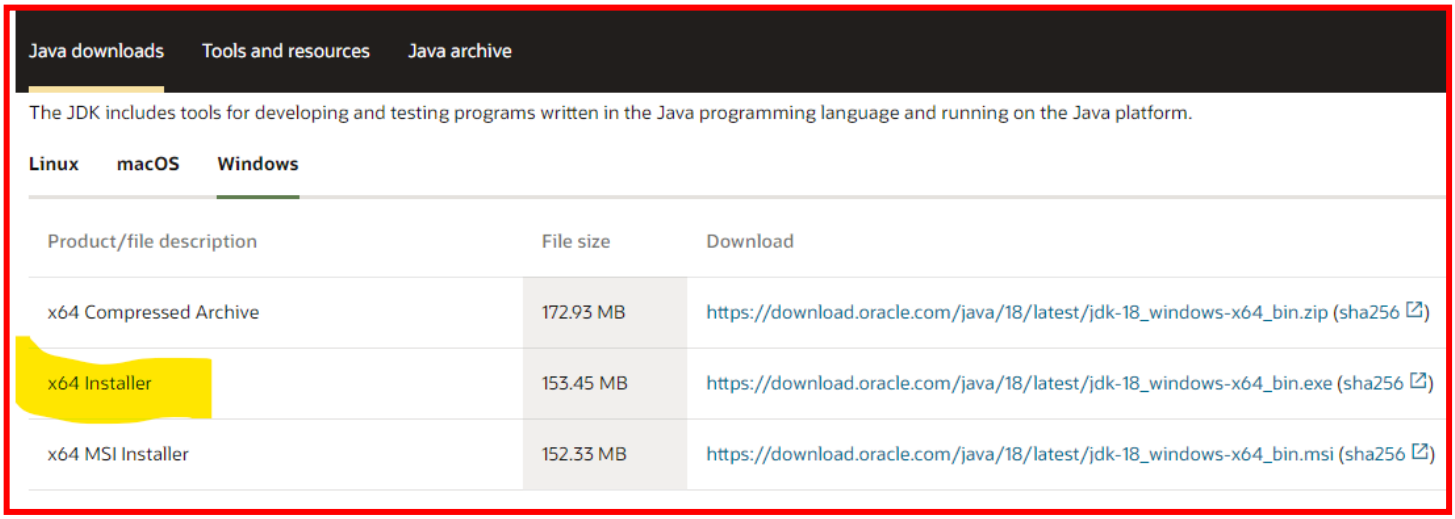
JAVA

Table des matières

Java

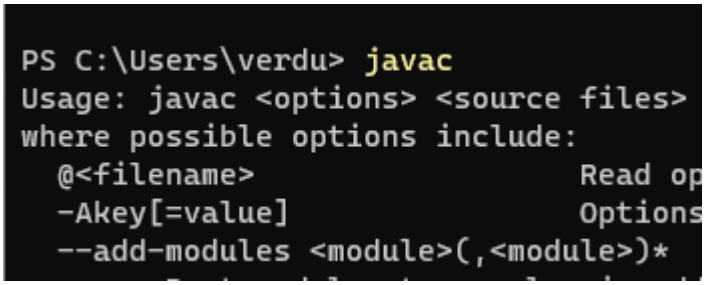
Installation du JDK

Téléchargement à partir du site d'Oracle



Sur l'ordi après installation (C:\Programmes \java)

Sous **C:\Program Files\Java\jdk-18.0.2.1** il y a un répertoire **bin** dans lequel sont les commandes java.



On se rend sous bin et on clique dans la barre d'adresse pour copier cette adresse (clic droit, copier).
Les commandes de bin sont accessibles par Clic droit sur démarrer **Terminal Windows (administrateur)**.
Mais avant il faut modifier les variables système de Windows pour qu'il sache où chercher ces commandes.
Clic droit sur démarrer **Rechercher → variables système**.
On choisit l'option variable système et nouvelle
Nom de la variable : **java** Contenu (coller, ajouter;%path%) **C:\Program Files\Java\jdk-18.0.2.1\bin ; %Path%**
A partir de là si on tape **javac** dans le terminal windows il reconnait la commande.

Où mettre nos programmes ?

On voit que terminal W a pour répertoire racine **c:\users\nom d'utilisateur** donc le mieux est d'ajouter un dossier **java** à ce répertoire. On enregistrera notre dossier source d'extension **.java** dans ce répertoire.
Depuis c:\users\nom d'utilisateur sous terminal W il suffira de taper **cd java** pour se rendre dans ce répertoire et la commande **dir** affichera le contenu du répertoire.
On écrira le programme sous l'éditeur Notepad++ et on l'enregistrera avec l'extension .java dans le dossier java.
Notez que le fichier doit porter le même nom que la classe avec une majuscule au début.

Compilation.

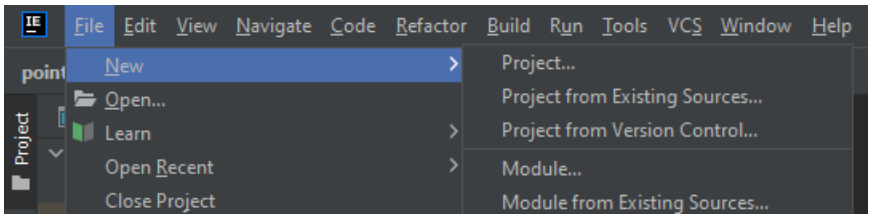
Sous terminal W il suffira de se rendre dans le répertoire java et si notre fichier s'appelle Prog.java on tapera **javac Prog.java**
Ce qui constitue un fichier **Prog.class** en bytecode.

Lancement.

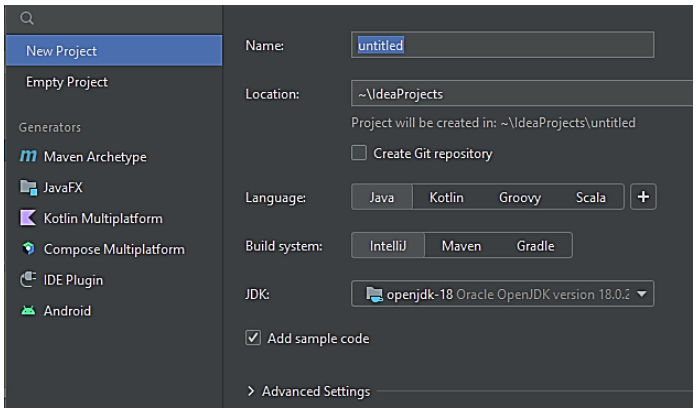
Sous terminal W il suffira de se rendre dans le répertoire java et si notre fichier s'appelle Prog.class on tapera **java Prog** ce qui provoquera l'exécution des instructions du programme avec une sortie écran si on veut voir quelque chose.

Intellij IDEA (Utilisation minimaliste)

1. Créer le projet



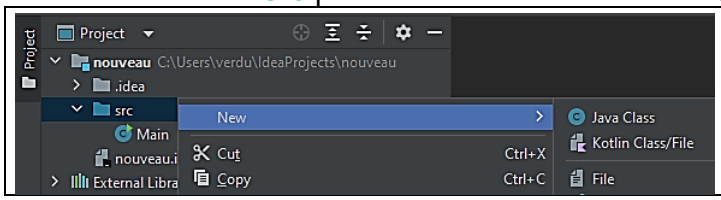
Pour créer un projet dans le menu d'IDEA
File
New
Project

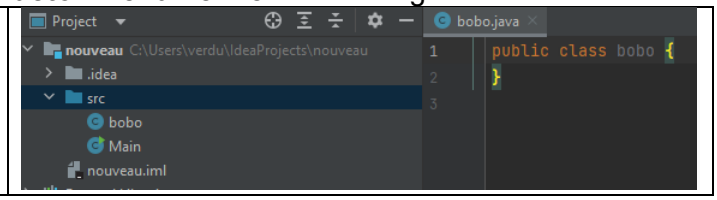


On donne le nom du projet (nouveau), l'emplacement, le langage et le JDK utilisés.
Puis on clique sur CREATE.

2. Créer une classe objet

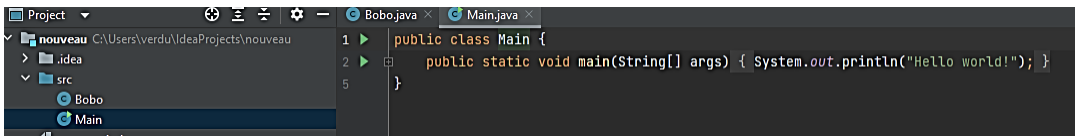
Fenêtre de gauche d'IDEA, sous **nouveau** (répertoire qui contient le projet nouveau avec une esquisse de Main.java toute prête). Pour créer un objet (une classe) on clique droit sur **src** (répertoire qui contient les sources) ; puis sur **New** et **Java Class**, on donne son nom (bobo) et dans l'éditeur l'esquisse de la classe apparaît (à droite **public class bobo { }**). Si on veut renommer le fichier **bobo.java** en **Bobo.java** on clique droit sur bobo sous **src** puis dans le menu contextuel **refactor / rename file** et on change bobo en Bobo.





3. Créer le programme main ()

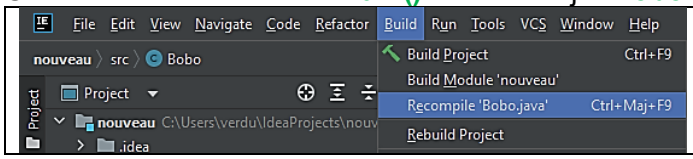
Pour créer un Main on double clique sur **main** sous **src** et l'esquisse Main apparait dans l'éditeur (ci-dessous à droite). On modifie ce qui doit l'être. On renommera **Main.java** comme on l'a fait pour **bobo.java**.

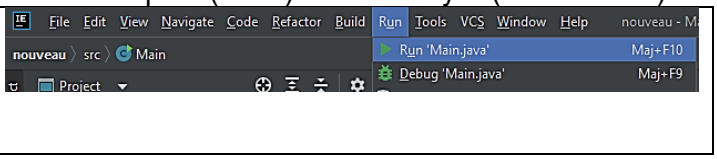


Au-dessus de l'éditeur un onglet **main** et un onglet **Bobo** permettent de passer de l'un à l'autre.

4. Compiler et essayer

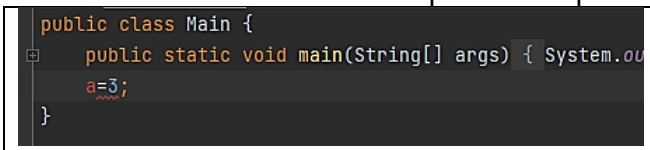
Une fois constitués notre **Main()** et notre objet **Bobo** il faut les compiler (Build) et les essayer (run du Main)

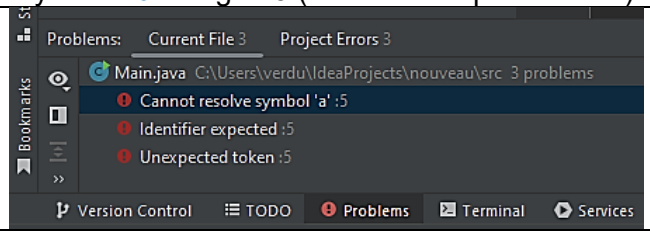




5. Traitement d'erreurs

Si on fait des fautes en programmant, en bas de l'écran un point rouge apparait devant Problems. On clique dessus et le correcteur nous dit qu'il ne comprend pas le symbole **a** en ligne 5 (affecté mais pas déclaré)





Les types de variables

- **boolean** : un booléen qui ne pourra prendre que les valeurs **true** ou **false**.
- **byte** : un entier relatif très court (entre −128 et 127).
- **short** : un entier relatif court (entre −32 768 et 32 767).
- **int** : un entier relatif (entre −2 147 483 648 et 2 147 483 647).
- **long** : un entier relatif long (entre −9 223 372 036 854 776 000 et 9 223 372 036 854 776 000).
- **float** : un nombre décimal (très grand).
- **double** : un nombre décimal à double précision (encore plus grand).
- **char** : un caractère (entre '\u0000' et '\uffff' numéro du caractère unicode en hexadécimal).
- **String** : une chaine de caractère. NB : ce n'est pas un type primitif mais une classe. S majuscule

En **rouge** les types les plus utiles. Une variable commence par une minuscule. Une classe par une majuscule.
String s = new String("abc); permet à s d'accéder aux méthodes de la classe String dont String() est le constructeur.

Déclaration / affectation **Toute instruction se termine par un ;**

Public static int n ; **n = 4 ;**	**byte temp;** **temp = 64;**	**String s= "bonjour " ;** **byte temp = 64 ;**	**float n =2.01f ; double n = 2.01 ;** **int nbre1 = 2, nbre2 = 3, nbre3 = 0;**

Constante : **final float PI = 3.14116f** **(PI en majuscules)**

Les opérateurs arithmétiques

Opérateur	Opération
+	addition
-	soustraction
*	multiplication
/	division
%	modulo

Pour écrire x =x+1 on écrit x+=1 (idem pour toutes)

Les méthodes de la classe MATH

Quelques exemples (il y en a beaucoup suivre ce lien)
X = Math.random(); // X aléatoire ∈ [0,1[
X = Math.sin(120); //La fonction sinus
X = Math.cos(120); //La fonction cosinus
X = Math.tan(120); //La fonction tangente
X = Math.abs(-120.25); //La fonction valeur absolue
X= Math.floor(a) // plus grand entier < a
X = Math.max(a,b) // le plus grand
X = Math.round(a) // entier le plus proche
X = Math.min(a,b) // le plus petit
X = Math.pow(d, 2); //La fonction puissance d^2.
X = Math.sqrt(a); // racine de a
Dans tous ces exemples X doit être déclaré en double

Chaines

+ concaténation
String a = "je suis " **+** "idiot " Ou si a et b sont des variables de type string : string c = a + b ;
double x = 3/2; Quand on écrit à l'écran : Système.out.println("le résultat est" +x); (chaine + nombre)

Test d'égalité stricte si a et b sont des variables de type string
La condition **a.equals(b)** donne true ou false

Conversion de type, par exemple transformer un entier n en double **f : f =(double)n;**
int nombre = Integer.parseInt(chaine) opération inverse **String s=String.valueOf(n);**

Les fonctions de traitement des chaines (les méthodes de la classe String)

java.lang.String		
Méthodes	**Retour**	**Explication**
charAt(int index)	char	Récupère un caractère particulier dans la chaîne dont la position est précisée en argument.
codePointAt(int index)	int	Renvoie le point de code qui démarre ou se termine à l'emplacement spécifié.
codePointCount(int début, int fin)	int	Renvoie le nombre de points de code entre début et fin-1. Les substitutions sans paires sont considérées comme des points de code.
compareTo(String autreChaîne) compareTolgnoreCase(String autreChaîne)	int	Compare la chaîne avec une autre chaîne passée en argument. Possibilité de ne pas tenir compte de la casse des caractères avec la deuxième méthode.
concat(String autreChaîne)	String	Concatène la chaine avec une autre.
endWith(String suffixe)	boolean	Vérifie si la chaîne de caractère se termine par le suffixe précisé en argument.
equals(String autreChaîne) equalsIgnoreCase(String autreChaîne)	Object	Compare la chaîne à une autre en ne tenant pas compte de la casse des caractères éventuellement.
format(String format, Object ... args)	String	Méthode statique qui permet de formater un texte, en relations avec les arguments proposés, dans un motif défini à la manière de la fonction printf() du C++.
indexOf(String sousChaîne) indexOf(int car) indexOf(String sousChaîne, int décalage) indexOf(int car, int décalage)	int	Cherche la première occurence d'un caractère ou d'une partie de la chaîne.
intern()	String	Cherche et renvoie une instance unique de la chaîne dans l'ensemble des chaînes partagées.
lastIndexOf(String sousChaîne) lastIndexOf(int car) lastIndexOf(String sousChaîne, int index) lastIndexOf(int car, int index)	int	Cherche la dernière occurence d'un caractère ou d'une sous-chaîne.
length()	int	Renvoie la longueur de la chaîne.
match(String motif)	boolean	Evalue que la chaîne de caractères correspond au critère spécifié par le motif proposé exprimé sous la forme d'expression régulière.
offsetByteCodePoints(int début, int pointCode)	int	Renvoie l'indice du point de code d'où pointe pointCode, depuis le point de code jusqu'à début.
replace(char ancien, char nouveau)	String	Remplace toutes les occurences d'un caractère dans une chaîne par un autre caractère.
replaceAll(String motif, String remplacement) replaceFirst(String motif, String remplacement)	String	Remplace toutes les occurences ou la première partie de la chaîne de caractère avec la chaîne de remplacement spécifiée en argument à partir d'un motif défini par une expression régulière.
split(String motif)	String[]	Décompose la chaîne en un tableau de chaînes en utilisant comme séparateur un élément d'expression régulière.
startWith(String prefixe)	boolean	Vérifie si la chaîne débute par un prefixe.
subString(int début) subString(int début, int fin)	String	Retourne une partie de la chaîne. Attention, le caractère indicé par début est pris en compte, alors que le caractère correspondant à fin ne l'est pas ; fin représente la limite supérieure exclusive.
toLowerCase()	String	Convertit la chaîne en minuscule.
toUpperCase()	String	Convertit la chaîne en majuscule.
trim()	String	Supprime les espaces blancs de début et de fin de chaîne.
valueOf(Type valeur)	String	Renvoie une chaîne de caractères correspondant à la valeur numérique passée en paramètre de la méthode. Type = int, long, double, float, Object, char[], char, boolean.

Utilisation si x variable de type string on écrit par exemple int n = x.length(); ou = "abcdef ".length();
x= x.replace("a","b"); y=x.subsString(0,3); L'index d'un caractère dans une chaine commence à 0.

Instructions conditionnelles

Opérateurs de comparaison

opérateur	signification	domaine d'application
<	plus petit que	types numériques, caractères
>	plus grand que	types numériques, caractères
<=	plus petit que ou égal à	types numériques, caractères
>=	plus grand que ou égal à	types numériques, caractères
==	égalité de valeurs (types primitifs) égalité de références (types objets)	tous les types de données disponibles dans le langage
!=	inégalité de valeurs (types primitifs) inégalité de références (types objets)	tous les types de données disponibles dans le langage

Opérateurs logiques

opérateur	signification	exemple	vrai si
&&	et	a && b	a et b sont vrais
||	ou	a || b	soit a soit b est vrai
!	non	!a	a est faux

Instructions conditionnelles et boucles

Int x

If(x%2==0){programme si x est pair ; } else {programme si x est impair ;}

Switch(x){
Case 2 :
Programme si x = 2 ;
Break ;
Case 3 :
Programme si x=3 ;
Break ;
Default :
Programme si x différent de 2 et de 3 ;
Break ;
}

Le break fait sortir le programme de la structure switch dés qu'une condition est réalisée.

int x = 10, y = 20;

int max = (x < y) ? y : x ; condition ternaire équivaut à si x<y max vaut y sinon max vaut x (ici max vaut 20)

for(int i =1 ; i<=10 ;i++){instructions réalisées pour i varie de 1 à 10 ;}

While(int i <10){instruction a réaliser tant que i <10 ; i++} si i <10 en rentrant il faut qu'il évolue vers 11+

Do{instructions ;}while(i <4) ; même chose mais si i>4 en rentrant instructions réalisées une fois

Tableaux et collections

Les tableaux

Les tableaux sont déclarés par le type suivi du nom terminé par 2 crochets [ ] = {liste des valeurs initiales} ;

```
int tableauEntier[] = {0,1,2,3,4,5,6,7,8,9};
double tableauDouble[] = {0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0};
char tableauCaractere[] = {'a','b','c','d','e','f','g'};
String tableauChaine[] = {"chaine1", "chaine2", "chaine3" , "chaine4"};
```

On peut créer un tableau vide

```
int tableauEntier[] = new int[6];
//Ou encore
int[] tableauEntier2 = new int[6];
```

Puis le remplir (ici avec des entiers) en écrivant par exemple **tableauEntier[0]=12 ; tableauEntier[1]=9 ;** …
L'index de la première case du tableau est 0. Ici l'index varie de 0 à 5.

Tableaux à plusieurs dimensions

Exemple
int pn[][] = { {0,2,4,6,8},{1,3,5,7,9} };
La première accolade contient les nombres pn[0][0] , pn[0][1],…, pn[0][4]
La seconde contient les nombres pn[1][0] , pn[1][1],…, pn[1][4]

Nombre d'éléments d'un tableau : **tableauEntier.length=6**. Premier indice : 0 Dernier indice = length –1 = 5.
On parcourt un tableau tabl
Soit par une boucle for sur les indices (i varie de 0 à tabl.length–1)
for(i=0;i<tabl.length;i++){x=tabl[i];action;}
Soit par une boucle foreach pour x dans tabl.
for (type de variable x : tabl){action;} Signification: pour tout élément x dans tabl réalise l'action.

Les collections

Les collections sont des ensembles d'objets. Il en existe plusieurs types

- **List** : collection d'éléments ordonnés qui accepte les doublons
- **Set** : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
- **Map** : collection sous la forme d'une association de paires clé/valeur
- **Queue et Deque** : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement
- **Collection** est l'interface racine dans l'hiérarchie de java framework Collections. Ce qui veut dire que chaque classe implémente Collection.
- **Collections** est une classe utilitaire membre de java framework Collections et qui a des méthodes statiques pour manipuler les objets qui implémentent l'interface Collection. Par exemple, le tri d'un ArrayList dans l'ordre croissant: Collections.sort(arraylist);

- Une liste chainée linkedList est une liste dans laquelle on entre soit par la première position, soit par la dernière et donc les éléments consécutifs gardent le lien initial
Elément avant : soit rentré avant soit rentré devant
Elément après : soit rentré après, soit rentré derrière

- Dans une Map, la clé peut être n'importe quel objet, un index numérique, une string, …
Quand on ajoute ou quand on lit un élément on manipule en même temps la clé et la valeur associée.

- Dans une queue les éléments sont stockés dans un certain ordre et le premier rentré est disponible pour un programme qui l'extrait de la queue. Alors le second entré prend la place du premier.
Premier rentré, premier sorti. (FIFO = first in, first out)

Notions initiales de programmation

Entrées = pour nous saisies au clavier (quelquefois lecture d'un port affichant des données)
Sorties = pour nous écriture sur l'écran (quelquefois écriture d'un port)
// = commentaire sur une ligne
\n "pour aller à la ligne **\n** dans la chaine"

Programme main appelé au démarrage du projet avec import initial

```
import java.util.Scanner;
public class Demo {
public static void main(String[] args){
Scanner sc = new Scanner(System.in);
System.out.println("Saisissez un entier : ");
int i = sc.nextInt();
System.out.println("Saisissez une chaîne : ");
sc.nextLine();
String str = sc.nextLine();
System.out.println("FIN ! ");
}
}
```

On importe les classes qui ne font pas partie du paquetage standard
On crée une **classe publique Demo**
Au sein de la classe **le programme main** sera celui qui sera exécuté en 1er

On crée un objet scanner nommé sc (méthodes de saisie au clavier)
Sc.nextInt() permet de saisir un entier au clavier (on le range dans i)
Sc.nextLine() permet de saisir une string au clavier (on la range dans str)
Sc.nextLine() sans variable d'accueil permet de vider le tampon texte

System.out.println("String") Imprime la String à l'écran

Une fois le programme source Demo.java écrit avec un traitement de texte, on le compile (javac Demo.java) ce qui constitue le programme exécutable Demo.class qu'on peut exécuter en tapant java Demo.

Programme main utilisant un sous- programme (fonction, méthode, ...)

Certains programmes peuvent appeler des fonctions, des sous programmes, des méthodes.
On les situe dans la classe mais à l'extérieur du programme main.

```
Public class Demo {
Public static void main (string[ ] args){
double x = 25.2356879 ;
double y =arrondi(x,2);
}
public static double arrondi(double A, int B) {
return Math.floor(A * Math.pow(10, B) )) /
Math.pow(10,B);
}
}
```

En vert la classe Demo qui contient …
En bleu le programme main

En rouge la fonction arrondi(A,B)
A est le nombre à arrondir, B est le nombre de décimales.
Math.floor(x) renvoie la partie entière d'un decimal positif
Math.pow(x,y) renvoie x^y
Donc arrondi(x,2) multiplie x par 100, prend la partie entière de ce nombre (floor) et la divise par 100 ce qui donne x arrondi à 2 décimales.

Main appelle la fonction arrondi dans double y =arrondi(x,2)

Fonctions pures

● instruction **return une valeur.** Que signifie retourner une valeur (nombre, chaine , …) ?
Si dans un programme de la classe on écrit **double y = arrondi(25.2356879,2)** ; y sera égal à 25.23.
La fonction a reçu 25,2356879 pour l'arrondir à 2 décimales et **elle a retourné (return)** 25,23 au programme appelant.
Certaines fonctions retournent un nombre, une chaine, un booléen dans ce cas on écrit ce type après public.
public static double arrondi(double A, int B) Ici la fonction **arrondi(x,y)** est **publique** (accessible par toutes les classes), elle retourne un **double** (instruction return) et on lui passe **comme arguments un double x en premier (le nombre à arrondir) et un entier y en second** (le nombre de décimales). Static : universel, commun à tous(toutes).

Méthodes ou sous-programmes

● Certains sous-programmes sont de **type void** ce qui signifie qu'ils ne renvoient aucun résultat. Ils font un travail c'est tout.

```
public class ImprimeTableau
{
public static void main(String[] args)
{
String[] tab = {"toto", "tata", "titi", "tete"};
parcourirTableau(tab);
}
static void parcourirTableau(String[] tabBis)
{
for(String str : tabBis){
System.out.println(str);
}
}
}
```

Voici un exemple ou on définit un tableau appelé tab dans le main() puis on l'envoie en argument à la méthode parcourirTableau qui est chargée d'imprimer le tableau à l'écran.
On remarque deux choses :

● **On envoie tab** à parcourirTableau mais l'argument qu'il reçoit ne s'appelle plus tab mais tabBis et il est défini comme un tableau de chaines (String[]). En fait tabBis = tab.
Mais dans le sous-programme parcourirTableau, on pourrait modifier tabBis sans que ça impacte tab.

● Remarquer la forme inhabituelle de la condition dans le for (String str : tabBis) cette écriture signifie "pour tout str dans tabBis exécute le programme entre { et }". Str va donc prendre les valeurs tabBis[0]= "toto " puis tabBis[1]= "tata " jusqu'à tabBis[3] = "tete ".

Les listes

List Contient des objets d'une classe (strings, éléments d'une énumération, …)

```
List<Couleur> COULEURS_CHAUDES = Arrays.asList(ROUGE, ORANGE, JAUNE);
```

Arrays.asList crée une liste de taille fixe à partir d'un tableau ou d'une énumération.

Chaque élément est accessible par un index.
Peut contenir des objets en double.
Modifiable par suppression, ajout, ordre
Création liste vide (4 types)

```
List myArrayList = new ArrayList();   //Type commun
List myLinkedList = new LinkedList();   //Type lié chaque élément est lié au suivant et au précédent
List myVector = new Vector(); //comme arrayList mais synchronisée (en cas de travail simultané) et plus lente
List myStack = new Stack();  //Type où l'index peut être une string (dictionnaire)
```

Acceptent tous types de données

```
List<Int> list = Arrays.asList(1, 2);
```

N'accepte que les entiers

```
List<String> list = new ArrayList<String>(){{list.add("a"); list.add("b");}};
```

Type ArrayList

```
import java.util.ArrayList; // import
ArrayList<String> cars = new ArrayList<String>(); // Création d'une liste de strings
```

Méthodes communes aux listes

Ajout	cars.add("Volvo");
Accès	cars.get(0);
Obtenir l'index	cars.indexOf("Opel");
Obtenir le dernier index	cars.lastIndexOf("Opel");
Change Item	cars.set(0,"Opel");
Supprime Item	cars.remove(0);
Supprime liste	cars.clear();
Taille liste	cars.size();
Tri ordre alpha	Collections.sort(cars);
Ajout de la collection motos à l'index x	cars.addAll(x,motos);
Créer une sous liste des index x à y	Cars.subList(x,y);

Dans tous les cas
System.out.printLn(nom de la liste) imprime ses éléments dans l'ordre des index [Vollvo, Renault, Suzuki, Mazda]
Pour vérifier qu'une liste de nom list contient un mot **list.contains("mot") = true ou false**.
Marche aussi si on remplace list par une variable de type string.

Type LinkedList

```
import java.util.LinkedList; // import
LinkedList<String> cars = new LinkedList<String>(); // Création d'une liste de strings
```

LinkedList dispose des méthodes suivantes en plus des méthodes précédents

addFirst() addLast() removeFirst() removeLast() getFirst() getLast()	Méthodes permettant D'ajouter De supprimer D'obtenir Le premier ou le dernier élément de la liste

Type HashMap (Dictionnaire)

```
import java.util.HashMap; // import the HashMap class
HashMap<String, String> capitales = new HashMap<String, String>();// liste de chaines indexées par des chaines
```

Permet de remplacer l'index numérique par un index chaine égale au nom de pays
Pour ajouter **capitales.put(**"France", "Paris")
Pour obtenir **capitales.get**("France") = "Paris"

Type Vector

A des méthodes propres mais sans grand intérêt

Les énumérations

Enumération Mois et 2 méthodes (dans mois.java)

```
public enum Mois {
    JANVIER, FEVRIER, MARS, AVRIL, MAI,
JUIN, JUILLET, AOUT, SEPTEMBRE, OCTOBRE,
NOVEMBRE, DECEMBRE;
    // Pour obtenir le mois à partir de
l'index
    public static Mois quelmois(int ind)
{return Mois.values()[ind-1];}
      // Pour obtenir l'index à partir d'un
membre de l'énumération
public static int quelindex(Mois m){
return m.ordinal()+1;}
}
```

Les index d'une énumération commencent à 0.
0 index de Janvier, 1 index de février, etc.
Soit l'énumération **Mois** (avec une majuscule)
Mois.values()[2] pointe sur MARS
On définit une variable de type Mois
Mois m;
Mois m = Mois.values()[2]; (on lui affecte la valeur MARS en disant que c'est le Mois d'index 2)
Int index = m.ordinal(); → index = 2 (on cherche l'index de m = MARS dans l'énumération)

String chaine = Mois.m.name() ; → **chaine = "MARS"** (on transforme un mois en chaine)
String chaine = Mois.MARS.name() ; → **même résultat**
m = Mois.valueOf("MARS"); → Opération inverse obtenir un membre de l'énumération à partir d'une chaine

Exemples d'utilisation des fonctions sur les énumérations (propres au langage) et méthodes propres à l'énumération Mois

public void setTmin(String nomMois, float valeur){....}

Pour affecter une valeur (float valeur) au tableau tmin[ ] à partir du nom du mois (string nomMois), il faut d'abord trouver l'index x du mois dans Mois. Et on fera tmin[x+1]=valeur. (Dans tmin l'index de janvier est 1 dans Mois c'est 0)

On transforme nomMois en majuscules	**nomMois=nomMois.toUpperCase();**
On transforme nomMois en Mois (m)	**Mois m =Mois.valueOf(nomMois);**
On cherche l'index x de m dans Mois	**int x = m.ordinal();**
On affecte la valeur passée en paramètre à tmin[x+1]	**tmin[x+1]=valeur;**
On aurait pu faire	**int x = Mois.quelindex(m); tmin[x] = valeur;**

Puisque quelindex ajoute 1 à l'ordinal. **Remarque** : la convention qui consiste à écrire quelIndex (I majuscule) au lieu de quelindex est indispensable dans les appels à java (valueOf(), toUpperCase()) mais non obligatoire quand, dans un programme, on donne un nom à une méthode ou à une variable. Pratique simplement conseillée.

Un autre exemple

```
public enum Couleur {

  ROUGE, ORANGE, JAUNE, VERT, BLEU, MAGENTA;
  private static final List<Couleur> COULEURS_CHAUDES = Arrays.asList(ROUGE, ORANGE, JAUNE); //création liste des chaudes
  public boolean isChaude() {
    return COULEURS_CHAUDES.contains(this); // true si la Couleur à laquelle on applique la méthode est dans le tableau
  }
  public boolean isFroide() {// true isChaude() renvoie false
    return !isChaude();
  }
  public Couleur getComplementaire() { //pour i ∈[0,2] la complémentaire de couleur[i] est couleur[i+2]
    Couleur[] values = Couleur.values(); // on crée un tableau values[ ] de couleurs
    int index = this.ordinal() + (values.length / 2); // index = index de la couleur + 3 (3 = moitié du nombre d'éléments de values[ ] )
    return values[index % values.length];   // index %6 est l'index de la couleur complémentaire
  }
}
// getComplementaire() plus simple on cherche x=this.ordinal();
If (x < 3) {return  Couleur.Value()[x+3] ;}else{Couleur.Value()[x-3];}
```

Utilisation:

```
Couleur couleur = Couleur.ROUGE; // on définit et on affecte une variable Couleur
System.out.println(couleur.isChaude()); // true
System.out.println(couleur.isFroide()); // false
System.out.println(couleur.getComplementaire()); // VERT
```

JAVA orienté objet

L'objet rectangle (public donc accessible partout).

```
public class Rectangle {
    double largeur, hauteur;
    public Rectangle(double initL, double initH){
        largeur=initL;
        hauteur=initH;
    }
    double perimetre() {return 2*(largeur+hauteur);}
    double surface() {return largeur*hauteur;}
    double diagonale() {
        return Math.sqrt(largeur*largeur+hauteur*hauteur)}
    void doubler() {largeur*=2; hauteur*=2;}
}
```

Rectangle Commence par une majuscule

2 variables d'instance appelées ATTRIBUTS

Une méthode spéciale appelée constructeur (même nom que la classe)

La classe rectangle

4 méthodes associées à la classe rectangle

Création d'instances (d'objets) selon le modèle rectangle

- Pour créer une instance (objet), on invoque un constructeur avec l'instruction *new.* P.e. :

  new Rectangle(5, 10); // création d'un rectangle de larg. 5, haut. 10

- Pour se référer à l'objet créé, nous avons besoin d'une référence (ou variable). La déclaration d'une référence se fait comme pour les types de base (int, double, char, boolean,…) en faisant précéder le nom de la référence par le nom de la classe:

  Rectangle r1, r2; // déclaration de deux rectangles

- L'assignation se fait au moment de la création de l'instance

  r1 = new Rectangle(5, 10); r2 = new Rectangle(2, 2);

- Il est possible de déclarer la référence et de créer l'objet en même temps:

  Rectangle r1 = new Rectangle(5, 10)

Invocations des méthodes sur une instance

- L'invocation d'une méthode se fait en désignant l'instance (par la référence) et en la faisant suivre du nom de la méthode et des paramètres effectifs données sous forme d'expression, s'il y a lieu:

  instance.méthode(expr1, expr2, …)

- Par exemple, pour obtenir la surface du rectangle r1, on écrira:

  r1.surface()

- Si la méthode possède des paramètres, les expressions passées en paramètre sont évaluées puis affectées aux paramètres correspondants de la méthode

- Si l'on veut invoquer une méthode sur l'instance courante, on écrira:

  this.méthode(expr1, expr2, …)

Exemple méthode **accesseur** (accès à un paramètre) **Public String getLargeur(){ return largeur; }**
Exemple méthode **mutateur** (modif paramètre) **Public void setHauteur(double h){hauteur=h;}**

On peut créer un objet Disque (public) selon le même modèle

```
Public class Disque {
double diametre;
static final double PI=3.14159;
Disque(double initD) {diametre=initD;}
double perimetre() {return PI*diametre;}
double surface() {return (PI*diametre*diametre)/ 4;}
double rayon() {return diametre/2;}
void doubler() {diametre*=2;} }
}
```

L'objet disque est défini par son seul diamètre (on aurait pu choisir le rayon).
Il définit une constante (final) non modifiable (static) de type double appelée PI. (il existe un Math.PI)

Le constructeur Disque() demande le diamètre.

La classe embarque 4 méthodes : perimetre(),surface(), rayon() et doubler()

Mot clé static : signifie non modifiable. PI peut être invoqué partout dans la classe et en dehors par Disque.PI ou Disque.methode() si elle est définie en Static. C'est le cas de Demo.main() par exemple. Ou d'une méthode qui permet d'écrire une chaine fixe à l'écran.
La classe ***Math*** ne contient que des attributs statiques (*PI*, *E*) et des méthodes statiques(*sqrt*, *exp*, *sin*, *cos*, *log*, …)
La classe ***BigInteger*** contient deux attributs *static* (*ONE* et *ZERO*) et une méthode statique *BigInteger valueOf(long)*
La classe ***Font*** contient plusieurs attributs *static* parmi lesquels *BOLD*, *ITALIC*, et *PLAIN* et des méthodes statiques parmi lesquelles *Font decode(String str)* qui fabrique une fonte à partie de son nom.
La classe ***Color*** contient des couleurs prédéfinies *WHITE*, *RED*, *BLACK*, *BLUE*, ...

Ensuite on peut utiliser les objets disque et rectangle dans un programme main

```
class TestRectDisk1{ public static void main (String args[]) {
Rectangle r1=new Rectangle(2,4);
Rectangle r2=new Rectangle(3,4);
System.out.println("diagonale de r1: "+r1.diagonale()); // pour appliquer la méthode à l'objet : objet.methode()
System.out.println("périmètre de r2: "+r2.perimetre());
r2.doubler();
System.out.println("périmètre de r2: "+r2.perimetre());
Disque d1,d2;
d1=new Disque(2); d2=new Disque(4);
System.out.println("rayon de d1: "+d1.rayon());
System.out.println("périmètre de d2: "+d2.perimetre());
d2.doubler();
System.out.println("périmètre de d2: "+d2.perimetre());
}}
```

Pour résumer :
Une fois le programme public class Objet{ } créé avec ses méthodes et son constructeur Objet().
Un autre programme peut **instancier** un objet de cette classe en écrivant Objet ob = new Objet();
Majuscule à Objet qui désigne une classe. Minuscule à ob qui désigne une variable.
Objet() est le constructeur de la classe. Pour le moment ob pointe sur null.
Affectation: Si on écrit ob ="chaine si le constructeur attend une String" ob pointe sur cette chaine.
On peut synthétiser les deux opérations en une seule en écrivant par exemple
Objet ob = new Objet("abc"); la variable ob désigne désormais l'objet paramétré par "abc".
Exemple String s = new String("abc");
Attention si on écrit
String s1="abc";
String s2="abc";
String s3 = new String("abc");
Un test (s1==s2) donnera "true" mais un test (s1==s3) donnera "false" parce qu'on utilisera une autre instance de la classe string initiale.

D'autres exemples rencontrés :
Scanner sc = new Scanner(System.in); // On crée un objet scanner sc de saisie au clavier (system.in).
int i = **sc.nextInt();** //la méthode, nextInt() attend la frappe clavier d'un entier qu'on rangera dans i
sc.nextLine(); //On vide le buffer ligne avant d'en lire une autre
String str = **sc.nextLine();** // la méthode, nextLine() attend la frappe clavier d'une chaine qu'on rangera dans str

Héritage de classe, visibilité

Supposons qu'on ait créé la classe **Personne** avec les attributs suivants

```
Private long numeroSS;
private String prénom;
private String nom;
private String email;
```

Maintenant on veut créer la classe **Travailleur**.
Un travailleur est une personne avec tous les attributs de la classe Personne auxquels on a ajouté un attribut nom de l'entreprise (nomEntreprise).
On va dire que la classe Travailleur dérive (hérite) de la classe personne avec une extension de ses attributs.
Un mot clé permet de créer l'héritage c'est extends.
On écrit

```
package fr.koor.poo;

public class Travailleur extends Personne {
   private String nomEntreprise;
  public Travailleur (long numeroSS, string prenom, string nom, string email, string nomEntreprise){
   super(numeroSS, prenom, nom, email);
  This.nomEntreprise = nomEntreprise;
}
```

On commence par écrire l'argument additionnel nomEntreprise.

Le constructeur travailleur() Fait appel à **super()** le constructeur de la classe mère.
Super.methode() ferait appel à une méthode de la classe mère.

La classe héritière a tous les attributs et méthodes de la classe parente plus ceux qui lui sont propres.
This. Signifie qui appartient à la classe en cours **Super**. Signifie qui appartient à la classe mère

Quelques explications sur le code précédent :
Package fr.koor.poo Le package est un ensemble de classes ayant un lien particulier entre elles.
Par exemple elles contribuent à former une application dans un domaine particulier. (Exemple java.Math)
Ou il s'agit de classes et de méthodes utiles dans un domaine particulier de la programmation.
Le package **java.lang** est implicitement appelé, il contient les instructions de base du langage.
L'instruction **package nomdepackage**; assigne les classes qui suivent à ce package.
L'instruction **import nomdepackage** permet d'appeler les classes externes qu'on compte utiliser dans notre programme et leurs méthodes.
On peut importer
Toutes les classe du package **import nomdepackage.***
Seulement une classe du package **import nomdepackage.classe**
Ou même une méthode **import nomdepackage.classe.methode**

Le rôle des mots clés de visibilité

TABLE 3.1 – Portée des autorisations

Élément	Autorisations
Variable	Lecture et écriture
Méthode	Appel de la méthode
Classe	Instanciation d'objets de cette classe et accès aux variables et méthodes de classe

TABLE 3.2 – Autorisations d'accès

	public	protected	défaut	private
Dans la même classe	Oui	Oui	Oui	Oui
Dans une classe du même package	Oui	Oui	Oui	Non
Dans une sous-classe d'un autre package	Oui	Oui	Non	Non
Dans une classe quelconque d'un autre package	Oui	Non	Non	Non

Dans une classe une variable Private n'est accessible que depuis les méthodes de la classe (ex attributs d'un objet).

Traitement des exceptions

```
public class Main {
  public static void main(String[ ] args) {
    try {  // Try définit un bloc de code à tester car pouvant produire des erreurs
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);//instruction pouvant produire une erreur
    } catch (Exception e) { // Catch succédant à try définit les instructions à tester en cas d'erreur
      System.out.println("Aucun nombre à cet indice dans le tableau.");//traitement de l'erreur
    }  finally {
      System.out.println("Toutes les erreurs passées en revue");//code exécuté quel que soit le résultat
     }

  }
}
```

throw est utilisée avec un **type d'exception** .
Il existe de nombreux types d'exceptions en Java :
`ArithmeticException`, `FileNotFoundException`, `ArrayIndexOutOfBoundsException`, `SecurityException`, etc . Exemple d'utilisation de throw :

```
if (age < 18) {
     throw new ArithmeticException("Vas te faire cuire un œuf tu n'as pas encore 18 ans");
   }
   else {
     System.out.println("si tu n'as pas menti sur ton âge le monde merveilleux du porno s'ouvre à toi");
   }
```

Vocabulaire du programmeur

Autocomplétion :
Aptitude de l'éditeur à compléter les commandes en fonction du début de l'écriture.

Refactoring :
Permet de gérer l'impact d'un changement de nom ou du déplacement d'une méthode.

Repository CVS
Le projet est sur un serveur CVS dans un répertoire et fait l'objet d'un travail d'équipe. Ce dispositif gère les changements de version, les révisions d'un fichier et résout les conflits.

Conseils aux débutants : programmation d'une classe publique

Par exemple **public class Chose{… }**

● Rendre ses attributs Publics par exemple **public double x,y ;** permet, après création d'un objet **obj** de la classe par la méthode **Chose obj = new Chose(x,y) ;** d'atteindre ces attributs par **obj.x** et **obj.y**.

● Sauf besoin implicite on créera les constructeurs sans indication de visibilité ce qui revient à les rendre accessibles dans la classe et dans le package.
Chose(double xx, double yy){x=xx ; y=yy ;}

● Ne pas oublier le **;** à la fin de chaque instruction.

Le traitement des fichiers (texte ou data)

● Les imports

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
```

associé à `throws IOException` dans l'appel du programme de lecture/écriture

● Création d'un tampon de lecture dont la source est fichier.txt (programme permettant de découper le fichier en lignes)

```
BufferedReader fichier  = new BufferedReader(new FileReader("fichier.txt"));
```

● Lecture ligne par ligne

```
String ligne ;
    BufferedReader fichier = new BufferedReader(new FileReader("fic.txt"));
    while ((ligne = fichier.readLine()) != null) { System.out.println(ligne) }
    fichier.close();
```

On peut mettre ces lignes dans un
Try {
…..
} catch (Exception e) {
    e.printStackTrace() }

Ici on imprime les lignes à l'écran , on pourrait les mémoriser dans un tableau (t[i] =ligne;)en incrémentant un entier i de 0 à n ou les examiner une à une à la recherche d'un mot par exemple.

Lire tout le fichier avec la classe Files `List<String> Files.readAllLines("path du fichier")` **(readAllBytes() existe aussi)**

● Création d'un tampon d'écriture dont la cible est fichier.txt (programme permettant d'écrire le fichier chaine après chaine)

```
BufferedWriter fichier = new BufferedWriter( new FileWriter("fichier.txt"));
```

● Ecriture (en mode ajout)

```
Int nombre = 25;
String ligne = ", jour de mon "+nombre+"eme anniversaire";
    BufferedWriter fichier = new BufferedWriter(new FileWriter(fichier.txt));
fichier.write("bonjour tout le monde");
    fichier.newLine();
    fichier.write("Nous sommes le "+ new Date());
    fichier.write(ligne);
        fichier.close();
```

newLine() insère un RCLF

On peut mettre ces lignes dans un
Try {
…..
} catch (Exception e) {
    e.printStackTrace() }

```
fichier.println("bonjour tout le monde");
```

println ajoute un RCLF (saut de ligne) à la fin de la chaine. (RC=retour charriot, LF = line feed)

Avec la classe Files **Files.write(path, tableau de lignes, Charset.forName(UTF-8));** écrit et écrase
Files.write(path, tableau de lignes, StandardOpenOption.APPEND); écrit et ajoute

Lire ou écrire un fichier découpé en octets (en Bytes)

Les classes **FileInputStream** et **FileOutputStream** permettent de lire/ d'écrire les fichiers par groupes d'octets.

```
FileInputStream fichier =new FileInputStream("monfichier.dat");
int octet = 0;
while (octet !=  -1 ) {
   octet = fichier.read(); // ou int = fichier.read(byte [ ], 0, 64)
}
```

```
fos = new FileOutputStream("cible.bin");
fos.write(octet)        //un octet est un int
fos.write(byte[ ])  attend en paramètre un tableau d'octets
fos.write(byte[ ],0,128)  écrit les octets no 0 à 127 du tableau
```

La classe File

Utilisation fichier = new File(fichier.txt); puis par exemple if(fichier.exists()) { } ou string nom =fichier.getName()

Dans Filereader, FileWriter, FileInputStream, FileOutputStream on peut remplacer la string qui désigne le fichier par un objet File.

Méthode	Rôle
boolean canRead()	Indiquer si le fichier peut être lu
boolean canWrite()	Indiquer si le fichier peut être modifié
boolean createNewFile()	Créer un nouveau fichier vide
boolean delete()	Détruire le fichier ou le répertoire. Le booléen indique le succès de l'opération
deleteOnExit()	Demander la suppression du fichier à l'arrêt de la JVM
boolean exists()	Indique si le fichier existe physiquement
String getAbsolutePath()	Renvoyer le chemin absolu du fichier
String getName()	Renvoyer le nom du fichier
String getPath()	Renvoyer le chemin du fichier
boolean isAbsolute()	Indiquer si le chemin est absolu
boolean isDirectory()	Indiquer si le fichier est un répertoire
boolean isFile()	Indiquer si l'objet représente un fichier
long length()	Renvoyer la longueur du fichier
String[] list()	Renvoyer la liste des fichiers et répertoires contenus dans le répertoire
boolean mkdir()	Créer le répertoire
boolean mkdirs()	Créer le répertoire avec création des répertoires manquants dans l'arborescence du chemin
boolean renameTo()	Renommer le fichier

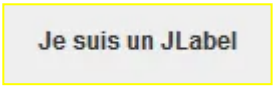
Interfaces graphiques Swing Awt

Création d'une frame sous swing

```
import javax.swing.*;
public class fenetre
{
public static void main(String[] args)
{
JFrame frame = new JFrame("Hello World");
JLabel label = new JLabel("Je suis un JLabel",
JLabel.CENTER);
frame.add(label);
JPanel panel = new JPanel();
frame.add(panel);
JButton btn1 = new JButton("Bouton 1");
Panel.add(btn1);
ButtonGroup group = new ButtonGroup();
JRadioButton radio1 = new JRadioButton("ON", true);
JRadioButton radio2 = new JRadioButton("OFF", false);
group.add(radio1);
group.add(radio2);
panel.add(radio1);
panel.add(radio2);
String[] langs = {"PHP", "Java", "Python", "C++", "Ruby"};
JComboBox cb = new JComboBox(langs);
panel.add(cb);
frame.pack()
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(250, 250);
frame.setVisible(true);
}
}
```

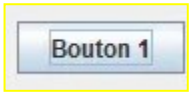
On importe swing

On crée frame une instance de JFrame, une fenêtre de titre "Hello World".



On définit un label (étiquette) et on l'ajoute au centre de la JFrame.

On ajoute un JPanel (conteneur des composants) à la JFrame.



On ajoute un bouton avec le texte Bouton 1 au panel.

On crée un groupe de radioboutons
On crée les 2 boutons du groupe

Radio1 est coché (true), pas radio2.

On ajoute les boutons au groupe.
Puis on ajoute les boutons au panel

On crée le tableau lang des choix du combo.
On crée le combo cb avec pour contenu lang.

On ajoute cb au panel

Pack optimise la situation des composants dans frame
Quand on la ferme on sort du programme.
On donne ses dimensions en pixels (on peut aussi positionner le coin haut gauche)
On la déclare visible.

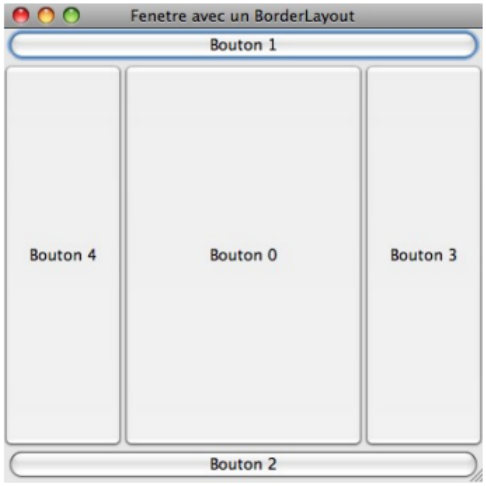
En pratique, on instancie rarement directement JFrame. On crée des instances de JFrame dans lesquelles on peut disposer les composants selon des schémas de disposition appelés "layouts" et "panels"
Et ce sont ses instances qu'on appelle selon nos besoins personnels.

On utilise plutôt SWING pour dessiner une frame.
AWT est plutôt utilisé pour gérer le comportement des composants (programmes déclenchés par le clic d'un bouton, le survol d'un composant, l'écriture d'une zone de texte, le choix dans un combo, le clic d'un radiobutton (choix d'une option) ou l'examen de cases à cocher,…)

Schémas de disposition des composants dans la frame

FlowLayout
Disposition dans l'ordre du programme

```
boutons = new JButton[5];
for (int i = 0; i < boutons.length; i ++)
 {boutons[i]= new JButton("Bouton "+i); =
setLayout(new FlowLayout());
for (int i = 0; i < boutons.length; i ++) {add(boutons[i]);
}
```

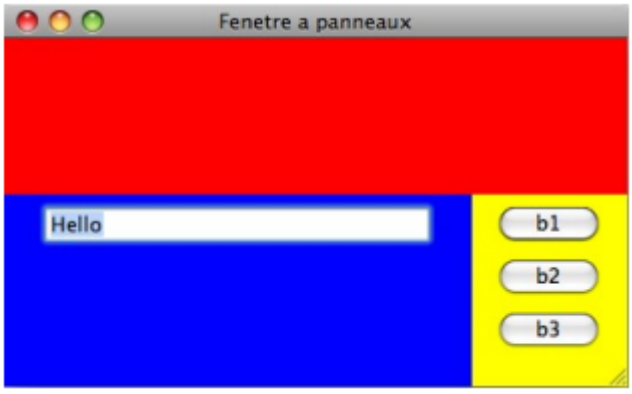


BorderLayout
Disposition en 5 zones (North, west, center, east, south)

```
boutons = new JButton[5];
for (int i = 0; i < boutons.length; i ++) boutons[i]=new
JButton("Bouton "+i);
add(boutons[0]); // au centre
add(boutons[1],"North");// ou BorderLayout.NORTH
add(boutons[2],"South");// ou BorderLayout.SOUTH
add(boutons[3],"East"); // ou BorderLayout.EAST
add(boutons[4],"West"); // ou BorderLayout.WEST }
```

GridLayout
Disposition en grille

```
setLayout(new GridLayout(3,4));
                    // 3 lignes, 4 colonnes
```



Panneaux JPanel

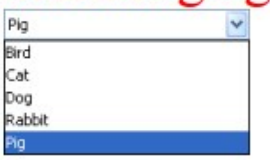
```
public class FenPanneaux extends JFrame {
protected JPanel pRouge, pBleu, pJaune;
protected JButton b1,b2,b3;
protected JTextField texte;
public FenPanneaux () {
setTitle("Fenêtre à panneaux");
pRouge = new JPanel();
pRouge.setBackground(Color.red);
pBleu = new JPanel();
pBleu.setBackground(Color.blue);
pJaune = new JPanel();
pJaune.setBackground(Color.yellow);
add(pRouge, "North");
add(pBleu, "Center"); // ou : add(pBleu);
add(pJaune, "East");
pRouge.setPreferredSize(new Dimension(400,100));
pJaune.setPreferredSize(new Dimension(100,200));
b1 = new JButton("b1"); pJaune.add(b1);
b2 = new JButton("b2"); pJaune.add(b2);
b3 = new JButton("b3"); pJaune.add(b3);
texte = new JTextField("Hello",20);
pBleu.add(texte);
pack(); }
public static void main(String args[]) {
FenPanneaux f = new FenPanneaux();
f.setVisible(true); } }
```

Composants, méthodes, propriétés

•Composants à usage général

JButton

JCheckBox

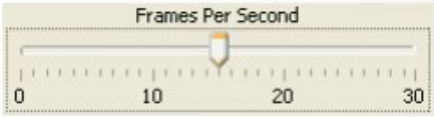


JComboBox

JList

JMenu

JRadioButton



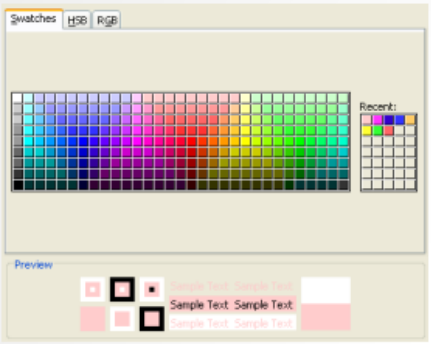
JSlider

JSpinner

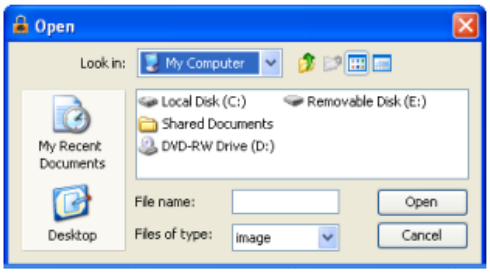
JTextField

JPasswordField

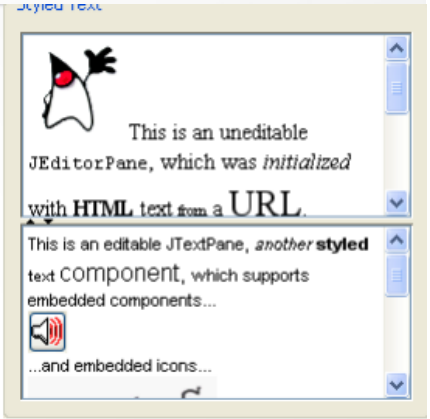
•Composants interactifs



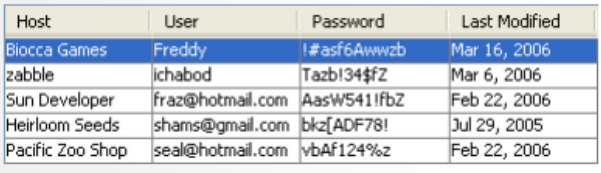
JColorChooser



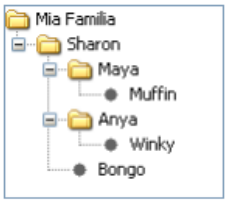
JFileChooser



JEditorPane ou JTextPane



JTable



JTree

•Composants non éditables

JProgressBar

JToolTip

JSeparator

JLabel

Evènements en lien avec les composants

Classes auditrices

ActionListener : clic de souris ou enfoncement de la touche Enter
ItemListener : utilisation d'une liste ou d'une case à cocher
MouseMotionListener : événement de souris
WindowListener : événement de fenêtre

Avec un clic souris sur un bouton → Action Listener

```
import java.awt.event.*;
Import javax.swing.*;
public class MonAppli implements ActionListener, MouseListener{
… création de Frame, panel p, …..
JButton b = new JButton("boutton");    p.add(b);
b.ActionListener(this);
…..
public void actionPerformed(ActionEvent evt) {
Object source = evt.getSource();
 if (source == b){programme à réaliser;}
}
}
```

On importe awt.event.* et swing.*
La classe indique les Listeners auxquels elle fait appel.

Le bouton indique à quel listener il va faire appel et dans quelle classe (ici This la classe en cours).

Quand il est sollicité, ActionListener fait appel à actionPerformed avec la description de l'évènement appelant evt. Evt.getSource() permet d'identifier le bouton appelant et à déclencher le programme correspondant.

Avec les cases à cocher ou une liste (ItemListener)

Dans le groupe de checkboxs on doit déclencher une action si l'une d'elle (cb1) change d'état.

```
JCheckBox cb1 = new JCheckBox("coupable");
cb.addItemListener(this);   //déclenche un évènement si la case change d'état
                            //On appelle la méthode itemStateChanged(ItemEvent e)
public void itemStateChanged(ItemEvent e) {
    e.getSource identifie la checkbox appelante cb1
    int statuscb = e.getStateChange(); permet de lire l'état de cb1 (valeurs 0 ou 1)
    statuscb == ItemEvent.SELECTED ou ItemEvent.DESELECTED
```

Mais à tout moment on peut lire l'état de cb1 avec cb1.isSelected() qui est true ou false.

Avec un combo

```
JComboBox comb = new JComboBox(choix);
panel.add(comb);
cb.addItemListener(this);   //déclenche un évènement si on fait un choix dans la liste déroulante
                            //On appelle la méthode itemStateChanged(ItemEvent e)
public void itemStateChanged(ItemEvent e) {
        Object obj = e.getItem();  // On capture l'item sélectionné
         String selection = (String)obj; // On le traduit en chaine sélectionnée
```

Avec une zone de texte TextField (appel TextListener sur touche entrée)

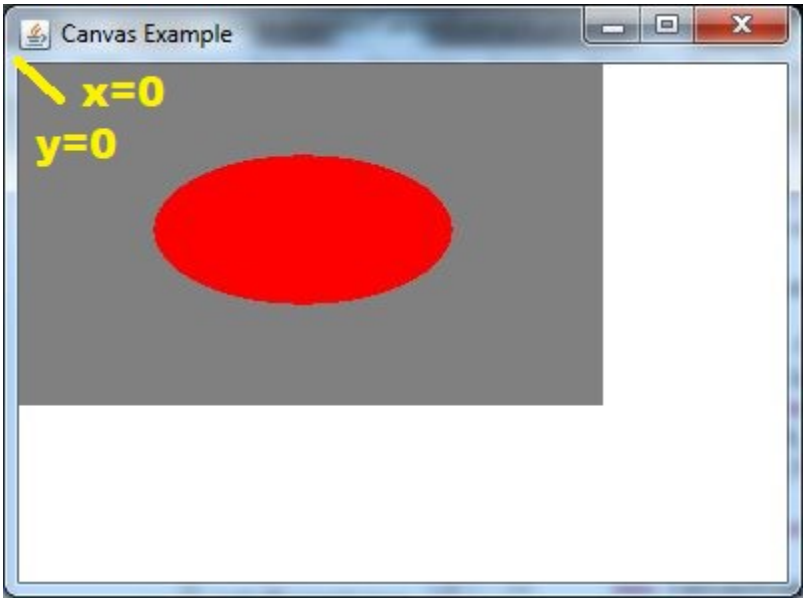
```
On appelle textValueChanged()
        public void textValueChanged(TextEvent txt) {
source est le TextField appelant
        Object source = txt.getSource();
Pour avoir le texte contenu dans source
        ((TextField)source).getText();
```

Mouvements &clics souris (MouseMotionListener | Mouse Listener)

```
button = new JButton("Survoler ce bouton");
button.setOpaque(true);
add(button);
button.addMouseListener(new MouseAdapter() {  // (appuyer, relâcher, cliquer, entrer et sortir sur un composant)
public void mouseEntered(MouseEvent evt) {button.setBackground(Color.ORANGE);}//au survol bouton orange
public void mouseExited(MouseEvent evt) {button.setBackground(null);}//quitte le survol bouton non coloré
```

Dessiner sur un CANVAS

La classe Canvas donne accès à un composant particulier qui crée dans l'écran une zone de dessin (Canvas) dans laquelle on dessine grâce à des instructions contenues dans **public void** paint(Graphics g){…}




```
import java.awt.*;

public class CanvasExample
{
public CanvasExample() //constructeur de la frame
 {
   Frame f = new Frame("Canvas Example");
   f.add(new MyCanvas());
   f.setLayout(null);
   f.setSize(400, 400);
   f.setVisible(true);
 }
  public static void main(String args[])
  { //ajout du canvas contenant le dessin (ovale rouge)
    new CanvasExample();
  }
}
```

```
// Mycanvas est une extension de la classe Canvas à laquelle on ajoute une taille, une couleur de fond et un dessin
class MyCanvas extends Canvas
{
      public MyCanvas() {
      setBackground (Color.GRAY);
      setSize(300, 200);
    }
  public void paint(Graphics g)
  {
    g.setColor(Color.red);
    g.fillOval(75, 75, 150, 75);
  }
}
```

Les principales méthodes de paint(Graphics g)

```
g.setColor(Color.black);
```

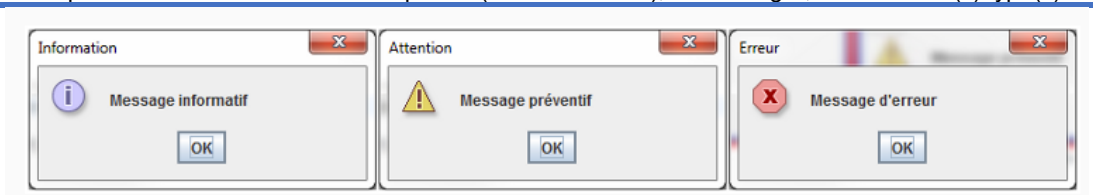
drawLine(x1,y1,x2,y2)
drawRect(x,y,width,height)
drawOval(x,y,width,height)
`int[] x={10,60,100,80,150,10};`
`int[] y={15,15,25,35,45,15};`
`g.drawPolygon(x,y,x.length);`
`drawArc(x,y,l,h,θd,θf)`
	fillRect(x,y,width,height)
	fillOval(x,y,width,height)
	`int[] x={10,60,100,80,150,10};`
	`int[] y={15,15,25,35,45,15};`
	`g.fillPolygon(x,y,x.length);`
		`Font fonte = new Font(" TimesRoman ",Font.BOLD,30);`
		`g.setFont(fonte);`
		`g.drawString("bonjour",50,50);`
			On peut également dessiner avec la souris
			MouseEvent e
			e.getx()
			e.gety()
			coord souris

Les images

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class AddImage {
public static void main(String[] args) {
JFrame frame = new JFrame("JLabel Test");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(530,600);
 frame.setLocationRelativeTo(null);
 frame.setVisible(true);
String imgUrl="icon.png";
 ImageIcon icone = new ImageIcon(imgUrl);
JLabel jlabel = new JLabel(icone, JLabel.CENTER);
frame.getContentPane().add(jlabel);
 frame.validate();
```

Les boîtes à message JOptionPane

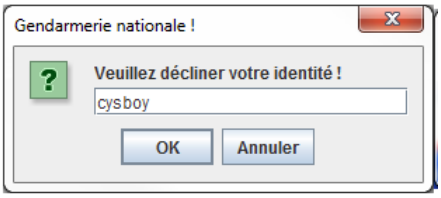
Les arguments de JOptionPane sont dans l'ordre : le parent (frame ou NULL),"le message", "le titre" et le(s) type(s) JOptionPane.type

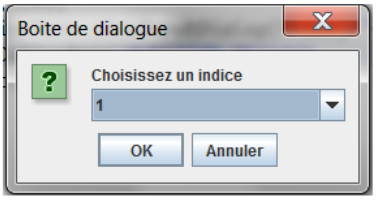



```
JOptionPane jop1, jop2, jop3;
//Boîte du message d'information
jop1 = new JOptionPane();
jop1.showMessageDialog(null, "Message informatif", "Information", JOptionPane.INFORMATION_MESSAGE);
//Boîte du message préventif
jop2 = new JOptionPane();
jop2.showMessageDialog(null, "Message préventif", "Attention", JOptionPane.WARNING_MESSAGE);
//Boîte du message d'erreur
jop3 = new JOptionPane();
jop3.showMessageDialog(null, "Message d'erreur", "Erreur", JOptionPane.ERROR_MESSAGE);
```


```
JOptionPane jop = new JOptionPane();
    int option = jop.showConfirmDialog(null,
      "Voulez-vous lancer l'animation ?",
      "Lancement de l'animation",
      JOptionPane.YES_NO_OPTION,
      JOptionPane.QUESTION_MESSAGE);
```

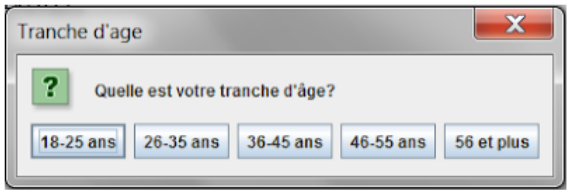
```
JOptionPane jop = new JOptionPane();
    int option = jop.showConfirmDialog(null,
      "Voulez-vous arrêter l'animation ?",
      "Arrêt de l'animation",
      JOptionPane.YES_NO_CANCEL_OPTION,
      JOptionPane.QUESTION_MESSAGE);
```






```
JOptionPane jop = new JOptionPane(), jop2 = new JOptionPane();
   String nom = // variable récupérant la réponse
jop.showInputDialog(null, "Veuillez décliner votre identité !",
"Gendarmerie nationale !",
JOptionPane.QUESTION_MESSAGE);

//affichage de la saisie
 jop2.showMessageDialog(null, "Votre nom est " + nom, "Identité",
JOptionPane.INFORMATION_MESSAGE
JOptionPane.OK_CANCEL_OPTION);
```

```
int[] indice = {0,1,2,3,4,5};
    JOptionPane jop = new JOptionPane();
    int rep = jop.showInputDialog(null,
      "Choisissez un indice",
      "Boite de dialogue",
      JOptionPane.QUESTION_MESSAGE,
      null,    // on conserve l'icône de Question_Message
      indice,   // tableau dans le combo
      indice[1]); // choix par défaut

La réponse est dans rep
```




```
String[] choix=
{"18-25 ans","26-35 ans","36-45 ans","46-55 ans","56 ans et plus"};
 int x = JOptionPane.showOptionDialog(null,
 "Quelle est votre tranche d'âge?",
 "Tranche d'age",
 JOptionPane.DEFAULT_OPTION,
 JOptionPane.QUESTION_MESSAGE,
 null, choix, choix[0]);
```

Option d'icône	Changer l'icône du message	Boutons de confirmation
JOptionPane.PLAIN_MESSAGE (pas d'icône) JOptionPane.ERROR_MESSAGE JOptionPane.INFORMATION_MESSAGE JOptionPane.WARNING_MESSAGE JOptionPane.QUESTION_MESSAGE	On a créé icon (ci-contre à gauche). Après JOptionPane.QUESTION_MESSAGE, null, → on garde l'icône normale Icon, → on remplace par notre image.	JOptionPane.DEFAULT_OPTION JOptionPane.YES_NO_OPTION JOptionPane.YES_NO_CANCEL_OPTION JOptionPane.OK_CANCEL_OPTION
`ImageIcon icon = new ImageIcon("chose.png");`		
		Réponses possibles dans int option YES_OPTION NO_OPTION CANCEL_OPTION OK_OPTION CLOSED_OPTION // fermeture message

Comparaison JAVA / HTML + JAVASCRIPT

La structure du programme HTML figure dans la colonne de gauche.
Les sections <Body> et <script> (javascript) sont citées in extenso.

<html>
<head>

Un programme Html entre <html> et </html> est divisé en sections
<head></head> qui contient CSS et Javascript
<body></body> qui contient les composants à afficher sur l'écran

```
<style type="text/css" >
.centre{top:30%;left:27%;width:20%;height:40%; color:white; background-color:black;}

.btn {définit bords arrondis, couleur rouge, gradient de couleur}
.btn:hover{couleur bleue}
.btn:active{…}
</style>
```

On donne du style aux composants grâce aux CSS
.centre contient les instructions définissant la couleur, la position, les dimensions et la couleur de texte de la div (comprendre panneau) de class="centre" dans le body.
.btn contient les instructions permettant d'enjoliver les boutons de class="btn" (au repos et survolés par la souris) situés dans le body.

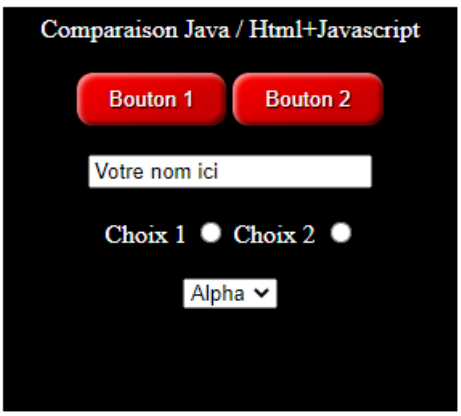
```
<script type="text/javascript">

function btu(){
alert("Vous avez cliqué sur le bouton 1");
}

function btd() {
alert("Vous avez cliqué sur le bouton 2");
}

function slct(){
        var t =document.getElementById('s1').value;
        alert("Vous avez fait le choix "+t);
}

function efface(){
        var t = document.getElementById('t1');
        t.value="";
}
</script>
</head>
```

On lie un composant du Body à un programme javascript quand un évènement se produit en écrivant dans la balise du composant une instruction de type **onclick="btu();"** qui signifie qu'un clic sur le bouton provoquera le lancement du programme (function btu()) du javascript. Celui-ci affiche un message.
Onchange="slct();" Provoque la réalisation du programme function slct() quand le choix du select (du combo) change.
Quand on affecte aux composants du Body un paramètre de type **id="s1"** il permet à javascript de retrouver quel est le composant qui a appelé le programme et de le mettre dans une variable x grâce à l'instruction
var x = document.getElementById('s1')
Et on peut lire ou modifier le composant en manipulant cette variable.
Par exemple x.value permet de lire ou de fixer la valeur du texte du composant d'id "s1".

<body>

```
<div class="centre"  align="center">
<label>Comparaison Java /Html+Javascript</label>
<br>
<input type="button" id="b1" class="btn"  onclick="btu();" value="Bouton 1">
<input type="button" id="b2" class="btn" onclick="btd();" value="Bouton 2">
<br>
<input type="text" id="t1" onclick="efface();" value="Votre nom ici">
<br>
Choix 1 <input type="radio" name="radio" id="r1" value="ch1">
Choix 2 <input type="radio" name="radio" id="r2" value="ch2">
<br>
<select name="select" id="s1" onchange="slct();">
<option>Alpha</option>
<option>Beta</option>
<option>Delta</option>
</select>
</div>
```

</body>
</html>



Résultat au lancement du html interprété par le navigateur, on retrouve sur l'écran le panneau noir (la div) défini par le CSS . centre avec des boutons enjolivés grâce au CSS .btn.
Dans le Body on retrouve la div contenant label, boutons, champ texte, radio boutons, et combo (select). align="center" centre les composants.
class="btn" affecte le style CSS .btn au bouton.
Id="s1" permet au javascript de reconnaître le composant appelant, de le lire et de le manipuler.

 Cette balise insère un saut de ligne.
En cliquant sur l'image ci-dessus on peut lancer et essayer le programme html.

Quelques exercices corrigés pour débuter.

Exercice 1

La clase chat est définie comme suit

```
public class Chat {
// 5 attributs définissent un chat
    private String nom;
    private String pelage;
    private boolean poilLong;
    private int anneeNaissance;
    private boolean pucee;
// le constructeur permet de définir un chat quand il reçoit ses 5 attributs
    public Chat(String var1, String var2, boolean var3, int var4, boolean var5) {
        this.nom = var1;
        this.pelage = var2;
        this.poilLong = var3;
        this.anneeNaissance = var4;
        this.pucee = var5;
    }
// La méthode chat.toString() permet de renvoyer une chaine avec tous les attributs
    public String toString() {
        return "Chat [nom=" + this.nom + ", pelage=" + this.pelage + ", poilLong=" +
this.poilLong + ", anneeNaissance=" + this.anneeNaissance + ", pucee=" + this.pucee +
"]";
    }
}
```

a- Dans la méthode principale, créez 2 instances de la classe Chat, correspondant à Azrael et Grosminet.
b- Affichez ensuite ces instances.

```
public class ManipulationsChats {

    public static void main(String[] var0) {
//création des 2 instances faisant appel au
constructeur chat(….)
        Chat var1 = new Chat("Azrael", "blanc
roux", false, 2017, false);
        Chat var2 = new Chat("Grosminet",
"jaune", true, 1849, false);
// Affichage avec toString(). On peut aussi
écrire System.out.println(var1.toString()) ;
        String var3 = var1.toString();
        System.out.println(var3);
        String var4 = var2.toString();
        System.out.println(var4);
    }
}
```

b- Compilez ces 2 classes : dans le répertoire exo1, javac *.java. Vous obtenez alors 2 fichiers suffixés par ".class".
c- Exécutez la méthode principale : dans le répertoire exo1, java ManipulationsChats.

Ça affiche les attributs des 2 chats formattés par toString().

Exercice 2

On s'intéresse ici à des points dans un repère orthonormé de dimension 2. Un point est représenté par son abscisse (coordonnée x) et son ordonnée (coordonnée y). L'origine du repère est le point de coordonnées (0,0).

Question 1. Implémentez une classe `Point` que vous munirez des éléments suivants :

- des attributs de type double pour l'abscisse et l'ordonnée
- un constructeur paramétré permettant de positionner l'abscisse et l'ordonnée d'un point grâce aux valeurs des paramètres
- un constructeur non paramétré qui construit le point origine
- une méthode toString qui retourne une chaîne de caractères représentant le point receveur, sous la forme : `Point(x,y)` où x et y sont respectivement l'abscisse et l'ordonnée du receveur.
- une méthode `module` qui calcule le module du point receveur, c'est à dire sa distance à l'origine
- une méthode `distanceDe` qui calcule la distance du point receveur à un autre point pris en paramètre
- une méthode `symetrique` qui retourne le point symétrique du point receveur, par symétrie centrale par rapport à l'origine.

Question 2. Implémentez une méthode `main` que vous placerez dans la classe `ManipulationsPoint` qui réalise les traitements suivants :

- création de deux instances de la classe `Point` : une pour l'origine, et une pour le point (3,4), que l'on placera dans des variables nommées respectivement o et p1.
- appel de la méthode `module` pour ces deux instances et affichage d'un message de type : `le module de (0,0) est 0` et `le module de (3,4) est 5`.
- appel de la méthode `symetrique` sur p1, stockage de la référence d'objet obtenue dans une variable nommée p2, appel de la méthode `distanceDe` sur p1 en passant p2 en paramètre, vérification que le résultat obtenu est 10.

Ça donne ça

Classe Points

```
public class Point {
    public double x,y;
Point(double xx, double yy) {
this.x=xx;this.y=yy;}
Point (){this.x=0;this.y=0;}
public String toString(){
return "Point("+this.x+","+this.y+")";}
public double module(){return
Math.sqrt(this.x*this.x+this.y*this.y);}
 public double distanceDe(double x1,double
y1){return Math.sqrt((this.x-x1)*(this.x-
x1)+(this.y-y1)*(this.y-y1));}
public Point sym(){double x2=-(this.x);
double y2=-(this.y);
return new Point(x2,y2);}

}
```

Classe ManipulationsPoints

```
class ManipulationsPoints {
   public static void main(String args[]){
        Point o = new Point();
        System.out.println(o.toString());
        Point p1 = new Point(3,4);
        System.out.println(p1.toString());
        System.out.println(
"le module de p1 est : "+p1.module());
        System.out.println(
"le module de o est : "+o.module());
        Point p2 = p1.sym();
        System.out.println(p2.toString());
        System.out.println(
"distance de p1 a p2 ="
+p1.distanceDe(p2.x,p2.y));
    }
}
```

Exercice 3

```
import java.util.Arrays;

public class SyntheseMeteo {
    private final String ville;
    private float[] tmin;
    private float[] tmax;

    public SyntheseMeteo(String ville){
        this.ville=ville;
    }

    public SyntheseMeteo(String ville, float[] tmin, float[] tmax){
        this(ville);
        this.tmin=tmin;
        this.tmax=tmax;
    }

    @Override
    public String toString() {
        return "SyntheseMeteo{" +
                "ville='" + ville + '\'' +
                ", tmin=" + Arrays.toString(tmin) +
                ", tmax=" + Arrays.toString(tmax) +
                '}';
    }

    public float temperatureMensuelleMax(){
        float result=tmax[0];
        for (float temp:tmax){
            if (temp>result){
                result=temp;
            }
        }
        return result;
    }
```

```
public class ManipulationsSyntheseMeteo {
    public static void main(String[] args) {
        float[] tmin={1, 2, 3, 4, 5, 12, 15, 16, 14, 12, 8, 3};
        float[] tmax={7, 8, 14, 17, 20, 30, 37, 36, 29, 27, 19, 17};
        SyntheseMeteo nimes=new SyntheseMeteo("Nîmes", tmin,
tmax);
        System.out.println(nimes.temperatureMensuelleMax());

        SyntheseMeteo lille=new SyntheseMeteo("Lille");
        lille.setTmin(1, -3);
    }

}
```

On s'intéresse à une classe SyntheseMeteo qui synthétise les informations météorologiques mensuelles d'une ville : son nom est dans l'attribut ville, et ses 12 températures minimales (respectivement maximales) stockées dans le tableau tMin (respectivement tMax).

Question 1. Exécutez le main de la classe ManipulationsSyntheseMeteo. Quelle erreur obtenez-vous ? Pourquoi ? Corrigez la classe SyntheseMeteo de manière à ne plus avoir cette erreur.

Ce qui provoque l'erreur : les 2 lignes du main()

```
SyntheseMeteo lille=new SyntheseMeteo("Lille");
lille.setTmin(1, -3);
```

Le constructeur qui n'utilise que le nom (appel 1ere ligne ci-dessus) ne définit pas l'objet avec les tableaux de température comme arguments ce qui fait que la méthode setTmin n'a pas de tableau où ranger sa valeur.

Correction

```
public SyntheseMeteo(String ville){
    this.ville=ville;
    this.tmin = new float[12]; //ajout Q1
    this.tmax = new float[12]; // ajout Q1
    }
```

On ajoute la création de 2 tableaux vides au constructeur qui n'utilise que la ville.
(lignes this.tmin et this.tmax)

Question 2. Regardez la méthode temperatureMensuelleMax. Le premier "tour" de la boucle for est inutile. Transformez la boucle for (qui est ici un *pour tout*) en une boucle for avec un indice de boucle i allant de 1 à la taille du tableau exclue.

Avant

```
public float temperatureMensuelleMax() {
/*      float result=tmax[0]; //mis en commentaire
    for (float temp:tmax){
      if (temp>result){
        result=temp;
      }
    }
    return result;
}
```

Après

```
    float result = tmax[0];
    for (int i = 1; i < tmax.length; i++) {
        if (tmax[i] > result) {
            result = tmax[i];
        }
    }
    return result;
}
```

Question 3. Ajoutez une méthode moisLePlusChaud qui retourne le mois le plus chaud sous forme d'un entier (1 pour janvier, 2 pour février, 12 pour décembre).

```
public int moisChaud(){  float result = tmax[0]; int mois = 1;
 for (int i =1; i < tmax.length; i++) {
        if (tmax[i] > result) {
            result = tmax[i]; mois=i+1;
        }
    }
 return mois;
}
```

On mémorise une température du tableau dans result , on mémorise le numéro de mois correspondant dans mois et on parcourt toute la boucle des températures .
Si on trouve une température plus grande que result on remplace result par sa valeur et on affecte le numéro du mois correspondant à mois.

Question 4. Modifiez la méthode précédente pour retourner un littéral d'une énumération Mois plutôt que le numéro du mois. Vous créerez une énumération `Mois` dans un fichier `Mois.java`. Pour convertir un numéro de moi en un littéral de l'énumération Mois, vous utiliserez la méthode `numMoisVersMois` qui est commentée dans la classe `SyntheseMeteo`.

Le prof propose de créer 2 fonctions

```
private Mois numMoisVersMois(int numMois){
    //return Mois.values()[numMois-1]; // Cette instruction aurait suffit mais la comprenez-vous ?
    switch (numMois){
        case 1:return Mois.JANVIER;
```

Attention, ici Mois.JANVIER n'est pas une chaine. C'est l'élément JANVIER de l'énumération des noms de mois, l'objet énumération étant appelé Mois avec une majuscule. D'ailleurs on remarque que la fonction numMoisVersMois retourne un Mois et non un string. Pour transformer un Mois en String il suffit d'écrire String chaine = Mois.JANVIER.name() ;

Et

```
 private int moisVersNumMois(Mois mois){
 //return Mois.JANVIER.ordinal()+1; // Cette instruction aurait suffit mais la comprenez-vous ?
 switch (mois){
     case JANVIER : return 1;
```

Ici on envoie un Mois à la fonction et elle retourne un int.

```
Pour créer un Mois à partir d'une chaine il faut écrire Mois m =Mois.valueOf("JUILLET");
```

Pour créer les deux fonctions dans syntheseMeteo il suffit de les dé-commenter .
Puis on les appelle avec pour paramètre un entier numMoisVersMois(12)= DECEMBRE ou un Mois moisVersNumMois(DECEMBRE)=12. Dans la pratique il faudra souvent transformer une chaine en Mois ou un Mois en chaine, ce qui fait que quand on utilise un SWITCH() l'énumération ne fait que compliquer les choses, il suffit d'envoyer directement la chaine du mois à moisVersNumMois et de faire en sorte que numMoisVersMois renvoie une chaine et non un mois. Les Switch n'utilisant que des int et des String.

Je me suis donc plutôt intéressé à l'utilisation des méthodes liées à une énumération qui simplifient beaucoup les choses. Chaque libellé de l'énumération est obtenu à partir de son index par Mois.values()[index] et si m est un Mois on obtient l'index en écrivant m.ordinal().

Enumération Mois et 2 méthodes (dans mois.java)

```
public enum Mois {
    JANVIER, FEVRIER, MARS, AVRIL, MAI,
JUIN, JUILLET, AOUT, SEPTEMBRE, OCTOBRE,
NOVEMBRE, DECEMBRE;
    // Pour obtenir le mois à partir de
l'index
    public static Mois quelmois(int ind)
{return Mois.values()[ind-1];}
      // Pour obtenir l'index à partir d'un
membre de l'énumération
public static int quelindex(Mois m){
return m.ordinal()+1;}
}
```

```
//Q3 Modifié retourne le numéro du mois plus chaud
 public int moisChaud(){  float result = tmax[0]; int mois = 0;
  for (int i = 0; i < tmax.length; i++) {
        if (tmax[i] > result) {
           result = tmax[i]; mois=i+1;
        }
    }
  return mois;
 }
//Q4 Modifié pour donner le nom du mois chaud et pas le numéro

 public String nomMoisChaud(){
Int index = moisChaud() ;
Mois m = Mois.quelmois(index ) ;
Return m.name() ;
}
```

On utilise moisChaud() pour obtenir l'index du mois, puis on utilise Mois.quelmois(index) pour obtenir le Mois m puis m.name() pour obtenir la String

Question 5. Dans la méthode main de `ManipulationsSyntheseMeteo`, ajoutez en dernière ligne l'instruction `lille.setTmin(13, 10);` et exécutez de nouveau le main. Analysez l'erreur obtenue.

Question 6. Modifiez la méthode setTmin afin que :

- il n'y ait pas d'accès au tableau si le numéro de mois passé en paramètre n'est pas compris entre 1 et 12.
- la valeur ne soit pas reportée dans le tableau si elle est aberrante (inférieure à -100 ou supérieure à 100)

```
//Q6 modifié pour sortie si mois ou température incorrects
  public void setTmin(int numMois, float valeur){
    if(numMois<1 || numMois>12){System.out.println("Numero de
mois incorrect");return;}
    if(valeur<-100 ||valeur >100) {System.out.println("Temperature
aberrante");return;}
    tmin[numMois+1]=valeur;
  }
```

L'erreur provient de l'utilisation dans setTmin(13,10) d'un mois no 13 alors que cet index n'existe pas dans ce tableau. Pour éviter les erreurs on contrôle numMois et valeur par un if en entrée de programme et si ça ne convient pas on édite un message et on sort (return)

Question 7. Ajoutez une autre méthode setTmin qui prend en paramètre un littéral d'énumération Mois plutôt qu'un numéro de mois. Pour convertir un Mois en un numéro de mois, on utilisera la méthode moisVersNumMois qui est commentée dans la classe `SyntheseMeteo`.

```
//Q7 accepte le nom du mois
public void setTmin(String nomMois, float valeur)
{
    nomMois=nomMois.toUpperCase();
    Mois m =Mois.valueOf(nomMois); // Pour
obtenir un membre de l 'énumération à partir
d'une string
    int x = Mois.quelindex(m);
    tmin[x+1]=valeur;
}
```

On aurait pu envoyer directement un membre de l'énumération à setMin sous la forme Mois.JANVIER mais en réalité on manipule souvent des chaines.
J'ai donc supposé que nomMois était une chaine et on la transformait en Mois dans la fonction.

ManipulationsSyntheseMeteo a été modifié pour contrôler nos modifications. Au total ça donne ça :

Classe synthèse météo

```
import java.util.Arrays;
public class SyntheseMeteo
{
    private String ville;
    private float[] tmin;
    private float[] tmax;
 public SyntheseMeteo(String ville){
     this.ville=ville;
    this.tmin = new float[12]; //ajout Q1
    this.tmax = new float[12]; // ajout Q1
    }
public SyntheseMeteo(String ville, float[] tmin, float[] tmax){
    this(ville);
    this.tmin= tmin;
    this.tmax=tmax;
  }
  public String toString() {
    return "SyntheseMeteo{" +
          "ville='" + ville + '\'' +
          ", tmin=" + Arrays.toString(tmin) +
          ", tmax=" + Arrays.toString(tmax) +
          '}';
  }

  public float temperatureMensuelleMax() {
/*      float result=tmax[0]; //mis en commentaire
    for (float temp:tmax){
       if (temp>result){
          result=temp;
       }
    }
    return result;
  }
 *///Q2 même métode mais en parcourant tmax[] en boucle
    float result = tmax[0];
    for (int i = 0; i < tmax.length; i++) {
       if (tmax[i] > result) {
          result = tmax[i];
       }
    }
    return result;
  }
//Q6 modifié pour sortie si mois ou température incorrects
  public void setTmin(int numMois, float valeur){
    if(numMois<1 || numMois>12){System.out.println("Numero de
mois incorrect");return;}
    if(valeur<-100 ||valeur >100) {System.out.println("Temperature
aberrante");return;}
    tmin[numMois+1]=valeur;
  }
//Q7 accepte le nom du mois
public void setTmin(String nomMois, float valeur)
{
    nomMois=nomMois.toUpperCase();
    Mois m =Mois.valueOf(nomMois); // Pour
obtenir un membre de l 'énumération à partir
d'une string
    int x = Mois.quelindex(m);
    tmin[x+1]=valeur;
}
//Q3 Modifié retourne le numéro du mois plus chaud
  public int moisChaud(){  float result = tmax[0]; int mois = 0;
  for (int i = 0; i < tmax.length; i++) {
       if (tmax[i] > result) {
          result = tmax[i]; mois=i+1;
       }
    }
  return mois;
  }
//Q4 Modifié pour donner le nom du mois chaud et pas le numéro

  public String nomMoisChaud(){
Int index = moisChaud() ;
Mois m = Mois.quelmois(index ) ;
Return m.name() ;
}
 }
```

Enumération Mois et 2 méthodes (dans mois.java)

```
public enum Mois {
    JANVIER, FEVRIER, MARS, AVRIL, MAI,
JUIN, JUILLET, AOUT, SEPTEMBRE, OCTOBRE,
NOVEMBRE, DECEMBRE;
    // Pour obtenir le mois à partir de
l'index
    public static Mois quelmois(int ind)
{return Mois.values()[ind-1];}
     // Pour obtenir l'index à partir d'un
membre de l'énumération
public static int quelindex(Mois m){
return m.ordinal()+1;}
}
```

Classe Manipulations

```
public class ManipulationsSyntheseMeteo {
    public static void main(String[] args) {
        float[] tmin={1, 2, 3, 4, 5, 12, 15,
16, 14, 12, 8, 3};
        float[] tmax={7, 8, 14, 17, 20, 30,
37, 36, 29, 27, 19, 17};
        SyntheseMeteo nimes=new
SyntheseMeteo("Nîmes", tmin, tmax);

System.out.println(nimes.temperatureMensuell
eMax());

System.out.println(nimes.moisChaud());

        SyntheseMeteo lille=new
SyntheseMeteo("Lille");
      lille.setTmin(1, -3);
      lille.setTmin("juillet",-35);

System.out.println(lille.toString());

System.out.println(Mois.quelmois(3));
// Pour obtenir un membre de l 'énumération
à partir d'une chaine (« JUILLET »)
        Mois m =Mois.valueOf("JUILLET");
//Pour obtenir une chaine à partir d'un
membre m de l'énumération (chaine='JUILLET')
        String chaine = m.name() ;
//Pour obtenir l'index de m dans
l'énumération (index = 7)
        Int index = m.ordinal() ;

System.out.println(Mois.quelindex(m));

        lille.setTmin(13, 10);

System.out.println(nimes.nomMoisChaud());
    }
}
```

Sorties écran de Manipulations

```
37.0
7
SyntheseMeteo{ville='Lille', tmin=[0.0, 0.0, -3.0, 0.0, 0.0, 0.0, 0.0, 0.0, -
35.0, 0.0, 0.0, 0.0], tmax=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0]}
MARS
7
Numero de mois incorrect
JUILLET
```