unicode, affichage de caractères
3         public static void main (String[] args) {
4         char c = '0';
5         // Affichage des chiffres
6         while (c <= '9'){
7                 System.out.print(c);
8                 c++;
9         }
```

```
10          System.out.println();
11          System.out.println("-----------------");
12
13          // Affichage des codes des chiffres
14          int i = '0';
15          while (i <= '9') {
16                  System.out.print(i + "␣");
17                  i++;
18          }
19          System.out.println();
20          System.out.println("-----------------");
21
22          // Affichage des lettres
23          char c='A';
24          while (c <= 'Z') {
25                  System.out.print(c);
26                  c++;
27          }
28
29          System.out.println();
30          System.out.println("-----------------");
31
32 }
```

Exercice 7 – Constantes et conversion de types

On peut sauter les questions 3 et 4, si on ne veut pas passer de temps sur les problèmes de codage.

- Une constante numérique entière est codée dans un `int`.
- Une constante numérique réelle est codée dans un `double`.
- Les caractères sont codés en Unicode sur 16 bits. Les majuscules ont un code (en base 10) compris entre 65 ('A') et 90 ('Z') ; les minuscules entre 97 ('a') et 122 ('z') ; les chiffres entre 48 ('0') et 58 ('9')
- Dans une expression numérique, un char est converti en entier non négatif.

Pour aborder quelques problèmes liés à la syntaxe et aux types primitifs, on considère la classe suivante :

```
1 public class TestExp {
2         public static void main(String[] args){
3                 int x = 0;
4                 double y = 1;
5                 double z = 2;
6                 if (x < y < z)
7                         System.out.println("OK");
8                 else System.out.println("FAUX");
9
10        }
11 }
```

Faire remarquer que les chaînes de caractères sont entre "..." ;

Q 7.1 Détailler la conversion générée par le compilateur lors de l'instruction `double y = 1`.

1 est une constante entière codée dans un int. Cet int est converti en double, la valeur résultante est affectée à y.

Q 7.2 Montrer que l'expression `x < y < z` est incorrecte (erreur de compilation). La modifier pour qu'il n'ait plus d'erreur.

(x < y) est une comparaison entre deux nombres. Le résultat est de nature booléenne c'est-à-dire égal à true ou false. Un booléen ne peut être inférieur ou supérieur à un nombre. Ce sont deux entités qui ne sont pas comparables.
if (x < y && y < z)
Pas besoin de parenthèses en plus (Cf. priorité des opérateurs)

Q 7.3 On écrit maintenant : `char x = 0;`. Comment s'analyse cette instruction ?
Quelle est la valeur binaire codée dans `x` ?

0 est une constante entière codée dans un int. x est une variable de type char codée sur 16 bits. On peut recopier un nombre entier dans un char, pourvu que la valeur puisse tenir sur 16 bits, c'est-à-dire soit inférieure ou égale à 65 535 (2^{16}-1). La valeur binaire prise par x est alors la valeur binaire du nombre situé à droite du symbole d'affectation. En l'occurrence zéro.
On peut dire que le code de x est zéro (tous les bits à 0). Le type char peut être utilisé dans les calculs.

Q 7.4 On écrit maintenant : `char x = '0'`. Analyser cette instruction. Quelle est la valeur décimale de `x` ? Exprimer cette valeur en binaire sur 16 bits. '

0' est une constante de type char codée selon le code Unicode sur 16 bits. Le code est recopié dans x. c'est-à-dire 48 en base 10, 30 en hexadécimal et 0000 0000 0011 0000 en binaire.

Q 7.5 Le code de 'A' est 65 (base 10) en Unicode, comme en code ASCII, mais représenté sur 16 bits, le code de 'B' est 66... Donner plusieurs solutions pour compléter le programme suivant :

```
1 public class TestMaj {
2         // cette classe teste si une lettre est en majuscule ou non
3         public static void main(String[] args){
4                 char x = 'K';
5                 if (.....)
6                         System.out.println("C'est une majuscule");
7                 else System.out.println("Ce n'est pas une majuscule");
8         }
9 }
```

```
1 (x >='A' && x <= 'Z')
2
3 (x>= 65 && x <= 65+25)
```

Q 7.6 Définir une variable booléenne `maj` vraie si la variable `x` de type `char` est une majuscule. Réécrire l'instruction `if`.

```
boolean maj = x >='A' && x <= 'Z';
if (maj) ....
```

Q 7.7 Deux classes de test ont été définies précédemment. Quels noms donner aux fichiers de ces deux classes ? Comment compiler ces fichiers ? Comment exécuter ce programme ?

Nom classe = nom fichier
javac ****.java ou javac *.java
java ***** (attention, il ne faut pas mettre le .class)
Quelle est la syntaxe des commandes de compilation et d'exécution ?
javac nom de fichier
java nom de la classe qui contient la méthode main

Exercice 8 – Priorité des opérateurs

```
int a, b, c, d;
a = 10; b = 3; c = 3;
double x, y, z;
```

Q 8.1 Parenthéser l'expression suivante :

```
a = b = c = d = 1;
```

L'affectation est le seul opérateur qui s'évalue de la droite vers la gauche (et non de la gauche vers la droite comme les autres)

Q 8.2 En vous référant à la table de priorité des opérateurs (en annexe page 193), évaluer :
`d = a*c/b;`

d = (a*c)/b, deux opérateurs de même niveau, de gauche à droite, d = 10

`d = a/b*c;`

d = (a/b)*c ; deux opérateurs de même niveau, de gauche à doite, mais division entière, d = 9

`y = 1 / 100;`

y = 0.0, car division entière

`x = 1; y = x / 100;`

y = 0.01, car division réelle

Q 8.3 Parenthéser et indiquer les différentes conversions générées par le compilateur pour évaluer :
`y = a + b * x / c;`

y = (a + ((b * x)/c))
conversion de b en double, conversion de c en double, division réelle, conversion de a en double

Q 8.4 Parenthéser l'expression suivante. Les parenthèses doivent illustrer clairement l'ordre d'évaluation des expressions en Java.
`y = 3 + a * x * y - 5 / z;`

y = ((3 + ((a * x) * y)) - (5 / z));

Q 8.5 Parenthéser l'instruction, puis en donner une forme simplifiée :
`y = c + 4 / 5 * Math.sin(x-3) - 3`

y = ((c + ((4 / 5) * sin(x-3))) - 3)
comme 4 / 5 donne 0, on a donc y = c - 3
Autres exos possibles :
1) Ecrire une boucle qui calcule la somme des n premiers entiers
2) Trouver tous les nombres de 3 chiffres a,b,c tels que abc =a^3+b^3+c^3
3) Afficher les tables de vérité

Exercice 9 – Variables

Dans son programme de fidélisation, la société SFCF s'intéresse à ses plus fidèles voyageurs. L'analyse des besoins a permis de retenir pour un voyageur les caractéristiques suivantes :
nom, prénom, numéro adhérent, date de naissance, situation familiale, nombre d'enfants, nombre de points cumulés, montant en euros équivalent, statut de "super grand voyageur", taux de réduction.
Le statut de "super grand voyageur" s'obtient dès que le nombre de points cumulés atteint 5000 points. La situation familiale est codée par le premier caractère des libellés : *C*élibataire, *M*arié, *P*acsé, *V*euf, *D*ivorcé.

Q 9.1 Choisir un nom de variable et donner un type primitif, si possible, pour chacune des caractéristiques (voir annexe pour les types primitifs).

int nuAdherent (peut être aussi une chaîne) ; char SitFam ; int nbEnfants ; int nbPointsCumul ; double mtEuros ; boolean statutSGV ; double tauxReduc ;
Dans l'ensemble du fascicule, on mettra assez systématiquement des double et non des float
Dire des choses sur l'art et la manière d'écrire les noms de variables
On peut aussi introduire ici les types String et Date (et en profiter pour introduire la convention d'écriture pour les classes : minuscules/majuscules)

Q 9.2 Initialiser les variables correspondant aux caractéristiques suivantes : situation familiale, nombre d'enfants, nombre de points cumulés, montant en euros équivalent, statut de "super grand voyageur", taux de

réduction. Par défaut, le voyageur est célibataire, n'est pas "super grand voyageur", n'a ni enfants, ni points cumulés et pas de réduction.

SitFam = 'C' ; nbEnfants = 0 ; nbPointsCumul = 0 ; mtEuros = 0 ; statutSGV = false ; tauxReduc =0 ;
Dire qu'on peut aussi faire l'initialisation en même temps que la déclaration.
char SitFam = 'C' ; int nbEnfants = 0 ; int nbPointsCumul = 0 ; double mtEuros = 0 ; boolean statutSGV = false ; float tauxReduc =0 ;
Faire remarquer que les caractères sont entre '...' et non "..."

Q 9.3 Lors de l'achat d'un billet, le client a gagné 500 points. Mettre à jour la variable correspondante et calculer le montant en euros équivalent sachant qu'un point vaut 1,50 €.
Lorsque le client constate qu'il peut obtenir un voyage gratuit, il choisit un voyage équivalent à 2000 points. Mettre à jour les variables en conséquence.

nbPointsCumul= nbPointsCumul+500 ; ou nbPointsCumul+=500 ;
mtEuros= nbPointsCumul*1.5 ; ou mtEuros+=500*1.5 ;
nbPointsCumul= nbPointsCumul-2000 ; ou nbPointsCumul-=21000 ;
mtEuros= nbPointsCumul*1.5 ; ou mtEuros-=2000*1.5 ;
Vérifier que les types sont compatibles et en profiter pour montrer les conversions de type implicites.

Q 9.4 Lorsque le nombre de points cumulés est supérieur à 5000, le client obtient le statut "super grand voyageur". Par ailleurs, s'il est "super grand voyageur", il bénéficie d'un taux de réduction à 5% s'il a moins de 3 enfants et à 10% sinon. S'il n'est pas célibataire, il bénéficie en outre d'une prime exceptionnelle de 100 points. Ecrire l'instruction qui permettra tester et de mettre à jour les variables concernées.

Insister sur la différence entre = et ==

```
1 if (nbPointsCumul >=5000) statutSGV=true;
2 // ou mieux:
3 statutSGV = nbPointsCumul >=5000;
```

Dans un premier temps, il peut être judicieux de faire uniquement le test sur le nombre d'enfants

```
1 if (nbEnfants<3)
2         tauxReduc=0.05;
3 else tauxReduc=0.10;
4
5 if (statutSGV) {
6         if (nbEnfants<3)
7                 tauxReduc=0.05;
8         else tauxReduc=0.10;
9 }
10 if (sitFam != 'C') nbPointsCumul += 100;
```

Leur faire remarquer que ce n'est pas statutSGV == true, une maladresse qu'ils font souvent

Q 9.5 Pour l'impression des billets, la société SFCF souhaite aussi avoir une variable identite contenant le nom et le prénom du voyageur. Déclarer et initialiser cette variable.

String identite = nom+" "+prenom ;
Insister sur la différence de signification du +. Dire qu'il s'agit bien ici d'une opération de concaténation.

Quizz 1 – Compilation et exécution

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

.java

QZ 1.2 Un fichier source Java est composé de ...

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

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

méthodes et variables

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

méthodes

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

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

QZ 1.6 Si vous n'utilisez pas l'environnement intégré, quelle commande tapez-vous pour compiler un fichier ?

javac NomFichier.java

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

main

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

public static void main(String[] args)

Quizz 2 – Conventions de nommage

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

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

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

Quizz 3 – Syntaxe des expressions

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

Oui, car en java, le type par défaut pour les réels est double. Si l'on souhaite utiliser un float, il faut lui préciser : `float f=1.12f;`
Test.java :3 : possible loss of precision
found : double
required : float
float f=1.12 ;
1 error

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

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

QZ 3.3 Sachant que le code ascii de '1' est 49, qu'affiche :

1. `System.out.println(1+"1")` ?

   11 (concaténation "1"+"1")

2. `System.out.println(1+'1')` ?

50 (addition 1+49)
Remarque : System.out.println('1'+"1") affiche 11

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

`true && 49 == 49 => true && true => affiche true`
Pour répondre à la question, il faut savoir quel est l'opérateur le plus prioritaire. La table de priorité des opérateurs page 193 permet de voir que l'opérateur `==` est plus prioritaire que l'opérateur `&&`.

QZ 3.4 Qu'affiche : `System.out.println("Bonjour \nvous \ttous !")` ?

Cette question est pour introduire : "n et "t. "Bonjour" puis nouvelle ligne, puis "vous", puis espace tabulation, puis "tous !"

QZ 3.5 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 ?

On regarde d'abord la borne inférieure. Si `Math.random()` retourne 0, je veux que cela fasse 10. Donc je pose l'équation :
si `Math.random()=0` alors on veut que : 0*(x-1)+y=10 => y=10 ;
si `Math.random()=1` alors on veut que : 1*(x-1)+10=30 => x-1=20 => x=11
`int v=(int)(Math.random()*11+20);`

TME : classe : définition, syntaxe

Objectifs

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

Soumission

A chaque TME, vous devez rendre un compte rendu (1 fichier texte) indiquant pour chaque exercice les classes réalisées et les résultats obtenus lors de l'exécution. N'oubliez pas d'indiquer votre nom et celui de votre binôme. Un modèle de compte rendu est disponible sur le site de l'UE. A la fin de chaque TME, vous devez soumettre votre compte-rendu en cliquant sur le lien "soumission" pour envoyer votre compte rendu.

Exercices

En TME, faire les exercices 10,11, 12,13

Exercice 10 – Préliminaires

Q 10.1 Mettez le site web de l'UE dans les favoris de votre navigateur pour que vous puissiez accéder rapidement au site web de l'UE à chaque séance de TME.

Q 10.2 Pour bien organiser vos fichiers, on souhaite créer un répertoire `2i002`, puis dans ce répertoire, créer un répertoire `TME1`. Tous vos fichiers du `TME1` devront se trouver dans ce répertoire. Pour cela, ouvrez un nouveau terminal, puis tapez les commandes qui permettent de réaliser les instructions suivantes :

1. créer un répertoire `2i002`,
2. se déplacer dans ce répertoire `2i002`,
3. créer un répertoire `TME1`,
4. lister les fichiers du répertoire `2i002` pour vérifier que le répertoire `TME1` a été créé,
5. se déplacer dans le répertoire `TME1`,
6. afficher le nom du répertoire courant.

Aide : l'annexe page 195 rappelle la liste des instructions que vous pouvez utiliser pour organiser vos fichiers et répertoires sous linux.

mkdir 2i002 ; cd 2i002 ; mkdir TME1 ; ls ; cd TME1 ; pwd

Q 10.3 Pour ouvrir un éditeur de texte afin d'écrire du code Java ou du texte, vous pouvez utiliser la commande : `gedit &` ou bien `gedit MaClasse.java &` pour ouvrir le fichier `MaClasse.java`. Attention : n'oubliez pas de mettre le `'&'` à la fin de la commande pour séparer le terminal et l'éditeur de texte.

INUTILE EN 2012 LES COMPTES ETAIENT BIEN CONFIGURES
Ouvrer un terminal. Taper exactement la commande suivante : `gedit .bashrc &`
Ecrivez à la fin du fichier `.bashrc` qui vient de s'ouvrir : `PS1='\W \$ '`
Sauvegarder, puis fermer gedit et fermer le terminal.
Ouvrir un nouveau terminal, le prompt contient maintenant le nom du répertoire courant.

Q 10.4 Aller sur le site de l'UE dans la partie "TME et Soumission". Sauvegarder dans votre sous-répertoire TME1 le fichier qui vous servira de modèle pour les comptes rendus (le fichier s'appelle : CompteRendu.txt). Renommer ce fichier avec vos noms et le numéro de TME. *Attention :* à chaque séance de TME, vous devez écrire un compte rendu qui doit contenir : **les instructions (copier-coller des classes que vous écrivez), les exécutions des programmes (copier-coller du résultat affiché par le terminal) et, éventuellement, les réponses aux questions demandées.** Ce compte rendu sera à rendre en fin de chaque TME (soumission).

Q 10.5 Lisez rapidement l'annexe 2 page 195, puis répondez aux questions suivantes :
– Pourquoi est-il important d'indenter vos programmes ?
– Pourquoi est-il important de sauvegarder et de compiler régulièrement vos programmes au lieu d'attendre d'avoir écrit le programme dans son entier ?

Exercice 11 – Premier programme

Erreurs fréquentes :
Lors de l'exécution : `java Bonjour.class`, alors qu'il ne faut pas le suffixe class
Ils ne sont pas dans le bon répertoire
Problème de majuscules
S'il y a plusieurs fichiers, il faut dans le CLASSPATH inclure le répertoire courant, c'est-à-dire .

Q 11.1 Ecrire la classe `Bonjour` (fichier `Bonjour.java`) dont la méthode main affiche "Bonjour !".

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

Q 11.1.1 Quelle est la commande pour compiler cette classe ? Compiler la classe. Quel est le nom du fichier créé par la compilation ?

javac `Bonjour.java`
Bonjour.class

Q 11.1.2 Quelle est la commande pour exécuter cette classe ?

java Bonjour

Q 11.1.3 Supprimer le fichier `Bonjour.class`. Sans recompiler, taper à nouveau la commande pour exécuter la classe. La classe est-elle exécutée ?

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

Q 11.2 Observer les erreurs de compilation :

Q 11.2.1 Introduire un espace au milieu du mot static. Compiler. D'après le message d'erreur, à quelle ligne est l'erreur ? à quel endroit est détectée l'erreur ? Remarque : pour cette erreur, l'explication de l'erreur par le compilateur ne correspond pas à la correction à effectuer. Les diagnostics du compilateur ne doivent donc pas être suivis à la lettre, mais indiquent seulement l'échec de l'analyse.

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

Q 11.2.2 Rétablir le mot `static` correctement et supprimer le " terminant le mot Bonjour. Compiler et observer.

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

Q 11.2.3 Après avoir supprimé du répertoire courant le fichier `Bonjour.class`, transformer la méthode `main` en `Main`. La compilation réussit-elle ? Peut-on exécuter le programme ? Expliquer.

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

Q 11.2.4 Supprimer l'accolade { de la méthode main. Compiler et lire les messages.

Si vous oubliez une accolade, le compilateur diagnostiquera une erreur souvent très difficile à retrouver. Donc, chaque fois que vous tapez une {, tapez immédiatement l'accolade fermante } et ouvrez des lignes intermédiaires (en tapant Entrée) pour taper le reste du texte. Pour cela il faut être en mode insertion. Si, dans la ligne inférieure de la fenêtre de l'éditeur, vous voyez OVR, vous êtes en mode overwrite, c'est-à-dire en mode superposition. Tapez alors sur la touche `Inser`.

Q 11.2.5 Supprimer le mot-clé `public`. La compilation réussit-elle ? Peut-on exécuter le programme ?

La compilation et l'exécution se passent correctement.

Q 11.2.6 Supprimer le mot-clé `static`. La compilation réussit-elle ? Peut-on exécuter le programme ?

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

Exercice 12 – Segment

On veut écrire des classes Java afin de pouvoir comparer la longueur de plusieurs segments de droite (sur une dimension).

Q 12.1 Un segment est une portion de droite délimitée par 2 extrémités. Ecrire la classe `Segment` qui contient :
- les variables d'instance `x` et `y` correspondant aux valeurs des deux extrémités,
- un constructeur : `public Segment(int extX,int extY)` qui initialise la variable `x` avec la valeur de `extX` et la variable `y` avec la valeur de `extY`,
- une méthode `public int longueur()` qui retourne la longueur du segment. Si l'extrémité `x` a une valeur plus petite que la valeur de l'extrémité `y` alors la longueur est `y-x`, sinon la longueur est `x-y`.
- la méthode `toString()` dont le but est de retourner une chaîne de caractères au format suivant : `"Segment [<x>,<y>]"` où `<x>` et `<y>` doivent être remplacés par les valeurs des extrémités `x` et `y` du segment courant.

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

Q 12.2 Écrire une classe `TestSegment` dont la méthode `main` crée le segment [6,8] et le segment [12,5], puis compare la longueur des 2 segments. Si le premier segment est plus long, ce programme affiche que le premier segment est plus long, sinon il affiche que le deuxième segment est plus long.

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

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

Exercice 13 – Solidarité villageoise

Un énorme rocher est tombé dans la nuit sur un petit village de l'ouest de la France, empêchant les cultures. Il est décidé de former une équipe de villageois pour déplacer le rocher (qui pèse 200 kg).

Q 13.1 Dans la classe `Villageois`, écrire les variables suivantes :
– `nom` (le nom du villageois, de type `String`),
– `poids` (le poids du villageois, type `double`),
– `malade` (type `boolean`, sa valeur sera `true` si le villageois est malade, `false` sinon).
Quel doit être le nom du fichier contenant cette classe ?

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

Le fichier doit s'appeller : Villageois.java

Q 13.2 Ajouter dans la classe `Villageois` le constructeur `public Villageois(String nomVillageois)` qui initialise le nom du villageois avec la valeur de `nomVillageois`, la variable `poids` avec un poids compris entre 50 et 150 kg (150 exclu), et la variable `malade` à `true` dans 20% des cas et à `false` sinon. Aide : utiliser la méthode `Math.random()` qui génère une valeur aléatoire de type `double` comprise entre `0.0` inclus et `1.0` exclu. Exemples : pour initialiser aléatoirement la variable `x` avec un nombre entre 20 et 100 exclu, on écrit l'instruction : `double x=Math.random()*80+20;` Pour modéliser un évènement qui doit se produire dans 40% des cas, on regarde si `Math.random() < 0.40`. Voir aussi la documentation de la classe `Math` page 193.

```
1 public Villageois(String nomVillageois) {
2         nom = nomVillageois;
3         poids = Math.random()*100 + 50; // poids aléa entre 50 et 200
4         malade = Math.random() < 0.2;
5 }
```

Q 13.3 Dans une nouvelle classe `TestVillageois`, ajouter une méthode `main`, qui crée quatre instances de la classe `Villageois` et les affiche. Quel doit être le nom du fichier contenant la classe `TestVillageois` ? Compilez et exécutez ce programme.

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

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

Le fichier doit s'appeller : TestVillageois.java

Q 13.4 Ajouter dans la classe `Villageois` et utiliser dans la classe `TestVillageois` les méthodes suivantes :

- `public String getNom()` qui retourne le nom de ce villageois,
- `public double getPoids()` qui retourne le poids du villageois,
- `public boolean getMalade()` accesseur de la variable malade,

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

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

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

        return text;
}
```

Q 13.5 Ajouter dans la classe `Villageois` la méthode `double poidsSouleve()` qui retourne le poids soulevé par ce villageois : le tiers de son poids s'il est en bonne santé, le quart s'il est malade.

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

Q 13.6 Ajouter dans la classe `TestVillageois`, les instructions pour calculer le poids total que peuvent soulever les 4 villageois, et pour afficher un message pour indiquer si les villageois ont réussi à soulever le rocher ou pas.

```
double poidsTotal=v0.poidsSouleve()+v1.poidsSouleve()
                 +v2.poidsSouleve()+v3.poidsSouleve();
if ( poidsTotal > 200 )
        System.out.println("Le rocher a ete souleve !");
else
        System.out.println("Le rocher n'a pas ete souleve !");
```

Exercice 14 – Affichage avec passage à la ligne

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

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

```
public class Lettre {
        private char carac;

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

Q 14.1 Dans la méthode main d'une classe `TestLettre`, écrire les instructions qui, pour chaque caractère de 'a' à 'z', affiche son code ascii (utiliser la méthode `getCodeAscii()`).
Aide : utiliser une boucle for avec un compteur de type `char`.

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

Q 14.2 On veut maintenant afficher l'alphabet comme ceci :

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

Pour cela, il suffit de répéter l'affichage d'un caractère en passant à la ligne tous les cinq caractères. A la suite dans le `main`, en utilisant la méthode `getCarac()` de la classe Lettre, effectuer cet affichage.

Aide : utiliser l'opérateur % (modulo, c-à-d reste de la division) et l'instruction : `System.out.print(`*chaine*`);` qui affiche chaîne sans passer à la ligne (contrairement à `System.out.println()`).

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

Exercice 15 – Libellé d'un chèque

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

Chgt 2012 : suppression de tous les static
NB : toutes les méthodes *outils* pourraient être private, c'est une bonne illustration pour les étudiants

```
//class Libelle pour libeller le montant d'un chèque.
//la methode statique enLettres(nb) renvoie nb "en toutes lettres"
//nb doit etre inferieur a 1000
//la methode main teste un certain nombre de cas....

public class Libelle {

public String moinsDe16(int nb) {
        switch (nb) {
                case 0 : return "zero" ;
                case 1 : return "un" ;
                case 2 : return "deux" ;
                case 3 : return "trois" ;
                case 4 : return "quatre" ;
                case 5 : return "cinq" ;
                case 6 : return "six" ;
                case 7 : return "sept" ;
                case 8 : return "huit" ;
                case 9 : return "neuf" ;
                case 10 : return "dix" ;
                case 11 : return "onze" ;
                case 12 : return "douze" ;
                case 13 : return "treize" ;
                case 14 : return "quatorze" ;
                case 15 : return "quinze" ;
                default : return "seize" ;
```

```
27          }
28  }
29  public String chiffreDesDizaines (int nb){
30          switch (nb) {
31                  case 0 : return "" ;
32                  case 1 : return "dix" ;
33                  case 2 : return "vingt";
34                  case 3 : return "trente" ;
35                  case 4 : return "quarante" ;
36                  case 5 : return "cinquante" ;
37                  case 6 : return "soixante" ;
38                  case 7 : return "soixante-dix" ;
39                  default : return "quatre-vingt" ;
40          }
41  }
42  public String moinsDe100(int nb) {
43          int dizaine = nb/10 ;
44          int unite = nb%10 ;
45          String lien = (unite == 1) && (dizaine<8) ? "␣et␣" : "-" ;
46          if (dizaine == 7 || dizaine == 9){
47                  dizaine--;
48                  unite = unite + 10 ;
49          }
50          return chiffreDesDizaines(dizaine) + lien + enLettres(unite) ;
51  }
52  public String moinsDe1000(int nb) {
53          int centaine = nb/100 ;
54          int suite = nb % 100 ;
55          String s = suite==0 ? "" : "␣" + enLettres(suite);
56          String c = centaine==1 ? "" : enLettres(centaine) + "␣";
57          String cent = centaine>1 && suite == 0 ? "cents" : "cent" ;
58          return  c + cent + s ;
59  }
60  public String enLettres(int nb){
61          if (nb<=16) return moinsDe16(nb) ;
62          else if (nb == 80) return "quatre-vingts" ;
63          else if (nb<100) return moinsDe100(nb) ;
64          else if (nb<1000) return moinsDe1000(nb) ;
65          else return "mille␣ou␣plus␣";
66  }
67  public static void main (String [] a) {
68          int [] tab={15,23,71,80,81,91,98,100,103,423,600,1000,1010};
69          System.out.println ("Quelques␣nombres␣en␣toutes␣lettres␣:\n␣");
70          for(int i=0;i<tab.length;i++) {
71                  System.out.println (tab[i] + "␣:␣" + enLettres(tab[i]));
72          }
73  }
74  }
```

Exercice 16 – Formule de Newton

La suite de Newton définie ci-dessous converge vers la racine carrée de `x` :
$U_0 = x/2$ et $U_i = (U_{i-1} + x/U_{i-1})/2$
Écrire une classe `SuiteNewton` qui calcule la racine de x avec une précision de ε, en utilisant la suite de Newton.

Ajout 2012 : suppression des static

```
1 public class SuiteNewton {
2         // Calcul d'une racine carrée par la suite
3         // de Newton Ui = (Ui-1 + x/Ui-1)/2
4         //---------------- U0 = x / 2
5         private double x;
6         private double epsilon;
7         public SuiteNewton(double x, double epsilon){
8                 this.x = x; this.epsilon = epsilon;
9         }
10
11        public double racine() {
12                int i = 0; // Pour le calcul du nombre d'itérations
13                double u = 0, v = x/2;
14                if(x < 0) return Double.NaN;
15                if (x != 0) {
16                        do {
17                                u = v;
18                                v = (u + x / u)/ 2;
19                                i++;
20                        } while (Math.abs(u-v) > epsilon);
21                }
22                System.out.println("Nombre d'itérations = " + i);
23                return v;
24        }
25 }
26
27 //================
28
29 public class TestSuiteNewton {
30        public static void main(String[] args) {
31                System.out.println("Calcul de la racine carrée");
32                double nombre = 5;
33                System.out.println("Racine(" + nombre + ")=" + racine(nombre, 1e
                      -4));
34                System.out.println("Math.sqrt(" + nombre + ")=" + Math.sqrt(nombre
                      ));
35        }
36 }
```

2 Encapsulation, surcharge

Objectifs

- `this`
- Surcharge de méthodes
- Surcharge de constructeurs
- Encapsulation
- Méthode `toString`

Rappel : On utilise vraiment static seulement à partir du thème 5.
Proposition :
En TD, faire les Quizz à la fin du Thème 1 - TD, puis au moins exercices 17
En TME, exercice 20

Exercices

Exercice 17 – 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 17.1 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.

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

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

Q 17.2 Expliquer l'affectation de la ligne 5 : que représente `this.volume` ? `volume` ?

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

Q 17.3 Expliquer la ligne 8.

this( ) avec parenthèse correspond à un appel de constructeur, ici c'est le constructeur à 1 paramètre.
Attention : this() ne s'utilise que dans un constructeur, jamais de : new this().
Attention : si on utilise un this(), il doit toujours être la première instruction dans le constructeur

Q 17.4 Compléter la méthode d'instance `remplir(Bouteille b)` qui ajoute le contenu de la bouteille `b` à la bouteille courante.

```
1 public void remplir (Bouteille b){
2         volume+=b.volume;
3 }
```

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

Q 17.5 Peut-on rajouter une méthode portant le même nom que la méthode précédente, mais prenant un paramètre de type `double` ? Si oui, écrire cette méthode.

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

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

Q 17.6 Quel va être le résultat de l'affichage des lignes 5, 6, 8 et 9 du programme ci-après ? Expliquer.

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

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

Exercice 18 – Adresse Web (Surcharge de constructeurs)

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

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

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

```
1 public class AdresseWeb {
2         private String protocole; // http , https ,...
3         private String domaine; // mondomaine.com
4         private String chemin; // chemin menant au fichier commencant par /
5
6         public AdresseWeb(String protocole, String domaine, String chemin) {
7                 this.protocole=protocole;
8                 this.domaine=domaine;
9                 this.chemin=chemin;
10        }
11        public AdresseWeb(String domaine, String chemin) {
12                this("http",domaine,chemin);
13        }
14        public AdresseWeb(String domaine) {
15                this(domaine,"");
16        }
17        public String toString() {
18                return protocole+"://www."+domaine+chemin;
19        }
20 }
```

Q 18.2 Ecrire la classe `TestAdresseWeb` qui crée 3 adresses Web en appelant à chaque fois un constructeur différent, puis affiche les URLs correspondantes. A quoi sert la surchage de constructeurs ?

L'énnonce ne précise pas qu'il faut une méthode main.
La surcharge de constructeurs permet d'éviter de répéter du code inutilement.

```
1 public class TestAdresseWeb{
```

```
2       public static void main(String [] args) {
3            AdresseWeb aw1=new AdresseWeb("https","site.fr","/dir/page1.html");
4            AdresseWeb aw2=new AdresseWeb("fleur.com","/tulipe.php");
5            AdresseWeb aw3=new AdresseWeb("cactus.fr");
6            System.out.println("Mes sites preferes :\n"+aw1+"\n"+aw2+"\n"+aw3);
7       }
8 }
```

Exercice 19 – Salutation ! (appels aux constructeurs, surcharge)

Le but de l'excercice (plutôt à faire en TME) est que les étudiants comprennent que les constructeurs sont appelés quand on fait un new Salutation()

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
7 public class TestSalutation {
8         public static void main(String [] args) {
9                 Salutation s1=new Salutation();
10        }
11 }
```

Q 19.1 Quel est le résultat affiché par ce programme ?

Affichage de : "Appel au constructeur sans parametre de la classe Salutation"

Q 19.2 Dans la classe `Salutation`, ajouter un deuxième constructeur (surcharge de constructeurs). Ce constructeur prend en paramètre une chaîne de caractères et l'affiche.

```
1         public Salutation(String m) {
2                 System.out.println(m);
3         }
```

Q 19.3 Dans la classe `TestSalutation`, ajouter plusieurs instances de la classe en appelant plusieurs fois chaque constructeur. Quel est le résulat affiché par ce programme ?

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

```
3                 Salutation s1=new Salutation();
4                 Salutation s2=new Salutation("Salutation␣!");
5         }
6 }
```

Exercice 20 – Course de relais 4 fois 100m

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

Q 20.1 Ecrire une classe `Coureur` qui a les variables d'instance suivantes :
- `nuDossard` de type `int` (numéro du dossard),
- `tempsAu100` de type `double` (nombre de secondes au 100m),
- `aLeTemoin` de type `boolean` (vrai ssi le coureur a le témoin).

Q 20.2 Ajouter dans la classe `Coureur`, les constructeurs suivants :
- un constructeur prenant un seul paramètre correspondant au numéro du dossard, qui initialise `tempsAu100` avec un nombre compris entre 12 et 16 exclu, et `aLeTemoin` avec `false`,
- un constructeur sans paramètre qui appelle le constructeur à un paramètre et qui initialise `nuDossard` avec un entier choisi aléatoirement entre 1 et 1000.
  Aide : pour générer un nombre aléatoirement, utiliser la méthode `Math.random()` qui rend un double compris entre 0 et 1 exclu. Exemple : pour initialiser aléatoirement la variable `x` avec un nombre entier entre 20 et 100 compris, on écrit l'instruction : `int x=(int)(Math.random()*81+20);`

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

Q 20.3 Dans un autre fichier, écrire une classe `TestCoureur` contenant la méthode `main`, point d'entrée du programme. Cette méthode crée 4 instances de la classe `Coureur` `c1`, `c2`, `c3` et `c4`. Vérifier que la méthode `main` compile.

```
1 public class TestCoureur {
2         public static void main(String [] args) {
3                 Coureur c1=new Coureur();
4                 Coureur c2=new Coureur();
5                 Coureur c3=new Coureur();
```

```
6                Coureur c4=new Coureur();
7
8                System.out.println("Presentation des coureurs :\n");
9                System.out.println(c1.toString());
10               System.out.println(c2.toString());
11               System.out.println(c3.toString());
12               System.out.println(c4.toString());
```

Q 20.4 Ajouter et tester au fur et à mesure dans la méthode `main()` de `TestCoureur` les méthodes suivantes :
- les accesseurs : `int getNuDossard()`, `double getTempsAu100()`, `boolean getALeTemoin()`,
- le modifieur : `void setALeTemoin(boolean b)` qui modifie la valeur de la variable d'instance `aLeTemoin`,
- la méthode `toString()` qui retourne une chaîne de caractères décrivant les caractéristiques de ce coureur. Exemple : `Coureur 56 tempsAu100 : 13,7s au 100m aLeTemoin : non`.

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

Q 20.5 Ajouter dans la classe `Coureur` les méthodes suivantes :
- `void passeTemoin(Coureur c)` qui affiche : `"moi, coureur xx, je passe le témoin au coureur yy"`, enlève le témoin à ce coureur et le donne au coureur `c` passé en paramètre.
- `void courir()` qui simule la course du coureur sur 100 mètres en affichant le message `"je suis le coureur xx et je cours"`.

```
1        public void passeTemoin(Coureur c) {
2                System.out.println("Moi, coureur "+ this.nuDossard
3                        +", je passe le temoin au coureur "+c.nuDossard + "\n");
4                setALeTemoin(false);
5                c.setALeTemoin(true);
6        }
7
8        public void courir() {
9                System.out.println("Moi, coureur "+ nuDossard + ", je cours\n");
10               // Clavier.dormir(1000*((int)(tempsAu100/5)));
11               try {
12                       Thread.sleep((int)(tempsAu100/5));
13               } catch (Exception e) {
14                 e.printStackTrace();
15               }
16       }
```

Pour les étudiants les plus avancés...
Si on veut faire un temps d'attente proportionnel au temps de course du coueur on peut utiliser : `Clavier.dormir(int duree)` qui arrête l'exécution de la méthode pendant une durée exprimée en millisecondes (voir la documentation de la classe Clavier en annexe 1 page 194).
Ajout 2012 : on a convenu de ne pas introduire clavier (au moins au début), il faut donc donner le code de sleep à utiliser en aveugle.
ou bien :

```
try {
        Thread.sleep((int)(tempsAu100/5));
} catch (Exception e) {
        e.printStackTrace(); // affichage en cas de problème
}
```

Q 20.6 Ajouter dans la méthode main les instructions qui permettent de :
– faire courir en relais 4 fois 100m les quatre coureurs dans l'ordre c1, c2, c3, c4.
– calculer et afficher le temps total mis par les coureurs pour faire les 400m.

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

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

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

Exercice 21 – Classe triangle

Cet exercice comprend de la composition d'objets.

Q 21.1 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 a 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.

```
1 public class Point {
2         private int posx, posy;
3
4         public Point () {
5                 posx = 0;     // ou posx = (int)Math.random() * 10
6                 posy = 0;     // ou posy = (int)Math.random() * 10
7         }
8         public Point (int valx, int valy) {
9                 posx = valx;
10                posy = valy;
11        }
12
13        public void setPosx (int x) { posx = x; }
14        public void setPosy (int y) { posy = y; }
15        public int getPosx (int x) { return posx; }
16        public int getPosy (int y) { return posy; }
17
18        public String toString() { return "("+posx+","+posy+")"; }
19        public int carre(int nb) { return nb * nb; }
20
21        public float distance(Point p) {
22                float res = carre(posx - p.posx) + carre(posy-p.posy);
23                return (float)Math.sqrt(res);
24        }
25        public void deplaceToi(int newx, int newy) {
26                posx = newx;
27                posy = newy;
28        }
29        public void deplaceToiRel(int deltaX, int deltaY) {
30                posx += deltaX;
31                posy += deltaY;
32                // ou : deplaceToi(x+deltaX, y+deltaY)
33        }
34        public boolean equals(Object o) {
35                Point p=(Point)o;
36                return (this.posx==p.posx) && (this.posy==p.posy);
37        }
38 }
```

Q 21.2 Tester cette classe en écrivant la méthode `main` qui crée des points et affiche leurs coordonnées.

```
1 public class TestPoint {
2     public static void main(String[] args) {
3         Point p1 = new Point(2, 5);
4         System.out.println("Coordonnee de p1 : "+p1.toString());
5         Point p2 = new Point(10, 5);
6         System.out.println("Coordonnee de p2 : "+p2.toString());
7
8         System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
```

```
9           p1.deplaceToi(0,0);
10          System.out.println ("Nouvelles coordonnees de p1 : "+p1.toString());
11          System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
12          p1.deplaceToiRel(2,5);
13          System.out.println ("Nouvelles coordonnees de p1 : "+p1.toString());
14          System.out.println ("Distance entre p1 et p2 est de "+p1.distance(p2));
15      }
16  }
```

Q 21.3 Ecrire une classe `Triangle` à trois variables d'instance prenant leur valeur dans la classe `Point`. Cette classe comprendra :
- Un constructeur par défaut.
- Un constructeur à trois paramètres : les trois sommets du triangle.
- Une méthode `getPerimetre()` qui retourne le périmètre du triangle.
- Redéfinir la méthode `public String toString()` qui retourne une chaîne de caractères décrivant le triangle (en utilisant la méthode `toString()` de la classe `Point`).

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

Q 21.4 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).

```
1 public class TestTriangle {
2     public static void main(String[] args) {
3         Point p1 = new Point(-1, 0);
4         System.out.println ("Coordonnee de p1 : "+p1.toString());
5         Point p2 = new Point(1, 0);
6         System.out.println ("Coordonnee de p2 : "+p2.toString());
7         Point p3 = new Point(2, 1);
8         System.out.println ("Coordonnee de p3 : "+p3.toString());
9         Triangle t = new Triangle(p1, p2, p3);
10        System.out.println ("Coordonnee du triangle : "+t.toString());
11        System.out.println ("Son cote A-B est de : "+t.baseAB());
12        System.out.println ("Son cote A-C est de : "+t.baseAC());
13        System.out.println ("Son cote B-C est de : "+t.baseBC());
14        System.out.println ("Son perimetre est de : "+t.perimetre());
15    }
16 }
```

Quizz 4 – Fleur (constructeur, `this`)

Etudier le programme ci-dessous puis répondre aux questions.

```
1 public class Fleur {
2         private String nom;
3         private String couleur;
4
5         public Fleur (String name, String couleur) {
6                 nom = name;
7                 this.couleur = couleur;
8         }
9
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 4.1 Donner les commandes pour compiler, puis exécuter ce programme.

Si on suppose une classe par fichier :
`javac Fleur.java Quizz.java`
`java Quizz` car la méthode main se trouve dans la classe Quizz

Attention : Respecter les majuscules et minuscules.

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

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

QZ 4.3 La variable d'instance `nom` aurait-elle pu être déclarée après la variable `couleur` ? après la méthode `getNom()` ? Si oui, est-ce que cela aurait fait une différence ? Peut-on intervertir les lignes 23 et 24 ?

Oui. Oui. Cela ne fait aucune différence. L'ordre des déclarations des variables et des méthodes n'est pas important.
On ne peut pas intervertir les lignes 23 et 24, car dans le corps d'une méthode l'ordre des déclarations est important.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Quizz 5 – Encapsulation

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

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

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

Quizz 6 – Méthode `toString()`

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

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

QZ 6.2 Soit la classe suivante :

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

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

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

3 Tableaux

Objectifs

- Tableau d'éléments de type simple
- Tableau de tableaux (deux dimensions)
- Tableau d'objets à une dimension

Exercices

Propositions :
En TD, faire le Quizz sur les tableaux, puis exercice 22 (tableau d'entiers)
En TME, exercice 2 (histogramme de notes), exercice 3 (Triangle de Pascal)
Faire aussi un exercice sur les tableaux d'objets Thème suivant Exercice 5 Villageois

Exercice 22 – Tableau d'entiers

Q 22.1 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).

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

Q 22.2 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 22.3 Tester ces méthodes au fur et à mesure dans la méthode `main` d'une classe `TestTableau`.

```
1 public String toString(){
2         if (tab.length==0) return "[]";
3         String ch = "[";
4         for (int i=0; i<tab.length-1; i++) {
5                 ch += tab[i];
6                 ch += ",";
7         }
8         ch += tab[tab.length-1];
9         ch += "]";
10        return ch;
11 }
12
13 // Calcul le rang (indice) de l'element maximum du tableau
14 public int rangMax() {
15        int rang = 0;
16        for (int i=1; i<tab.length; i++) {
17                if (tab[i] > tab[rang])
18                rang = i;
19        }
20        return rang;
21 }
22
23 // calcule la somme des elements du tableau
24 public int somme() {
25        int som = 0;
26        for (int i=0; i<tab.length; i++) {
27                som += tab[i];
28        }
29        return som;
30 }
```

Q 22.4 On veut maintenant tester l'égalité de deux objets de la classe `TableauInt` Dans cette optique, créer dans la méthode main des instances de la classe `TableauInt` t1, t2, t3, t4 en respectant les contraintes suivantes :
– les instances `t1` et `t2` de la classe `TableauInt` ont des références égales.
– l'instance `t3` a les mêmes éléments que `t1`, mais pas la même référence.
– l'instance `t4` n'a pas les mêmes éléments que `t1` (et donc pas la même référence).

```
1 public class TestTableau {
2         public static void main(String[] args) {
3                 TableauInt t = new TableauInt ();
4                 System.out.println("tab de lg "+10 +" : "+t.toString());
5                 TableauInt t1 = new TableauInt (20);
6                 TableauInt t2 = t1;     // meme reference
7                 TableauInt t3 = new TableauInt(20); // meme element
8                 TableauInt t4 = new TableauInt(); // contenu aleatoire
9                 System.out.println("La valeur de t1 : "+t1.toString());
10                System.out.println("La valeur de t2 : "+t2.toString());
11                System.out.println("La valeur de t3 : "+t3.toString());
12                System.out.println("La valeur de t4 : "+t4.toString());
13                System.out.println("----------------------\n");
14                System.out.println("Egalite t1 et t2 : "+t1.egal(t2));
15                System.out.println("Egalite t1 et t3 : "+t1.egal (t3));
```

```
16                  System.out.println("Egalite t1 et t4 : "+t1.egal (t4));
17                  System.out.println("----------------------\n");
18                  System.out.println("Egalite t1 et t2 : "+t1.equals(t2));
19                  System.out.println("Egalite t1 et t3 : "+t1.equals(t3));
20                  System.out.println("Egalite t1 et t4 : "+t1.equals(t4));
21          }
22 }
```

Q 22.5 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.

```
1 // retourne l'egalite de deux tableaux (memes elements)
2 public boolean egal(TableauInt ref) {
3         for (int i=0; i<tab.length; i++) {
4                 if (tab[i] != ref.tab[i])
5                         return false;
6         }
7         return true;
8 }
```

Q 22.6 Créer un tableau d'entiers `t5` ayant pour valeurs {1, 2, 3} et un tableau d'entiers `t6` ayant les mêmes valeurs. Evaluer l'expression `t5.equals(t6)` et expliquer le résultat.

Voir poly de cours section 3.4.3 Comment comparer et 4.1 La classe Object

```
1 int [] t5 = {1,2,3};
2 int [] t6 = {1,2,3};
3 t5.equals(t6);
```

Rend false car la méthode equals de la classe `Object` teste l'égalité des références. Si vous voulez comparer le contenu de deux tableaux, il faut écrire une méthode ad hoc.

Exercice 23 – 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 23.1 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,

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

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

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

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

Q 23.2 Ajouter à cette classe, une méthode `afficheHistogrammeTableau()` qui affiche l'ensemble des valeurs du tableau histogramme.

Q 23.3 Ajouter à cette classe, une méthode `afficheHistogramme()` qui affiche le résultat sous forme d'un histogramme, c'est-à-dire en associant à chaque élément du tableau une ligne comprenant autant de `*` que la valeur de cet élément. (comme dans l'exemple de l'énoncé)

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

```
20          for (int i=0;i<hist.length;i++)
21                  s=s.concat(hist[i]+"␣");
22          return s;
23  }
```

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

Q 23.4 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.

```
1 public class TestHisto {
2         public static void main(String [] args){
3                 Histo physique=new Histo(150);
4                 physique.afficheHistogrammeTableau();
5                 physique.afficheHistogrammeTableau("Voici␣l'histogramme");
6                 physique.afficheHistogramme();
7                 System.out.println(physique);    // toString() implicite
8         }
9 }
```

Exercice 24 – 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 24.1 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. Aide : cette classe `contient` une variable de type tableau d'entiers à deux dimensions. Dans le constructeur, la réservation de la mémoire pour ce tableau ce fait en plusieurs étapes. Premièrement, de la mémoire est réservée pour un tableau de n lignes où *chaque ligne est un tableau d'entiers*. Ensuite, pour chaque ligne, de la mémoire est réservé pour le tableau d'entiers. Chaque ligne a une taille différente ce qui permet de ne réserver que la place mémoire nécessaire au triangle de Pascal.

Q 24.2 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 24.3 Ecrire une classe `TestTrianglePascal` qui crée plusieurs instances de la classe `TrianglePascal` et les affiche.

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

```
1 public class TrianglePascal {
2       private int taille;
3       private int [][] triangle;
4
5       public TrianglePascal(int n) {
6             if (n > 0){
7                   triangle = new int [n][];
8                   taille = n;
9             } else {
10                  System.out.println("anomalie dans la taille (par defaut : 5)");
11                        triangle = new int [5][];
12                  taille = 5;
13            }
14      }
15      public void remplirTriangle(){
16            for (int i = 0 ; i < taille ; i++){
17                  triangle[i] = new int[i+1];
18            }
19            for (int i = 0 ; i < taille ; i++){
20                  for (int j = 0 ; j < (i+1) ; j++){
21                        if ((j == 0) || (i == j))
22                              triangle[i][j] = 1;
23                        else
24                              triangle[i][j] =triangle[i-1][j-1] + triangle[i-1][j];
25                  }
26            }
27      }
28
29      public String toString(){
30            String s = "";
31            for(int i=0 ; i<triangle.length ; i++){
32                  for(int j = 0 ; j<triangle[i].length ; j++){
33                        s += triangle[i][j];
34                  }
35                  s += "\n";
36            }
37            return s;
38      }
39 }
40
41 public class TestTrianglePascal{
42      public void main (String[] args) {
43            TrianglePascal t1 = new TrianglePascal (6);
44            t1.remplirTriangle();
45            System.out.println(t1.toString());
46            // test cas d'anomalie
47            TrianglePascal t2 = new TrianglePascal (0);
48            t2.remplirTriangle();
49            System.out.println(t2.toString());
50      }
51 }
```

Exercice 25 – Villageois (tableau d'objets)

On a déjà vu cet exercice. Ce qui nous intéresse ici, c'est l'initialisation du tableau d'objets (`new` du tableau et ajout des villageois).

On reprend l'exercice 13 page 26 "Solidarité Villageoise" pour y ajouter le concept d'équipe de villageois (tableau d'objets Villageois). Récupérez votre classe `Villageois` et compilez-la.

Q 25.1 On considère que les villageois sont regroupés au sein d'une équipe qui contient entre 0 à 20 volontaires. Ecrire la classe `Equipe` qui contient les variables suivantes :
- le nom de l'équipe,
- un tableau de villageois,
- le nombre de villageois présents dans l'équipe.

On peut faire l'exercice sans le static pour la constante `MAXVILLAGEOIS`. Mais on peut aussi choisir de fonctionner en aveugle : "vous utilisez public/private static final + TYPE pour définir une constante"

```
public class EquipeVillageois {
        private static final int MAXVILLAGEOIS=20;
        private Villageois [] tabV=null;
        private int nbV;
        private String nom; // Nom de l'equipe
```

Cette classe contient un constructeur prenant en paramètre le nom de l'équipe qui crée une équipe qui pour l'instant ne contient aucun villageois,

```
public EquipeVillageois(String nom){
        tabV=new Villageois[MAXVILLAGEOIS];
        nbV=0;
        this.nom=nom;
}

// constructeur absurde supprime en 2013
//public EquipeVillageois(String nom, int nbVillageois) {
//      this(nom);
//      for(int i=0;i<nbVillageois;i++)
//              embaucher(new Villageois());
//}
```

Q 25.2 Ajouter dans la classe `Equipe` une méthode `void embaucher(Villageois v)` qui intègre le villageois `v` dans cette équipe.

```
public void embaucher(Villageois v) {
        if ( nbV < MAXVILLAGEOIS ) {
                tabV[nbV]=v;
                nbV++;
                System.out.println("L'equipe "+nom+" embauche le "+v);
```

```
6         }
7 }
```

Q 25.3 Ajouter dans la classe `Equipe` une méthode `poidsSouleve()` qui calcule le poids soulevé par toute l'équipe.

```
1 public double poidsSouleve(){
2         double poids=0;
3         for(int i=0;i<nbV;i++)
4                 poids+=tabV[i].poidsSouleve();
5         return poids;
6 }
```

Q 25.4 une méthode `toString()` qui retourne une chaîne de caractères ressemblant au texte ci-dessous (aide : utiliser `toString()` de la classe Villageois) :

```
L'equipe Bleu contient les villageois :
villageois Pierrix, poids : 188Kg, malade : non, poids souleve : 62Kg
villageois Paulix, poids : 63Kg, malade : non, poids souleve : 21Kg
L'equipe souleve un poids total de 83Kg
```

```
1 public String toString(){
2         String s="L'equipe "+nom+" contient les villageois :\n";
3         for(int i=0;i<nbV;i++)
4                 s+=tabV[i]+"\n";
5         s+="L'equipe souleve un poids total de "+poidsSouleve();
6         return s;
7 }
```

Q 25.5 Dans une classe `TestEquipeVillageois`, créez plusieurs équipes avec un nombre de villageois différent. Pour chaque équipe, vérifiez si l'équipe a pu soulever un rocher de 150Kg.

```
1 public class TestEquipeVillageois {
2         public static void main(String[] args) {
3                 EquipeVillageois ev1=new EquipeVillageois("Bleu");
4                 ev1.embaucher(new Villageois("Pierrix"));
5                 ev1.embaucher(new Villageois("Paulix"));
6                 System.out.println(ev1.toString());
7                 EquipeVillageois ev2=new EquipeVillageois("Rouge");
8                 for(int i=0; i<5; i++) ev2.embaucher(new Villageois());
9                 System.out.println(ev2.toString());
10                ev1.afficherInfo();
11                ev2.afficherInfo();
12        }
13 }
```

Exercice 26 – Le jeu de la vie (tableau à 2 dimensions)

Présentation du jeu
Le monde est un damier de 20 cases sur 20 appelées cellules. Une cellule est soit vivante soit morte. Au départ, il y a 80% de cellules mortes. Chaque cellule a 8 voisins (monde torique type chambre à air), c'est-à-dire par exemple que le voisin de droite d'une cellule du bord droit du damier a comme voisine de droite la cellule opposée du damier. Il en est de même pour toutes les cellules du bord du damier, qui ont pour voisines les cellules du bord opposé.
Les règles de vie à chaque nouvelle génération sont les suivantes :
- une cellule meurt si elle a strictement moins de 2 voisins (mort par isolement) ou strictement plus de 3 (mort par surpopulation).
- une cellule naît sur une case vide si celle-ci a exactement 3 voisins.

On visualisera le monde en représentant une cellule vivante par le caractère `"*"` et une cellule morte par le caractère `"."`.

Q 26.1 Classes de base :
- Définir la classe `Monde` qui a quatre variables : la largeur du damier `maxX`, sa hauteur `maxY`, le numéro de la génération courante `noGener` et un tableau de booléens à 2 dimensions `tabCells` (true signifiant que la cellule est vivante, false qu'elle est morte).
- Définir le constructeur `Monde(double seuil)` de cette classe qui crée un monde de `maxX` cellules sur `maxY` cellules avec un pourcentage approximatif de cellules vivantes déterminé par le seuil.
  On pourra utiliser la méthode `random` de la classe `Math` qui génère une valeur aléatoire de type double comprise entre 0.0 inclus et 1.0 exclu.
- Écrire la méthode `public String toString()` qui retourne une chaîne de caractères décrivant ce monde avec son numéro de génération et le tableau de l'état de ses cellules.
- Définir une classe `TestJeuVie` avec la méthode `main` qui crée un monde avec 80% (seuil = 0.8) de cellules mortes et le visualise.

Q 26.2 Ajouter dans la classe `Monde` la méthode `nbVoisins(int nuLign,int nuCol)` qui retourne le nombre de cellules voisines vivantes de la cellule `tabCells(nuLign, nuCol)`. Penser à utiliser la fonction % (modulo). Calculer le nombre de voisins d'une cellule revient à compter le nombre de cellules vivantes dans le carré de 3x3 dont le centre est cette cellule. Un moyen simple de gérer la sortie du damier, dans un monde torique, est de prendre la formule : `(i + maxX) modulo maxX ou (i + maxY) modulo maxY`. Tester cette méthode.

Q 26.3 Ajouter, dans la classe Monde, la méthode `void genSuiv()` qui, à partir du tableau `tabCells` de la classe `Monde`, crée un nouveau tableau en y appliquant les règles du jeu de la vie données plus haut afin d'obtenir la génération suivante, puis bascule `tabCells` sur ce nouveau tableau. Tester le jeu de la vie.

même remarque que précédemment pour les constantes : on ne s'attarde pas sur static (ni final) -¿ soit on les supprime, soit on fonctionne en aveugle.

```
1 public class Monde {
2         public static final int MAXX = 20;
3         public static final int MAXY = 20;
4         private int noGen=0;
5         private boolean [][] tabCells;
6
7         public Monde(double seuil) {  // seuil : taux de cellules mortes
8                 tabCells = new boolean[MAXX][MAXY];
9                 for (int i=0 ; i<MAXX ; i++)
10                        for (int j=0 ; j<MAXY; j++)
11                                if (Math.random() > seuil)
12                                        tabCells[i][j] = true; // vivante
13                                else
```

```
14                                      tabCells[i][j] = false; // morte
15          }
16          public String toString() {
17                  String ch = "\nMonde␣generation␣" + noGen + "␣\n";
18                  for (int i=0; i<MAXX; i++) {
19                          for (int j=0; j<MAXY; j++)
20                                  ch += tabCells[i][j]?"*␣":".␣";
21                          ch += "\n";
22                  }
23                  ch += "\n";
24                  return ch;
25          }
26
27          /* Calcul du nombre de voisins d'une cellule.
28          * On suppose que le monde est un tore (chambre a air)
29          * On calcule le nombre de cellules vivantes dans un carre 3x3,
30          * puis on soustrait si la case centrale etait vivante. */
31          public int voisins(int x, int y) {
32                  int r = 0 ;
33                  for (int i = x-1 ; i <= x+1 ; i++) {
34                          int k = (i+MAXX) % MAXX; // pour gerer sortie du tableau
35                          for (int j = y-1 ; j <= y+1 ; j++) {
36                                  int l = (j+MAXY) % MAXY; // pour gerer sortie du
                                        tableau
37                                  if (tabCells[k][l]) r++ ;
38                          }
39                  }
40                  if (tabCells[x][y])  r-- ;
41                  return r;
42          }
43
44          /* Calcul de la generation suivante
45          * Pour chaque cellule, on calcule le nombre de voisins
46          * puis on applique les règles de vie et de mort de la simulation */
47          public void genSuiv() {
48                  boolean [][] tabCells2 = new boolean[MAXX][MAXY]; //variable
                        locale
49                  // init de tabCells2 a false
50                  for (int i=0 ; i<MAXX ; i++)
51                          for (int j=0 ; j<MAXY ; j++)
52                                  tabCells2[i][j]=false;
53                  for (int i=0 ; i<MAXX ; i++) {
54                          for (int j=0 ; j<MAXY ; j++) {
55                                  int n = voisins(i,j) ;
56                                  if (tabCells[i][j]) {
57                                          if ((n == 2) || (n == 3)) { tabCells2[i][j
                                                ]=true ; }
58                                  } else {
59                                          if (n==3) { tabCells2[i][j]=true ; }
60                                  }
61                          }
62                  }
63                  tabCells = tabCells2;  // on bascule sur le nouveau tableau
64                  noGen++;    // on incremente le no de generation
65          }
66  }
```

```java
public class JeuDeLaVie {
        public static void main(String[] args) {
                Monde m = new Monde(0.8);
                for (int i=0;i<10;i++) {
                        System.out.println(m);
                        try {Thread.sleep(2000); }
                        catch(InterruptedException e){
                                e.printStackTrace();
                        }
                        m.genSuiv() ;
                }
        }
}
```

Exercice 27 – 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 27.1 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 27.2 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 27.3 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

Q 27.4 Produire une exécution qui montre les cas d'erreur que vous traitez.

```java
public class Machin {
        private String nom;
        private int valeur;

        public Machin(String nom,int valeur){
                this.nom=nom;
                this. valeur = valeur;
        }
        public String getNom() {return nom;}
        public int getValeur() {return valeur;}

        public String toString() {
                return "(" + nom + "," + valeur + ")";
        }
}
```

```
16
17 public class Pile {
18         private Machin[] tab;
19         private int indSommet;  // indice du sommet de pile
20
21         public Pile(int tailleMax) {
22                 tab = new Machin[tailleMax];
23                 indSommet = -1;
24         }
25
26         public void empiler(Machin m){  // empiler en queue
27                 if (estPleine()) {
28                         System.out.println("la pile est pleine");
29                         return;
30                 } else {
31                         indSommet++;
32                         tab[indSommet] = m;
33                 }
34         }
35         public Machin depiler(){        // depiler si possible
36                 if (estVide()){
37                         System.out.println("tableau est vide");
38                         return null;
39                 } else {
40                         indSommet =  indSommet - 1;
41                         return tab[indSommet + 1];
42                 }
43         }
44
45         public boolean estVide(){return (indSommet < 0);}
46         public boolean estPleine(){return (indSommet >= tab.length);}
47
48         public String toString(){
49                 String s = "";
50                 if (estVide()) return "la pile est vide!\n";
51                 for (int i=0;i<=indSommet;i++)
52                         s += (" "+tab[i]);
53                 return s+"\n";
54         }
55 }
56
57 public class TestPileTableau{
58         public static void main(String[] args) {
59                 // creation des differents elements de type Machin
60                 Machin a = new Machin("a",1);
61                 Machin b = new Machin("b",2);
62                 Machin c = new Machin("c",3);
63                 Machin d = new Machin("d",4);
64                 Machin e = new Machin("e",5);
65                 Machin f = new Machin("f",6);
66                 Pile t = new Pile(10);
67                 System.out.println(t.toString());
68                 t.empiler(a);
69                 System.out.println("apres empilement de (a,1) :\n");
70                 System.out.println(T.toString()); // pile => 1 elements
71                 t.empiler(b);
```

```
72                  System.out.println("apres empilement de (b,2) :\n");
73                  System.out.println(T.toString()); // pile => 2 elements
74                  t.empiler(c);
75                  System.out.println("apres empilement de (c,3) :\n");
76                  System.out.println(T.toString()); // pile => 3 elements
77                  t.depiler();
78                  System.out.println("apres depilement de (c,3) :\n");
79                  System.out.println(T.toString()); // pile => 2 elements
80                  t.empiler(b);
81                  System.out.println("apres empilement de (b,2) :\n");
82                  System.out.println(T.toString()); // pile => 3 elements
83                  t.empiler(e);
84                  System.out.println("apres empilement de (e,5) :\n");
85                  System.out.println(T.toString()); // pile => 4 elements
86                  t.depiler();
87                  System.out.println("apres depilement :\n");
88                  System.out.println(T.toString()); // pile => 3 elements
89          }
90 }
```

Exercice 28 – Représentation mémoire d'objets composés et de tableaux

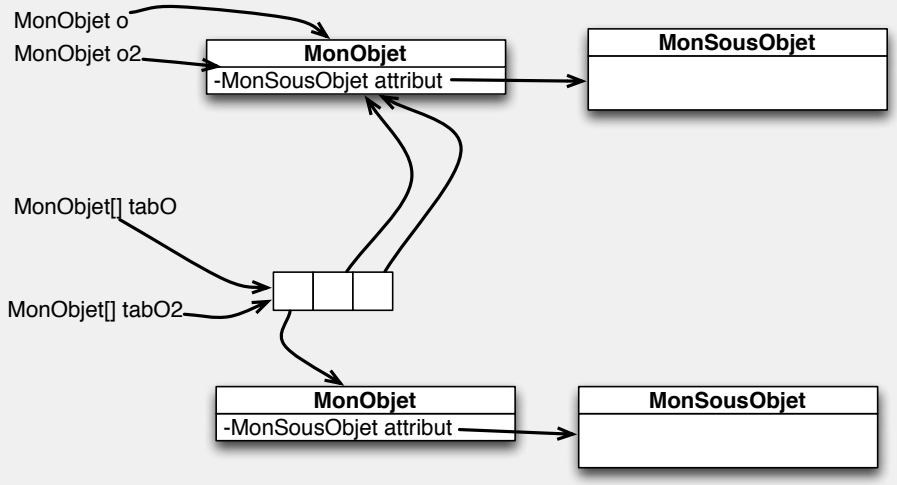
Soit une classe `Truc` possédant un attribut de type `AttrTruc` et un constructeur sans argument se chargeant de l'instanciation de l'attribut.

Q 28.1 Donner la représentation mémoire correspondant à l'exécution du code suivant :

```
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;
```

Combien y a-t-il d'instances de `Truc` et de `AttrTruc` ?

Ca donne une belle toile d'araignée...



Q 28.2 Application : `Truc` est en fait une `Voiture` dont la position est repérée par un `Point` initialisé en $(0,0)$ à la création d'une voiture. La voiture possède une méthode `void avancer()` qui lui permet de bouger selon une stratégie qui lui est propre. Le tableau de `Voiture` pourrait alors représenter une course opposant différentes voitures. La course se déroule alors tour par tour : à chaque tour, nous parcourons l'ensemble du tableau pour faire avancer les voitures. Au bout de 10 coups (par exemple) nous faisons le point pour savoir quelle voiture est aller le plus loin sur un circuit donné.
Pourquoi la structure mémoire de la question précédente pose problème dans ce cas là ?

Lorsque l'on fait avancer la première voiture ⇒ OK Mais dès que l'on fait avancer la seconde, la troisième bouge aussi ! ⇒ on va avoir l'impression que la voiture 3 se téléporte !

Q 28.3 Clonage & composition : Proposer une solution basée sur le clonage pour résoudre ce problème. Vous détaillerez le code à ajouter dans chaque classe.

Attention, il faut cloner les voitures, mais aussi les points qui servent à positionner la voiture

```
1 //Dans Point
2 public Point clone(){
3   return new Point(this.x,this.y);
4 }
5 //Dans Voiture
6 public Voiture clone(){
7   Voiture v2 = new Voiture();
8   v2.position = this.position.clone(); // ne pas oublier de cloner!
9   return v2;
10 }
11 // dans le main
12 Voiture[] course = new Voiture[3];
13 course[0] = new Voiture();
14 course[1] = v; course[2] = v.clone();
```

Dans les lignes 7+8 on créée une instance de point qui ne sert à rien... Si l'on souhaite éviter le problème :

```
1 //Dans Voiture
2 protected Voiture(Point p){ // attention a ne pas changer la vision client de l'
      objet... => failles de securite
3   this.position = p;
4 }
5 public Voiture clone(){
6   return new Voiture(this.position.clone());
7 }
```

Q 28.4 Clonage du tableau : Comparer les représentations mémoire de `course2` et `course3` par rapport à l'instance `course`. Ces représentations sont-elles satisfaisantes ? Sinon, proposer des modifications dans le code.

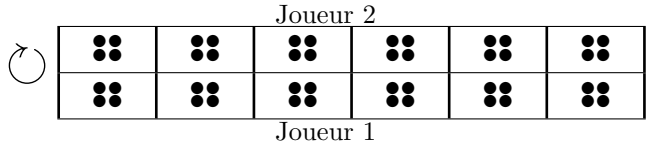
```
1 Voiture[] course = new Voiture[3];
2 course[0] = new Voiture(); course[1] = new Voiture(); course[2] = new Voiture();
3 Voiture[] course2 = course;
4 Voiture[] course3 = new Voiture[3];
5 for(int i=0; i<3; i++) course3[i] = course[i];
```

`course2` ne sert à rien, `course3` copie le tableau mais pas les instances de voitures !

```
1 for(int i=0; i<3; i++) course3[i] = course[i]; // pas terrible
2 for(int i=0; i<3; i++) course3[i] = course[i].clone(); // OK
```

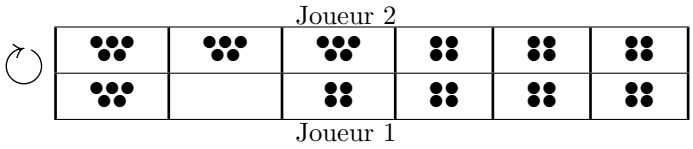
Exercice 29 – Awélé

Règle du jeu :
L'awélé est un jeu à deux joueurs qui se joue avec un plateau formé de 12 cases et de 48 billes réparties au début du jeu à raison de 4 par case (voir figure).

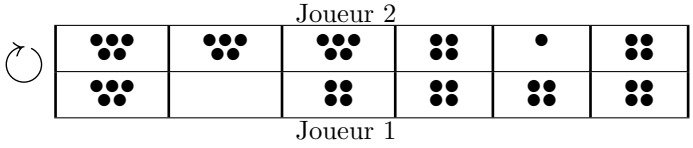


Chaque joueur possède les 6 cases Nord ou les 6 cases Sud du jeu. Les cases sont attribuées au joueur en début de la partie. Les cases du plateau sont toujours parcourues dans le sens indiqué sur la figure (sens des aiguilles d'une montre sur la figure).
Le jeu se joue de la façon suivante :

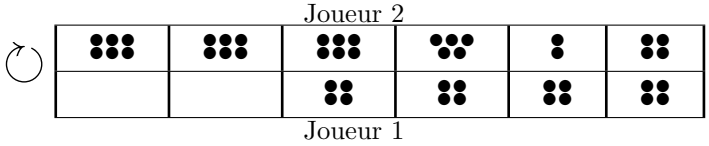
– Un joueur choisit l'une de ses propres cases, puis répartit son contenu en déposant une bille et une seule dans chacune des cases consécutives. Par exemple, si le joueur 1 choisit la case 5 (deuxième case en partant de la gauche des cases au Sud) pour jouer, cela donne le résultat suivant :



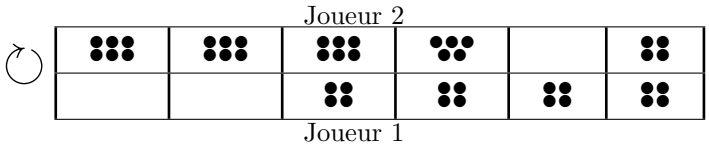
– Un joueur marque un coup gagnant lorsqu'il dépose sa dernière bille (du coup en cours) dans une case adverse qui ne contient qu'une seule bille. Dans cette situation, le joueur gagne alors ces deux billes et les retire du plateau.
Situation avant le coup gagnant :



Le joueur 1 choisit de jouer sa case 6, on obtient alors :



Le joueur 1 retire alors deux billes du plateau (il gagne deux billes) qui devient :



Le joueur peut aussi cumuler des gains si la case précédente a aussi deux billes à la fin du coup. Il en gagne alors deux de plus. Les joueurs jouent tour à tour jusqu'à ce que le nombre total de billes sur le plateau soit inférieur à 12. Le gagnant est le joueur qui a remporté le plus de billes.

Le jeu électronique :

Le jeu électronique de l'Awélé se joue lui aussi à deux joueurs. Le plateau est affiché à l'écran. Pour manipuler les billes, les joueurs désignent les cases du plateau soit en donnant le numéro de la case, soit en cliquant à la souris. Le jeu électronique contrôle le bon déroulement du jeu. Au début du jeu, il affecte les cases du plateau aux joueurs. Il fait jouer les deux joueurs à tour de rôle, comptabilise les points de chacun et désigne le gagnant. Pour jouer, un joueur désigne la case dont il veut répartir les billes. Le jeu répartit automatiquement ses billes dans les cases suivantes et gère les gains en attribuant les billes au joueur gagnant.

Q 29.1 Déterminer les classes nécessaires à la modélisation du jeu électronique en précisant les attributs et les méthodes de chacune des classes.

NB : il existe bien sûr d'autres modélisations
Classes : Case, Jeu, Joueur, Awele (comprend le main)
Case :
Variables d'instance : nbBilles, proprio ;
Constructeurs : Case(), Case(Joueur j) ;
Accesseurs : getNbBilles (), getProprio() ;
Modifieurs : setNbBilles(int n), setProprio(Joueur j) ;
Affichage : toString() ;
Méthodes : ajouteBilles(int i), retireBilles(int i), estVide() ;
Joueur :
Variables d'instance : int num ou String nom, int gain ;
Constructeurs : Joueur(), Joueur(int n) ;
Accesseurs : getNum(), getGain() ;
Modifieurs : setNum(int n), setGain(int g) ;
Affichage : toString() ;
Méthodes : ajouterGain(int i) ;
Jeu :
Variables d'instance : plateau, joueurs, aLaMain ;
Constructeur : Jeu() ;
Accesseurs : getALaMain() ;
Modifieurs : ;
Affichage : toString() ;
Méthodes : choisirCase(), jouerCoup(int n), changerJoueur(), jouer(), voirSiFin(), trouverGagnant() ;
Awele :
Méthode : main
Il faut se poser des questions du style :
Qui détermine le gagnant ? (le jeu ou le main ?)
Qui connaît quoi ? (le jeu connaît-il les joueurs ? le joueur connaît-il le jeu ?)
Après la modélisation, on se posera les questions du style :
Quelle structure de données pour le plateau, pour les joueurs ?
- un tableau
- une liste chaînée (ce dernier choix aura pour conséquence d'avoir une variable d'instance suivant dans la classe Case)

Jouer, c'est : choisir une case, joueur le coup, voir si la partie est finie et déterminer le gagnant quand la partie est finie (ceci peut être fait par le main si Joueur retourne vrai si fini).
VoirSiFin, c'est : soit compter le nombre de billes restant sur le plateau (boucle à chaque coup), soit compter le nombre de billes gagnées par les deux joueurs (somme à chaque coup), soit avoir une variable pour mémoriser le nombre de billes restantes sur le plateau (mise à jour si gain)
Main, c'est : créer un nouveau jeu, lancer le jeu (jouer),
Choix pour représenter le plateau :
(unknown char) Liste chaînée (unknown char)Tableau

Avantages (unknown char) Circulaire comme dans le jeu (unknown char) Facile pour repérer dans quel camp
(unknown char) on est
(unknown char) Facile pour trouver la case désignée
(unknown char) Facile pour voir le contenu du précédent
(unknown char) Accès direct à la 1ère case

Inconvénients (unknown char) Difficile de voir le contenu (unknown char) Calcul pour assurer
(unknown char) de la case précédente (unknown char) l'aspect circulaire du jeu
(unknown char) Boucle systématique pour aller (unknown char)
(unknown char) à la 1ère case (unknown char)
La programmation est donné dans le thème sur les tableaux
Se reporter au thème précédent

Q 29.2 Définir une classe `Case` caratérisée par un nombre de billes (`nbBille`), et son propriétaire (`prop` de type `Joueur`). Écrire un constructeur sans paramètre et un constructeur ayant comme paramètre un joueur, ainsi qu'une méthode `toString()`.

```
1 public class Case {
2         private int nbBille;
3         private Joueur prop;
4 
5         public Case() {
6                 nbBille=4;
7         }
8         public Case(Joueur j){
9                 this();
10                prop=j;
11        }
12 
13        public void setNbBille (int n){
14                nbBille=n;
15        }
16        public int getNbBille (){   return nbBille;   }
17        public void ajoutBille(int i){  nbBille+=i;   }
18        public Joueur getProp(){  return prop;  }
19        public void setProp (Joueur j){  prop=j;  }
20 
21        public String toString(){
22                return ("Cette␣Case␣a␣" + getNbBille()+
23                        "␣billes,␣son␣proprietaire␣est␣"+getProp()+"\n");
24        }
25 }
```

Q 29.3 Définir la classe `Joueur` qui se caractérise par le numéro et le gain du joueur. Écrire un constructeur sans paramètre et un constructeur ayant comme paramètre le numéro du joueur.

```
1 public class Joueur {
2         private int num;
```

```
3         private int gain;
4
5         public Joueur() {
6                 gain=0;
7         }
8         public Joueur(int i) {
9                 this();
10                num=i;
11        }
12        public int getNum(){return num; }
13        public void setNum(int i){ num=i; }
14        public void ajoutGain(int n){ gain +=n; }
15        public int getGain(){ return gain; }
16
17        public String toString(){
18                return "Joueur␣"+num+"␣a␣actuellement␣"+gain+"␣billes";
19        }
20}
```

Q 29.4 Définir une classe `Jeu` caractérisée par un plateau de cases, un tableau de joueurs et un indicateur pour savoir qui a la main (de type booléen). Écrire un constructeur qui initialise la partie comme indiquée sur la première figure et la méthode `toString()` pour visualiser le plateau.

```
1public class Jeu {
2         private Case [] plateau;
3         private Joueur [] joueurs;
4         private Joueur aLaMain;
5
6         public Jeu() {
7                 plateau=new Case[12];
8                 joueurs=new Joueur[2];
9                 joueurs[0]=new Joueur(1);
10                joueurs[1]=new Joueur(2);
11
12        // Initialisation du plateau, les cases sont repartis aux 2 joueurs
13        // c'est le constructeur de Case qui initialise le nbBilles a 4
14                for (int i=0; i<6;i++){
15                        plateau[i]=new Case (joueurs[0]);
16                }
17                for (int i=6; i<12;i++){
18                        plateau[i]=new Case (joueurs[1]);
19                }
20        }
21        public String toString(){
22                String s="Plateau␣de␣jeu␣\n␣Premier␣joueur\n";
23                for (int i=0; i<6; i++){
24                        s=s+plateau[i].toString();
25                }
26                s=s+ "Deuxieme␣joueur\n";
27                for (int i=6; i<12; i++){
28                        s=s+plateau[i].toString();
29                }
30                return s;
```

```
31          }
32 }
```

Q 29.5 Compléter la classe `Jeu` pour, dans le constructeur, définir le joueur qui a la main. Écrire la méthode changeJoueur() qui donne la main à l'autre joueur et une méthode `choisirCase()` qui demande au joueur qui à la main de choisir ue case et qui valide ce choix. Ecrire une méthode `unCoup(int i)` qui réalise le coup à partir de la case numéro i.

Q 29.6 Compléter la classe `Jeu` en y ajouter une méthode `finDePartie()` qui affiche si on est en fin de partie et une méthode `boolean jouer()` qui assure le bon déroulement de la partie tant que la partie n'est pas finie.

```
1 public class Jeu {
2         private Case [] plateau;
3         private Joueur [] joueurs;
4         private Joueur aLaMain;
5 
6         public Jeu() {
7                 plateau=new Case[12];
8                 joueurs=new Joueur[2];
9                 joueurs[0]=new Joueur(1);
10                joueurs[1]=new Joueur(2);
11                // Initialisation du plateau, les cases sont donnees aux deux
                      joueurs
12                // remarque : c'est le constructeur de Case qui initialise le
                      nbBilles à 4
13                for (int i=0; i<6;i++){
14                        plateau[i]=new Case (joueurs[0]);
15                }
16                for (int i=6; i<12;i++){
17                        plateau[i]=new Case (joueurs[1]);
18                }
19                // definition du joueur qui commence
20                aLaMain=joueurs [(int)(Math.random()*2)];
21        }
22 
23        public Joueur getJoueurCourant(){
24                return aLaMain;
25        }
26        public Joueur changeJoueur(Joueur j){
27                if (j==joueurs[0]) return joueurs[1];
28                else return joueurs[0];
29        }
30        public String toString(){
31                String s="Plateau de jeu \n Premier joueur\n";
32                for (int i=0; i<6; i++){
33                        s=s+plateau[i].toString();
34                }
35                s=s+ "Deuxième joueur\n";
36 
37                for (int i=6; i<12; i++){
38                        s=s+plateau[i].toString();
39                }
40                return s;
```

```
41          }
42
43          public int choixCase (){
44                  System.out.println(this);
45                  System.out.println("Vous␣etes␣le␣joueur␣"+aLaMain+"\n");
46                  System.out.println("Quelle␣case␣voulez␣vous␣?␣");
47                  int i= Integer.valueOf(ES.lire()).intValue();
48                  while (plateau[i].getProp()!=getJoueurCourant()){
49                          System.out.println("cette␣case␣n'est␣pas␣a␣vous,␣rejouez")
                                ;
50                                  i= Integer.valueOf(ES.lire()).intValue();
51                  }
52                  return i;
53          }
54
55          public void unCoup (int p){
56                  System.out.println ("Joueur␣"+aLaMain+ "case␣"
57                          +p+"\n");
58                  int n=plateau[p].getNbBille();
59                  // on enlève les billes de cette case
60                  plateau[p].setNbBille(0);
61                  int i;
62                  // on repartit les billes
63                  // Si p+n <12 alors on n'aura pas besoin d'en repartir que les
                        cases du debut
64                  if (p+n<12){
65                          for ( i= p+1; i<=p+n;i++){
66                                  plateau[i].ajoutBille(1);
67                          }
68                  }
69                  else {
70                          int k=p+n-12;
71                          for (i=p+1;i<12;i++){
72                                  plateau[i].ajoutBille(1);
73                          }
74                          for (i=0;i<=k;i++){
75                                  plateau[i].ajoutBille(1);
76                          }
77                  }
78                  System.out.println(this);
79                  System.out.println("Valeur␣de␣i␣"+i);
80                  // on regarde si le joueur gagne des billes
81                  // La derniere case touchee a ete la case i-1,
82                  // on regarde si elle contient 2 billes et si elle est a l'
                        adversaire
83
84                  if (i-1==0) i=12;
85                  while ((i-1>=0)&&(plateau[i-1].getNbBille()==2)&&
86                                  (plateau[i-1].getProp()!=aLaMain)) {
87                          int truc=i-1;
88                          System.out.println("Joueur␣" +aLaMain +
89                                  "Valeur␣de␣i-1␣" + truc +"nombre␣de␣billes␣"+
90                                  plateau[i-1].getNbBille()+'\n');
91                                  aLaMain.ajoutGain(2);
92                          System.out.println("JoueurGagnant" + aLaMain);
93                          plateau[i-1].setNbBille(0);
```

```
94                          i--;
95                  }
96          }
97
98          public boolean jouer(){
99                  boolean fini=false;
100                 int p=choixCase();
101                 unCoup(p);
102                 //on regarde si la partie est finie
103                 int somme=0;
104                 for ( int i=0; i<12; i++){
105                         somme+=plateau[i].getNbBille();
106                 }
107                 if (somme<12) fini=true;
108                 aLaMain= changeJoueur(aLaMain);
109                 return fini;
110         }
111
112         public void finDePartie (){
113                 if (joueurs[0].getGain()>joueurs[1].getGain())
114                         System.out.println("Le joueur "+ joueurs[0]+
115                                 " est le gagnant");
116                 else
117                         if(joueurs[0].getGain()==joueurs[1].getGain())
118                         System.out.println("Les joueurs "+
119                                  joueurs[0]+ joueurs[1]+
120                                  " sont les gagnants");
121                         else
122                         System.out.println("Le joueur "+
123                                 joueurs[1]+" est le gagnant");
124         }
125 }
```

Q 29.7 Définir la classe `Awele` qui lance une nouvelle partie et qui gère la fin de partie.

```
1 public class Awale {
2         public static void main (String [] a){
3                 Jeu partie=new Jeu();
4                 System.out.println(partie.toString());
5                 boolean fin =false;
6                 while (!fin) fin=partie.jouer();
7                         partie.finDePartie();
8         }
9 }
```

Quizz 7 – Tableaux

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

```
1 double [] tabD=new double [8];
2 double [] tabD2={1.0,2,3,4,5,6,7,8};
```

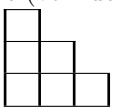
QZ 7.2 Comment peut-on obtenir le nombre d'éléments du tableau tabD ?

tabD.length

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

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

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).



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

```
1 int [][] tabTriangle=new int [3][];
2 for(int i=0;i<3;i++) tabTriangle[i]=new int [i+1];
3 // Autres possibilites
4 int [][] tabTriangle2={new int[1],new int [2], new int [3]};
5 int [][] tabTriangle3={{1},{2,2}, {3,3,3}};
```

QZ 7.5 On considère la classe `Bouteille` vue dans l'exercice 17 page 32. Créez un tableau de 2 bouteilles, la première bouteille aura un volume de 3 litres et la deuxième de 1,5 litre.

```
1 Bouteille [] tabB=new Bouteille[2];
2 tabB[0]=new Bouteille(3);
3 tabB[1]=new Bouteille();
4 // ou bien directement :
5 Bouteille [] tabB2={new Bouteille(3),new Bouteille()};
```

Quizz 8 – Tableaux d'objets

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

Qu'affichent les instructions suivantes ?

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

System.out.println(tabSimple[3]) ; affiche 0.
System.out.println(tabObject[3]) ; affiche null, car le handle tabObject[3] ne fait référence à aucun objet.
Le 1er : System.out.println(tabObject[3].toString()) ; provoque une erreur à l'exécution, car le handle tabObject[32] ne fait référence à aucun objet, il n'y a donc pas de méthode toString() (Exception NullPointeurException).
Le 2ieme : System.out.println(tabObject[3].toString()) ; affiche le toString()

Quizz 9 – Mot-clé `final`

Rappel : En Java, les constantes sont des variables dont on ne peut pas changer la valeur. Ces constantes sont soient initialisées lors de la compilation et ne changeront jamais, soient initialisées lors de l'exécution du programme et ne changeront plus ensuite. Le mot-clé `final` permet de déclarer une constante. Par convention, le nom des constantes s'écrit tout en majuscule.

```
1 final int CST1=10;
2 final int CST2;
3 CST1=15;
4 CST2=(int)(Math.random()*20);
5 CST2=25;
```

QZ 9.1 Est-il correcte de ne pas avoir initialisé la constante `CST2` lors de sa déclaration ?

Oui, en Java, une constante peut être initialisée après sa déclaration.

QZ 9.2 Quelles sont les instructions qui produisent une erreur ?

CST1=15 ; provoque une erreur car la valeur d'une constante initialisée ne peut être modifiée.
CST2=25 ; même raison.

CST2=(int)(Math.random()*20) ; ne provoque pas d'erreur car une constante peut avoir une valeur différente à chaque exécution du programme.

4 Composition, copie d'objets

Objectifs

- Composition d'objets
- Copie d'objets
- Constructeur de copie (surcharge de constructeurs)

Exercices

Propositions :
En TD : exercice 30, exercice 31 + Quizz page 80
En TME : exercice 33 (Tracteur)
Bien faire comprendre aux étudiants que lorsque une classe `contient` des objets, il faut faire une copie de ces objets.

Exercice 30 – Pion

Soient les classes suivantes :

```
1 public class Pion {
2         private String nom ;
3         private double posx ; // position du pion
4
5         public Pion(String n) {
6                 nom=n;
7                 posx=Math.random();
8         }
9         public void setNom(String n) { nom=n; }
10        public String getNom() { return nom; }
11 }
12 //=============================================
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 30.1 Que s'affiche-t-il ?

Affiche "Dormeur"

Q 30.2 Proposer une ou des solutions pour éviter le problème précédent.

Solution 1 : Faire une méthode d'instance copie() :

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

Puis dans le main :

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

Solution 2 : faire un « constructeur de copie »

```
public Pion (Pion r) {
        this(r.nom);
        posx=r.posx; /* ou this.posx=r.posx; */
}
```

Puis dans le main :
autrePion=new Pion (unPion);
Constructeur par copie : constructeur qui prend en paramètre 1 objet de même type, et dont le but est de faire une copie des variables d'instance => si la variable d'instance à copier est un objet, il faut faire appel au constructeur par copie de l'objet (Voir exo Tracteur et Machine a Schtroupfer)

Exercice 31 – Feu tricolor

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

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

Q 31.1 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.

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

```
class FeuTricolor {
    private Lampe verte,orange,rouge;
    public FeuTricolor() {
        verte=new Lampe();
        orange=new Lampe();
        rouge=new Lampe();
        // ou bien : this(new Lampe(),new Lampe(),new Lampe());
    }
    public FeuTricolor(Lampe v,Lampe o,Lampe r) {
        verte=v;
        orange=o;
```

```
12          rouge=r;
13      }
14 }
15 // Creation des objets avant, puis passage en parametre
16          Lampe l1=new Lampe();
17          Lampe l2=new Lampe();
18          Lampe l3=new Lampe();
19          FeuTricolor ft1=new FeuTricolor(l1,l2,l3);
20          FeuTricolor ft2=new FeuTricolor(new Lampe(),new Lampe(),new Lampe());
```

Q 31.2 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 }
```

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

Q 31.3 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);
```

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

Exercice 32 – 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 possiblité 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 }
```

BUG 2012

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

Q 32.1 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 32.2 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 32.3 Ecrire la méthode `void divorcer()` qui fait divorcer cette personne si c'est possible.

Q 32.4 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
```

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

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

        public String toString() {
                String civilite;
                if (conjoint == null)
```

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

Exercice 33 – Tracteur (composition d'objets et copie d'objets)

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

Q 33.1 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()`.

```
public class Roue{
        private int diametre;

        public Roue(int diam){ diametre = diam; }
        public Roue(){ this(60); }
        public String toString(){
                return "Roue de diametre: " + diametre + "\n";
        }
}
```

Q 33.2 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.

Voir le main ci-dessous.

Q 33.3 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)`.

```
public class Cabine{
        private int volume;
        private String couleur;

        public Cabine(int vol, String coul){
                volume = vol;
                couleur = coul;
        }
        public String toString(){
                return "Cabine de Volume : " + volume + " et de Couleur : " +
                    couleur;
        }
        public void setCouleur(String couleur) {
                this.couleur=couleur;
        }
}
```

Q 33.4 Ajouter dans la méthode main la création d'une cabine bleue.

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

Q 33.5 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.

```
1 public class Tracteur{
2         private Cabine cab;
3         private Roue r1,r2,r3,r4; //On peut faire aussi avec un tableau de 4 roues
4         public Tracteur(Cabine c,Roue petite1,Roue petite2,Roue grande1,Roue
              grande2){
5                 cab = c;
6                 r1=petite1; r2=petite2; r3=grande1; r4=grande2;
7         }
8
9         public Tracteur(){
10                this(new Cabine(400,"verte"),new Roue(),new Roue(),
11                        r3=new Toue(150),new Roue(150));
12        }
13        public String toString(){
14                return "Tracteur :\n"+ cab + "\nLes roues :\n"+r1 + r2 + r3 + r4;
15        }
16
17        public void peindre(String couleur) {
18                cab.setCouleur(couleur);
19        }
20 }
```

Q 33.6 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 33.7 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.

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

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

```
1 public Roue(Roue r){ this(r.diametre); }
2 public Cabine(Cabine c) { this(c.volume,c.couleur); }
3 public Tracteur(Tracteur t) {
```

```
4         this(new Cabine(t.cab), new Roue(t.r1),new Roue(t.r2),new Roue(t.r3),new
              Roue(t.r4));
5 }
```

Exercice 34 – Machine à schtroumpfer (copie d'un objet composite)

On veut construire une machine à schtroumpfer les alipois. Une telle machine a une couleur et est composée d'une partie avant et d'une partie arrière. La partie avant est composée de deux pièces de type A et d'une pièce de type B. La partie arrière est composée d'une pièce de type A et d'une pièce de type C. Une pièce A a un numéro de série et est formée d'une matière (type `String`). Une pièce B a un numéro de série et a une longueur (type `double`). Une pièce C a un numéro de série et est de luxe ou ordinaire (booléen). Voici les classes `PieceA`, `PieceB`, et `PieceC` telles qu'elles sont fournies :

```
1 // classe PIECE A
2 public class PieceA {
3         private int noSerie;
4         private String matiere;
5
6         public PieceA(int n) { noSerie=n; matiere="bois"; }
7         public PieceA(int n,String m) { noSerie=n; matiere=m; }
8
9         public String toString() {
10                return "pieceA␣no␣"+ noSerie+ "␣en␣"+ matiere+"\n";
11        }
12 }
13
14 // classe PIECE B
15 public class PieceB {
16        private int noSerie;
17        private double longueur;
18
19        public PieceB(int n) { noSerie=n; }
20        public PieceB(int n,double l) { noSerie=n; longueur=l; }
21
22        public void setLongueur(double l) { longueur=l; }
23
24        public String toString() {
25                return "pieceB␣no␣"+ noSerie+ "␣de␣longueur␣"+ longueur +"\n";
26        }
27 }
28
29 // classe PIECE C
30 public class PieceC {
31        private int noSerie;
32        private boolean luxe;
33
34        public PieceC(int n) { noSerie=n; }
35        public PieceC(int n,boolean b) { noSerie=n; luxe=b; }
36
37        public void setLuxe(boolean b) { luxe=b; }
38
39        public String toString() {
40                String s="pieceC␣no␣"+ noSerie;
41                if (luxe) s +="␣de␣luxe␣\n";
```

```
42                  else s +="␣ordinaire\n";
43                  return s;
44          }
45 }
```

NB : Pour le TME : les sources des classes des Pieces sont téléchargeables à partir du site de l'UE

Q 34.1 Écrire les classes `Avant`, `Arriere` et `Machine` (constructeurs, toString...)

Q 34.2 Écrire une méthode main qui crée une machine `m1`. Il faudra évidemment créer auparavant toutes les pièces, et l'avant et l'arrière dont elle est formée. L'afficher à l'écran.

Q 34.3 Dans cette méthode main déclarer une machine `m2` qui sera la copie de `m1`, par l'instruction `m2=m1;` Modifier ensuite la couleur de `m2` et la longueur de la pièce B de son avant. Afficher `m1` et `m2`. Que s'affiche-til ? Expliquez le problème et ce qui s'est passé.

Q 34.4 Pour éviter le problème précédent, on va créer dans chaque classe un constructeur de copie. Un tel constructeur prend en argument un objet de la classe et crée un clone de cet objet (qui est donc dupliqué entièrement). Ecrire des constructeurs de copie pour chaque classe. NB : il existe une alternative au constructeur par copie qui est couramment utilisée en JAVA, le clonage. Proposer une méthode `clone` sans argument qui retourne une nouvelle instance de l'objet courant.

Q 34.5 Créer maintenant une copie `m3` de la machine `m2` avec son constructeur de copie, puis modifier à nouveau la couleur et la longueur de la pièce B de l'avant. Affichez `m2` et `m3`. Que s'affiche-t-il maintenant ? Expliquez.

```
1 // Constructeur de copie de PieceA
2         public PieceA(PieceA p) {
3                 this(p.noSerie,p.matiere); // Appel du constructeur de PieceA à
                      deux paramètres
4         }
5 // clonage dans la classe PieceA
6         public PieceA clone(){
7                 return new PieceA(noSerie, matiere);
8         }
9
10 // Constructeur de copie de PieceB
11        public PieceB(PieceB p) {
12                this(p.noSerie, p.longueur); // Appel du constructeur de PieceB à
                      deux paramètres
13        }
14
15 // Constructeur de copie de PieceC
16        public PieceC(PieceC p) {
17                this(p.noSerie,p.luxe);
18        }
```

```
1 public class Arriere {
2         private PieceA pA;
3         private PieceC pC;
4
5         public Arriere(PieceA p1, PieceC p2) { pA=p1;pC=p2; }
6
7         // Constructeur de copie de Arriere
8         public Arriere(Arriere a) {
9                 pA=new PieceA(a.pA); //appel du constructeur de copie de PieceA
```

```
10                  pC=new PieceC(a.pC); //appel du constructeur de copie de PieceC
11          }
12          public String toString() { return "" + pA + pC; }
13  }
```

```
1 public class Avant {
2           private PieceA comp1,comp2;
3           private PieceB comp3;
4
5           public Avant(PieceA p1, PieceA p2, PieceB p3) {
6                   comp1=p1;comp2=p2;comp3=p3;
7           }
8
9           // Constructeur de copie de Avant
10          public Avant(Avant a) {
11                  comp1=new PieceA(a.comp1); //appel du constructeur de copie de
                        PieceA
12                  comp2=new PieceA(a.comp2); //appel du constructeur de copie de
                        PieceA
13                  comp3=new PieceB(a.comp3); //appel du constructeur de copie de
                        PieceB
14          }
15
16          // Pour permettre la modification de la longueur de comp3
17          public void modifierLongueurPieceB(double l) {
18                  comp3.setLongueur(l);
19          }
20          public String toString() {
21                  return "" + comp1 + comp2 + comp3;
22          }
23  }
```

```
1 public class Machine {
2           private Avant avant;
3           private Arriere arriere;
4           private String coul;
5
6           public Machine(Avant av, Arriere ar, String c) {
7                   avant=av;
8                   arriere=ar;
9                   coul=c;
10          }
11
12          // Constructeur de copie de Machine
13          public Machine (Machine m) {
14                  avant=new Avant(m.avant); //appel du constructeur de copie de
                        Avant
15                  arriere=new Arriere(m.arriere); //appel du constructeur de copie
                        de Arriere
16          }
17
```

```
18          // Pour permettre la modification de la longueur de avant
19          public void modifierLongueurAvant(double l) {
20                  avant.modifierLongueurPieceB(l);
21          }
22
23          public void setCoul(String c) {coul=c; }
24
25          public String toString() {
26                  return "machine␣"+ coul + ":\navant␣:␣\n" +avant + "\narriere␣:␣\n
                        "+arriere;
27          }
28  }
```

```
1 public class TestMachine {
2           public static void main(String [] args) {
3                   PieceA a1, a2, a3;
4                   PieceB b1;
5                   PieceC c1;
6
7                   a1=new PieceA(1,"fer");
8                   a2=new PieceA(2, "coton");
9                   b1=new PieceB(7,5.8);
10
11                  Avant avant=new Avant(a1,a2,b1);
12                  a3=new PieceA(3);
13                  c1=new PieceC(8,true);
14                  Arriere arriere=new Arriere(a3,c1);
15
16                  // Creation de la machine
17                  Machine m1= new Machine(avant,arriere, "rouge");
18                  System.out.println("machine␣m1␣:␣"+ m1);
19
20                  // maintenant on souhaite creer une nouvelle machine
21                  // avec les memes caracteristiques (un clone).
22                  // Une mauvaise solution est :
23                  Machine m2=m1;
24                  m2.setCoul("verte");
25                  m2.modifierLongueurAvant(10.38);
26                  System.out.println("m1␣apres␣changement␣de␣couleur␣de␣m2␣:\n"+m1);
27
28                  System.out.println("la␣copie␣m2␣:␣"+ m2);
29                  // m1 a aussi changé !!!! Donc en modifiant m2,
30                  // on a aussi modifié m1.
31                  // m1 et m2 accèdent au même objet !
32                  // Maintenant on veut cloner la machine m2,
33                  // on utilise le constructeur de copie de la machine
34                  // qui appelle les constructeurs de copie de toutes les pièces
35
36                  Machine m3=new Machine(m2);
37                  m3.setCoul("bleu");
38                  m3.modifierLongueurAvant(3.5);
39                  System.out.println("m2␣n'a␣pas␣change␣:\n"+m2);
40                  System.out.println("la␣copie␣m3␣:␣\n"+ m3 );
41          }
```

```
42 }
```

Exercice 35 – Course avec équipe (composition d'objets)

On propose ici d'étendre l'exercice 20 intitulé "La course de relais 4 fois 100m" pour y ajouter le concept d'équipe de coureur.

Remarque : ici l'équipe est sans tableau, car c'est un relais de 4 coureurs

Q 35.1 Ecrire une classe `Equipe` (composée de quatre coureurs) qui comprend :
- `ville` de type `String` qui est le nom de la ville de l'équipe.
- les 4 coureurs de type `Coureur`.

Y ajouter les méthodes suivantes :
- un constructeur `Equipe(String v)` qui crée une équipe pour la ville v de quatre coureurs.
- la méthode `public String toString()` qui retourne une chaîne de caractères décrivant l'équipe avec ses coureurs. On appellera pour ce faire la méthode `toString()` de la classe `Coureur` sur les objets `Coureur` de cette équipe.

Q 35.2 Tester ces méthodes en créant dans le main une équipe et en l'affichant.

Q 35.3 Ecrire une méthode `double tempsFinal()` qui rend le temps mis par cette équipe pour courir le 400m.

Q 35.4 Ecrire une méthode `Equipe rencontrer (Equipe e)` qui retourne l'équipe gagnante (c'est-à-dire la référence de l'équipe gagnante) dans la compétition opposant cette équipe (receveuse du message) à l'équipe passée en paramètre. L'équipe gagnante est celle qui a mis le moins de temps à courir les 400m. Si les temps sont égaux, la méthode retourne null.

Q 35.5 Créer dans la méthode main deux équipes `e1` et `e2`, les faire courir et afficher les temps mis par chaque équipe ainsi que le résultat de la compétition.

```
1 public class Equipe {
2         private String ville;
3         private Coureur coureur1, coureur2, coureur3, coureur4;
4
5         public String getVille() {return ville;}
6         public void setVille(String  v) {this.ville = v;}
7
8         public Equipe(String v) {
9                 ville = v;
10                coureur1 = new Coureur();
11                coureur2 = new Coureur();
12                coureur3 = new Coureur();
13                coureur4 = new Coureur();
14        }
15        public String toString() {
16                return "\nListe␣des␣coureurs␣de␣la␣ville␣" + ville + "␣:\n"
17                        +coureur1.toString()    +coureur2.toString()
18                        +coureur3.toString()    +coureur4.toString();
19        }
20        public double tempsFinal() {
```

```
21                  return  coureur1.getTempsAu100()+coureur2.getTempsAu100()
22                          +coureur3.getTempsAu100() +coureur4.getTempsAu100();
23          }
24          public Equipe rencontrer(Equipe e) {
25                  if( this.tempsFinal() == e.tempsFinal())
26                          return null;
27                  else {
28                          if (this.tempsFinal() < e.tempsFinal()) return this;
29                          else return e;
30                  }
31          }
32 }
```

Et on ajoute dans la méthode main :

```
1 System.out.println("ET MAINTENANT, LA COMPETITION :");
2
3 Equipe e1=new Equipe("Trifouilly-les-Oies");
4 Equipe e2=new Equipe("La-Fontaine-aux-Loups");
5
6 Equipe gagnante = e1.rencontrer(e2)
7 if (gagnante != null){
8         System.out.print("L'equipe gagnante est celle de " + gagnante.getVille());
9         System.out.println( "!!!!\n");
10 } else {
11         System.out.println("Les deux equipes sont ex aequo!!!!\n");
12 }
```

Quizz 10 – 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 10.1 La classe `A` contient-elle un constructeur ?

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

QZ 10.2 Combien y-a-t-il d'objets créés ?

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

QZ 10.3 Combien y-a-t-il de références (appelées aussi *handles*) utilisés ?

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

QZ 10.4 Que se passe-t-il si on rajoute l'instruction `a3=null;` ? `a2=null;` puis `a1=null;` ?

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

5 Variables et méthodes de classes

Objectifs

- Variables et méthodes de classe
- `static`
- Classe enveloppe

Propositions :
En TD : exercice 36, exercice 37 (Numero), puis choisir parmi les autres en fonction du niveau des étudiants
En TME : exercice 38 (Cône de révolution), puis au choix

Exercices

Exercice 36 – Membres d'instance ou de classe

Q 36.1 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).

nom	variable instance	VI
aboyer()	méthode instance	MI
siteWebSurChiens	variable classe	VC
siteWebDuChien	variable instance	VI
siteWebSPA	variable classe	VC
couleurDuPoil	variable instance	VI
courir()	méthode instance	MI
chercherLivreSurChiens()	méthode classe	MC
vidéothèqueSurChiens	variable classe	VC
metsPrefere	variable instance	VI
bibliographieSurChiens	variable classe	VC
dateNaissance	variable instance	VI
manger()	méthode instance	MI
regarderDVD()	cela depend	

regarderDVD() est une MI si c'est le chien qui regarde le DVD et une MC si c'est l'utilisateur qui regarde un DVD sur les chiens.

Exercice 37 – 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 37.1 Quel est le nom de la variable de classe ? Comment la reconnaît-on ?

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

Q 37.2 Pourquoi la variable cpt a-t-elle été initialisée ?

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

Q 37.3 Quel est l'affichage obtenu par l'exécution des lignes 4, 6, 8 et 9 du programme suivant :

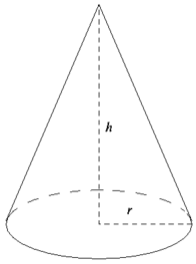
```
1 public class TestTruc{
2         public static void main (String[] args){
3                 Truc n1=new Truc();
4                 System.out.println(n1.getCpt());
5                 Truc n2=new Truc(25);
6                 System.out.println(n1.getCpt()+"␣"+n2.getCpt());
7                 Truc n3=new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

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

Q 37.4 Peut-on ajouter une instruction à la fin du programme de la question précédente afin d'afficher la valeur de la variable `cpt` sans utiliser d'instance ? Même question pour la variable `num`.

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

Exercice 38 – 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 38.1 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.

```
1 public class Cone {
2         private double r;
3         private double h;
4         private static final double PI=3.1415;
5                         // NB: c'est mieux d'utiliser Math.PI
6         private static int nbCones=0;
```

Q 38.2 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)`,

Constructeur :

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

- le constructeur sans paramètre qui initialise le rayon et la hauteur du cône entre 0 et 10 (non compris). Ce constructeur doit appeler le premier constructeur. Aide : utiliser l'instruction `Math.random()` qui retourne un double entre 0 (compris) et 1 (non-compris).

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

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

– la méthode `double getVolume()` qui retourne le volume V=1/3πr^2h du cône,

```
public double getVolume() {
        return 1.0/3*PI*r*r*h;
}
// Attention : 1/3=0 car la division entre 2 nombres de types int est la division entière
```

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

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

– l'accesseur de la variable `nbCones`,

Cette méthode est une méthode de classe. Comme cela, on peut faire : Cone.getNbCones() sans utiliser d'instance.

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

Q 38.3 Ecrire une classe `TestCone` qui contient une méthode `main` dans laquelle vous :
– créerez 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.

```
public static void main(String [] args) {
        Cone c1=new Cone();
        Cone c2=new Cone(5.4,7.2);
        System.out.println(c1+"\n"+c2);
        System.out.println("Le nombre de cones crees est : "+ Cone.getNbCones() );
        System.out.println("Le nombre de cones crees est : "+ c1.getNbCones() );
        System.out.println("Le nombre de cones crees est : "+ c2.getNbCones() );
}
```

Exercice 39 – Trio de personnes (composition et tableau d'objets)

Voici la définition d'une classe Personne, dans laquelle les personnes ont comme nom : "Individu" suivi d'une lettre.

```
1 public class Personne {
2         private static char lettre='A';
3         private String nom;
4
5         public Personne() {
6                 nom="Individu" + lettre;
7                 lettre++;
8         }
9         public String toString() { return nom; }
10 }
```

NB : Pour le TME, le texte de la classe `Personne` *est téléchargeable à partir du site de l'UE*

Q 39.1 Quel est l'affichage produit par la classe `TestProjet` suivante ? A quoi sert la variable lettre ? Pourquoi la variable lettre est-elle déclarée static ?

```
1 public class TestProjet {
2         public static void main(String [] args) {
3                 Personne p1=new Personne(), p2=new Personne();
4                 System.out.println("Personnes␣:␣"+p1+"␣"+p2);
5         }
6 }
```

Le but de cette question est que les étudiants comprennent à quoi sert la variable static lettre.
L'affichage produit est : "Personnes : IndividuA IndividuB"

Q 39.2 Un trio est composé de 3 personnes (tableau de 3 personnes) et d'un numéro (entier) déterminé en fonction d'une variable compteur qui est incrémentée à chaque création d'un Trio. Ecrire une classe `Trio` avec un constructeur sans paramètre et une méthode `toString()`. Par exemple, pour le trio 1, cette méthode retourne :

```
Trio 1 : IndividuC IndividuD IndividuE
```

```
1 public class Trio {
2         private static int compteur=0;
3         private int numeroTrio;
4         private Personne [] tab;
5
6         public Trio() {
7                 compteur++;
8                 numeroTrio=compteur;
9                 tab=new Personne [3];
10                for(int i=0;i<3;i++) {
11                        tab[i]=new Personne();
12                }
13        }
14
15        public String toString() {
16                return "Trio␣"+numeroTrio+"␣:␣"+tab[0]+"␣"+tab[1]+"␣"+tab[2];
17        }
18 }
```

Q 39.3 Un projet est composé d'un nom de projet et d'un objet Trio. Ecrire une classe `Projet` avec un constructeur ayant le nom du projet comme unique paramètre et une méthode `toString()`. Par exemple, pour le projet dont le nom est `P3X-774`, cette méthode retourne la chaîne :

```
Projet P3X-774 Trio 1 : IndividuC IndividuD IndividuE
```

```
1 public class Projet {
2         private String nom;
3         private Trio leTrio;
4
5         public Projet(String nom) {
6                 this.nom=nom;
7                 leTrio=new Trio();
8         }
9         public String toString() {
10                return "Projet␣"+nom+"␣"+leTrio;
11        }
12 }
```

Q 39.4 Ajouter dans la méthode `main` de la classe `TestProjet`, les instructions nécessaires pour obtenir l'affichage suivant :

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

```
1 public class TestProjet {
2         public static void main(String [] args) {
3                 Personne p1=new Personne();
4                 Personne p2=new Personne();
5                 System.out.println("Personnes␣:␣"+p1+"␣"+p2);
6                 Trio t1=new Trio();
7                 Trio t2=new Trio();
8                 System.out.println(t1+"\n"+t2);
9                 Projet proj1=new Projet("P3X-774");
10                Projet proj2=new Projet("P3R-233");
11                System.out.println(proj1+"\n"+proj2);
12        }
13 }
```

Q 39.5 Dans les classes `Personne`, `Trio` et `Projet`, rajouter les variables et méthodes nécessaires pour afficher dans la méthode `main` le nombre de personnes créées, le nombre de trios créés et le nombre de projets créés.

Dans chaque classe, on rajoute si nécessaire une variable compteur et cpt++ dans le constructeur, ainsi qu'un accesseur statique pour ces variables.
Dans la classe Personne :

```
1 public static int getCptPersonnes() {
2         return (int)(lettre-'A');
3 }
```

Puis on rajoute dans main :
Personne.getCptPersonnes()
Trio.getCompteur()
Projet.getCptProjet()

Exercice 40 – Méthodes de classe

Q 40.1 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).
  Rappel : vous n'avez pas besoin de connaître la valeur du code ASCII d'un caractère pour utiliser cette valeur.
- la méthode de classe `chaine()` qui retourne une chaîne de caractères construit à partir de la concaténation de 10 lettres de l'alphabet choisis aléatoirement (appeler la méthode `lettre()`).

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

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

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

Q 40.3 Dans la méthode `main()` de la classe `Alea`, afficher le résultat retourné par la méthode `chaine()`. Même question pour la méthode `main()` d'une classe `TestAlea`.

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

Exercice 41 – 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.

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

Exercice 42 – Somme de 2 vecteurs

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

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

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

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

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

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

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

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

Exercice 43 – Somme de 2 tableaux

Cet exercice est semblable à un exercice précédent, mais ici la classe `Vect` aura comme un attribut un tableau. On veut faire la somme de deux tableaux, c'est-à-dire que l'on veut créer un nouveau tableau dont chacun des éléments est la somme des éléments de mêmes rangs dans chacun des tableaux.

Q 43.1 Créer une classe `Vect` qui contient :
- une variable `tab` de type tableau d'entiers,
- un constructeur qui aura en paramètre le nombre d'éléments du tableau et qui réservera l'espace mémoire pour ce tableau (ce constructeur sera déclaré exceptionnellement private, on ne pourra l'appeler seulement qu'à partir d'un autre constructeur),
- un constructeur sans paramètre qui appel le premier constructeur afin de créer un tableau de 10 éléments, puis qui initialise chaque élément avec une valeur aléatoire comprise entre 0 et 99,
- une méthode `toString()`.

Q 43.2 Ajouter une méthode d'instance qui fait la somme de deux tableaux, éléments par éléments.

Q 43.3 Ajouter une méthode de classe qui fait la somme de deux tableaux, éléments par éléments.

Q 43.4 Ecrire une classe `TestVect` qui crée deux objets de la classe `Vect`, puis qui affiche la somme de leurs deux tableaux en utilisant la méthode d'instance et en utilisant la méthode de classe.

```
1 public class Vect {
2         private int [] tab;
3
4         private Vect(int n) { ~// Attention : exceptionnellement declare private
5                 tab=new int [n];
6         }
7         public Vect() {
8                 this(10);
9                 for (int i=0; i<tab.length ; i++)
10                tab[i] = (int)(Math.random() * 100);
11        }
12        public String toString() {
13                String s = "";
14                for (int i=0; i<tab.length ; i++)
15                s += tab[i] + "␣";
16                return s;
17        }
18        public Vect somme (Vect v) {
```

```
19                  Vect res = new Vect(v.tab.length);
20                  for (int i=0; i<tab.length ; i++)
21                          res.tab[i] = this.tab[i] + v.tab[i];
22                  return res;
23          }
24          public static Vect somme (Vect v1, Vect v2) {
25                  Vect res = new Vect(v1.tab.length);
26                  for (int i=0; i<res.tab.length ; i++)
27                          res.tab[i] = v1.tab[i] + v2.tab[i];
28                  return res;
29          }
30 }
31
32 public class TestVect {
33          public static void main (String[] args) {
34                  Vect v1 = new Vect();
35                  Vect v2 = new Vect();
36                  Vect v3 = v1.somme(v2);
37                  Vect v4 = Vect.somme(v1,v2);
38                  System.out.println("v1="+v1+"\nv2="+v2+"\nv3="+v3+"\nv4="+v4);
39          }
40 }
```

Exercice 44 – Classe enveloppe

Chaque type simple a une classe *enveloppe* correspondante, ce qui permet d'accéder utilement à des méthodes prédéfinies de la classe. Nous allons travailler avec la classe enveloppe Double (voir sur le site de l'UE la "Documentation Java") du type simple `double`. Toutes les méthodes qu'on écrira seront des méthodes de classe.

Q 44.1 Créer une classe `TestTableauDouble` avec la méthode `main` dans laquelle vous déclarerez et initialiserez deux tableaux de `double`, l'un trié et l'autre non trié (c'est vous qui choisissez les valeurs).

Q 44.2 Créer une classe `TableauDouble` dans laquelle vous définirez les méthodes de classe ci-après. Vous aurez besoin de la documentation java de la classe `Double` que vous trouverez à partir du site web de l'UE (voir notamment la documentation des méthodes : `int compareTo (Double d)` et `double doubleValue()`).
Ecrire et tester au fur et à mesure dans le main les méthodes de classe suivantes :

- `Double[] versDouble(double[] tab)` qui renvoie le tableau de `Double` obtenu en transformant chaque élément de `tab` en `Double`,
- `void afficher(Double [] tab)` qui affiche les valeurs du tableau,
- `Double maxTab(Double[] tab)` qui rend le plus grand élément du tableau,
- `int premierEgalA(Double[] tab, double d)` qui rend le rang du premier élément égal à d s'il existe, sinon -1,
- `boolean estTrie(Double [] tab)` qui vérifie que le tableau est trié,
- `boolean memeLong(Double[]t1, Double[]t2)` qui teste si les deux tableaux ont le même le nombre d'éléments (longueur effective).

```
1 // méthode statique, donc le tableau doit être passé en paramètres
2 public class TableauDouble {
3          public static void afficher(Double [] tab) {
4                  for (int i=0; i<tab.length;i++) {
```

```
5                       System.out.print(tab[i].doubleValue()+" ");
6               }
7               System.out.println();
8       }
9       public static void afficher(double [] tab) {
10              for (int i=0; i<tab.length;i++) {
11                      System.out.print(tab[i]+" ");
12              }
13              System.out.println();
14      }
15      public static Double [] versDouble(double [] tab) {
16              Double [] d=new Double [tab.length];
17              for (int i=0; i<tab.length; i++) {
18                      d[i]=new Double(tab[i]);
19              }
20              return d;
21      }
22      public static Double maxTab(Double[] tab) {
23              Double max=new Double(Double.MIN\_VALUE);
24              for (int i=0; i<tab.length; i++) {
25                      if (tab[i].compareTo(max) > 0)
26                              max =tab[i];
27              }
28              return max;
29      }
30      public static int premierEgalA(Double[] tab, double d) {
31              for (int i=0; i<tab.length; i++) {
32                      if (tab[i].compareTo(d) > 0)
33                              return i;
34              }
35              return -1;
36      }
37      public static int rangMax(Double [] tab) {
38              double max=Double.MIN\_VALUE;
39              int rang=0;
40              for (int i=0; i<tab.length; i++)
41                      if (tab[i].doubleValue() > max) {
42                              max =tab[i].doubleValue();
43                              rang=i;
44                      }
45              return rang;
46      }
47      public static boolean estTrie(Double [] tab){
48              for (int i=0; i<tab.length-1;i++)
49                      if (tab[i].doubleValue() > tab[i+1].doubleValue())
50                              return false;
51              return true;
52      }
53 }
54
55 //=============================================================
56 public class TestTableauDouble {
57      public static void main(String [] args) {
58              Double leMax;
59              int leRang;
60              boolean trie;
```

```
61                  int taille=7;
62          double []tab1={-34.8970d, 0.0d, 0.56d, 3d, 10d, 33d, 45.3d};// trie
63          double []tab2={12.2d, 54.6d, 0.3d, 4.908d, 56d, 1d, 0d}; // non trie
64
65                  // creation des tableaux de Double :
66                  Double [] tab1D=TableauDouble.versDouble(tab1);
67                  Double [] tab2D=TableauDouble.versDouble(tab2);
68                  System.out.println("Tableau tab1D de Double:");
69                  TableauDouble.afficher(tab1D);
70                  leMax=TableauDouble.maxTab(tab1D);
71                  System.out.println("leMax= " + leMax);
72                  leRang=TableauDouble.rangMax(tab1D);
73                  System.out.println("leRang= " + leRang);
74                  if (TableauDouble.estTrie(tab1D) )
75                          System.out.println("tab1D est trie");
76                  else
77                          System.out.println("tab1D n'est pas trie");
78                  System.out.println("Tableau tab2D de Double:");
79                  TableauDouble.afficher(tab2D);
80                  leMax=TableauDouble.maxTab(tab2D);
81                  System.out.println("leMax= " + leMax);
82                  leRang=TableauDouble.rangMax(tab2D);
83                  System.out.println("leRang= " + leRang );
84                  if (TableauDouble.estTrie(tab2D) )
85                          System.out.println("tab2D est trie");
86                  else
87                          System.out.println("tab2D n'est pas trie");
88          }
89  }
```

Exercice 45 – 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 45.1 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 45.2 Écrire une méthode de classe `boolean pair(int n)` qui vérifie que n est pair.

Q 45.3 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 45.4 Écrire les méthodes `rendVoyelle()` et `rendConsonne()` qui rendent respectivement une voyelle et une consonne de façon aléatoire.

Q 45.5 Écrire une méthode `generNom()` qui rend un nom de longueur aléatoire comprise entre 3 et 6. On génèrera alternativement une consonne, une voyelle, une consonne, etc...

Q 45.6 Écrire une classe `TestNom` dont la méthode `main` génère et affiche, dans une boucle, une dizaine de nom.

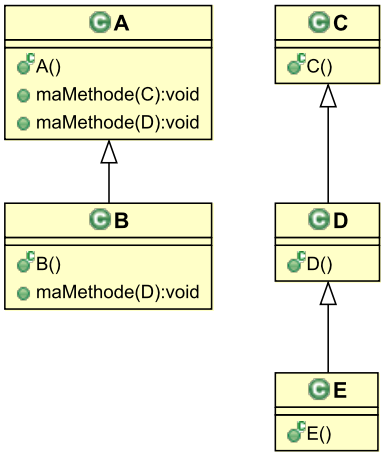
```
1 public class Nom {
2         private static char[] consonnes= {'B','C','D','F','G','H','J','K',
3                 'L','M','N','P','Q','R','S','T','V','W','X','Z'};
```

```
4         private static char[] voyelles={'A','E','I','O','U','Y'};
5
6         public static int rendAlea(int inf, int sup) {
7                 // rend un nat entre inf et sup, inf<=sup
8                 return (int)(inf + Math.random()*(sup-inf+1));
9                 // OU => Random r=new Random();return r.nextInt(sup-inf+1);
10        }
11        public static char rendVoyelle() {
12                return voyelles[rendAlea(0,voyelles.length-1)];
13        }
14        public static char rendConsonne() {
15                return consonnes[rendAlea(0,consonnes.length-1)];
16        }
17        public static boolean pair(int n) {
18                return ((n\%2) == 0);
19        }
20        public static String generNom() {
21                int lg;
22                String s="";
23                // tirage de la longueur entre3 et 6 :
24                lg=rendAlea(3,6);
25                for (int i=0; i<lg; i++) {
26                        if (pair(i)) s+= rendConsonne();
27                        else s+=rendVoyelle();
28                }
29                return s;
30        }
31}
32//=============================================
33public class TestNom {
34        public static void main(String [] args) {
35                int nbGeneres=10;
36                for (int i=0; i<nbGeneres; i++) {
37                        System.out.println("Nom gener : "+ Nom.generNom() + "\n");
38                }
39        }
40}
```

Exercice 46 – Sélection de méthode

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

Q 46.1 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);
```

Je numérote les 3 solutions de haut en bas
sur l'instance `a` :
- 1 : pas de piège
- 2 : pas de piège
- 1 : PIEGE : on est dynamique sur les instances qui appellent les méthodes mais pas sur les arguments ! `cd` est de type C : sélection de la méthode correspondante
- 2 : pas de piège (mais gare à l'exécution si on fait mal le cast)
- 2 : pas de signature exacte correspondant à l'appel ⇒ recherche de la signature compatible la plus proche (E est un D), D est plus proche de E (par rapport à C→E)

```
1 je suis dans A (arg C)
2 je suis dans A (arg D)
3 je suis dans A (arg C) // PIEGE: on est dynamique sur les instances qui appellent
      les methodes mais pas sur les arguments ! cd est de type C: selection de la
      methode correspondante
4 je suis dans A (arg D)
5 je suis dans A (arg D) // pas de signature exacte correspondant a l'appel ⇒
      recherche de la signature compatible la plus proche (E est un D), D est plus
      proche de E (par rapport a C→E)
6
7 je suis dans A (arg C)
8 je suis dans B (arg D)
9 je suis dans A (arg C)
10 je suis dans B (arg D)
11 je suis dans B (arg D)
12
13 je suis dans A (arg C)
14 je suis dans B (arg D)
15 je suis dans A (arg C)
16 je suis dans B (arg D)
17 je suis dans B (arg D)
```

Même comportements sur les instances `b` et `ab`
Remarque sur les priorités : entre une signature exacte dans la classe mère et une signature approchée dans la classe de l'instance, on prend la signature exacte (l9 et l15)

Q 46.2 Conversions implicites (ou pas)
On envisage 3 ajouts de méthodes :

Cas 1 :

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

Cas 2 :

```
1 // dans A
2 public void meth(int i)
3 // dans B
4 public void meth(double d)
```

Cas 3 :

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

Le code à exécuter est maintenant le suivant :

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

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

Cas 1 :
Tout est OK à la compilation + exécution : conversion implicite int⇒double

Cas 2 :

```
1 a.meth(2);   // OK 1
2 a.meth(2.);   // KO compil
3 b.meth(2); // OK 1
4 b.meth(2.); // OK 2
5 ab.meth(2); // OK 1
6 ab.meth(2.); // KO compil
```

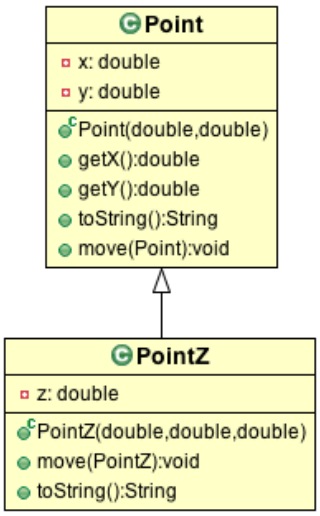
Pas de conversion double ⇒ int (trop dangereux)

Cas 3 :

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

Exercice 47 – Redéfinition piégeuse

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




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

Q 47.1 Pourquoi cette hiérarchie de classe est-elle discutable ?

Il faut savoir si un PointZ est une spécialisation (cas particulier) de Point ou l'inverse.
- Si on considère qu'il s'agit d'un cas particulier (c'est un Point qui a pour particularité d'avoir une composante en Z) ⇒ OK
  un PointZ **EST UN** Point
- Si on considère que PointZ = point 3D... Il y a un soucis. En effet, du point de vue logique, un Point2D EST UN Point3D et pas l'inverse...

Q 47.2 Syntaxe : les lignes 15, 19 et 23 sont-elles correctes ? Sinon, proposez des modifications. En ligne 23, peut-on utiliser directement `x` et `y` sans passer par les accesseurs ? Pourquoi ?

Le prog est correct.
Le super.get... en ligne 23 est évidemment optionnel.
On ne peut pas utiliser directement `x` et `y` car ce sont des attributs privés de la classe mère... Il faudrait qu'ils soient `protected`.

Q 47.3 Que pensez-vous du programme suivant ?

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

```
10 System.out.println(p3d);
```

Q 47.3.1 Qu'est-ce qui s'affiche ?

Q 47.3.2 Est-ce que ça vous semble logique ?

Q 47.3.3 Expliquer en détail ce qui s'est passé au niveau de la compilation et de l'exécution.

Affichage :

```
1 [1.0, 2.0]
2 [1.0 2.0 3.0]
3 [2.0, 3.0]
4 [2.0 3.0 3.0] // composante en Z non modifiee!!
```

1. Compilation : présélection sur le type des variables

   ```
   1 p3d.move(depl); // -> move(Point p)
   ```

2. Execution :
   (a) Recherche de `move(Point p)`
   (b) La méthode existe dans la super-classe, elle est exécutée
   (c) PointZ peut toujours remplacer un Point, il rentre dans la méthode

Quizz 11 – Variables et méthodes de classes

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

```
1 public class Cercle {
2     public static final double PI=3.14159;
3     private static int nbCercles=0;
4     public final int numero;
5     private int rayon;
6     public Cercle(int r) {
7         rayon=r;
8         nbCercles++;
9         numero=nbCercles;
10    }
11    public double surface() {
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 11.1 Cocher les réponses qui provoquent une erreur à la compilation si dans la classe `TestCercle`, je remplace `EXPRESSION` par :

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

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

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

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

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

QZ 11.3 Soit la classe `Test2Cercle` suivante :

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

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

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

6 Héritage et modélisation

Objectifs

- Héritage
- Généralisation / Spécialisation
- Redéfinition de méthodes
- Modélisation de problème

Il vaut mieux voir les classes abstraites dés que possible pour pouvoir faire plus d'exercice.

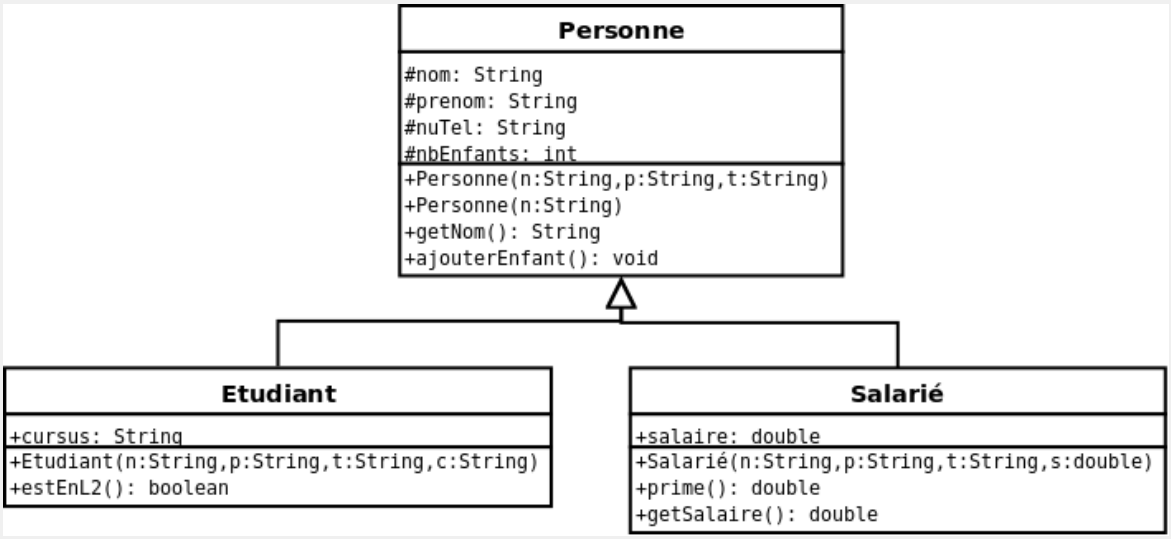
Exercices

Exercice 48 – 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 recoivent un salaire.

Q 48.1 Donner la hiérarchie des classes.

- Classe `Personne` (attributs : `nom`, `prenom`, `tel`, `nbEnfants`).
- Classe `Etudiant` qui hérite de `Personne` et qui a un attribut supplémentaire : `cursus`.
- Classe `Salarie` qui hérite de `Personne` et qui a un attribut supplémentaire : `salaire`.



Q 48.2 Quels sont les membres (variables et méthodes) hérités ?

La classe `Etudiant` hérite de tous les attributs et méthodes de `Personne`. La classe `Enseignant` hérite de tous les attributs et méthodes de `Personne`.

Q 48.3 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 }
```

Remarque : les variables d'instance ont été déclarées protected pour être accessible à partir des sous-classes. Si on les met en private, il faut alors utiliser des accesseurs dans les sous-classes.

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

Q 48.4 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 ?

```
1 class Salarie extends Personne {
2    private double salaire;
3    public Salarie(String n, String p, String t, double s) {super(n,p,t);salaire=s
         ;}
4    public double prime() {return 5*salaire*nbEnfants/100;}
5    public double getSalaire() {return salaire;}
6  }
```

NB : plutot que d'accéder aux variables protected, c'est plus propre de faire un accesseur dans la classe mère... Ca vaut le coup de le rappeler aux étudiants

Q 48.5 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  }
```

- Erreur sur prime de Personne (ligne 5) et de Etudiant (ligne 9).
- Erreur sur getSalaire (ligne 6)
- Erreur sur création Pauline (ligne 12), Ahmed (ligne 7)

Exercice 49 – Orchestre

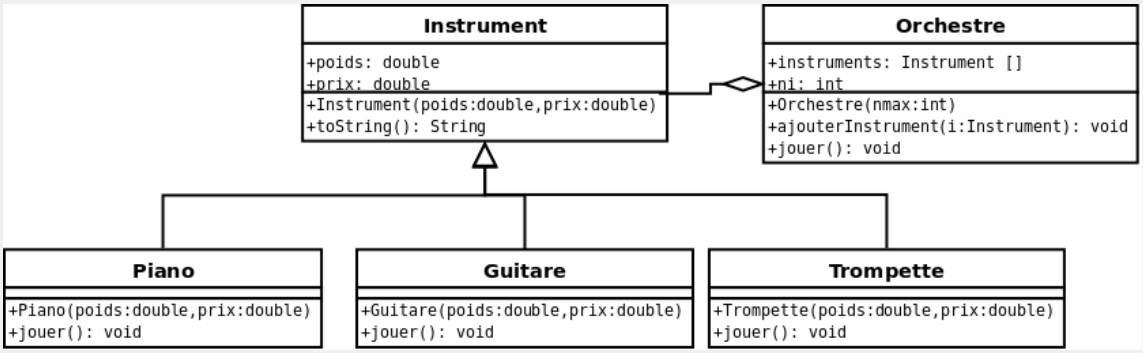
Exercice de base sur l'héritage pour introduire l'utilité des classes abstraites et de la redéfinition de méthodes.

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

Q 49.1 Dessiner l'arbre d'héritage du problème.

```
Instrument
    --- Piano
    --- Guitare
    --- Trompette
```

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



Q 49.2 Écrire une classe `Instrument` contenant deux variables d'instance de type double pour stocker le poids et le prix de l'instrument, respectivement. Munir la classe d'un constructeur à deux paramètres pour initialiser les variables d'instance, ainsi que de la méthode toString(). Quelle est la particularité de la méthode toString() d'un point de vue de l'héritage ?

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

```
public class Instrument {
        private double poids;
        private double prix;
        public Instrument(double poids, double prix) {
                this.poids = poids;
                this.prix = prix;
        }
        @Override
        public String toString() {
                return "Instrument [poids=" + poids + ", prix=" + prix + "]";
        }
}
```

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

Q 49.3 Écrire les classes `Piano`, `Guitare`, `Trompette`. Ces classes comporteront une méthode jouer() qui affichera, par exemple pour `Guitare`, "La guitare joue".

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

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

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

```
25          }
26 }
```

Q 49.4 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 lorque ceci est possible.

Au lieu d'utiliser nmax en variable d'instance, on peut utiliser `instruments.length`

```
1 public class Orchestre {
2         private Instrument[] instruments;
3         private int nmax;
4         private int ni;
5         public Orchestre(int nmax) {
6                 this.nmax = nmax;
7                 ni = 0;
8                 instruments = new Instrument[nmax];
9         }
10        public void ajouterInstrument(Instrument i){
11                if(ni<nmax){
12                        instruments[ni] = i;
13                        ni++;
14                }
15                else{
16                        System.err.println("Le tableau est plein!");
17                }
18        }
19        public void jouer(){
20                for(int i=0;i<ni;i++){
21                        instruments[i].jouer();
22                }
23        }
24 }
```

Q 49.5 Dessiner le diagramme UML correspondant.

L'idée du diagramme UML est de différencier les relations d'héritage et d'inclusion qui posent pas mal de soucis en général.

Q 49.6 Ajouter à la classe `Orchestre` une méthode jouer() qui fait jouer l'ensemble des intruments le constituant. Quel est le problème dans le code actuel et comment remédier à ce problème ?

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

Q 49.7 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) ?

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

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

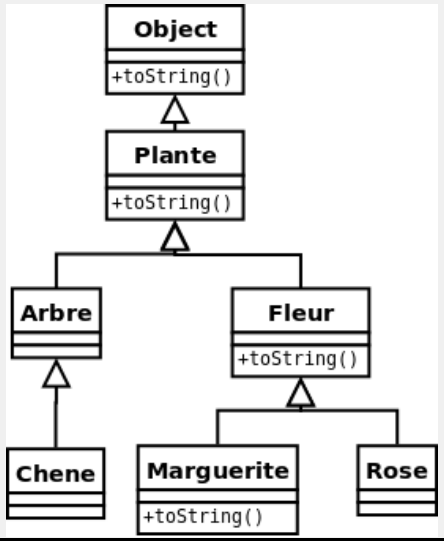
Exercice 50 – Botanique

Exercice sur héritage de méthodes et redéfinition.

Q 50.1 Dessiner l'arbre d'héritage et dire ce qu'affiche le programme suivant :

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

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



Q 50.2 En tirer des conclusions sur l'héritage et la redéfinition de méthode.

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

Q 50.3 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);
```

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

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

Exercice 51 – Concours

Un concours est organisé dans une assemblée de personnes. On pose une question (par exemple, "quel est le nom du premier mois du calendrier républicain ?" - réponse : vendémiaire) et une question subsidiaire (par exemple : "quel est le nombre de personnes qui vont trouver la bonne réponse ?").
Le ou les gagnants sont ceux qui, ayant bien répondu à la première question, ont trouvé ou se sont le plus rapprochés de la réponse à la question subsidiaire. On affichera les résultats en donnant le nom, le prénom et l'âge des gagnants.

Remarque : Dans cet exercice, on a besoin d'une structure de données type tableau ou vecteur pout gérer l'ensemble des personnes participant au concours.
L'exercice est donc à faire en TD, en écrivant en commentaire ce que l'on veut faire sans donner la programmation exacte. Si on veut le faire en TP, il faut utiliser la classe `Clavier` et leur dire de faire un affichage explicatif dans les méthodes qu'ils ne détaillent pas.

Q 51.1 Identifier ce qui va devenir une classe ou un attribut.

```
Classe Concours
     Attributs : question, reponse, questionSub, responseSub, nbPersonnes, participants
     Methodes : constructeur, toString, ajoutPersonne, traiteConcours (qui appelle
     compteRep et trouveEcart)

Classe Personne
     Attributs : nom, age, reponse1, reponse2
     Methodes : constructeur, accesseurs, toString

Classe TestConcours
     Attributs :
     Methodes : main
```

Q 51.2 Définir la ou les classes nécessaires pour traiter ce concours.

Remarque : Les attributs ont volontairement été mis en private et donc il faudra écrire des accesseurs et des modifieurs. La classe `Concours` utilise et modifie des attributs de la classe Personne.
L'exercice est donc à faire en TD, en écrivant en commentaire ce que l'on veut faire sans donner la programmation complète.

Q 51.3 Écrire les méthodes pour traiter le concours.

Q 51.4 Écrire une classe de test qui crée une dizaine de personnes et propose ce concours.

```
1 public class Concours {
2     private final int MAX = 100;    // nb max de participants
3     private String question;    // question du concours
4     private String reponse;    // reponse a la question
5     private String questionSub;  // question subsidianre
6     private int reponseSub;    // reponse subsidiaire
7     private int nbPersonnes;    // nombre de participants
8     private Personne[] participants;  // tableau de participants
```

```
9
10    public Concours() {
11      question = Clavier.saisirLigne(
12      "Quelle est la question :");
13      reponse = Clavier.saisirLigne(
14      "Quelle est la reponse a la question :");
15      questionSub = Clavier.saisirLigne(
16      "Quelle est la question subsidiaire :");
17      reponseSub = 0;  // en fait, elle sera calculee
18      nbPersonnes = 0;
19      participants = new Personne[MAX];
20    }
21    public String toString() {
22      String s = "question du concours :" + question + "\n";
23      s += "question subsidiaire :" + reponse + "\n";
24      s += "nombre de participants :" + nbPersonnes;
25      return s;
26    }
27    public void ajoutPersonne (Personne p) {
28      participants[nbPersonnes] = p;
29      nbPersonnes++;
30      System.out.println("Bonjour " + p.getNom());
31      p.setRep1 (Clavier.saisirLigne(question + " :"));
32      p.setRep2 (Clavier.saisirEntier(questionSub + " :"));
33    }
34    public void compteRep() {
35      // comptage des bonnes reponses a la question du concours
36      Personne p;
37      reponseSub = 0;
38      for (int i=0; i < nbPersonnes; i++) {
39        p = participants[i];
40        if (p.getRep1().equalsIgnoreCase(reponse)) reponseSub++;
41      }
42      System.out.print ("Le nombre de personnes ayant trouve " +
43      "la bonne reponse est :" + reponseSub);
44    }
45    public int trouveEcart() {
46      // recherche de l'ecart minimum entre nb de bonnes reponses
47      // et reponses subsidiaires
48      int min = MAX;
49      int diff;     // difference
50      for (int i=0; i < nbPersonnes; i++) {
51        p = participants[i];
52        if (p.getRep1().equalsIgnoreCase(reponse)) {
53          diff = Math.abs(p.getRep2() - reponseSub);
54          if (diff < min)
55          min = diff;
56        }
57      }
58      return min;
59    }
60    public void traiteConcours() {
61      this.compteRep();
62      int ecart = this.trouveEcart();
63      System.out.print ("L'ecart minimum est de " + ecart);
64      // recherche des gagnants
```

```
65        for (int i=0; i < nbPersonnes; i++) {
66          p = participants[i];
67          if (p.getRep1().equalsIgnoreCase(reponse) &&       (Math.abs(p.getRep2()-
              reponseSub)
68          == min)) {
69            System.out.print ("Nom du gagnant : " + p.getNom()
70            + "qui est age de " + p.getAge());
71            System.out.println ("et qui avait predit "
72            + p.getRep2() + "bonnes reponses");
73          }
74        }
75      }
76  }     // fin de classe
77
78  public   class Personne {
79      private String nom;
80      private int age;
81      protected String rep1;  // reponse a la question
82      protected int rep2;  // reponse a la question subsidiaire
83      public Personne () {
84        nom = Clavier.saisirLigne("Quel est votre nom : ");
85        age = Clavier.saisirEntier("Quel est votre age : ");
86      }
87      public Personne (String n, int a) {
88        nom = n;
89        age = a;
90      }
91      public String getNom() {return nom; }
92      public int getAge() {return age; }
93      public String getRep1() { return rep1; }
94      public int getRep2() { return rep2;}
95      public void setRep1(String r) { rep1 = r; }
96      public void setRep2(int n) { rep2 = n; }
97      public String toString() {
98        String s = nom + "age de " + age + "\n";
99        s += "reponse 1 : " + rep1 + "\n";
100       s += "reponse 2 : " + rep2 + "\n";
101       return s;
102     }
103   }
104   public class TestConcours {
105     public static void main(String[] args) {
106       Concours c = new Concours();
107       System.out.println();
108       System.out.println("Les questions du concours");
109       System.out.println(c.toString());
110       // creation des personnes et ajout dans le concours
111       Personne p1 = new Personne("Pierre", 25);
112       c.ajoutPersonne(p1);
113       Personne p2 = new Personne("Martine", 34);
114       c.ajoutPersonne(p2);
115       Personne p3 = new Personne("Emile", 83);
116       c.ajoutPersonne(p3);
117       Personne p4 = new Personne("Jeanne", 12);
118       c.ajoutPersonne(p4);
119       Personne p5 = new Personne("Adrienne", 42);
```

```
120         c.ajoutPersonne(p5);
121         System.out.println();
122         System.out.println("Les personnes");
123         System.out.println(p1.toString()+"\n"+p2.toString()
124         +"\n"+p3.toString()+"\n"+p4.toString()+"\n"+p5.toString());
125         c.traiteConcours();
126       }
127   }
```

Exercice 52 – Final : les différentes utilisations

Q 52.1 Questions de cours

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

Un attribut dont la valeur ne peut jamais être changée

```
1 // Usage 1
2 public class Truc{
3   private final int i = 1;
4 // Usage 2
5 public class Truc{
6   private final int i;
7   public Truc(){
8     i = 1; // init possible dans le constructeur
9   }
```

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

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

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

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

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

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

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

Q 52.2 Application sur la classe Point

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7 Héritage et classe abstraite

Objectifs

- Héritage
- Classes et méthodes abstraites
- Redéfinition de méthodes existantes

Propositions :
En TD : exercice 53 (Shape), exercice 54 (Ménagerie)
En TME : exercice 55 (Figure2D) ou exercice 56 (Robots pollueurs)

Exercices

Exercice 53 – Figures (héritage, constructeurs, méthode abstraite)

Rappel de cours : une méthode abstraite est une méthode sans corps, elle n'a qu'une en-tête. Si une classe est déclarée `abstraite` alors on ne peut pas créer d'objet de cette classe (pas de `new ClasseAbstraite()`). Si une classe possède une méthode abstraite, cette classe doit être déclarée abstraite. Si une classe hérite d'une méthode abstraite, elle doit soit être déclarée abstraite, soit définir le corps de la 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 necessaire, car implicitement appele
13      radius = 1 ;
14    }
15    public Circle(double x, double y, double r) { super(x,y) ; radius = r ; }
16    public String toString() {
17      return super.toString() + " Rayon : " + radius ;
18    }
19 }
20
21 public class MainShape {
22    public static void main(String [] args) {
23      Circle c1,c2 ;
24      c1 = new Circle(1,1,3) ;
25      c2 = new Circle() ;
26      System.out.println(c1.toString() + "\n" + c2.toString());
27    }
28 }
```

Q 53.1 De quels membres (variables d'instance et méthodes) de `Shape` hérite la classe `Circle` ?

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

Q 53.2 La compilation de la classe `Circle` échoue, expliquer pourquoi.

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

Q 53.3 Ajouter une méthode `surface()` à la classe `Circle` et modifier en conséquence la méthode `toString`.

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

Q 53.4 Créer une classe `Rectangle` qui hérite de `Shape`.

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

Q 53.5 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.

```
1 Shape[] tab = new Shape[3];
2 tab[0] = new Circle();
3 tab[1] = new Rectangle();
```

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

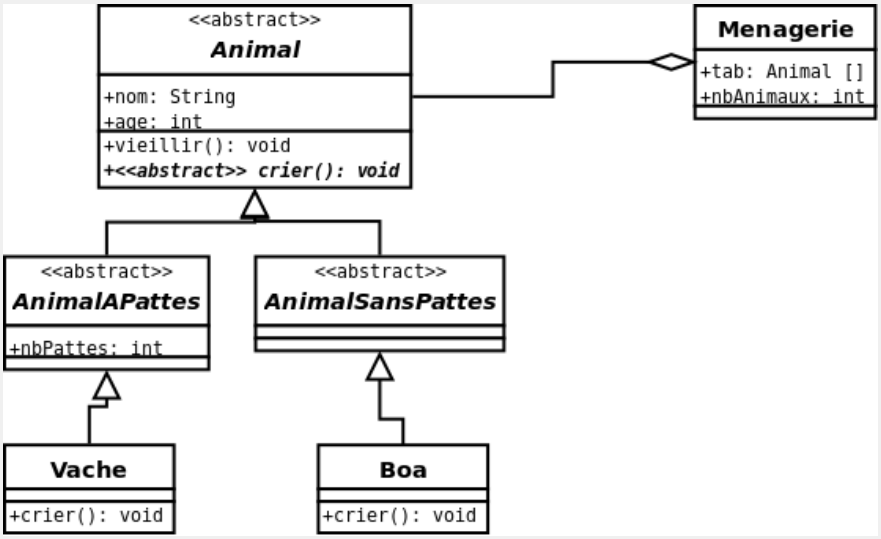
Exercice 54 – Ménagerie (tableaux, héritage, constructeur)

Exercice sur redéfinition de méthode et classe `abstraite`

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

Q 54.1 Etablir graphiquement la hiérarchie des classes ci-dessus. Déterminer celles qui peuvent être déclarées abstraites ?

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



Q 54.2 Ecrire la classe `Animal` avec deux constructeurs (un prenant en paramètre le nom et l'âge, l'autre prenant en paramètre le nom et qui fixe l'âge à 1 an), la méthode `toString`, une méthode vieillir qui fait vieillir l'animal d'une année, et une méthode `crier()` qui affichera le cri de l'animal. Peut-on écrire ici le corps de cette méthode ?

// Cette classe `est` abstraite car elle contient une methode abstraite

```
public abstract class Animal {
  private String nom;
  private int age;
  public Animal(String nom, int age) {
```

```
5       this.nom=nom;
6       this.age=age;
7     }
8     public Animal(String nom) {
9       this(nom,1);
10    }
11    public void vieillir() {
12      age++;
13    }
14    public abstract void crier(); /** Methode abstraite */
15    public String toString() {
16      return "Je␣m'appelle␣"+nom+",␣j'ai␣"+age +"ans";
17    }
18  }
```

Q 54.3 Ecrire toutes les sous-classes de la classe `Animal` en définissant les méthodes `toString()` et les méthodes `crier()` qui affichent le cri de l'animal.

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

```
36        return super.toString()+" et je suis un boa";
37      }
38    }
```

De meme, pour Saumon et Canard

Q 54.4 Ecrire une classe `Menagerie` qui gère un tableau d'animaux, avec la méthode `void ajouter(Animal a)` qui ajoute un animal au tableau, et la méthode `toString()` qui rend la liste des animaux.

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

Q 54.5 Ajouter une méthode `void midi()` qui fait crier tous les animaux de cette ménagerie.

Q 54.6 Ecrire la méthode `vieillirTous()` qui fait vieillir d'un an tous les animaux de cette ménagerie.

```
1  /** Comme tous les animaux contiennent la methode crier,
2  * on peut parcourir le tableau sans faire attention au type de l'animal. */
3
4  public void midi() {
5      for (int i=0; i<nbAnimaux; i++) {
6          tab[i].crier();
```

```
7       }
8   }
9   public void vieillirTous() {
10      for (int i=0; i<nbAnimaux; i++)
11          tab[i].vieillir();
12  }
```

Q 54.7 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.

Correction du main :

```
1   public class TestMenagerie {
2     public static void main(String [] args) {
3       Menagerie m=new Menagerie(20);
4       Animal b1= new Boa("Beatrice");
5       Animal b2= new Boa("Bernard");
6       Animal v1= new Vache("Marguerite");
7       Animal v2= new Vache("Blanchette");
8       m.ajouter(b1);
9       m.ajouter(b2);
10      m.ajouter(v1);
11      m.ajouter(v2);
12      System.out.println("Il␣est␣midi␣:");
13      m.midi();
14      System.out.println(m);
15      m.vieillirTous();
16      System.out.println(m);
17    }
18  }
```

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

Exercice 55 – 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
```

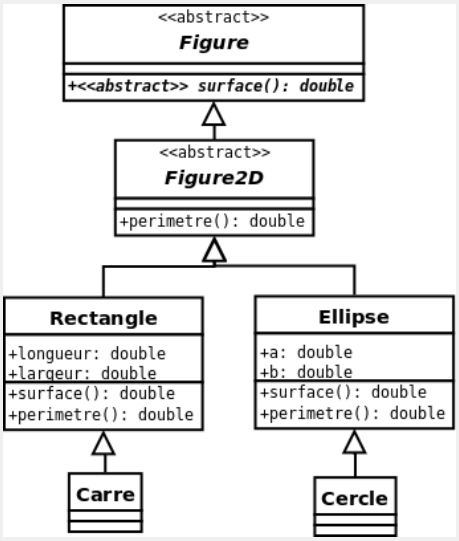
Ces classes devront respecter les principes suivants :
- Toutes les variables d'instance sont de type `double` et caractérisent uniquement la taille des objets, pas leur position.
- Chaque objet sera créé par un constructeur qui recevra les paramètres nécessaires (par exemple la longueur et la largeur 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 longeur a du demi-grand axe et la longueur b du demi-petit axe. Sa surface est $\pi * a * b$ et son périmètre est $2\pi\sqrt{\frac{(a^2+b^2)}{2}}$.

Rappel : dans la classe `Math`, il existe la constante `Math.PI` et la méthode `Math.sqrt()` qui retourne la racine carrée d'un nombre (voir annexe page 193).

Q 55.1 Quelles sont les particularités d'une méthode abstraite et les conséquences pour la classe et les classes dérivées ?

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



Q 55.2 Donner pour chacune des classes, en utilisant correctement les notions d'héritage et de classe abstraite :
- la définition de la classe,
- la déclaration des variables d'instance,
- le constructeur,
- les méthodes de la classe.

```
1 public abstract class Figure {
2     public abstract double surface();
3     public String toString() { return "c'est une figure";}
4 }
5 public  abstract class Figure2D extends Figure {
6     public abstract double perimetre();
7 }
8 public  class Rectangle extends Figure2D {
9     private double longueur, largeur;
10    public Rectangle (double l1, double l2) {
11      super();
12      longueur = l1;
13      largeur = l2;
14    }
15    public String toString() {
16      return "c'est un rectangle";
17    }
18    public double surface() {
19      return longueur*largeur;
20    }
21    public double perimetre() {
22      return 2*(longueur+largeur) ;
23    }
24 }
25 public class Carre extends Rectangle {
26    Carre(double cote) {
27      super(cote, cote);
28    }
29 }
30 public class Ellipse extends Figure2D {
31    private double a, b;
32    public Ellipse (double a, double b) {
33      super();
34      this.a = a;
35      this.b = b;
36    }
37    public String toString() {
38      return "c'est une ellipse";
39    }
40    public double surface() {
41      return Math.PI*a*b;
42    }
43    public double perimetre() {
44      return 2*Math.PI*Math.sqrt(a*a+b*b)/2;
45    }
46  }
47 public class Cercle extends Ellipse {
48    Cercle(double rayon) {
49      super(rayon, rayon);
50    }
51  }
```

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

Q 55.3 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.

```
1 public  class TestFigure {
2     public static void main (String [] args) {
3       Rectangle r = new Rectangle( 10,4);
4       System.out.println (r.toString()+" dont la surface est : "+ r.surface() + "
            et le perimetre : "+ r.perimetre());
5       Carre c = new Carre(25);
6       System.out.println (c.toString()+" dont la surface est : "+ c.surface() + "
            et le perimetre : "+ c.perimetre());
7       Ellipse e = new Ellipse(24, 12);
8       System.out.println (e.toString()+" dont la surface est : "+ e.surface() + "
            et le perimetre : "+ e.perimetre());
9       Cercle cc = new Cercle(15);
10      System.out.println (r.toString()+" dont la surface est : "+ r.surface() + "
            et le perimetre : "+ r.perimetre());
11    }
12 }
```

Exercice 56 – Les Robots Pollueurs

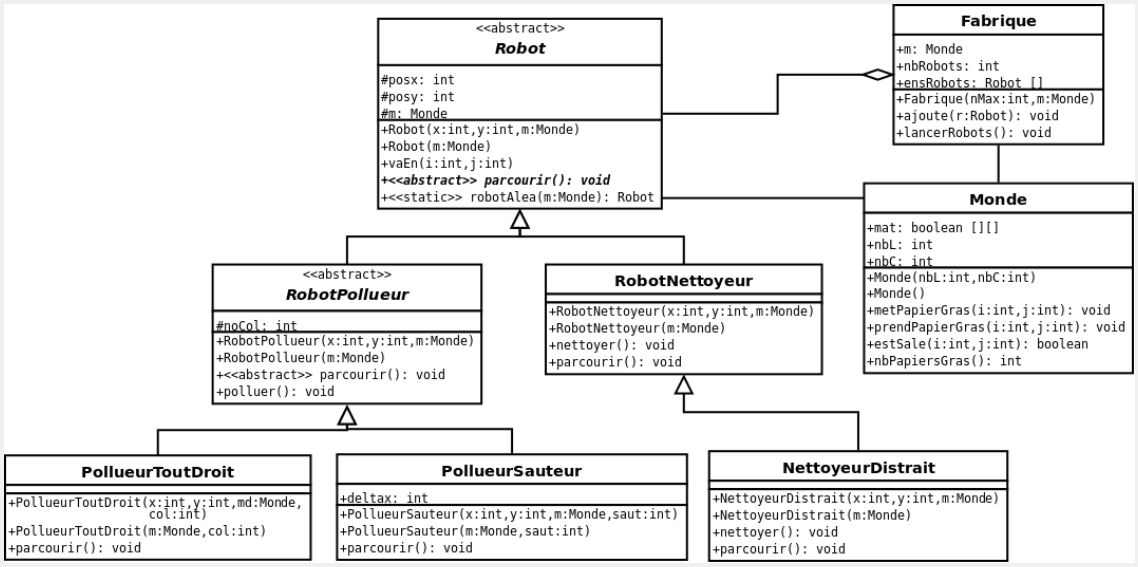
Le monde est constitué d'une matrice de cases. Différents types de robots agissent dans ce monde : des robots pollueurs et des robots nettoyeurs. Les robots pollueurs se baladent dans le monde et déposent des papiers gras sur les cases où ils se trouvent. Les robots nettoyeurs, quant à eux, ne supportent pas la vue d'un papier gras et l'enlèvent dès qu'ils en voient un sur une case.

Les robots pollueurs sont de deux sortes : robots sauteurs et robots qui vont tout droit. Chaque robot pollueur suit son parcours et dépose un papier gras sur chaque case rencontrée. Le parcours précis des robots pollueurs sera donné dans la suite.

Les robots nettoyeurs parcourent méthodiquement le monde en "boustrophédon", c'est-à-dire qu'ils parcourent la première ligne case par case, puis arrivé en bout de ligne font demi-tour, parcourent la deuxième ligne en sens inverse, et ainsi de suite jusqu'à la dernière case de la dernière ligne. Ils enlèvent, s'il s'en trouve un, le papier gras de la case où ils se trouvent. Parmi les robots nettoyeurs, certains sont distraits et n'enlèvent qu'un papier sur deux.

Les constructeurs, accesseurs, modifieurs et les méthodes `toString()` seront créés dans chaque classe suivant les besoins.

Q 56.1 Dessiner la hiérarchie des classes correspondant à la description ci-dessus.



Q 56.2 Ecrire la classe `Monde` qui contient les variables d'instance, le constructeur et les méthodes suivantes (déterminer si elles sont d'instance ou de classe) :

- le nombre de lignes `nbL`, le nombre de colonnes `nbC`, et une matrice booléenne `mat` de `nbL` lignes et `nbC` colonnes (vrai signifiant la présence d'un papier gras).
- un constructeur avec paramètres et un sans paramètres `Monde()` : ce constructeur par défaut crée un monde 10*10 sans papiers gras.
- `public String toString()` : retourne une chaîne de caractères décrivant le monde. On représentera un papier gras par le caractère `'o'`, rien par le caractère `'.`(point). Le caractère de passage à la ligne est `"\n"`.
- `metPapierGras(int i, int j)` : met un papier gras dans la case (i, j).
- `prendPapierGras(int i, int j)` : enlève le papier gras de la case (i, j).
- `estSale(int i, int j)` : teste si la case (i, j) a un papier gras.
- `nbPapiersGras()` qui rend le nombre de papier gras dans le monde.

```
1
2 public  class LeMonde {
3     private int nbL=10;
4     private int nbC=10;
5     private boolean [][] mat;
6     public LeMonde() {
7         mat=new boolean[nbL][nbC];
8         for (int i=0;i<nbL;i++)
9             for (int j=0;j<nbC;j++)
10                mat[i][j]=false;
11    }
12    public int getNbC() { return nbC;}
13    public int getNbL() { return nbL;}
14    public String toString() {
15        String s="";
16        for (int i=0;i<nbL;i++) {
17            for (int j=0;j<nbC;j++)
18                if (mat[i][j])
19                    s=s+"o";
20                else
```

```
                    s=s+".";
                s=s+"\n";
            }
            return s;
        }
        public boolean estSale(int i, int j) {return mat[i][j];}
        public void affiche() {System.out.println(toString()); }
        public void metPapierGras(int i, int j) {mat[i][j]=true;}
        public void prendPapierGras(int i, int j) {mat[i][j]=false;}
        int nbPapiersGras() {
            int n=0;
            for (int i=0;i<nbL;i++)
                for (int j=0;j<nbC;j++)
                    if (mat[i][j])
                        n++;
            return n;
        }
}
```

Q 56.3 Ecrire dans une classe `TestRobot` la méthode `main` et y créer un monde de 10 lignes et 10 colonnes. Y tester les méthodes `metPapierGras`, `prendPapierGras`, `estSale`.
Dans la suite on n'oubliera pas de reporter la hiérarchie construite en question 1 dans la définition des différentes classes (mot clé extends). Chaque fois qu'on écrira une variable ou une méthode, on en enrichira le dessin de la hiérarchie des classes.

```
public class TestRobot {
    public static void main (String [] args) {
      System.out.println("ca␣commence␣....");
      LeMonde monde=new LeMonde();
      RobotPollueur rp1, rp2;
      RobotNettoyeur rn1, rn2;
      rp1=new PollueurSauteur(monde, 5);
      rp2=new PollueurToutDroit(monde, 8);
      rn1=new RobotNettoyeur(monde);
      rn2=new NettoyeurDistrait(monde);
      rp1.parcourir();rp2.parcourir();
      System.out.println("etat␣du␣monde␣apres␣PSauteur(5)␣et␣PToutDroit(8)␣:\n");
      monde.affiche();
      rn1.parcourir();
      System.out.println("etat␣du␣monde␣apres␣RNettoyeur:\n");
      monde.affiche();
      rp1.parcourir();rp2.parcourir();
      System.out.println("etat␣du␣monde␣apres␣PSauteur(5)␣et␣PToutDroit(8)␣:\n");
      monde.affiche();
      //System.out.println("frapper une touche+return");
      //rep=ES.lireLigne();
      rn2.parcourir();
      System.out.println("etat␣du␣monde␣apres␣NDistrait:\n");
      monde.affiche();
      rn1.parcourir();
      System.out.println("etat␣du␣monde␣apres␣Nettoyeur:\n");
      monde.affiche();
    }
```

```
29  }
```

Q 56.4 Ecrire la classe `Robot` qui est abstraite car elle n'aura aucune instance, et qui contient les champs et méthodes suivantes :
- `posx, posy` : position du robot sur le monde.
- `m` : variable de type Monde. En effet il faut que le robot connaisse le monde pour pouvoir s'y déplacer et agir.
- deux constructeurs : `Robot(int x,int y,Monde m)` qui crée un robot à la position (x,y) et `Robot(Monde m)` qui crée un robot avec une position aléatoire. Le constructeur `Robot(Monde m)` doit appeler l'autre constructeur.
- `vaEn(int i, int j)` : se déplace en (i, j).
- `parcourir()` : méthode abstraite qui sera définie dans les sous-classes.

```
1 public abstract class Robot {
2   protected int posx; // dans l'ideal: private+accesseur
3   protected int posy;
4   protected LeMonde m;
5   public static final int nbTypesRobot=4;
6   public static final int nbTRMoins1=nbTypesRobot-1;
7   public Robot(int x, int y, LeMonde md) {
8       posx=x;
9       posy=y;
10      m=md;
11  }
12  public Robot(LeMonde m) {// cree un robot a une pos aleatoire
13      this((int)(Math.random()*m.getNbL()),
14              (int)(Math.random()*m.getNbC()), m);
15  }
16  public void vaEn(int x, int y) {
17      posx=x; posy=y;
18  }
19  public String toString() {
20      return "pos=("+posx+","+posy+")";
21  }
22  public abstract void parcourir() ;
23 }
```

Q 56.5 Ecrire la classe `RobotPollueur` (également abstraite car elle n'aura pas d'instance) qui contient les champs et méthodes :
- `polluer()` : met un papier gras là où ce robot se trouve dans le monde.
- constructeurs

```
1 public  abstract class RobotPollueur extends Robot{
2 //    protected int noCol; => NON!!! GROSSE ERREUR, CORRECTION 2013
3     public RobotPollueur(int x, int y, LeMonde md) {
4         super(x, y, md);
5     }
6     public RobotPollueur(LeMonde md) {
7         super(md);
8     }
```

```
9     public abstract void parcourir();
10    public void polluer() {
11        m.metPapierGras(posx, posy);
12    }
13  }
```

Q 56.6 Ecrire la classe `PollueurToutDroit` qui contient les méthodes suivantes :

- un constructeur
- `parcourir()` : cette méthode définit la méthode parcourir abstraite de la classe Robot. Elle décrit un parcours de ce robot dans le monde. Le robot se positionne d'abord dans sa colonne de départ en case (0, ColDepart), puis il va tout droit vers le sud en visitant chaque case de la colonne et dépose un papier gras sur chacune d'elle. Il s'arrête dans la dernière case de la colonne.

Déclarer dans la méthode main deux variables de type Robot qui référenceront deux instances de PollueurToutdroit et tester leur méthode parcourir(). Afficher l'état du monde après chaque parcours. Comment est choisi le corps de la méthode parcourir pour ces deux instances ?

```
1 public class PollueurToutDroit extends RobotPollueur {
2   public PollueurToutDroit(int x, int y,LeMonde md) {
3       super(x, y, md);
4 //      noCol=col; => CORRECTION 2013
5   }
6   public PollueurToutDroit(LeMonde md, int col) {
7       super(md);
8 //      noCol=col; NON
9       posy = col;
10  }
11  public String toString() {
12      return "Pollueur tout Droit "+ posy;
13  }
14  public void parcourir() {
15    for (int i=0;i<m.getNbL(); i++) {
16        vaEn(i, posy);
17        polluer();
18    }
19  }
20 }
```

Q 56.7 Ecrire la classe `PollueurSauteur`, qui contient les champs et méthodes suivants :

- `deltax` représentant la taille du saut effectué à chaque déplacement du sauteur ;
- un constructeur ;
- `parcourir()` : cette méthode définit la méthode parcourir abstraite de la classe `Robot`. Elle décrit un parcours de ce robot dans le monde. Le robot se positionne d'abord dans sa colonne de départ en case `(0, coldepart)`, puis il saute de façon analogue au cavalier des Echecs et va en `(1, coldepart+deltax)`, puis en `(2, colDepart)`, puis en `(3, colDepart+deltax)`, etc. Si la colonne sort de l'échiqier, on revient au début. Par exemple, la colonne `nbC`, qui n'existe pas, sera tranformée en `nbC modulo nbC`, c'est-à-dire colonne 0. Chaque case rencontrée est souillée d'un papier gras. Le robot s'arrête lorsqu'il atteint la dernière ligne.

Déclarer dans la méthode main deux variables de type Robot qui référenceront deux instances de `PollueurSauteur` avec des `colDepart` et des `deltax` différents et tester leur méthode `parcourir()`.

```
1  public class PollueurSauteur extends RobotPollueur {
2      protected int deltax; // taille du saut
3      public PollueurSauteur(int x, int y, LeMonde md, int saut) {
4          super(x, y, md);
5          deltax=saut;
6      }
7      public PollueurSauteur(LeMonde md, int saut) {
8          super(md);
9          deltax=saut;
10     }
11     public String toString() {
12         return "Pollueur Sauteur "+ deltax;
13     }
14     public void parcourir() {
15         int j;
16         for (int i=0;i<m.getNbL();i++) {
17             vaEn(i,deltax*(i%2));
18             polluer();
19         }
20     }
21 }
```

Q 56.8 Ecrire la classe `RobotNettoyeur`, qui contient les méthodes suivantes :
- `nettoyer()` : enlève le papier gras de la case où se trouve ce robot,
- un constructeur,
- `parcourir()` : cette méthode définit la méthode parcourir abstraite de la classe `Robot`. Elle décrit un parcours complet de ce robot dans le monde. Il part de la case $(0,0)$ et parcourt le monde en "boustrophédon". Si la case est sale, il enlève le papier gras qui s'y trouve.

Créer de même dans la méthode `main` deux `RobotNettoyeur` dont les références sont de type `Robot` et tester leur méthode `parcourir()` après le passage des robots pollueurs.

```
1 public class RobotNettoyeur extends Robot {
2     public RobotNettoyeur(int x, int y,LeMonde md) {
3         super(x,y,md);
4     }
5     public RobotNettoyeur(LeMonde md) {
6         super(md);
7     }
8     public void nettoyer() { // enleve le papier ou il se trouve
9         m.prendPapierGras(posx,posy);
10    }
11    public String toString() {
12        return "Nettoyeur "+ super.toString();
13    }
14    public void parcourir() { // parcours en boustrophedon
15        vaEn(0, m.getNbC()-1);
16        for (int i=0;i<m.getNbL(); i++) {
17            if ((i%2)==0) {
18                for (int j=0; j<m.getNbC(); j++) {
19                    vaEn(i, j);
20                    if (m.estSale(i,j))
21                        nettoyer();
```

```
22                  }
23              } else {
24                  for (int j=m.getNbC()-1; j>=0; j--) {
25                      vaEn(i, j);
26                      if (m.estSale(i,j))
27                          nettoyer();
28                  }
29              }
30          }
31      }
32  }
```

Q 56.9 Ecrire la classe `NettoyeurDistrait` qui contient les méthodes suivantes :
- un constructeur,
- `parcourir()` : cette méthode redéfinit la méthode `parcourir()`. Elle décrit un parcours complet de ce robot dans le monde. C'est le même parcours que celui des robots nettoyeurs mais comme il est distrait, il n'enlève qu'un papier sur deux.

Créer dans le main un `NettoyeurDistrait` dont la référence est de type `Robot` ; tester sa méthode `parcourir()` après le passage d'un pollueur.
Indications :
- prendre un compteur qu'on incrémentera pendant le parcours sur chaque case sale.
- utiliser l'opérateur % qui exprime la fonction modulo.

```
1 public class NettoyeurDistrait extends RobotNettoyeur {
2     public NettoyeurDistrait(int x, int y, LeMonde md) {
3         super(x, y, md);
4     }
5     public NettoyeurDistrait(LeMonde md) {
6         super(md);
7     }
8     public String toString() {
9         return "Distrait␣"+ super.toString();
10    }
11    public void parcourir() {
12        int k=0; // compteur de papiers gras
13        for (int i=0;i<m.getNbL(); i++) {
14            if ((i%2)==0) {
15                for (int j=0; j<m.getNbC(); j++) {
16                    vaEn(i, j);
17                    if (m.estSale(i,j))
18                        k++;
19                    if (k%2==1)
20                        nettoyer();
21                }
22            } else {
23                for (int j=m.getNbC()-1; j>=0; j--) {
24                    vaEn(i, j);
25                    if (m.estSale(i,j))
26                        k++;
27                    if (k%2==1)
28                        nettoyer();
29                }
```

```
30              }
31          }
32      }
33  }
```

Q 56.10 Supposons qu'il y ait au départ n papiers gras dans le monde. Combien faudra-t-il lancer de nettoyeurs distraits pour qu'il n'y ait plus un seul papier ?

$log(n)$, car à chaque passage, la moitié des papiers est enlevée.

Partie facultative

Il s'agit dans cette partie d'enrichir le programme précédent des Robots pour en faire un jeu. L'utilisateur pourra parier sur l'état de propreté du monde après le lancement d'un ensemble de robots tirés au sort. L'état de propreté est défini par un entier qui est le seuil de propreté : si le nombre de papiers gras est en dessous de de ce seuil, le monde sera considéré comme propre, et comme sale sinon.

Q 56.11 Écrire dans la classe `Robot` une méthode `Robot robotAlea(LeMonde m)` qui rend un robot déterminé aléatoirement.

```
1 public static Robot robotAlea(LeMonde m) {// rend un robot tire au hasard
2     Robot r=null;
3     int tirage=(int)(Math.random()*nbTypesRobot);
4     switch(tirage) {
5       case 0 : r=new RobotNettoyeur(m);break;
6       case 1 : r=new NettoyeurDistrait(m);break;
7       case 2 :
8           int col=(int)(Math.random()*m.getNbC());
9           r=new PollueurToutDroit(m,col);
10          break;
11      case nbTRMoins1 :
12        int delta=(int)(Math.random()*3)+1;
13        r=new PollueurSauteur(m,delta);
14    }
15    return r;
16  }
```

Q 56.12 Écrire une classe `Fabrique` qui a comme champ (entre autres) un tableau de `Robot`. On y mettra un constructeur, la méthode `ajoute(Robot r)` qui ajoute un robot dans la table, et une méthode `lancerRobots()`, qui lancera successivement la méthode `parcourir()` pour chaque robot du tableau, et affichera l'état du monde après chaque parcours.

```
1 public class Fabrique {
2   private LeMonde m;
3   private int nbRobots=0;
4   private  Robot [] ensRobots;
5   public Fabrique(int nMax,LeMonde m) {
6      // cree un tableau de robots aleatoites
```

```
7         this.m=m;
8         Robot r;
9         ensRobots=new Robot[nMax];
10        for (int i=0; i<nMax ;i++) {
11          r=Robot.robotAlea(m);
12          ajoute(r);
13        }
14      }
15      public void ajoute(Robot r) {
16        if (nbRobots >= ensRobots.length) {
17          System.out.println("plus de place dans le tableau!\n");
18          return;
19        }
20        ensRobots[nbRobots]=r;
21        nbRobots++;
22      }
23      public void lancerRobots() {
24        // lance les robots successivement et affiche
25        //le monde apres chaque parcours
26        Robot r;
27        for (int i=0; i<ensRobots.length ;i++) {
28          r=ensRobots[i];
29          System.out.println("robot lance : \n"+r);
30          r.parcourir();
31          m.affiche();
32        }
33      }
34    }
```

Q 56.13 Écrire une classe `Jeu`, qui contiendra les paramètres du jeu : taille du monde, nombre de robots total, seuil de propreté, ainsi que les entités du jeu : le monde, la fabrique, le pari du joueur, l'état du monde après le passage des robots. On y écrira les méthodes suivantes :
- `initJeu()` qui initialise le monde et la fabrique
- `prendrePari()` qui demande à l'utilisateur de choisir entre deux options « le monde sera propre » ou bien « le monde sera sale » (après le passage des robots)
- `lancerJeu()` qui prend le pari et lance les robots de la fabrique
- `finjeu()` qui détermine si le joueur a gagné et affiche le résultat.

```
1 public class Jeu {
2     // parametres du jeu :
3     private static int seuilProprete=15;
4     private static int tailleMonde=10;
5     private static int nbRob=8;
6     // elements du jeu :
7     private static LeMonde monde;
8     private static Fabrique f;
9     private static int pari;
10    private static boolean propre;
11    public static void initJeu() {
12      monde=new LeMonde();
13      f=new Fabrique(5,monde);
14    }
15    public static void lancerJeu() {
```

```
16        prendrePari();
17        f.lancerRobots();
18      }
19      public static void prendrePari() {
20        seuilProprete=(int)(Math.random()*tailleMonde*tailleMonde);
21        System.out.println("je␣prends␣le␣pari,␣le␣seuil␣de␣proprete␣est␣"+
22        seuilProprete + "\n");
23        System.out.println("1␣:␣le␣monde␣sera␣propre\n");
24        System.out.println("2␣:␣le␣monde␣sera␣sale\n");
25        System.out.println("Entrez␣votre␣reponse␣(1␣ou␣2)␣:\n");
26        pari=ES.lireEntierln();
27        System.out.println("vous␣avez␣parie␣:␣"+pari+"\n");
28      }
29      public static void finJeu() {
30        int propreInt;
31        int n=monde.nbPapiersGras();
32        propre=(n <= seuilProprete);
33        if (propre) propreInt=1;
34        else propreInt=2;
35        System.out.println("Il␣y␣a␣" + n + "␣papiers␣gras,␣");
36        if (propreInt==pari)
37        System.out.println("Vous␣avez␣gagne!\n");
38        else
39        System.out.println("Vous␣avez␣perdu!\n");
40      }
41    }
```

Q 56.14 Lancer le jeu dans le main.

```
1 public class TestRobot {
2   public static void main (String [] args) {
3     System.out.println("ca␣commence␣....");
4     LeMonde monde=new LeMonde();
5     System.out.println("************************************");
6     System.out.println("PARTIE␣2␣:␣JEU");
7     System.out.println("********************************");
8     Jeu.initJeu();
9     Jeu.lancerJeu();
10    Jeu.finJeu();
11    System.out.println("c'est␣fini,␣au␣revoir...");
12  }
13 }
```

Q 56.15 Enrichir le jeu avec de nouveaux robots : robot rebelle qui ne fait pas ce qu'on lui dit, robot qui tombe en panne, robot traitre qui tout d'un coup change de camp, etc..

Quizz 12 – Classe et méthode abstraite

QZ 12.1 Les instructions suivantes sont-elles correctes ? Expliquez.

```
1 public abstract class Z {}
```

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

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

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

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

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

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

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

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

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

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

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

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

Quizz 13 – Vocabulaire sur l'héritage

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

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

8 Héritage et liaison dynamique

Objectifs

- Transtypage d'objet
- Méthode `equals()`
- `instanceof`
- Liaison dynamique
- Contexte de méthode
- Héritage de classe prédéfinie

Remarque : On a regroupé les exercices sur l'héritage, il y a beaucoup de possiblités. C'est une notion difficile pour les étudiants, on y passera au moins deux semaines.
Propositions :
En TD : exercice 57 (Transtypage), exercice 58 (méthode equals), exercice **??** (Liaison dynamique) et exercice 59 (contexte méthode). Voir aussi le quizz 14 (Héritage)
En TME : finir exercices du thème précédent ou exercice 60 (Vehicules à moteur)

Exercices

Exercice 57 – 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 57.1 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; // cast implicite
7     c1 = (Chien)m1; // cast explicite
8     c1 = m1;
9     Mammifere m2=new Mammifere();
10    Chien c2=(Chien)m2; // cast explicite
11  }
12 }
```

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

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

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

Exercice 58 – 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 58.1 Qu'affiche l'exécution du main ?

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

Q 58.2 Rédéfinir la méthode `boolean equals(Object ob)` de la classe `Object` dans la classe `Point`, de façon qu'elle teste l'égalité des coordonnées et non des références. Les instructions de test sont fournies dans la méthode `main`.

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

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

Affiche alors :

```
p1=p2 : true
```

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

Q 58.3 Que se passe-t-il si dans la méthode `main`, on rajoute à la suite les instructions suivantes ? Comment résoudre le problème rencontré ?

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

Le programme affiche :

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

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

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

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

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

Exercice 59 – Contexte de méthode

Soient les classes suivantes (la variable i de la classe `B` masque la variable i héritée de `A`) :

```
1 public class A {
2   public int i=10;
3   public void f(){System.out.println(i);}
4   public void g(){System.out.println(i);}
5 }
6 public class B extends A {
7   public int i=123;
8   public void f(){System.out.println(i);}
```

```
9   public void h(){System.out.println(i);}
10 }
11 public class TestContexteMethode{
12   public static void main(String[] args){
13     A a = new A();
14     B b = new B();
15     a.f();
16     a.g();
17     b.f();
18     b.g();
19     b.h();
20     a=b;
21     a.f();
22     a.g();
23     a.h();
24     ((B)a).h();
25   }
26 }
```

Q 59.1 Qu'affiche ce programme ? Expliquez.

```
1 a.f();    // 10
2 a.g();    // 10 car g() emmene son contexte
3 b.f();    //123
4 b.g();    // 10 etonnant, non? c'est le contexte de A qui est pris
5 b.h();    //123
6
7 a=b;
8 a.f();    //123 f() choisie d'apres le type reel(B) de a
9 a.g();    // 10
10 a.h();   // erreur a la compilation : cannot find symbol method h()
11 ((B)a).h();  // 123
```

Exercice 60 – 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 60.1 : Construire le graphe hiérarchique des classes décrites ci-dessus.

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

Q 60.2 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).

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

Q 60.3 Ecrire une méthode `rouler(double distance)` qui fait avancer un véhicule. A quel niveau de la hiérarchie faut-il l'écrire ?

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

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

Q 60.4 Ecrire les méthodes `void approvisionner(double nbLitres)`, et `boolean enPanne()` (en panne s'il n'y a plus d'essence). A quel niveau de la hiérarchie faut-il les écrire ?

Il faut les écrire dans la classe `AMoteur` :

```
1  public void approvisionner(double nbLitres) {
2    if (niveauEssence+nbLitres > capaciteReservoir)
3       System.out.println("ca deborde..!");
4    else {
5       niveauEssence +=nbLitres;
6       System.out.println("ajoute "+ nbLitres + " litres dans " + this);
7    }
8  }
9  public boolean enPanne() {
10   if (niveauEssence==0) System.out.println("plus d'essence!\n");
11   return (niveauEssence==0);
12 }
```

Q 60.5 Ecrire la classe `Velo` avec consructeur et méthode `toString()` et une méthode `void transporter(String depart, String arrivee)` qui affiche par exemple "le vélo n°2 a roulé de Dijon à Châlon".

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

Q 60.6 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.

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

```
8       return "␣Voiture␣"+super.toString();
9     }
10    public void transporter(int n, int km) {
11      if (enPanne()) {
12        return;
13      }
14      System.out.println(this + "␣transporte␣"+ n +"␣personnes␣sur␣"+ km + "␣km");
15    }
16  }
```

Q 60.7 : 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".

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

Q 60.8 Peut-on factoriser la déclaration de la méthode transporter, et si oui, à quel niveau ?

Non, les signatures sont toutes différentes

Q 60.9 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 }
```

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

```
1 public class MainVehicule {
2     public static void main(String[] args) {
3       Vehicule v1=new Velo(17); // nb de vitesses
4       Vehicule v2=new Voiture(40.5,5); // km,nb de Places
5       Vehicule v3=new Camion(100.0,100.0); // km,volume
6       System.out.println("Vehicules␣:␣"+v1+v2+v3);
7       System.out.println();
8       ((AMoteur)v2).approvisionner(35.0); // litres d'essence
9       ((AMoteur)v3).approvisionner(70.0);
10      System.out.println();
11      ((Velo)v1).transporter("Dijon","Valence");
12      ((Voiture)v2).transporter(5,300);
13      ((Camion)v3).transporter("tuiles",1000);
14    }
15  }
```

L'exécution donne :

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

Quizz 14 – Héritage et liaison dynamique

Soient les 4 classes suivantes :

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

et les déclarations suivantes :

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

QZ 14.1 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.
- `a1.f();`
- `p1.f();`

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

a1.f() ;p1.f() ;p1.g() ; sont correctes.
a1.g() ; est incorrecte car la méthode g() n'existe pas dans la classe Animal.
Pour vous en convaincre, changez le nom de la méthode f() par vieillir() et le nom de la méthode g() par nager(). Les animaux et les poissons vieillissent, mais seuls les poissons nagent

QZ 14.2 Que retournent les instructions suivantes ?
- `a1.toString()`
- `p1.toString()`
- `c1.toString()`
- `z1.toString()`

- `a1.toString()` retourne "Animal"
- `p1.toString()` retourne "Poisson"
- `c1.toString()` retourne "Animal" (comme toString() n'est pas définie dans la classe Cheval, c'est la méthode toString() de Animal qui est utilisée)
- `z1.toString()` retourne Zoo@1bc4459 (comme toString() n'est pas définie dans la classe Zoo, c'est la méthode toString() de Object qui est utilisée)

QZ 14.3 Parmi les instructions suivantes, lesquelles provoquent une erreur à la compilation ? Expliquez.
- `Animal a2=p1;`
- `Animal a3=(Animal)p1;`
- `Poisson p2=a1;`
- `Poisson p3=(Poisson)a1;`

Animal a2=p1 ;
Animal a3=(Animal)p1 ;
Poisson p3=(Poisson)a1 ;
=> pas d'erreurs à la compilation,
mais Poisson p3=(Poisson)a1 ; provoque une erreur à l'exécution car l'objet référencé par a1 est réellement un animal pas un poisson
Poisson p2=a1 ; provoque une erreur à la compilation, car on ne peut pas affecter un Animal dans un Poisson

9 TME SOLO

Objectifs

- TD : Révision sur l'héritage
- TME : TME solo (tout, y compris l'héritage)
- Divers :
  - instanceof
  - ArrayList
  - package

Rajouter des exercices sur la visibilité des variables protected en fonction des packages
Exercices 63 (Chemin de Fer) ou 64 (Aquarium)

Exercices

Exercice 61 – 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 61.1 Recherchez la classe `Random`. Combien a-t-elle de constructeurs ? Combien a-t-elle de méthodes ? A quel package appartient cette classe `Random` ? La classe `Math` appartient-elle au même package que la classe `Random` ? Aide : les packages sont écrits tout en minuscule.

2 constructeurs, 10 méthodes + 11 méthodes héritées de la classe Object
java.util
Non, la classe `Math` appartient au package java.lang.

Q 61.2 Recherchez la classe `ArrayList`. D'après la documentation, combien a-t-elle de champs ? Combien a-t-elle de constructeurs ? Combien environ a-t-elle de méthodes ? De quelles classes hérite-t-elle ? A quel package appartient cette classe `ArrayList` ?

1 champs hérité
3 constructeurs
20 méthodes+26 méthodes héritées
Voici l'arbre d'héritage :

```
java.lang.Object
    extended by java.util.AbstractCollection<E>
          extended by java.util.AbstractList<E>
              extended by java.util.ArrayList<E>java.lang.Object

->java.util.AbstractCollection<E>
```

```
->java.util.AbstractList<E>

->java.util.Vector<E>
```

Elle appartient au package : java.util

Q 61.3 Il est possible de créer une documentation pour les classes que vous créez. Pour cela, il faut utiliser la commande javadoc. Récupérez sur le site web de l'UE le fichier `Clavier.java`. Placez ce fichier dans un répertoire vide, puis tapez la commande : javadoc `Clavier.java`, puis : `firefox index.html` Comparez les commentaires du fichier `Clavier.java` et la page web affichée.

L'idée c'est qu'ils comprennent que les commentaires du fichier java sont ajoutés dans la doc HTML.

Exercice 62 – 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 62.1 Créez 3 classes A, B et C chacune dans un fichier différent. Déclarez ces classes public. Mettez la classe `A` dans le package pack1 et les classes B et C dans le package pack2. Ajoutez rapidement une méthode avec des commentaires à chaque classe (pour cela, il faut mettre les commentaires entre `/** ... */` avant le nom de la méthode ou de la classe). Générez une (et une seule) documentation pour ces 3 classes.

Il faut mettre en haut de chaque fichier :

```
package pack1; // pour A
package pack2; // pour B et C
javadoc A.java B.java C.java
```

Attention : seules les classes déclarées public sont dans la doc HTML

Q 62.2 Créez un objet de la classe `A` dans la classe B. Compilez les fichiers. Quelle instruction faut-il ajouter ?

Il faut importer la classe `A` dans la classe B. Au début de la classe B, on ajoute l'instruction : `import pack1.A;`
Il faut compiler les deux fichiers en même temps : `javac A.java B.java`

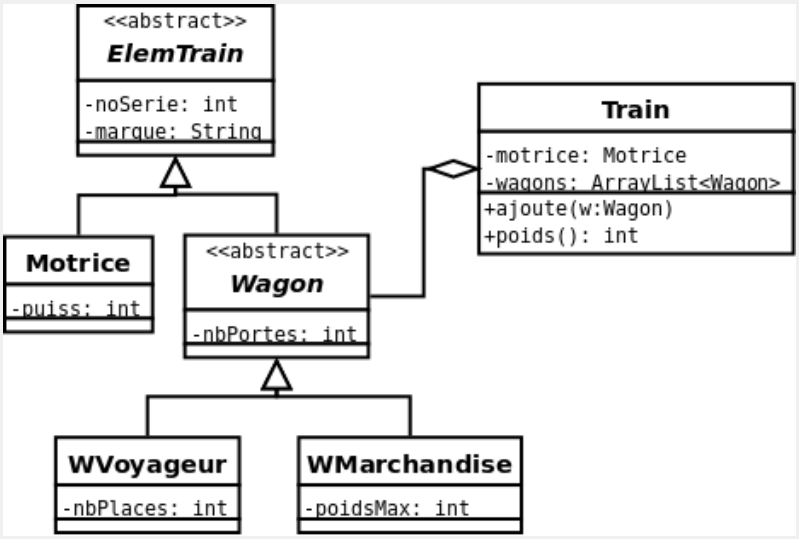
Exercice 63 – Compagnie de chemin de fer (ArrayList, instanceof)

Remarque : Exercice utilisant la classe **ArrayList, instanceof**

Une compagnie de chemin de fer veut gérer la formation de ses trains, à partir de la description suivante. Un train est formé d'éléments de train. Un élément de train possède un numéro de série et une marque. Un élément

de train est soit une motrice, soit un wagon. Une motrice a une puissance. Un wagon a un nombre de portes. Un wagon peut être soit un wagon voyageurs, auquel cas il possède un nombre de places, soit un wagon de marchandise, auquel cas il possède un poids maximum représentant la charge maximale qu'il peut transporter.

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



Q 63.2 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 ».

```
1 public abstract class ElemTrain {
2     private int noSerie;
3     private String marque;
4     public ElemTrain(int no, String m) {
5         noSerie=no;
6         marque=m;
7     }
8     public String toString() { return ""+ noSerie; }
9 }
10 public class Motrice extends ElemTrain {
11     private int puiss;
12     public Motrice(int no, String m, int p) {
13         super(no, m);
14         puiss=p;
15     }
16     public String toString() {
17         return "Motrice␣"+super.toString();
18     }
19 }
20 public abstract class Wagon extends ElemTrain{
21     private int nbPortes;
22     public Wagon(int no, String m, int nbPo) {
23         super(no, m);
24         nbPortes=nbPo;
25     }
```

```
26 }
27
28 public class WVoyageur extends Wagon {
29   private int nbPlaces;
30   public WVoyageur(int no, String m, int nbPo, int nbPl) {
31       super(no,m,nbPo);
32       nbPlaces=nbPl;
33   }
34   public String toString() {
35       return "Wagon␣Voyageur␣"+super.toString();
36   }
37 }
38
39 class WMarchandise extends Wagon {
40   private int pdsMax;
41   public WMarchandise(int no, String m, int nbPo, int pds) {
42       super(no, m, nbPo);
43       pdsMax=pds;
44   }
45   public int getPdsMax() {
46       return pdsMax;
47   }
48 }
```

Q 63.3 Un Train possède une motrice et une une suite de wagons (on gèrera cette suite obligatoirement par la classe `ArrayList` (voir la documentation page 194). Ecrire la classe `Train` avec au minimum un constructeur a un paramètre de type `Motrice` qui construit un train réduit à cette motrice, et ayant donc un ensemble vide de wagons.

```
1 import java.util.ArrayList;
2 public class Train {
3   public Motrice motrice;
4   public ArrayList <Wagon> wagons;
5   public Train(Motrice m) {
6       motrice=m;
7       wagons=new ArrayList<Wagon>();
8   }
9   public void ajoute(Wagon w) {
10      wagons.add(w);
11  }
12
13  public String toString() {
14      return motrice+"␣"+wagons;
15  }
16  public int poids() {
17      Object o;
18      int p=0; // accumulateur
19      int c=wagons.size();
20      for (int i=0; i<c; i++) {
21          o=wagons.get(i);
22          if (o instanceof WMarchandise)
23              p+=((WMarchandise)o).getPdsMax();
24      }
```

```
25        return p;
26    }
27 }
```

Après avoir expliquer le fonctionnement de `instanceof`, il est utile d'insister sur la laideur du code produit :) En général, on préfère des fonctions abstraite de haut niveau pour gérer ce type de problème. Avec instanceof, le code n'est pas évolutif, si on ajoute un nouveau type de wagon, il faudra faire une modification ici aussi : c'est la caractéristique d'un *mauvais code*.

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

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

Q 63.6 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 63.7 Ecrire la méthode principale `public static void main(String[] args)` dans une classe `MainTrain`. Cette méthode crée une motrice, des wagons de voyageur et des wagons de marchandise, crée un train formé de ces éléments, affiche la composition de ce train ainsi que le poids transporté.

```
1 class MainTrain {
2   public static void main(String[] args) {
3     Motrice m=new Motrice(5634,"MMM",2000); // no serie, marque, puiss
4     // no serie, marque, nbPortes, nb voyageurs :
5     WVoyageur wv=new WVoyageur(7845,"WWW",6,100);
6     // no serie, marque, nbPortes, poids max :
7     WMarchandise wm=new WMarchandise(9997, "WWW",2,1000);
8     WMarchandise wm2=new WMarchandise(3087, "WWW2",3,2000);
9     WMarchandise wm3=new WMarchandise(3114, "WWW3",3,1500);
10    Train t=new Train(m);
11    t.ajoute(wv);
12    t.ajoute(wm);
13    t.ajoute(wm2);
14    t.ajoute(wm3);
15    System.out.println("composition du train :");
16    System.out.println(t.toString());
17    System.out.println("poids maximum charge par le train :"+t.poids());
18  }
19 }
20
21 /* EXECUTION :
22 composition du train :
23 Motrice 5634 [Wagon Voyageur 7845, Wagon Marchandise 9997,
24 Wagon Marchandise 3087, Wagon Marchandise 3114]
25 poids maximum charge par le train : 4500
26 Press any key to continue...
27 */
```

Exercice 64 – Extrait de l'examen 2007-2008 S1 : Aquarium

Exceptions, ArrayList, instanceof

L'objet de ce problème consiste à réaliser un aquarium virtuel de taille 500×500 dans lequel évoluent des thons et des requins. On suppose que l'on dispose de la classe `Point` ci-dessous :

```
1 public class Point {
2   public int x,y;
3   public Point(int x, int y){ this.x=x;this.y=y; }
4   public Point() {       /* initialise x et y entre 0 et 499 */
5     this((int)(Math.random() * 500),(int)(Math.random() * 500));
6   }
7   public String toString() { return "("+x+","+y+")␣"; }
8   public double distanceTo(Point p) {
9     int dx = p.x-x;
10    int dy = p.y-y;
11    return Math.sqrt(dx*dx+dy*dy);
12  }
13 }
```

La classe `ArrayList` est une une classe prédéfinie en java qui se trouve dans le package `java.util`. Elle permet de stocker des objets (un peu comme dans un tableau mais sans se préoccuper de la taille du tableau). Pour utiliser cette classe, il faut indiquer dans quel package elle se trouve, en rajoutant 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 `<...>`. L'annexe page 194 donne un extrait des méthodes de la classe `ArrayList`.
Par exemple, voici un exemple d'utilisation des méthodes de cette classe quand ont veut créer une liste de chaines de caractères `(String)` :

```
1 ArrayList<String> liste=new ArrayList<String>();
2 liste.add(new String("Bonjour"));
3 String s=liste.get(0); // recuperation objet a la position zero
4 System.out.println("Taille␣du␣tableau~:␣"+liste.size());
```

Vous utiliserez les deux classes ci-dessus pour ce problème.

Q 64.1 Classe `Poisson`

Q 64.1.1 Définissez une classe abstraite `Poisson` qui possède un attribut `protected position` de type `Point`, avec son constructeur qui assigne à ce poisson une position aléatoire entre $(0, 0)$ et $(499, 499)$. Y mettre l'accesseur public de la position ainsi qu'une méthode abstraite `void move(Point cible)` qui sera définie dans des classes filles. Le point passé en paramètre est le point visé par le mouvement.

```
1  public abstract class Fish {
2    protected Point position;
3    public Fish() {
4      position = new Point();
5    }
6    public abstract void move(Point target);
7    public Point getPosition() {
8      return position;
9    }
10 }
```

Q 64.1.2 Définissez dans la classe `Poisson` une méthode `void verifPosition()` qui replace le poisson dans l'aquarium s'il n'y est pas : si son abscisse ou son ordonnée sont en dehors de l'intervalle $[0, 499]$, on l'y ramènera par un modulo 500.

```java
public void checkBorders() {
  // c'est trop moche les attributs public! mais on ne va pas refaire l'enonce
  position.x %= 500;
  position.y %= 500;
}
```

Q 64.1.3 Définissez une autre classe `PoissonInconnuException` qui étend `Exception` et servira à gérer les cas où un poisson n'est pas d'un type connu.

```java
public class PoissonInconnuException extends Exception {
  PoissonInconnuException(String s) {super(s);
  }
```

Q 64.2 Classe `Requin`

Q 64.2.1 Définissez une classe `Requin` qui hérite de `Poisson` et représente un requin. La méthode `toString()` rend la chaîne `"requin"` suivie des coordonnées du requin, par exemple `"requin(450,200)"`. Définissez-y la méthode `void move(Point cible)` qui assure le déplacement du requin. Le comportement de cette méthode consiste à parcourir la moitié du chemin qui le sépare du point cible, puis à vérifier que le requin est toujours dans l'aquarium (et si ce n'est pas le cas, de l'y replacer) en appelant la méthode définie plus haut `verifPosition()`.

```java
public class Shark extends Fish {
  public Shark() {
    super();
  }
  public void move(Point target) {
    position = new Point((position.x + target.x)/2,(position.y + target.y)/2);
    checkBorders();
  }
  public String toString() {
    return "requin" + getPosition();
  }
  ..............
```

On peut aussi modifier directement la position sans en créer une nouvelle :

```java
public void move(Point target) {
  position.x = (position.x + target.x)/2;
  position.y = (position.y + target.y)/2;
  checkBorders();
}
```

Q 64.3 Classe `Thon`

Q 64.3.1 Définissez une classe `Thon` qui hérite de `Poisson` et représente un thon. Écrivez-y, en plus du constructeur, la méthode `void move(Point cible)`, qui assure le déplacement du thon. Le comportement est le suivant : si le point cible est à une distance supérieure à 60, le thon se déplace aléatoirement en ajoutant à chaque coordonnée de sa position une valeur aléatoire comprise entre -15 et +15. Sinon, le thon parcourt la

moitié du chemin qui le sépare du point. Puis on remet si besoin le poisson dans l'aquarium en appelant la méthode `verifPosition()`. La méthode `toString()` rend la chaîne "thon" suivie des coordonnées du thon, par exemple `"thon(450,200)"`.

```
1 public class Tuna extends Fish {
2   public Tuna() {
3     super();
4   }
5   public void move(Point target) {
6     double d = position.distanceTo(target);
7     if (d>60) {
8       position=new Point(position.x + (int)(1+Math.random()*30)-15,
9       position.y + (int)(1+Math.random()*30)-15);
10    }
11    else {
12      position=new Point((position.x+target.x)/2, (position.y+target.y)/2);
13    }
14    checkBorders();
15  }
16  public String toString() {
17    return "thon"+this.getPosition();
18  }
19 }
```

On peut aussi modifier directement la position sans en créer une nouvelle.

Q 64.4 Classe `PoissonList`
Cette classe `PoissonList` est destinée à gérer la liste des poissons présents dans l'aquarium.

Q 64.4.1 Définissez la classe `PoissonList` qui hérite de `ArrayList`, avec un constructeur sans paramètres et un constructeur de copie, comme dans la classe `ArrayList` dont elle hérite.

```
1 import java.util.*;
2 class FishList extends ArrayList<Fish> {
3   public FishList() { super(); }
4   public FishList(FishList f) { super(f); }
```

Q 64.4.2 Dans cette classe `PoissonList` ajouter une méthode `nbThons()` qui rend le nombre de thons dans cette liste.
Indication : on pourra utiliser l'opérateur `instanceof` qui permet de savoir si un objet est instance d'une classe : l'expression `unObjet instanceof UneClasse` rend `true` si et seulement si l'objet `unObjet` est une instance de la classe `UneClasse`.

```
1    public int nbThons() {
2      int k=0;
3      for (int i=0; i<size(); i++){
4        if (get(i) instanceof Tuna){
5          k++;
6        }
7      }
```

```
8       return k;
9     }
```

Q 64.4.3 Dans cette classe, ajoutez une méthode `int rangPoissonProche(int index)` qui renvoie l'indice du poisson le plus proche dans l'aquarium du poisson dont l'indice est passé en paramètre.

```
1   public int getClosestFish(int index) {
2     int retour = 0;
3     Fish current = get(index);
4     double dmin = Double.MAX_VALUE; // ou 1000, ca suffit
5     for (int i=0;i<size();i++) {
6       if (i!=index) {
7         Fish autre = get(i);
8         double d =current.getPosition().distanceTo(autre.getPosition());
9         if (d<dmin) {
10          dmin = d;
11          retour = i;
12        }
13      }
14    }
15    return retour;
16  }
```

Q 64.4.4 Dans cette classe, ajoutez une méthode `void bougeTousPoissons()`. Cette méthode déplace tous les poissons (thons et requins) en appelant leur méthode `move(Point cible)`, où cible est soit la position du poisson le plus proche (au sens de la question précédente) si celui-ci est un thon, soit le centre de l'aquarium - le point de coordonnées (250,250) - si le poisson le plus proche est un requin (autrement dit : tout poisson "fuit" les requins, tout poisson est attiré par les thons).

```
1     public void moveAllFishes() {
2       for (int i=0;i<size();i++) {
3         Fish f1 = get(i);
4         int closest = getClosestFish(i);
5         Fish f2 = get(closest);
6         Point target = new Point(250,250);
7         if (f2 instanceof Tuna) {
8           target = f2.getPosition();
9         }
10        f1.move(target);
11      }
12    }
```

On dira que deux poissons sont "voisins" si la distance entre eux est inférieure à 2. Chaque fois qu'un requin est "voisin" d'un thon, il le mange et le thon disparaît. Chaque fois que deux thons sont "voisins" l'un de l'autre, ils se reproduisent et un nouveau thon est ajouté à une position aléatoire de l'aquarium entre $(0,0)$ et $(499,499)$. On veut gérer ces cas d'apparition et de disparition de poissons en créant une nouvelle liste de poissons. Ceci sera fait par la méthode `faireUnPas()` dont voici la description : Elle commence par mettre à jour les positions de tous les poissons en appelant la méthode `bougeTousPoissons()` et elle crée un double L2 de cette liste de

poissons. Puis elle parcourt la liste originale, et pour chaque poisson de cette liste et son poisson le plus proche (au sens de la question précédente) elle applique les règles d'ajout et de suppression décrites ci-dessous, et fait la modification dans la nouvelle liste. Lorsqu'elle a parcouru toute la liste elle renvoie la nouvelle `PoissonList` obtenue ainsi.
Règles d'ajout et de suppression :
– S'il s'agit de deux thons, on ajoute un nouveau thon dans la nouvelle liste de poissons.
– S'il s'agit d'un thon et d'un requin, on supprime le thon dans la nouvelle liste de poissons.
– S'il s'agit de deux requins, il ne se passe rien.
– S'il ne s'agit ni de thon ni de requin, on lève une instance de la classe `PoissonInconnuException` qui sera traitée dans le main.

Q 64.4.5 Toujours dans la classe `PoissonList`, complétez la méthode ci-dessous.

```
1  public PoissonList faireUnPas() throws PoissonInconnuException {
2    bougeTousPoissons();
3    // creation d'un double de this :
4    // parcours de this :
5    for (int i=0;i<size();i++) {
6      // on recupere le poisson courant et son plus proche dans l'aquarium :
7      // traitement du couple si les deux poissons sont differents :
```

```
1      public PoissonList faireUnPas() throws PoissonInconnuException {
2        bougeTousPoissons();
3        // creation d'un double de this :
4        FishList v2 = new FishList(this);
5        // parcours de this :
6        for (int i=0;i<size();i++) {
7          // on recupere le poisson courant et son plus proche dans l'aquarium :
8          Fish f1 = get(i);
9          int closest = getClosestFish(i);
10         // traitement du couple si les deux poissons sont differents :
11         if (closest>i) { // pour ne traiter q'une fois le couple
12           Fish f2 = ((Fish)get(closest));
13           double d = (f1.getPosition()).distanceTo(f2.getPosition());
14           if (d>2) continue;
15           if (f1 instanceof Tuna && f2 instanceof Tuna) {// thon et thon
16             v2.add(new Tuna());
17             continue;
18           }
19           if (f1 instanceof Tuna && f2 instanceof Shark) {//thon et requin
20             v2.remove(i);
21             continue;
22           }
23           if (f1 instanceof Shark && f2 instanceof Tuna){ // requin et thon
24             v2.remove(closest);
25             continue;
26           }
27           if (f1 instanceof Shark && f2 instanceof Shark){ //requin requin
28             continue;
29           }
30           throw new PoissonInconnuException("poisson inconnu");
31         }
32       }
33       return v2;
34     }
```

Q 64.5 Classes `Aquarium` et `TestAquarium`

Q 64.5.1 Définissez une classe `Aquarium` qui contient un attribut liste de type `PoissonList` représentant la liste des poissons présents dans l'aquarium. Écrivez-y le constructeur `Aquarium` qui prend en argument deux entiers, `nbThons` et `nbRequins`, représentant le nombre initial de thons et de requins dans la simulation. Le constructeur remplit la liste de poissons avec le nombre adéquat de thons et de requins. Écrivez-y aussi la méthode `toString()` qui rend la liste des poissons contenus dans l'aquarium.

```
1  public class Aquarium {
2    private FishList list;
3    public Aquarium(int nbtunas, int nbsharks) {
4      list = new FishList();
5      for (int i=0;i<nbtunas;i++) {
6        list.add(new Tuna());
7      }
8      for (int i=0;i<nbsharks;i++) {
9        list.add(new Shark());
10     }
11   }
12   public String toString() {
13     return list.toString();
14   }
15 }
```

Q 64.5.2 Ecrivez dans une classe `TestAquarium` la méthode principale, `public static void main(String [] args)`, qui récupère en arguments de la ligne de commande le nombre de thons et le nombre de requins, appelle le constructeur de `Aquarium` avec ces valeurs, affiche la liste des poissons avec leurs coordonnées, appelle la méthode `faireUnPas` et réaffiche la liste des poissons. Pensez à attraper les exceptions qui sont susceptibles d'être levées et à les traiter en affichant un message idoine.

Rappels de cours :

- les arguments passés sur la ligne de commande sont récupérables par le tableau de chaînes de caractères args, paramètre de la méthode main.
- la méthode `int Integer.parseInt(String s)` rend l'entier représenté par la chaîne `s`, ou bien lève une exception `NumberFormatException` si la chaîne `s` ne repésente pas un entier.

Voici deux exemples d'exécution possibles :

```
>java TestAquarium 5 6m
  donnee non entiere
>java TestAquarium 3 2
  la liste des poissons :
  [thon(474,286) , thon(30,301) , thon(27,417) , requin(181,127) , requin(98,400)]
  liste des poissons apres un pas:
  [thon(471,277) , thon(43,305) , thon(25,411) , requin(112,216) , requin(61,405)]
```

Q 64.5.3 Dans la classe `Aquarium`, ajoutez une méthode `void run()` qui réalise une boucle perpétuelle dans laquelle on met sans cesse à jour la liste des poissons (méthode `faireUnPas()`)avec une temporisation de 300 ms (utilisez la méthode `sleep` de la classe `Thread`) et on affiche le nombre total de poissons ainsi que le nombre de thons et de requins. On traitera dans cette méthode les exceptions éventuellement levées.

Rappel de cours : la méthode `static void sleep(long millis)` de la classe `Thread` suspend l'exécution pendant millis millisecondes et peut lever l'exception `InterruptedException`, classe dérivée de `Exception`.

```
1  public class TestAquarium {
2    public static void main(String [] args)  {
3      Aquarium m=null;;
4      try {
5        int nbtunas = Integer.parseInt(args[0]);
6        int nbsharks = Integer.parseInt(args[1]);
7        m= new Aquarium(nbtunas,nbsharks);
8        System.out.println("la liste des poissons : \n"+ m);
9        m.list=m.list.makeStep();
10       System.out.println("liste des poissons apres un pas:\n"+ m);
11     } catch(NumberFormatException e) {
12       System.out.println("donnee non entiere");
13       System.exit(-1);
14     } catch(PoissonInconnuException e) {
15       System.out.println(e.getMessage());
16     }
17   }
18 }
```

Quizz 15 – 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 15.1 Expliquez la ligne 1.

Java est fourni avec un ensemble de classes. Par exemple, les classes String, Math, System. Ces classes sont regroupées en fonction de leurs fonctionnalités dans des ensembles appelés packages. L'instruction `import` permet d'utiliser les classes d'un certain package. L'instruction `import java.util.ArrayList;` permet d'utiliser la classe `ArrayList` du package `java.util`.

QZ 15.2 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 194). Ajoutez-y un constructeur qui prend en paramètre le nombre `n` d'objets à créer à l'initialisation de la liste. Ce constructeur crée aléatoirement 50% d'objets de type `B` et 50% d'objets de type `C` et les ajoute à la liste.

```
1 public class ListeDeA {
2         private ArrayList<A> liste;
3         public ListeDeA (int n) {
4                 liste=new ArrayList<A>();
5                 for(int i=0;i<n;i++) {
6                         if (Math.random()<0.5) {
7                                 liste.add(new B());
8                         } else {
9                                 liste.add(new C());
10                        }
11                }
12        }
13
14
15 public class TestArrayList {
16        public static void main(String [] args) {
17                ListeDeA l=new ListeDeA(8);
18                l.afficherListe();
19                l.afficherMethode();
20        }
21 }
```

QZ 15.3 Ajoutez à la classe `ListeDeA` une méthode `afficherListe()` qui appelle la méthode `afficher()` de chacun des objets de la liste. Utilisez cette méthode dans une méthode `main`.

```
1         public void afficherListe() {
2                 for(A a: liste) {
3                         a.afficher();
4                 }
5         }
```

QZ 15.4 Ajoutez à la classe `ListeDeA` une méthode `afficherMethode()` qui pour chaque objet de la liste appelle la méthode `methodeB()` si cet objet est un objet de type `B`, et appelle la méthode `methodeC()` si cet objet est un objet de type `C`. Utilisez cette méthode dans une méthode `main`.

```
1         public void afficherMethode() {
2                 for(int i=0;i<liste.size();i++) {
3                         A a=liste.get(i);
4                         if (a instanceof B) {
5                                 B b=(B)a;
6                                 b.methodeDeB();
7                         } else if (a instanceof C) {
8                                 C c=(C)a;
9                                 c.methodeDeC();
```

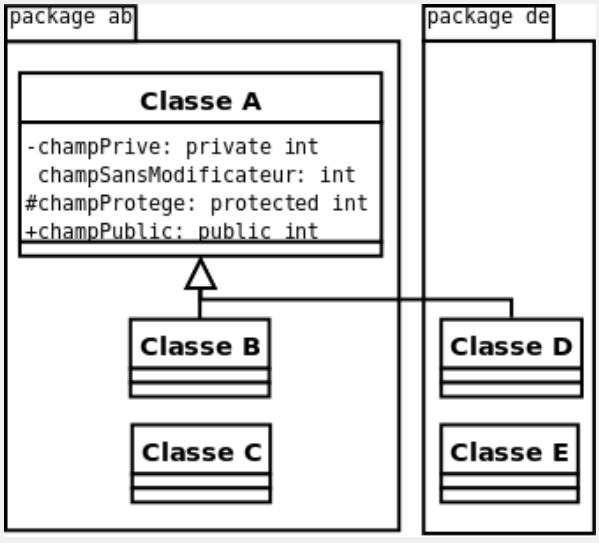
```
10                          } else {
11                                  System.out.println("erreur ni B n C");
12                          }
13                  }
14          }
```

Quizz 16 – 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 }
```

QZ 16.1 Donner la déclaration des classes `B`, `C`, `D` et `E`, et faire un schéma.



Obligatoirement chaque classe dans un fichier différent.

```
1 package abc;
2 public class B extends A { }
3
4 package abc;
5 public class C { }
6
7 package de;
8 public class d extends A { }
9
10 package de;
11 public class E { }
```

QZ 16.2 Compléter le tableau ci-dessous en cochant les cases pour lesquelles les variables d'instance de la classe `A` sont visibles.

	Classe `A`	Classe `B`	Classe `C`	Classe `D`	Classe `E`
`champPrive`					
`champSansModifieur`					
`champProtege`					
`champPublic`					

	Classe `A`	Classe `B`	Classe `C`	Classe `D`	Classe `E`
`champPrive`	X				
`champSansModifieur`	X	X	X		
`champProtege`	X	X	X	X	
`champPublic`	X	X	X	X	X

QZ 16.3 Si la classe `A` n'était pas déclarée `public`, est-ce que cela change la visibilité des variables ?

Oui, les variables ne sont visibles que dans le paquetage. C'est un cas que l'on rencontre rarement.

10 Exceptions

Objectifs

- Exceptions
- Lancement (throw), capture (try-catch) et propagation (throws) d'exceptions prédéfinies
- Lancement, capture et propagation d'exceptions créées par le programmeur
- Passage de paramètres sur la ligne de commande

Propositions :
TD : un ou deux exercices parmi exercices 65, 66, et 67 (Entier Borné)
TME : deux exercices parmi 68 (MonException), **??** (Gestion de Personnes), **??** (Banque), 69 (MonTableau) et 70 (Etudiant).

Exercices

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

- Pour capturer une exception :

```
try {
    instructions qui peuvent lever une exception
} catch (MonException me) {
    System.out.println(me.toString());
} catch (AutreException ae) {
    System.out.println(ae.getMessage());
} finally {
    instructions toujours executees
}
```

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

Exercice 65 – Capture dans le main d'une exception prédéfinie (`try catch`)

Q 65.1 Soit classe `TestAttrapePas0` ci-dessous. Que se passe-t-il lors de l'exécution ?

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

Une exception de dépassement des bornes du tableau n'est pas attrapée par le programmeur et est donc levée directement par java après l'affichage des 5 valeurs du tableau.

```
// EXECUTION :
java TestAttrapePas0
1 2 3 4 5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at TestAttrapePas0.main(TestAttrapePas0.java:8)
```

Q 65.2 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 : depassement des bornes a la position 5"` quand l'exception se produit.

```
1 public class TestAttrapePas0 {
2     public static void main(String[] args){
3         int[] tab= {1,2,3,4,5};
4         try {
5             for (int i=0; i<15; i++)
6                 System.out.print(tab[i] + "␣");
7         } catch(ArrayIndexOutOfBoundsException e) {
8             System.out.println("Exception␣:␣depassement␣des␣bornes␣position␣"+e.
                  getMessage());
9         }
10    }
11 }

// EXECUTION /
> java TestAttrapeDepassTab
1 2 3 4 5
Exception : depassement des bornes position 5

Si le try... catch est de part et d'autre du System.out, on aura 10 fois le message
```

Exercice 66 – `Try, catch, throw, throws`, création d'une exception utilisateur

Q 66.1 Écrire une classe `TestAttrapePas1` dans laquelle on définira une méthode de classe `moyenne(String[] tab)` qui, étant donné un tableau de chaînes de caractères représentant des notes (entiers entre 0 et 20), rend la moyenne entière de ces notes. Testez cette méthode dans un main, en affichant la moyenne des notes passées en argument sur la ligne de commande, sans capturer l'exception éventuellement levée.

Indications :

- Utiliser la méthode `Integer.parseInt` qui transforrme une chaîne de caractères en entier et lève une exception `NumberFormatException` si la chaîne n'est pas un entier.
- Les arguments qui sont passés en ligne de commande sont récupérables par le tableau `String[] args` passé en paramètre de la méthode main.

```java
/** Levee des exceptions predefinies NumberFormatException ou
    ArithmeticException, qui ne sont pas capturees */
public class TestAttrapePas1 {
  public static int moyenne(String [] tab) {
    int note,somme=0,n=0;
    for (int i=0;i<tab.length; i++) {
      note=Integer.parseInt(tab[i]);   //exception non capturee
      somme=somme+note;
      n=n+1;
    }
    return (somme/n);     // exception non capturee
  }
  public static void main(String[] args) {
    System.out.println("La moyenne est : "+moyenne1(args));
  }
}
```

Q 66.2 Que donnent les exécutions suivantes :

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

```
1)  EXECUTION QUAND TOUT VA BIEN :
    java AttrapePas1 10 12 16 18
    La moyenne est : 14
2) EXECUTION QUAND TOUT VA MAL :
    java TestAttrapePas1 12 1j 10 13 15
    Exception in thread "main" java.lang.NumberFormatException:
     For input string: "1j"
       at java.lang.NumberFormatException.forInputString(Unknown Source)
         at java.lang.Integer.parseInt(Unknown Source)
           at java.lang.Integer.parseInt(Unknown Source)
            at TestAttrapePas1.moyenne1(TestAttrapePas1.java:10)
             at TestAttrapePas1.main(TestAttrapePas1.java:18)

3) DEUXIEME EXECUTION QUAND TOUT VA MAL, MAIS AUTREMENT :
ON A OUBLIE DE PASSER LA LISTE DES NOTES
   >java TestAttrapePas1
    Exception in thread "main" java.lang.ArithmeticException: / by zero
      at TestAttrapePas1.moyenne1(TestAttrapePas1.java:14)
         at TestAttrapePas1.main(TestAttrapePas1.java:18)
```

Q 66.3 Dans une classe `TestAttrape2`, reécrire une méthode `moyenne(String[] tab)`qui calcule la moyenne des notes de tab, mais capture cette fois l'exception levée si une note n'est pas entière et la traite en affichant le message « la note n'est pas entière ».

1. Où peut-on attraper l'exception `NumberFormatException` ?
2. Que se passe-t-il si aucune des notes n'est pas entière ou s'il n'y a aucune note ?

1. 1ere version en attrapant l'exception dans le main :

```
/**   TestAttrape2.java :  l'exception est levee dans la boucle,
* mais attrapee dans le main, ce qui interrompt
*  le programme des qu'une note n'est pas entiere  */
public class TestAttrape2 {
  // inchange par raport a la question 1
  public static int moyenne(String [] liste) {
    int note,somme=0,n=0;
    for (int i=0;i<liste.length; i++) {
      note=Integer.parseInt(liste[i]);  // levee de l'exception
      somme=somme+note;
      n=n+1;
    }
    return (somme/n);   // exception non capturee
  }
  public static void main(String [] args) {
    try {
      System.out.println("La moyenne est :"+moyenne(args));
    } catch(NumberFormatException e) {
      System.out.println("la note n'est pas entiere");
    }
  }
}
```

```
EXECUTION :
java TestAttrape2_0 10  12 lm 63 54 pp
la note n'est pas entiere
```

2. 2eme version en attrapant l'exception dans la boucle :

```
/* TestAttrape2.java : capture, A L'INTERIEUR de la BOUCLE,
* d'une exception predefinie. Le programme n'est ainsi
* donc pas interrompue.*/
public class TestAttrape2 {
  public static int moyenne(String [] liste) {
    int note,somme=0,n=0;
    for (int i=0;i<liste.length; i++) {
      try {
        note=Integer.parseInt(liste[i]);
        somme=somme+note;
        n=n+1;
      } catch(NumberFormatException e) {
        System.out.println("la "+(i+1)+" eme note n'est pas entiere\
            textbackslashn");
      }
    }
    return (somme/n);  // exception non capturee
  }
  // inchange par rapport a la qestion1
  public static void main(String [] args) {
    System.out.println("La moyenne est :"+moyenne1(args));
  }
}
```

```
EXECUTION :
  java TestAttrape2 11 lm 16 1e pp 18
```

```
    la 2 eme note n'est pas entiere
    la 4 eme note n'est pas entiere
    la 5 eme note n'est pas entiere
    La moyenne est : 15
  Execution :
    java TestAttrape2 mm reg 6r c5 mm
    la 1 eme note n'est pas entiere
    la 2 eme note n'est pas entiere
    la 3 eme note n'est pas entiere
    la 4 eme note n'est pas entiere
    la 5 eme note n'est pas entiere
    Exception in thread "main" java.lang.ArithmeticException:  by zero
  Il y a levée de l'exception "ArithmeticException" car division par 0
```

Q 66.4 Écrire une classe `AucuneNoteEntiereException` dérivée de la classe `Exception`. Dans une classe `TestAttrape3` réécrire la méthode moyenne qui lancera une instance de la classe `AucuneNoteEntiereException` lorsque ce cas se présentera. Cette exception sera capturée dans le main.

```
1  public class AucuneNoteEntiereException extends Exception {
2    public AucuneNoteEntiereException (String s) {
3      super(s);
4    }
5    public String toString() {
6      return "aucune note n'est valide\n";
7    }
8  }
```

```
1 /* Lancement et capture d'une exception creee par l'utilisateur */
2    public class TestAttrape3 {
3      public static int moyenne(String [] liste)
4      throws AucuneNoteEntiereException {
5        int note,somme=0,n=0;
6        for (int i=0;i<liste.length; i++) {
7          try {
8            note=Integer.parseInt(liste[i]);
9            somme=somme+note;
10           n=n+1;
11         } catch(NumberFormatException e) {
12           System.out.println("la "+(i+1)+" eme note n'est pas entiere\n");
13         }
14       }
15       if (n==0) throw new AucuneNoteEntiereException("Lancement de
             AucuneNoteEntiereException:");
16       return (somme/n);
17     }
18     public static void main(String [] args) {
19       try {
20         System.out.println("La moyenne est : "+moyenne(args));
21       }   catch(AucuneNoteEntiereException e) {
22         System.out.println(""+ e.getMessage()+ e);
23       }
24     }
25   }
```

Q 66.5 Que donne l'exécution de la commande `javaTestAttrape3 mm reg 6r c5 mm` ?

```
java TestAttrape3 mm reg 6r c5 mm
  la 1 eme note n'est pas entiere
  la 2 eme note n'est pas entiere
  la 3 eme note n'est pas entiere
  la 4 eme note n'est pas entiere
  la 5 eme note n'est pas entiere
Lancement de AucuneNoteEntiereException : aucune note n'est valide
```

Q 66.6 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` ?

```
1  /* On capture plusieurs type d'exceptions d'ou
2  * plusieurs blocs catch */
3  public class PasEntre0et20Exception extends Exception {
4    public PasEntre0et20Exception(String s){super(s);}
5  }
6  public class TestIntervalle {
7    public static int moyenne(String [] liste) throws AucuneNoteEntiereException
         {
8      int note,somme=0,n=0;
9      for (int i=0;i<liste.length; i++) {
10       try {
11         note=Integer.parseInt(liste[i]);
12         if (note < 0)
13               throw new PasEntre0et20Exception("negative");
14         if (note > 20)
15               throw new PasEntre0et20Exception("superieure a 20");
16         somme=somme+note;
17         n=n+1;
18       } catch(PasEntre0et20Exception e) {
19         System.out.println("la " + (i+1)+" eme note est "+e.getMessage());
20       } catch(NumberFormatException e) {
21         System.out.println("la "+(i+1)+ "eme note n'est pas entiere\n");
22       }
23     }
24     if (n==0)
25       throw new AucuneNoteEntiereException("Lancement de
             AucuneNoteEntiereException :");
26     return (somme/n);
27   }
28   public static void main(String [] args) {
29     try {
30       System.out.println("La moyenne est : "+moyenne(args));
31     } catch(AucuneNoteEntiereException e) {
```

```
32          System.out.println(""+ e.getMessage()+ e);
33        }
34      }
35    }
```

```
java TestIntervalle -10 -3 45 -78 -6 21
la 1 eme note est negative
la 2 eme note est negative
la 3 eme note est superieure a 20
la 4 eme note est negative
la 5 eme note est negative
la 6 eme note est superieure a 20
Lancement de AucuneNoteEntiereException : aucune note n'est valide
```

Exercice 67 – 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 67.1 Écrire dans une classe `TestEntierBorne` la méthode `main` qui saisit une valeur entière. On utilisera obligatoirement la méthode `saisirLigne` de la classe `Clavier` non standard qui affiche un message et lit un `String`, puis la méthode `parseInt` de la classe `Integer` (voir la documentation en ligne pour cette méthode) pour transformer la chaîne saisie en entier. Dans le cas où la saisie n'est pas un entier, cette méthode peut lever l'exception `NumberFormatException`.
Que se passe-t-il à l'exécution si la saisie n'est pas entière ? Expliquez.

Une erreur `NumberFormatException` est levée

Q 67.2 Traiter maintenant l'exception levée dans le main. Ajouter les instructions pour que le main s'endorme pendant n secondes en utilisant la méthode `sleep`de la classe `Thread` qui lève une exception de type `InterruptedException`.

```
1 try {
2   n=Integer.parseInt(Clavier.lireLigne("Entrez␣un␣nb␣secondes␣(entier)␣:␣"));
3   System.out.println("entier␣lu␣:␣"+n);
4   n1000=1000*n;
5   System.out.println("debut␣de␣l'arret␣de␣"+n+"␣secondes");
6   Thread.sleep(n1000);
7 } catch(NumberFormatException e) {
8   System.out.println("la␣saisie␣n'est␣pas␣un␣entier");
9 }catch(InterruptedException e) {
10  System.out.println("pb␣ds␣sleep(n)");
11 }
```

Q 67.3 Écrire la classe `EntierBorne` qui est une classe « enveloppe » du type simple int, i.e. qui "enveloppe" une variable d'instance de type int dans un objet de cette classe. Écrire le constructeur à un paramètre de type `int`qui peut lever l'exception `HorsBornesException` si la valeur qui est passée en paramètre est plus grande que 10000 ou plus petite que -10000, et la méthode `toString()`. On définira pour cela la classe `HorsBornesException`.

```
1   class HorsBornesException extends Exception {
2     public HorsBornesException(String s) {super(s);}
3   }
4   class EntierBorne { //les bornes pour les el de la classe :
5     static int maxEntier=10000,minEntier=-10000;
6     int valeur;// la valeur de l'Entier
7     public EntierBorne(int i) throws HorsBornesException {
8       if (i<minEntier)
9          throw new HorsBornesException("Constructeur:tu es trop petit");
10      if (i>maxEntier)
11         throw new HorsBornesException("Constructeur:tu es trop grand");
12      valeur=i;
13    }
14    public int getValeur() {return valeur;}
15    public String toString() { return ""+valeur+" "; }
16  }
```

Q 67.4 Définir la méthode `EntierBorne somme(EntierBorne i)` qui rend un objet `EntierBorne` dont la valeur est la somme de cet élément et du paramètre. Elle pourra lever sans la capturer l'exception `HorsBornesException` si la somme est trop grande.

```
1  public EntierBorne somme(EntierBorne i) throws HorsBornesException {
2    // rend un objet EntierBorne somme de cet el et de i
3    int val=this.valeur+i.valeur;
4    if (val<minEntier)
5       throw new HorsBornesException(" somme:je suis trop petit ");
6    if (val>maxEntier)
7       throw new HorsBornesException(" somme:je suis trop grand ");
8    return new EntierBorne(val);
9  }
```

Q 67.5 Définir la méthode `EntierBorne divPar(EntierBorne i)` qui rend un objet EntierBorne dont la valeur est la division entière de cet élément par le paramètre i. Elle pourra lever l'exception `HorsBornesException` ou l'exception `DivisionParZeroException`.

```
1  public EntierBorne divPar(EntierBorne i) throws ArithmeticException,
2  HorsBornesException {
3    // rend un objet EntierBorne division de cet el par i
4    if (i.valeur == 0) throw new ArithmeticException("division par zero");
5    int val=(int)(this.valeur/i.valeur);
6    return new EntierBorne(val);
7  }
```

Q 67.6 On définira ensuite la méthode `EntierBorne factorielle()` qui calcule la factorielle de cet élément. Elle pourra, en plus de l'exception `HorsBornesException`, lever l'exception `IllegalArgumentException` dans le cas où n serait négatif.

```
1  public EntierBorne factorielle() throws Exception {
2    if (this.getValeur() <0)
3       throw new IllegalArgumentException("x doit etre >= 0");
4    int fact=1;
5    for (int i=2; i<=getValeur(); i++)
6       fact *= i;
7    return new EntierBorne(fact);
8  }
```

Q 67.7 Créer un jeu de tests pour ce programme, en réfléchissant aux différents cas possibles et les tester dans le main.

```
1  public class TestEntierBorne {
2    public static void main(String [] args) {
3      int n=0; // nbbre saisi de secondes
4      int n1000=0; // n en milisecondes
5      int m=0; //
6      EntierBorne eb=null,eb1=null,eb2=null;
7      try {
8        n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier (nb de secondes d
             'arret): "));
9        System.out.println("entier lu : "+n);
10       n1000=1000*n;
11       System.out.println("debut de l'arret de "+n+" secondes");
12       Thread.sleep(n1000);
13       throw new Exception("Tout va bien");
14     } catch(NumberFormatException e) {
15       System.out.println("la saisie n'est pas un entier");
16     }catch(InterruptedException e) {
17       System.out.println("pb ds sleep(n)");
18     }
19     catch(Exception e) {
20       System.out.println("Fin de l'arret de "+n+" secondes");
21     }
22     System.out.println();
23     System.out.println("TEST SOMME : ");
24     try {
25       n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
26       System.out.println(" "+n);
27       eb=new EntierBorne(n); // peut lever exception HorsBornes
28       m=Integer.parseInt(Clavier.saisirLigne("Entrez un autre entier : "));
29       System.out.println(" "+m);
30       eb1=new EntierBorne(m); // peut lever exception HorsBornes
31       eb2=eb.somme(eb1); //leve exception HorsBornes si somme trop grande
32       throw new Exception("Tout va bien");
33     } catch(HorsBornesException e) {
34       System.out.println(e.getMessage());
35     } catch(NumberFormatException e) {
36       System.out.println("la saisie n'est pas entiere");
```

```
37        } catch(Exception e) {
38          System.out.println("EntierBornes : "+eb+" "+eb1);
39          System.out.println("somme des EntierBorne : "+eb2);
40        }
41        /*-------------------------------------------------*/
42        System.out.println();
43        System.out.println("TEST DIVISION : ");
44        try {
45          n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
46          System.out.println(" "+n);
47          eb=new EntierBorne(n);
48          m=Integer.parseInt(Clavier.saisirLigne("Entrez un autre entier : "));
49          System.out.println(" "+m);
50          eb1=new EntierBorne(m); // peut lever HorsBornesException
51          eb2=eb.divPar(eb1);
52          throw new Exception("Tout va bien");
53        } catch(NumberFormatException e) {
54          System.out.println("la saisie n'est pas entiere");
55        } catch(HorsBornesException e) {
56          System.out.println(e.getMessage());
57        } catch(ArithmeticException e){
58          System.out.println("Entiers Bornes : " + eb + eb1);
59          System.out.println(e.getMessage());
60        } catch(Exception e){
61          System.out.println("EntierBornes : " + eb + eb1);
62          System.out.println("division des EntierBorne : "+eb2);
63        }
64        //-------------------------------------------------
65        System.out.println();
66        System.out.println("TEST FACTORIELLE : ");
67        try {
68          n=Integer.parseInt(Clavier.saisirLigne("Entrez un entier : "));
69          System.out.println(" "+n);
70          eb=new EntierBorne(n); // peut lever exception HorsBornes
71          //leve HorsBornes si factorielle trop grande
72          // ou IllegalArgument si negatif
73          eb2=eb.factorielle();
74          throw new Exception("Tout va bien");
75        } catch(HorsBornesException e) {
76          System.out.println(e.getMessage());
77        } catch(NumberFormatException e) {
78          System.out.println("la saisie n'est pas entiere");
79        } catch(Exception e) {
80          System.out.println("EntierBorne : "+eb);
81          System.out.println("factorielle de EntierBorne : "+eb2);
82        }
83      }
84    }
```

```
EXECUTION 1:
Entrez un entier(nb de secondes d'arret) :
3
3
debut de l'arret de 3 secondes
Fin de l'arret de 3 secondes
TEST SOMME :
```

```
Entrez un entier :
5
5
Entrez un autre entier :
10000
10000
somme:je suis trop grand
TEST DIVISION :
Entrez un entier :
5000
5000
Entrez un autre entier :
10
10
EntierBornes : 5000 10
division des EntierBorne : 500
TEST FACTORIELLE :
Entrez un entier :
9
9
Constructeur:tu es trop grand
**********************************
EXECUTION 2 :
Entrez un entier(nb de secondes d'arret) :
2
2
debut de l'arret de 2 secondes
Fin de l'arret de 2 secondes
TEST SOMME :
Entrez un entier :
78
78
Entrez un autre entier :
-789
-789
EntierBornes : 78  -789
somme des EntierBorne : -711
TEST DIVISION :
Entrez un entier :
987
987
Entrez un autre entier :
0
0
Entiers Bornes : 987 0
division par zero
TEST FACTORIELLE :
Entrez un entier :
6
6
EntierBorne : 6
factorielle de EntierBorne : 720
```

Exercice 68 – throw, throws, finally

Q 68.1 Donnez l'affichage produit par le programme ci-après. Expliquez les résultats.

Cet exercice a pour but d'expliquer le fonctionnement de finally. Regarder l'affichage obtenue. Chaque méthode test() permet de comprendre un peu mieux le fonctionnement des exceptions et de finally.

```
1 public class MonException extends Exception {
2    public MonException(String s) {
3       super(s);
4       System.out.println("\n␣MonException␣:␣constructeur");
5    }
6 }
7
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 }
```

Attention : finally est TOUJOURS EFFECTUE qu'il y ait une exception ou non (voir test5).

Attention : comme le montre le test4, finally ne capture pas l'exception.

Mais si l'exception est capturée, alors le programme continue avec l'instruction suivant le finally, sinon le bloc finally est exécutée, puis l'exception est déléguée à la méthode appelante.

```
 VOICI L'AFFICHAGE OBTENUE
MonException : constructeur
main : test1 : capture de l'exception MonException: lancee dans test1
MonException : constructeur
test2 : capture de l'exception : MonException: lancee dans test2
test2 : fin de la methode
MonException : constructeur
test3 : capture de l'exception : MonException: lancee dans test3
test3 : finally est effectue
```

```
test3 : fin de la methode
MonException : constructeur
test4 : finally est effectue
main : test4 : capture de l'exception MonException: lancee dans test4
test5 : finally est effectue
test5 : fin de la methode
Fin du programme
```

Exercice 69 – 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 69.1 Définir une classe `MonTableau` qui possède les variables `tab` (tableau d'entiers) et `lgReelle` (entier) donnant le nombre de cases de `tab` réellement utilisées dans le tableau. Au départ, `lgReelle` vaut 0. Ecrire un constructeur prenant en paramètre la taille du tableau, et une méthode `ajouter(int n)` qui ajoute la valeur n à la suite du tableau sans vérifier s'il reste de la place.

```
1 public class MonTableau {
2     private int [] tab;
3     private int lgReelle;
4
5     public MonTableau(int max) {
6         tab=new int[max];
7         lgReelle=0;
8     }
9
10    public void ajouter(int i) {
11        tab[lgReelle]= i;
12        lgReelle++;
13    }
14 }
```

Q 69.2 Ecrire la méthode `main` qui crée un objet `MonTableau` de 3 cases et y ajoute 10 entiers. Exécutez le programme. Que se passe-t-il ?

```
1 public static void main(String[] args){
2     MonTableau t=new MonTableau(3);
3     for(int i=1;i<=10;i++) {
4         t.ajouter(i);
5     }
6 }

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at MonTableau.ajouter(MonTableau.java:11)
        at MonTableau.main(MonTableau.java:19)
```

Q 69.3 Capturer dans la méthode `main` l'exception précédemment levée, et afficher le texte `"Depassement des bornes a la position 3"` en utilisant la méthode `getMessage()` de l'exception levée.

```
1 public static void main(String[] args){
2     MonTableau t=new MonTableau(3);
3     try {
4         for(int i=1;i<=10;i++) {
5             t.ajouter(i);
6         }
7     } catch (ArrayIndexOutOfBoundsException e) {
8         System.out.println("Depassement␣des␣bornes␣position␣:␣"+e.getMessage());
9     }
10 }
```

Q 69.4 Définir un nouveau type d'exception appelée `TabPleinException`.

```
1 public class TabPleinException extends Exception {
2     public TabPleinException(String s) {
3         super(s);
4     }
5 }
```

Q 69.5 Modifier la méthode `ajouter` pour lever cette exception quand le tableau est plein. Capturer cette exception dans la méthode `main`. Que retourne les méthodes `getMessage()` et `toString()` de cette exception ?

```
1     public void ajouter(int i) throws TabPleinException {
2         if (lgReelle < tab.length) {
3             tab[lgReelle]= i;
4             lgReelle++;
5         } else {
6             throw new TabPleinException("Le␣tableau␣est␣plein");
7         }
8     }
9
10 public static void main(String[] args){
11     MonTableau t=new MonTableau(3);
12     try {
13         for(int i=1;i<=10;i++) {
14             t.ajouter2(i);
15         }
16     } catch (ArrayIndexOutOfBoundsException e) {
17         System.out.println("Depassement␣des␣bornes␣position␣:␣"+e.getMessage());
18     } catch (TabPleinException tpe) {
19         System.out.println("Depassement␣:␣"+tpe.getMessage());
20         System.out.println("Depassement␣:␣"+tpe.toString());
21     }
22 }
```

La méthode `getMessage()` retourne le texte passée dans le constructeur. La méthode `toString()` de `Exception` retourne NomDeLaClasse+le texte passée dans le constructeur.

Exercice 70 – 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 70.1 Écrire la classe `Etudiant` correspondant à la description ci-dessus avec un constructeur à un paramètre, le nom. La méthode `toString()` rend le nom de l'étudiant suivi de ses notes.

```
1 public class Etudiant {
2     private String nom;
3     private int [] tabNotes;
4     private int nbNotes;
5     public Etudiant(String n) {
6         nom=n;
7         tabNotes=new int[5];
8         nbNotes=0;
9     }
10    public String toString() { // rend le nom et les notes
11        String s= "";
12        for (int i=0; i<nbNotes; i++) {
13            s += tabNotes[i] + "␣";
14        }
15        return nom + "␣" + s ;
16    }
17 }
```

Q 70.2 Ajouter la méthode `void entrerNote(int note)` qui entre la note dans la liste des notes de cet étudiant. Elle lèvera une exception `TabNotesPleinException` (à définir) dans le cas où le tableau de notes de cet étudiant serait plein. Cette exception sera capturée dans le `main`.

```
1 public class TabNotesPleinException extends Exception {
2     public TabNotesPleinException(String s) {
3         super(s);
4     }
5 }
```

```
1     public void entrerNote(int note) throws TabNotesPleinException {
2         if (nbNotes<tabNotes.length) {
3             tabNotes[nbNotes]=note;
4             nbNotes++;
5         } else {
6             throw new TabNotesPleinException("le␣tableau␣de␣notes␣de␣l'etudiant␣"
                  + this.nom+ "␣est␣plein");
7         }
8     }
```

Q 70.3 En supposant que la classe qui contient le `main` s'appelle `TestEtudiants`, on veut passer sur la ligne de commande une liste d'étudiants avec leurs notes, par exemple :

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

On supposera que chaque donnée est correcte (pas de mélange entre lettres et chiffres), et que la première donnée est un nom.
Ces données sont de deux types : chaîne de caractères et entier. On va utiliser le fait qu'un entier ne fait pas lever d'exception à la méthode `Integer.parseInt` alors qu'une chaîne de caractères lui fait lever l'exception `NumberFormatException`.
Rappel : la méthode `int Integer.parseInt(String s)` rend l'entier représenté par la chaîne `s`, ou bien lève une exception `NumberFormatException` si la chaîne `s` ne repésente pas un entier.
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
```

```java
public class TestEtudiants {
    public static void main(String [] args) {
      int note;
      // lecture des donnees sur ligne de commande :
      for (int i=0;i<args.length; i++) {
        try {
          note=Integer.parseInt(args[i]);
          System.out.print(args[i]+" c'est une note, ");
        }
        catch(NumberFormatException e) {
          System.out.println("\n" + args[i]+" c'est un nom, ");
        }
      }
    }
}
```

Q 70.4 On souhaite gérer dans la classe `Etudiant` une liste au sens `ArrayList` d'étudiants. Une liste d'étudiants ne dépend pas d'un étudiant en particulier. Qu'en concluez-vous sur le type de variables que doit être la liste d'étudiants ? Ajouter les instructions nécessaires dans la classe `Etudiant`.

La liste d'étudiants doit être statique.

```java
class Etudiant {
    private static ArrayList<Etudiant> vEtu=new ArrayList<Etudiant>();
    public Etudiant(String n) {
        ...
        vEtu.add(this);
    }
    public static int getNbEtudiants() {
```

```
8           return vEtu.size();
9       }
10      public static String listeEtudiants() {
11          return vEtu.toString();
12      }
13  }
```

Q 70.5 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 ]
```

```
1   public class TestEtudiants {
2     public static void main(String [] args) {
3       int note;
4       Etudiant eCourant=null;
5       // lecture et stockage des donnees de la ligne de commande :
6       for (int i=0;i<args.length; i++) {
7         try {
8             note=Integer.parseInt(args[i]);
9             eCourant.entrerNote(note);
10        } catch(TabNotesPleinException e) {
11            System.out.println(e.getMessage());
12        } catch(NumberFormatException e) {
13            eCourant = new Etudiant(args[i]);
14        }
15      }
16      //affichage des etudiants :
17      System.out.println("les␣"+ Etudiant.getNbEtudiants() + "etudiants␣:␣\n" +
            Etudiant.listeEtudiants());
18    }
19  }
```

11 Manipulation de flux entrée / sortie

Objectifs

- Gestion d'arborescences de fichiers
- Flux de lecture / écriture
- Manipulation (création, lecture, écriture, ...) de fichiers textuels
- Mise en mémoire tampon
- Entrée / Sortie standard

Propositions :
TD : Exo 71 sur les objets `File`, Exo 72 (au moins questions 2,3 et 4), Exo 73 sur la copie de fichiers, Exo 74 sur la mise en mémoire tampon, Exo 75 (Production de compte rendu de TME) et Exo 76 autour de la classe Clavier

TME : Finaliser Exo 72 sur le traitement de texte et realiser l'Exo 75 sur la production automatique de compte rendu de TME

La classe File

Le paquetage `java.io` définit un grand nombre de classes pour gérer les entrés / 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 :

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

Exercice 71 – 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 71.1 Dire ce qu'affiche l'execution suivante : `java TestFile "./2i002/TME11/Files/fichier1.txt"`
- Si le répertoire `"./2i002/TME11/Files"` existe
- Si le répertoire `"./2i002/TME11/Files"` n'existe pas

```
// Si le répertoire existe :
Le fichier existe : non
Le fichier existe : oui
/home/.../2i002/TME11/Files/fichier1.txt
./fichier1.txt

// Si le répertoire n'existe pas
Le fichier existe : non
java.io.IOException: Le chemin d'acces specifie est introuvable

// L'exception est envoyee a la creation du fichier
```

Q 71.2 Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier

```
// Ajouter les deux lignes suivantes juste apres la construction de f :

File r=f.getParentFile();
if (r!=null){
    r.mkdirs();
}
```

Q 71.3 Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la classe `File`

```
public static void pwd(){
        File f=new File(".");
        System.out.println(f.getAbsolutePath());
}
```

Q 71.4 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)

```
1 public static void ls(File f){
2         File[] childs=f.listFiles();
3         for(int i=0;i<childs.length;i++){
4             File fic=childs[i];
5             if (fic.isFile()){
6                 System.out.println(fic.getAbsolutePath());
7             }
8         }
9 }
```

Q 71.5 Écrire une méthode `lsRecursif(File f)` permettant d'afficher tous les noms de fichiers contenus dans l'arborescence prenant sa racine au niveau du répertoire passé en paramètre (ne pas afficher les répertoires)

```
1 public static void lsRecursif(File f){
2         File[] childs=f.listFiles();
3         for(int i=0;i<childs.length;i++){
4             File fic=childs[i];
5             if (fic.isFile()){
6                 System.out.println(fic.getAbsolutePath());
7             } else {
8                 lsRecursif(fic);
9             }
10        }
11 }
```

Les flux

Outre la classe `File`, le paquetage `java.io` (i pour input, o pour output) définit une multitude de classes permettant la manipulation de flux de lecture/écriture. Ces flux permettent des échanges de données entre le programme et d'autres entités, qui peuvent être :
- une variable du programme (par exemple, pour la construction de chaînes de caractères)
- la console de l'utilisateur (`System.in` : entrée standard, `System.out` : sortie standard)
- un fichier (création, lecture, écriture, modifications, ...)
- la mémoire
- ...

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

Ces classes de flux sont néanmoins des classes abstraites. Les classes à utiliser sont préfixées par :
- la source pour les flux entrants (`FileInputStream`, `FileReader`, `InputStreamReader`, `StringReader`...)

– la destination pour les flux sortants (`FileOutpuStream`, `FileWriter`, `OutputStreamWriter`, `StringWriter`...)

La classe `Reader` définit principalement les méthodes suivantes :

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

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

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

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

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

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

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

Exercice 72 – 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 72.1 Écrire une méthode `String saisie()` qui demande à l'utilisateur de saisir une ligne de texte tant que la ligne entrée par l'utilisateur est différente de la chaîne "_fin_". Cette méthode retourne une chaîne de caractères contenant la concaténation de toutes les lignes saisies. Proposez une première solution utilisant des concaténations de `String`. Puis proposez une deuxième solution utilisant un seul objet `StringWriter`.

Il existe 4 façons de concaténer des chaines de caractères : l'opérateur "+", la méthode `String.concat()`, et les classes `StringBuffer` et `StringBuilder`. Mais avant de les comparer, il faut comprendre la base du problème : **l'immutabilité** de la classe `String`. `String` étant une classe immutable, une variable de type `String` ne peut pas être modifiée. Lorsque l'on croit modifier une `String`, en vérité, on crée une nouvelle `String` à partir de l'ancienne qui sera soumise au garbage collector.

Exemple : `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour");a=new String(a+" tous le monde");`
Le premier objet est détruit lorsqu'on crée le deuxième.

```
public static String saisie(){
        String texte="";
        String ligne="";
        while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0)
            {
            texte+=ligne+"\n";
        }
        return(texte);
}

// On peut faire remarquer qu'il serait plus efficace d'utiliser un StringWriter
public static String saisie(){
        StringWriter texte=new StringWriter();
        String ligne="";
        while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0)
            {
            texte.write(ligne+"\n");
        }
        return(texte.toString());
}

// ou un java.lang.StringBuilder
public static String saisie(){
        StringBuilder texte=new StringBuilder();
        String ligne="";
        while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0)
            {
            texte.append(ligne+"\n");
        }
        return(texte.toString());
    }
```

Q 72.2 Écrire une méthode `affiche(String fichier)` affichant le contenu du fichier dont le nom est passé en paramètre.

```
public static void affiche(String fichier) throws IOException{
        File file=new File(fichier);
        FileReader lecteur=new FileReader(file);
        try {
            int c;
            StringBuilder texte=new StringBuilder();
            while((c=lecteur.read())!=-1){
                texte.append((char)c);
            }
            System.out.println(texte.toString());
        }
        finally{
            lecteur.close();
        }
}
```

```
16
17 // --- Ou avec un tableau de char ---
18 public static void affiche(String fichier) throws IOException{
19         File file=new File(fichier);
20         FileReader lecteur=new FileReader(file);
21         try {
22             int nb=0;
23             char[] buf=new char[100];
24             StringBuilder texte=new StringBuilder();
25             while((nb=lecteur.read(buf))>0){
26                 texte.append(buf);
27                 buf=new char[100];
28             }
29             System.out.println(texte.toString());
30         }
31         finally{
32             lecteur.close();
33         }
34 }
```

Q 72.3 Écrire une méthode `afficheLignes(String fichier)` affichant, en numérotant les lignes, le contenu du fichier passé en paramètre.

```
1 public static void afficheLignes(String fichier) throws IOException{
2         File file=new File(fichier);
3         FileReader lecteur=new FileReader(file);
4         try {
5             int c;
6             int nbLines=1;
7             StringBuilder texte=new StringBuilder();
8             boolean nligne=true;
9             while((c=lecteur.read())!=-1){
10                 if (nligne){
11                     texte.append(nbLines+" : ");
12                     nbLines++;
13                     nligne=false;
14                 }
15                 char car=(char) c;
16                 texte.append(car);
17                 if (car=='\n'){
18                     nligne=true;
19                 }
20             }
21             System.out.println(texte.toString());
22         }
23         finally{
24             lecteur.close();
25         }
26 }
```

Q 72.4 Écrire une méthode `ecrireTexte(String fichier)` permettant de créer un nouveau fichier contenant un texte saisi par l'utilisateur.

```java
public static void ecrireTexte(String fichier) throws IOException{

        File file=new File(fichier);
        FileWriter ecrivain=new FileWriter(file);
        try {
            String saisie=saisie();
            ecrivain.write(saisie);
            ecrivain.flush(); // ecriture dans le fichier maintenant
        }
        finally{
            ecrivain.close();
        }
}
```

Q 72.5 Écrire une méthode `ajouteTexte(String fichier)` permettant d'ajouter, en fin de fichier passé en paramètre, du texte saisi par l'utilisateur.

```java
public static void ajouteTexte(String fichier) throws IOException{

        File file=new File(fichier);
        File tmp=new File("tmp_file.txt");

        FileWriter ecrivain=new FileWriter(tmp);
        FileReader lecteur=new FileReader(file);
        try {
                int nb=0;
                char[] buf=new char[100];
                while((nb=lecteur.read(buf))>0){
                    ecrivain.write(buf,0,nb-1);
                    ecrivain.write("\n");
                    ecrivain.flush();
                    buf=new char[100];

                }

                String saisie=saisie();
                ecrivain.write(saisie);
                ecrivain.flush();
        }
        finally{
                ecrivain.close();
                lecteur.close();
                if (!file.delete()){
                    throw new IOException("Probleme suppression fichier");
                }
                if (!tmp.renameTo(file)){
                    throw new IOException("Probleme renommage du fichier");
                }
        }
}

// Ou avec le constructeur de FileWriter qui accepte en second argument un boolean
      append
```

```
36
37 public static void ajouteTexte(String fichier) throws IOException{
38
39         File file=new File(fichier);
40         FileWriter ecrivain=new FileWriter(file,true);
41
42         try {
43                 String saisie=saisie();
44                 ecrivain.write(saisie);
45                 ecrivain.flush();
46         }
47         finally{
48                 ecrivain.close();
49         }
50 }
```

Q 72.6 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`.

```
1 public static void replace(int num, String newLigne, String fichier) throws
      IOException{
2         File file=new File(fichier);
3         File tmp=new File("tmp_file.txt");
4
5         if (file.exists()){
6             FileWriter ecrivain=new FileWriter(tmp);
7             FileReader lecteur=new FileReader(file);
8             try {
9                 int c;
10                int nbLines=0;
11                StringBuilder ligne=new StringBuilder();
12
13                while((c=lecteur.read())!=-1){
14                    char car=(char) c;
15                    ligne.append(car);
16                    if (car=='\n'){
17                        nbLines++;
18                        if (nbLines==num){
19                            ecrivain.write(newLigne+"\n");
20                        } else {
21                            ecrivain.write(ligne.toString());
22                        }
23                        ligne=new StringBuilder();
24                    }
25                }
26                if (nbLines<num) {
27                    System.out.println("Pas de ligne "+num);
28                }
29            }
30            finally{
31                ecrivain.close();
32                lecteur.close();
33                if (!file.delete()){
```

```
34                      throw new IOException("Probleme suppression fichier");
35                  }
36                  if (!tmp.renameTo(file)){
37                      throw new IOException("Probleme renommage du fichier");
38                  }
39              }
40          } else {
41              System.out.println("Pas de ligne "+num);
42          }
43
44  }
```

Q 72.7 Écrire un programme proposant à l'utilisateur un menu lui permettant d'éditer un fichier dont le chemin est passé en argument. Exemple :

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

```
1 public static int menu(String fichier){
2         System.out.println("Fichier "+fichier);
3         System.out.println("1. Ajouter texte");
4         System.out.println("2. Afficher fichier");
5         System.out.println("3. Remplacer ligne");
6         System.out.println("4. Quitter");
7         int choix=Clavier.saisirEntier("Votre choix ?");
8         return(choix);
9 }
10
11 public static void main(String[] args){
12         int choix;
13         String fichier=args[0];
14         try {
15             File file=new File(fichier);
16             if (!file.exists()){
17                 File parent=file.getParentFile();
18                 if (parent!=null){
19                     parent.mkdirs();
20                 }
21                 file.createNewFile();
22             }
23
24             while((choix=menu(fichier))!=4){
25                 if ((choix<1) || (choix>4)){
26                     System.out.println("Choix invalide");
27                     continue;
28                 } else if (choix==1){
29                     ajouteTexte(fichier);
30                 } else if (choix==2){
31                     afficheLignes(fichier);
32                 } else if (choix==3){
```

```
33                      int num=Clavier.saisirEntier("Quele ligne a remplacer ?");
34                      String newLigne=Clavier.saisirLigne("Entrez la nouvelle ligne "
                            +num);
35                      replace(num,newLigne,fichier);
36                  }
37              }
38          } catch(IOException e) {
39              System.out.println(e);
40          }
41  }
```

Exercice 73 – Copie de fichiers binaires

Q 73.1 Écrire un programme permettant de copier un fichier binaire passé en premier argument sous le nom passé en second.

```
1 import java.io.*;
2
3 public class CopyFiles {
4    public static void copyFile(File srcFile, File destFile) throws IOException {
5       InputStream oInStream = new FileInputStream(srcFile);
6       OutputStream oOutStream = new FileOutputStream(destFile);
7
8       // Transfer bytes from in to out
9       byte[] oBytes = new byte[1024];
10      int nLength;
11
12      while ((nLength = oInStream.read(oBytes)) > 0) {
13         oOutStream.write(oBytes, 0, nLength);
14         oOutStream.flush();
15      }
16      oInStream.close();
17      oOutStream.close();
18   }
19   public static void main(String[] args){
20      try {
21         copyFile(new File(args[0]),new File(args[1]));
22      } catch(IOException e) {
23         System.out.println(e);
24      }
25   }
26 }
```

La mise en mémoire tampon

La mise en mémoire tampon des données lues permet d'améliorer les performances des flux sur une entitée. 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 74 – Mise en mémoire tampon

Q 74.1 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.

```
BufferedReader in= new BufferedReader(new FileReader(new File("text.txt")));
```

Q 74.2 Écrire une méthode `afficheLignesFichier(String fichier)` qui affiche ligne après ligne le texte du fichier dont le chemin est passé en paramètre.

```
public static void afficheLignesFichier(String fichier) throws IOException,
    FileNotFoundException{
        BufferedReader lecteur=new BufferedReader(new FileReader(new File(fichier)
            ));
        try {
            String ligne="";
            while((ligne=lecteur.readLine())!=null){
                System.out.println(ligne);
            }
        }
        finally{
                lecteur.close();
        }
}
```

Q 74.3 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.

```
// --- Avec heritage ---
import java.io.*;
public class Ecrivain extends PrintWriter {
    public Ecrivain(String fichier) throws IOException{
        super(new BufferedWriter(new FileWriter(new File(fichier))));
    }
    public void finalize(){
        close();
    }
}

// --- Sans heritage ---
```

```
13 import java.io.*;
14 public class Ecrivain {
15     private PrintWriter pw=null;
16     public Ecrivain(String fichier) throws IOException{
17         pw=new PrintWriter(new BufferedWriter(new FileWriter(new File(fichier))));
18     }
19     public void close(){
20         if (pw!=null){
21             pw.close();
22             pw=null;
23         }
24     }
25     public void finalize(){
26         close();
27     }
28     public void println(String s){
29         if (pw!=null){
30             pw.println(s);
31         } else {
32             System.out.println("Flux ferme");
33         }
34     }
35     public void print(String s){
36         if (pw!=null){
37             pw.print(s);
38         } else {
39             System.out.println("Flux ferme");
40         }
41     }
42 }
```

Exercice 75 – 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 prenoms, noms et numeros d'etudiants des utilisateurs du programme (une ligne par étudiant). Le fichier doit se terminer par une ligne "Groupe : <numero du groupe>". Enfin, chaque répertoire d'exercice contient deux fichiers "intitule.txt" et "executions.txt", le premier contenant l'énoncé de l'exercice, le second contenant les résultats d'exécution des programmes ainsi que les observations qui ont pu avoir été faites.

Q 75.1 Écrire un programme `RenduTMEProducer` prenant en argument le chemin du répertoire de TME concerné par le compte rendu et produisant en racine de ce répertoire un fichier "compteRenduTME.txt" de la forme de celui que vous avez l'habitude de rendre en fin de TME.

```
1 import java.io.*;
2 import java.util.ArrayList;
3 public class RenduTMEProducer {
```

```
4
5       public static ArrayList<File> getAllJavaFiles(File f){
6           ArrayList<File> ret=new ArrayList<File>();
7           File[] childs=f.listFiles();
8           for(int i=0;i<childs.length;i++){
9               File fic=childs[i];
10              if (fic.isFile()){
11                  String path=fic.getAbsolutePath();
12                  int iext=path.lastIndexOf(".");
13
14                  if ((iext>0) && (iext<path.length()-4)){
15                      String ext=path.substring(iext+1,iext+5);
16                      if (ext.compareTo("java")==0){
17                          ret.add(fic);
18                      }
19                  }
20              } else {
21                  ret.addAll(getAllJavaFiles(fic));
22              }
23          }
24          return(ret);
25      }
26
27      public static String getTexteFromFile(File f) throws IOException{
28          StringBuilder sb=new StringBuilder();
29          BufferedReader lecteur=new BufferedReader(new FileReader(f));
30          try {
31              String ligne="";
32              while((ligne=lecteur.readLine())!=null){
33                      sb.append(ligne+"\n");
34              }
35          }
36          finally{
37              lecteur.close();
38          }
39          return(sb.toString());
40      }
41      public static void produceCompteRendu(String rep) throws IOException{
42          File tme=new File(rep);
43          if (!tme.exists()){
44              throw new IOException("Le repertoire de TME donne n existe pas");
45          }
46          tme=new File(tme.getAbsolutePath());
47          if (!tme.isDirectory()){
48              throw new IOException("Le chemin donne n est pas un repertoire");
49          }
50
51          File 2i002=tme.getParentFile();
52          if (2i002.getName().compareTo("2i002")!=0){
53              throw new IOException("Le repertoire parent du repertoire donne n'est
                    pas 2i002");
54          }
55
56          File etudiants=new File(2i002.getAbsolutePath()+"/etudiants.txt");
57          if (!etudiants.exists()){
58              throw new IOException("Le repertoire 2i002 ne contient pas de fichier
```

```
                nomme␣etudiants.txt");
59          }
60
61          File rendu=new File(tme.getAbsolutePath()+"/compteRenduTME.txt");
62          System.out.println(rendu.getAbsolutePath());
63          boolean ok=true;
64          if (rendu.exists()){
65              ok=false;
66              String ecrase=Clavier.saisirLigne("Le␣repertoire␣"+rep+"␣contient␣deja
                    ␣un␣compte␣rendu␣de␣TME.␣Voulez␣vous␣l'ecraser␣?␣(Oui/Non)");
67              ecrase=ecrase.toLowerCase();
68              if (ecrase.compareTo("oui")==0){
69                  ok=true;
70              }
71          }
72          if (ok){
73              PrintWriter ecrivain=new PrintWriter(new BufferedWriter(new FileWriter
                    (rendu)));
74              try {
75                  ecrivain.println("Compte␣Rendu␣2i002␣\n");
76                  String texteEtudiants=getTexteFromFile(etudiants);
77                  ecrivain.println(texteEtudiants+"\n");
78                  ecrivain.println("Numero␣du␣TME␣:␣"+tme.getName()+"\n");
79
80                  File[] exos=tme.listFiles();
81                  for(int i=0;i<exos.length;i++){
82                      File exo=exos[i];
83                      if (exo.isDirectory()){
84                          System.out.println("exo␣:␣"+exo.getAbsolutePath());
85                          ecrivain.println("//␣--------------------");
86                          ecrivain.println("Intitule␣de␣l'exercice␣:");
87                          File intitule=new File(exo.getAbsolutePath()+"/intitule.
                                txt");
88                          if (intitule.exists()){
89                              ecrivain.println(getTexteFromFile(intitule));
90                          }
91                          ecrivain.println("");
92                          ecrivain.println("Classes␣:␣\n");
93                          ArrayList<File> classes=getAllJavaFiles(exo);
94                          for(File classe:classes){
95                              ecrivain.println(getTexteFromFile(classe)+"\n");
96                          }
97                          ecrivain.println("Executions␣et␣observations␣:");
98                          File execution=new File(exo.getAbsolutePath()+"/executions
                                .txt");
99                          if (execution.exists()){
100                             ecrivain.println(getTexteFromFile(execution));
101                         }
102                         ecrivain.println("");
103                     }
104                 }
105             } catch(IOException e) {
106                 ecrivain.close();
107                 rendu.delete();
108                 throw(e);
109             }
```

```
110             finally{
111                 ecrivain.close();
112             }
113         }
114     }
115
116     public static void main(String[] args){
117         try {
118             produceCompteRendu("ExempleArborescence/2i002/TME11");
119         } catch(IOException e) {
120             System.out.println(e+"\n");
121             e.printStackTrace();
122         }
123
124     }
125 }
```

Entrée / Sortie standard

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

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

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

Exercice 76 – Classe Clavier

Q 76.1 Sachant que le paquetage `java.io` contient une classe de flux `InputStreamReader` permettant de lire des caractères à partir d'un flux entrant d'octets, réécrire le code de la classe `Clavier`, notamment :

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

```
1 import java.io.* ;
2
3 /**Cette classe implante des saisies au clavier par lecture d'une ligne. */
4 public class Clavier {
5
6     private static final BufferedReader in =
7         new BufferedReader (new InputStreamReader (System.in)) ;
8
9     /** Affiche le message et retourne un int lu au clavier. */
10    public static int saisirEntier (String mess){
11        while(true){
```

```
12              try {
13                  return Integer.parseInt (saisirLigne ( mess)) ;
14              } catch (NumberFormatException e) {
15                  mess = "Recommencez␣:␣" ;
16              }
17          }
18      }
19
20      /** Affiche le message et retourne une ligne lue au clavier. */
21      public static String saisirLigne (String mess) {
22          System.out.println (mess) ;
23          try {
24              return in.readLine () ;
25          } catch (IOException e){
26              return null; // provisoire !!
27          }
28      }
29  } // Clavier
```

Quizz 17 – `String`, classe immutable

QZ 17.1 Combien d'objets sont créés dans les instructions ci-après ?
`String a="Bonjour"; a=a+" tous le monde";`

Explications à vérifier.
Il y a deux objets créés. `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour");a=new String(a+" tous le monde");`
Le premier objet est détruit lorsqu'on crée le deuxième.

QZ 17.2 Donnez une solution équivalente en utilisant un `StringWriter`.

Un seul objet nécessaire.

```
1 import java.io.StringWriter;
2
3 StringWriter sw=new StringWriter();
4 sw.write("Bonjour");
5 sw.write("␣tous␣le␣monde");
6 System.out.println(sw.toString());
```

1 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

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

Commentaires

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

Divers

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

Principales instructions

Instruction

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

Instruction de contrôle - Conditionnels

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

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

Instruction de contrôle - Boucles

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

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

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

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

Tableau de codage des types simples

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

Table de priorité des opérateurs

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

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

La classe `Math` (standard)

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

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

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

La classe `String` (standard)

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

La classe `Arraylist` (standard)

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

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

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 : ");`

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

2 Environnement Linux

Création et gestion de répertoires sous Linux

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

Démarrage sous Linux

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

3 Annales

Examen du 10 janvier 2011 Durée : 2h00

Exercice 77 – Partie A : Questions de cours (20 points)

Q 77.1 (9 points) On considère les 2 classes suivantes :

```
1  class Transport {
2    private int cptTransports=0;
3    private int numero;
4    protected String compagnie;
5    public Transport(String compagnie) {
6      compagnie=compagnie;
7      cptTransports++;
8      numero=cptTransports;
9    }
10   public abstract void seDeplacer();
11 }
12 class Bus extends Transport {
13   public Bus() {
14     compagnie="Inconnue";
15     numero=5;
16   }
17   public Bus() {
18     super(compagnie);
19   }
20   public Bus(String compagnie) {
21     this(compagnie);
22   }
23 }
```

Ces 2 classes contiennent 9 erreurs, dont 7 sont détectées à la compilation. Pour chaque erreur, donnez le numéro de la ligne et expliquez, puis si possible, donnez une correction pour l'erreur. Remarque : plusieurs erreurs peuvent se trouver sur la même ligne.

Il y a 6 erreurs détectées à la compilation. Le fait que "seDeplacer() n'est pas définie dans Bus" n'est pas détecté à la compilation tant que la classe `Transport` n'est pas déclarée abstract. Les deux autres erreurs sont le compteur pas static et ligne 6 : `compagnie=compagnie;`
Ligne 1 : la méthode `seDeplacer()` est déclarée abstract, la classe `Transport` devrait être aussi déclarée abstract : `abstract class Transport ...`
Ligne 2 : `cptTransports` compte le nombre de transports créés, ce devrait être une variable de classe : `private static int cptTransports=0;`
Ligne 6 : la valeur de la variable locale compagnie est ajoutée à elle-même. Il manque le mot-clé `this` : `this.compagnie=compagnie;`
Ligne 14 : il manque l'appel au constructeur de la classe mère : `super("Inconnue");` Ici, l'appel est obligatoire, car la classe mère ne possède pas de constructeur sans paramètre, l'appel au constructeur par défaut ne peut être effectué.
Ligne 15 : la variable numero est déclarée "private" dans la classe mère, on ne peut pas l'utiliser dans la classe fille
Ligne 17 : ce constructeur a la même signature que le premier constructeur. Deux méthodes ne peuvent avoir la méthode signature.

Ligne 18 : la variable compagnie n'est pas une variable locale à la méthode, mais la variable protected de la classe mère. Cette variable n'a pas encore été initialisée. Une correction possible est : `this("CompagnieInconnue");`
Ligne 21 : appel récursif, le construction s'appelle lui-même. Une correction possible est : `super(compagnie);`
Autre erreur : Manque la définition de la méthode abstraite seDeplacer() dans la classe Bus. La classe `Bus` doit soit être déclarée abstract soit définir la méthode : `public void seDeplacer() System.out.println("Je roule !!!");`

Q 77.2 (6 points) Soit le programme suivant qui ne contient **pas** d'erreurs aux lignes 1 à 9 :

```
1  abstract class Personne { }
2  class Etudiant extends Personne {
3    private static int cptEtudiants=0;
4    private String niveau;
5    public void afficherNiveau() { System.out.println(niveau); }
6    public static void afficherNbEtudiants() {
7      System.out.println("Il␣y␣a␣"+cptEtudiants+"etudiants.");
8    }
9    public static void main(String [] args) {
10     Etudiant e1=new Etudiant();
11     Personne p1=new Personne();
12     Personne p2=e1;
13     Etudiant e2=p2;
14     niveau="L2";
15     cptEtudiants++;
16     e1.afficherNiveau();
17     afficherNiveau();
18     p2.afficherNiveau();
19     e1.afficherNbEtudiants();
20     afficherNbEtudiants();
21   }
22 }
```

Q 77.2.1 Parmi les lignes 10 à 13, lesquelles sont fausses ? Expliquez, et corrigez si possible.

La ligne 11 est fausse, car comme la classe `Personne` est déclarée abstract, on ne peut pas faire un `new Personne()`.
La ligne 13 est fausse, car les types sont incompatibles, on ne peut pas affecter un handle de type `Personne` à un handle de type `Etudiant`. Il faut faire un cast explicite : `Etudiant e2=(Etudiant)p2;`

Q 77.2.2 La ligne 14 est-t-elle fausse ? Si oui, expliquez et proposez une solution, sinon expliquez pourquoi elle est juste.

La ligne 14 est fausse, car niveau est une variable d'instance, on ne peut pas l'utiliser directement dans une méthode static. Pour l'utiliser dans une méthode static, il faut utiliser une instance. Par exemple : `e1.niveau="L2";` Remarque : comme la méthode main appartient à la classe Etudiant, on peut utiliser la variable privée etudiant dans cette méthode.

Q 77.2.3 Même question que la question 77.2.2 pour la ligne 15.

La ligne 15 est juste, car `cptEtudiants` est une variable static, et la méthode main est une variable static de la même classe. On peut donc l'utiliser direcement sans instance.

Q 77.2.4 Quelles lignes sont fausses parmi les lignes 16 à 20 ? Expliquez.

Ligne 17 : `afficherNiveau()` est une méthode d'instance, on ne peut pas l'utiliser dans une méthode static sans utiliser d'instance. Par exemple : `e1.afficherNiveau();`
Ligne 18 : le handle p2 est de type `Personne`, il ne connaît pas la méthode `afficherNiveau()` de la classe `Etudiant`.

Q 77.3 (5 points) Soit le contenu du fichier `Test.java` suivant :

```
1  import java.awt.*;
2  import javax.swing.*;
3  class Publicite extends JPanel {
4    private String message;
5    public Publicite(String message) {
6      super();
7      this.message=message;
8      setPreferredSize(new Dimension(message.length()*40,200));
9    }
10   public void paintComponent(Graphics g) {
11     super.paintComponent(g);
12     g.setFont(new Font("arial",Font.BOLD,64));
13     g.drawString(message,50,120);
14   }
15 }
16 public class Test {
17   public static void main(String [] args) {
18     JFrame f=new JFrame();
19     Publicite pub=new Publicite("Bienvenue␣!");
20     f.getContentPane().add(pub);
21     f.pack();
22     f.setVisible(true);
23   }
24 }
```

Q 77.3.1 A quoi servent les lignes 1 et 2 ?

Les classes JFrame, JPanel, Color ... sont définies dans des packages. Les lignes 1 et 2 importent ces classes afin de pouvoir les utiliser dans les classes de l'utilisateur.

Q 77.3.2 Expliquez brièvement ce que fait chacune des lignes 18, 20, 21 et 22.

Ligne 18 : création d'une fenêtre graphique
Ligne 20 : ajout du panneau pub à la fenêtre graphique
Ligne 21 : la taille de la fenêtre est ajustée à la taille du panneau
Ligne 22 : affichage de la fenêtre dans l'écran

Q 77.3.3 Que fait ce programme ?

Ce programme crée une fenêtre graphique, puis affiche le message Bienvenue en arial dans la fenêtre.

Q 77.3.4 A quoi sert la méthode `paintComponent` ? Quand est-elle appelée ? Faut-il faire un appel explicite à cette méthode ?

La méthode paintComponent permet de dessiner dans le panneau. Elle est appelée par le système chaque fois que la fenêtre doit être redessinée.

Q 77.3.5 Expliquez brièvement ce que fait chacune des lignes 11 à 13.

Ligne 11 : appel à la méthode paintComponent de la classe `JPanel` afin de redessiner le panneau
Ligne 12 : changement de la fonte d'écriture
Ligne 13 : écriture du message dans le fenêtre

Exercice 78 – Partie B : Problème (40 points)

Remarques préliminaires
- Sauf indications contraires, toutes les variables d'instance et de classe seront déclarées privées, toutes les méthodes seront déclarées publiques.
- Si besoin, vous pouvez ajouter des variables et méthodes. Pour gagner du temps, vous n'êtes pas obligé d'ajouter la méthode `toString()` et les accesseurs à chaque classe, sauf s'ils sont demandés.

On considère les ordinateurs d'une entreprise. Un ordinateur a une adresse IP. Les ordinateurs personnels sont des ordinateurs qui ont un seul processeur et qui sont associés au nom d'une personne. Les supercalculateurs sont des ordinateurs qui possèdent entre 2 et 8 processeurs. Une entreprise possède des ordinateurs.

Q 78.1 (3 points) Donnez le schéma de la hiérarchie d'héritage Java des classes : `Processeur, Ordinateur, OrdinateurPersonnel, SuperCalculateur, Entreprise` et `Object`.

Q 78.2 (4 points) Un processeur a un identifiant unique et a une fréquence. Ecrire la classe `Processeur` qui contiendra au moins les variables et méthodes suivantes :
- `cpt` : un compteur initialisé à 1 000 000,
- `id` : l'identifiant unique du processeur au format "CPUXXXXXXX" où XXXXXXX est un nombre supérieur à 1 000 000,
- `frequence` : le nombre d'opérations effectuées par le processeur en une seconde, ce nombre sera exprimé en GHz et stocké dans une variable de type double,
- un constructeur qui prend en paramètre la fréquence du processeur,
- un constructeur sans paramètre qui appel le premier constructeur pour initialiser aléatoirement la fréquence du processeur entre 1 GHz et 5 GHz (non compris). Aide : utiliser `Math.random()` qui retourne un double entre 0 et 1 (non compris).
- l'accesseur de la variable frequence,
- une méthode `String toString()` qui retourne une chaîne de caractères représentant le processeur. Par exemple, pour le processeur dont l'id est "CPU1000001" et dont la fréquence est 3.06GHz, cette chaîne a le format : `"Processeur CPU1000001 (3.06GHz)"`.

```
1 public class Processeur {
2       private static int cpt=1000000;
3       private String id;
```

```
4       private double frequence; // nb operations par seconde
5       public Processeur(double frequence) {
6         this.frequence=frequence;
7         cpt++;
8         id="CPU"+cpt;
9       }
10      public Processeur() {
11        this(Math.random()*4+1); // Entre 1 et <5 GHz
12      }
13      public double getFrequence() {return frequence; }
14      public String toString() {
15        return "Processeur␣"+id+"␣("+frequence+"GHz)";
16      }
17  }
```

Q 78.3 (4 points) Un ordinateur a une adresse IP. Ecrire la classe `Ordinateur` qui contiendra au moins les variables et méthodes suivantes :
- `adresseIP` : une chaîne de caractères,
- un constructeur prenant en paramètre l'adresse IP de l'ordinateur,
- une méthode `String toString()` qui retourne l'adresse IP de l'ordinateur,
- une méthode `double getFrequence()` dont le but est de retourner le nombre d'opérations que peut effectuer l'ordinateur en une seconde. Si l'ordinateur possède un seul processeur alors la fréquence de l'ordinateur est égale à celle du processeur. Si l'ordinateur possède plusieurs processeurs, alors la fréquence de l'ordinateur est égale à 90% de la somme totale des fréquences des processeurs de l'ordinateur.

Questions : La fréquence de l'ordinateur dépend du nombre de processeurs de cet ordinateur, qu'en concluez-vous ? Que faudra-t-il faire dans les classes qui héritent de la classe `Ordinateur` ?

Comme on ne connait pas dans la classe `Ordinateur` le nombre de processeurs de l'ordinateur, on en conclue que la méthode `getFrequence()` doit être déclarée abstraite, et donc que la classe `Ordinateur` doit aussi être déclarée abstraite. Toute classe qui hérite de `Ordinateur` doit définir `getFrequence()` ou être déclarée abstraite.

```
1 /** Classe abstraite */
2 public  abstract class Ordinateur {
3     private String adresseIP;
4     public Ordinateur(String adresseIP){
5       this.adresseIP=adresseIP;
6     }
7     public String toString()} {
8     return adresseIP;
9   }
10  /** Methode abstraite*/
11  public abstract double getFrequence();
```

Q 78.4 (3 points) 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 quatrième nombre de l'adresse IP. Si ce nombre est supérieur à 255 alors le troisième nombre est incrémenté. Remarque : cette méthode s'occupe seulement du troisième et quatrième chiffres de l'adresse IP, elle ne s'occupe pas du cas où la prochaine IP est celle après 192.168.255.255.

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

Q 78.5 (5 points) Un ordinateur personnel possède un processeur et connaît le nom de son propriétaire. Ecrire la classe `OrdinateurPersonnel` qui hérite de la classe `Ordinateur` et qui contient notamment les variables et méthodes suivantes :

- `pro` : le processeur de l'ordinateur,
- `nomProprio` : le nom du propriétaire de l'ordinateur,
- un constructeur qui prend en paramètre le nom du propriétaire, qui affecte une adresse IP à l'ordinateur à l'aide de la classe AdresseIP, et qui initialise le processeur de l'ordinateur avec une fréquence initialisée aléatoirement entre 1 et 5 (non compris) GHz.
- une méthode `String toString()` qui retourne une chaîne au format suivant : `"Ordinateur personnel 192.168.0.2 Processeur CPU1000024 (1.79GHz) appartenant à Jacques"`

```
1 /** Il ne faut pas oublier de definir la methode getFrequence(). */
2 public class OrdinateurPersonnel extends Ordinateur {
3       private Processeur pro;
4       private String nomProprio;
5       public OrdinateurPersonnel(String nomProprio) {
6         super(AdresseIP.getAdresseIP());
7         pro=new Processeur();
8         this.nomProprio=nomProprio;
9       }
10      public double getFrequence() {
11        return pro.getFrequence();
12      }
13      public String toString() {
14        return "Ordinateur␣Personnel␣"+super.toString()+"␣"+pro+"␣appartenant␣a␣"+
              nomProprio;
15      }
16      public String getNomProprio() {
17        return nomProprio;
18      }
19 }
```

Q 78.6 (5 points) Un super calculateur est un ordinateur qui possède entre 2 et 8 processeurs (le nombre de processeurs est choisi aléatoirement (voir `double Math.random()`)). Ecrire la classe `SuperCalculateur` qui contient notamment un tableau de processeurs `tabPro` et un constructeur sans paramètre.

```
/** Il ne faut pas oublier de faire heriter la classe SuperCalculateur de la
    classe  Ordinateur et de definir la methode getFrequence().*/
public class SuperCalculateur extends Ordinateur {
    private Processeur [] tabPro;
    public SuperCalculateur() {
      super(AdresseIP.getAdresseIP());
      tabPro=new Processeur[(int)(Math.random()*7)+2]; // Entre 2 et 8
      for(int i=0;i<tabPro.length;i++) {
        tabPro[i]=new Processeur();
      }
    }
    public double getFrequence() {
      double frequenceTotale=0;
      for(int i=0;i<tabPro.length;i++)
      frequenceTotale+=tabPro[i].getFrequence();
      return 0.9*frequenceTotale;
    }
}
```

Q 78.7 Une entreprise a un nom et des ordinateurs (représentés par un tableau d'ordinateurs appelé `tabOrdi`). Le nombre d'ordinateurs que peut avoir une entreprise est limité par la place disponible dans ses locaux. A sa création, une entreprise a donc un nombre `nbMaxOrdi` maximal d'ordinateurs qu'elle peut posséder et un nombre `nbOrdi` réel d'ordinateurs qu'elle possède au moment de sa création. Des ordinateurs peuvent être ensuite ajoutés à l'entreprise. Si l'on souhaite ajouter un ordinateur alors qu'on a atteint le nombre maximal d'ordinateurs, le système lève l'exception : `DepassementNbOrdiMaxException`.

Q 78.7.1 (2 points) Ecrire la classe `DepassementNbOrdiMaxException` qui contient un constructeur prenant en paramètre un entier (le nombre d'ordinateurs maximal qui a été atteint) et qui, en cas de dépassement, affichera le message d'erreur : `"Le nombre d'ordinateurs maximal est depasse (nbOrdiMax=200)"`.

```
public class DepassementNbOrdiMaxException extends Exception {
    public DepassementNbOrdiMaxException(int nbOrdiMax) {
      super("Le nombre d'ordinateurs max est depasse (nbOrdiMax="+nbOrdiMax+")");
    }
}
```

Q 78.7.2 (9 points) Soit la classe `Nom` suivante qui retourne aléatoirement un nom.

```
public class Nom {
    private static String [] noms={"Pierre","Paul","Jacques"};
    public static String getNom() {
        return noms[(int)(Math.random()*noms.length)];
    }
}
```

Ecrire la classe `Entreprise` qui possède notamment les méthodes suivantes :

- `ajouterOrdinateur (Ordinateur ordi)` qui ajoute un ordinateur au tableau d'ordinateurs. Si le tableau est plein, alors l'exception `DepassementNbOrdiMax Exception` est levée et l'erreur est déléguée à la méthode appelante.

- `int ajouterNOrdinateurs(int n)` qui ajoute `n` ordinateurs à l'entreprise. En moyenne 20% des ordinateurs ajoutés sont des supercalculateurs, les autres sont des ordinateurs personnels. Cette méthode doit utiliser la méthode `ajouterOrdinateur` et retourne le nombre d'ordinateurs que l'on a réellement pu ajouter à l'entreprise.
- un constructeur avec 3 paramètres : le nom de l'entreprise, le nombre d'ordinateurs maximal que peut avoir l'entreprise et le nombre d'ordinateurs que l'entreprise a lors de sa création (dont 20% de supercalculateurs, les autres étant des ordinateurs personnels).
- `double getFrequence()` qui retourne la somme des fréquences de tous les ordinateurs de l'entreprise.
- `afficherNoms()` qui affiche le nom de toutes les personnes de l'entreprise qui ont un ordinateur personnel. Aide : l'opérateur `instanceof` renvoie `true` si l'objet spécifié est du type d'objet spécifié. Usage : *`nomObjet`* `instanceof` *`TypeObjet`*

```
1 public class Entreprise {
2     private String nomEntreprise;
3     private Ordinateur [] tabOrdi;
4     private int nbOrdi;
5     public void ajouterOrdinateur (Ordinateur ordi) throws
          DepassementNbOrdiMaxException {
6         if ( nbOrdi >= tabOrdi.length )
7             throw new DepassementNbOrdiMaxException(nbOrdi);
8         tabOrdi[nbOrdi]=ordi;
9         nbOrdi++;
10        System.out.println("Ajout de "+ordi);
11    }
12    public int ajouterNOrdinateurs(int n) {
13        int i=0;
14        try {
15            for(i=0;i<n;i++) {
16                if (Math.random()<0.2)
17                    ajouterOrdinateur(new SuperCalculateur());
18                else
19                    ajouterOrdinateur(new OrdinateurPersonnel(Nom.getNom()));
20            }
21        } catch(DepassementNbOrdiMaxException d) {
22            System.out.println(d);
23        }
24        return i;
25    }
26    public Entreprise(String nom,int nbOrdiMax,int nbOrdi) {
27        nomEntreprise=nom;
28        tabOrdi=new Ordinateur[nbOrdiMax];
29        this.nbOrdi=ajouterNOrdinateurs(nbOrdi);
30    }
31    public double getFrequence() {
32        double somme=0;
33        for(int i=0;i<nbOrdi;i++) {
34            somme+=tabOrdi[i].getFrequence();
35        }
36        return somme;
37    }
38    public void afficherNoms(){
39        System.out.println("Personnes qui ont un ordinateur personnel :");
40        for(int i=0;i<nbOrdi;i++) {
41            if ( tabOrdi[i] instanceof OrdinateurPersonnel ) {
42                OrdinateurPersonnel op=(OrdinateurPersonnel)tabO[i];
43                System.out.println(op.getNomProprio());
```

```
44              }
45          }
46      }
47  }
```

Q 78.7.3 (2 points) Dans la classe `TestEntreprise`, ajouter une méthode `main` qui crée l'entreprise `MaSuperEntreprise` qui a de la place pour 10 ordinateurs et qui au départ possède 5 ordinateurs. Ajouter l'ordinateur personnel de Thomas à l'entreprise. Afficher ensuite la fréquence totale disponible dans l'entreprise, puis le nom des personnes qui ont un ordinateur personnel.

```
1 public class TestOrdinateur {
2     public static void main(String []args) {
3         Entreprise e=new Entreprise("MaSuperEntreprise",10,5);
4         try {
5             e.ajouterOrdinateur(new OrdinateurPersonnel("Thomas"));
6         } catch (DepassementNbOrdiMaxException d) {
7             System.out.println(d+" : l'ordinateur de thomas ne peut pas etre
                  ajoute");
8         }
9         System.out.println("Frequence totale : "+e.getFrequence()+" GHz");
10        e.afficherNoms();
11    }
12 }
```

Q 78.8 (3 points) L'entreprise souhaite ajouter un supercalculateur qui possède la même configuration (mêmes fréquences des processeurs) qu'un supercalculateur qu'elle possède déjà. Que doit-on ajouter dans les classes ? Donnez les méthodes ajoutées.

On va utiliser des constructeurs par copie.
Dans la classe `Processeur`, on rajoute le constructeur par copie :

```
1 public Processeur(Processeur p) {
2     this(p.frequence); // frequence identique, mais identifiant different
3 }
```

Dans la classe `SuperCalculateur`, on rajoute le constructeur par copie :

```
1 public SuperCalculateur(SuperCalculateur sc) {
2     super(AdresseIP.getAdresseIP()); // Pas la meme IP
3     tabPro=new Processeur[sc.tabPro.length];
4     for(int i=0;i<tabPro.length;i++)
5         tabPro[i]=new Processeur(sc.tabPro[i]);
6 }
```