République Tunisienne
Ministère de l'Éducation

Centre National de Formation des Formateurs en Éducation

Module de formation

Intitulé : Initiation au langage JAVA

Discipline : Informatique

Équipe de travail formé par :

- …
- …

Année 2016

Sommaire

Problématique 4
Présentation de la formation 5
Plan de la formation 7
Programmation orientée objet 11
1. Introduction 11
2. Le concept de classe 11
a. Déclaration d'une Classe 11
b. Déclaration des attributs et des méthodes 12
3. Les objets 16
La création d'un objet : instancier une classe 16
a. Cycle de vie d'un objet 16
b. Les références et la comparaison d'objets 16
c. L'objet nul 17
d. Variable this 17
e. Modificateurs d'accès 19
Introduction et Installation 20
1. De la programmation classique vers la programmation orientée objet 20
2. Introduction 21
3. Les techniques de base de la programmation en Java 21
a. Environnement de JAVA 21
b. Caractéristiques de JAVA 22
4. Installation sous Windows 23
a. Créer un projet sous Netbeans – Étape 1 23
b. Créer un projet sous Netbeans – Étape 2 24
c. Créer un projet sous Netbeans – Étape 3 24
d. Créer un projet sous Netbeans – Étape 4 25
e. Créer un projet sous Netbeans – Étape 5 25
f. Créer un projet sous Netbeans – Étape 6 26
Les Structures de données en JAVA 29
1. Les identificateurs 29
2. Variables 29
a. Les types primitifs 30
b. Déclaration et initialisation d'une variable 30
c. Type objets 32
3. Les opérations d'entrées/sorties 34
a. Écriture 34
b. Lecture à partir du clavier 34
4. Les structures de contrôle conditionnelles 35
5. Les structures de contrôle itératives 35
a. La boucle For 35

b. La boucle Do While 36
6. Les débranchements 36
7. Les exceptions en Java 37
8. Les fonctions/méthodes 37
9. La surcharge des méthodes 39
Gestion des entrées sorties en Java 40
1. Introduction 40
2. Flux (Stream) 40
3. Lecture et écriture des flux 41
4. Écriture d'un fichier texte 41
5. Lecture d'un fichier texte 42
6. Entrées/Sorties standard (console) 42
7. Les flux binaires de sortie et d'entrée 43
Les Concepts de base de la programmation orientée objet en JAVA 44
I. Héritage 44
a. Règles de l'héritage 44
II. Polymorphisme 51
III. Interface 53
a. Définition 53
IV. Encapsulation 56
V. Abstraction 56
Réalisation d'interfaces graphiques Avec NetBeans 58
1. Création d'un projet 58
2. Ajout d'une interface dans le projet 59
a. Ajouter un composant visuel à votre projet. 60
b. Compiler le projet 61
c. Les éléments de base d'une fenêtre 61
d. Communication avec l'utilisateur 66
Connexion à une Base de données (JDBC) 70
I. Introduction 70
II. Étapes de connexion à une BD 70
a. Chargement du pilote 70
b. Connexion au SGBD 71
c. Création de l'objet Statement 72
Série d'exercices 75
Correction de la série d'exercices 84
Bibliographie 118

Problématique

Les nouvelles technologies tiennent une place prépondérante au quotidien des élèves, des enseignants et dans l'enseignement. Pour cela, nos enseignants sont attirés par l'utilisation des nouvelles technologies pour optimiser l'apprentissage en salle de classe et s'adapter à l'environnement externe, la pédagogie active qui est soutenue par la majorité de nos programmes scolaires à travers les recommandations.

Dans ce contexte, il y a une évolution extraordinaire dans le domaine informatique qui a touché toutes les branches (systèmes d'exploitation, réseaux informatiques, sécurité informatique, base de données, méthodes de conception, …). En parallèle, il y a une évolution rapide au niveau des langages de programmation pour d'adapter aux besoins des programmeurs et des systèmes d'information (naissance du concept orienté objet, développement des applications réparties, …) et dans le but de s'adapter aux nouvelles technologies des langages de programmation et d'améliorer la section « Sciences de l'Informatique » niveau troisième et quatrième année qui se base sur quatre matières fondamentales, essentiellement « Algorithmique et programmation » qui s'enseigne sur deux ans. Cependant, il y a aussi des observations effectuées en classe et l'écoute active des propos des enseignants et des apprenants qui se résument en un point nécessitant l'utilisation d'un langage de programmation plus évolué et plus riche qui s'exécute sans tenir compte de l'environnement de son exécution.

En même temps, les difficultés résumées dans la partie « Problématique » constituent un vrai obstacle pour toute tentative allant dans ce sens.

Le thème « Initiation Java » pourrait aplanir ces difficultés et ce en visant des compétences fondamentales qui permettront à l'enseignant de mieux agir et d'être ainsi plus efficace.

Présentation de la formation

Cette formation est conçue comme une introduction aux paradigmes de la programmation orientée objet.

Elle présente les notions de base de cette technologie : classe, objet, polymorphisme, abstraction, encapsulation, surcharge. Ces concepts seront illustrés avec le langage JAVA.

Cette formation permet aussi aux formés d'avoir un panorama complet de ses Domaines d'applications. Elle est extrêmement pratique, elle présente les principales classes utilitaires nécessaires aux algorithmes de base, à la gestion des entrées/sorties, à la création d'interface ou à l'accès à une base de données.

La formation prévue devrait s'effectuer dans un laboratoire d'informatique avec un public réduit (deux par poste au maximum). Elle s'étale sur cinq journées où se chevauchent apprentissage, discussions et productions. Les tableaux ci-joints présentent en détail la consistance de chaque journée. La formation prévue est axée sur l'exploitation du langage de programmation ***JAVA*** en utilisant l'éditeur ***NetBeans*** ou ***Elipse*** pour donner l'opportunité aux formés de développer des applications.

Prérequis :

Algorithmique, Structures de données,

Méthodologie :

- Exposé informel
- Laboratoire

Moyens pédagogiques :

- Tableau
- Vidéo projecteur
- Laboratoire (Ordinateurs équipés de NetBeans7.2)

Objectifs généraux	Objectifs spécifiques	Critères d'évaluation de la performance
1- S'initier aux concepts orientés objets et leurs intérêts pratiques.	À partir des notes du cours et des discussions entre les apprenants le formé doit être capable de dégager les lacunes de l'approche classique de programmation et de distinguer les principales caractéristiques de POO	Aucune confusion entre les principes de la programmation classique et la programmation orientée objet
2- Comprendre les concepts orientés objet en termes de concept, syntaxe Java et utilisation.	À partir des notes du cours, le formé doit être capable de concevoir des classes et des objets en Java	Les exercices relatifs à l'implémentation d'une classe en Java doivent être réussis
3- Comprendre l'utilité et l'application des concepts d'encapsulation et de surcharge.	À partir des notes du cours, le formé doit maîtriser les concepts d'encapsulation et surcharge	Les exercices relatifs à l'accès aux membres privés doivent être maîtrisés
4- Comprendre la technique d'héritage et ses intérêts pratiques.	À partir des notes du cours, le formé doit être capable de concevoir et d'implémenter des classes dérivées.	Aucune erreur n'est permise au niveau de la définition et de la manipulation d'une classe dérivée
5- Comprendre le concept de polymorphisme et de ses intérêts pratiques	À partir des notes du cours, le formé doit comprendre l'intérêt pratique de polymorphisme et de la relation avec l'héritage	Aucune erreur n'est permise au niveau de la liaison entre l'héritage et le polymorphisme
6- Comprendre et distinguer entre les 2 concepts de classes abstraites et interfaces	À partir des notes du cours, le formé doit comprendre l'utilité pratique des classes abstraites et d'interfaces et doit distinguer les nuances entre ces deux concepts	Aucune confusion entre les 2 concepts n'est permise
7- Faire découvrir la notion de gestion des exceptions	À partir des notes du cours, le formé doit comprendre l'utilité pratique des exceptions.	Les exercices relatifs à l'implémentation des mécanismes des exceptions en Java doivent être réussis

Plan de la formation

Journée I

Thème n°1 : Introduction à la notion de la programmation orientée objet		
Les activités de la formation	**Durée**	**Remarque(s)**
- Discuter avec les formés des problèmes rencontrés dans la programmation procédurale - Les langages de programmation utilisés actuellement dans l'enseignement de la matière « algorithmes & programmation » sont-elles en adéquation avec les nouvelles technologies ? - Le langage JAVA est une issue possible - Présentation des objectifs de la POO - Comparaison entre un langage procédural et un langage objet.	1h	Travail en groupe
Thème n°2 : Présentation du langage JAVA		
Activité	**Durée**	**Remarque(s)**
- Présenter au public une collection de langages de programmation orientée objet - Aperçu sur les caractéristiques fondamentales du langage Java - Environnement Java (Java Application Programming Interface, Java Runtime Environment) - Installation des outils de développement - Premier exemple en Java	1h30	Présentation (Data show)
Thème n°3 : Syntaxe du langage JAVA		
Activité	**Durée**	**Remarque(s)**
- Compilation et exécution - Découverte de l'environnement Eclipse - Constantes - Variables et Types primitifs - Tableaux et matrices - Chaînes de caractères - Les opérateurs - Les structures de contrôle Conditionnelles - Les structures de contrôle Itératives - Instruction Break et Continue	2h30	Présentation (Data show)

Journée II

Thème n°1 : Éléments de programmation Java		
Les activités de la formation	**Durée**	**Remarque(s)**
- Class - Package - Variables et méthodes : - Visibilité des champs - Variables et méthodes de classe	1h	Travail en groupe
Thème n°2 : Héritage et polymorphisme		
Activité	**Durée**	**Remarque(s)**
- Principe de l'héritage - Polymorphisme - Encapsulation	1h30	Présentation (Data show)
Thème n°3 : Interfaces et méthodes		
Activité	**Durée**	**Remarque(s)**
- Interfaces - Classes abstraites - Classes et méthodes génériques - Itératives - Instruction Break et Continue	2h30	Laboratoire informatique

Journée III

Thème n°1 : Gestion des exceptions		
Activité	**Durée**	**Remarque(s)**
- Déclaration - Interception et traitement - Classes d'exception - Classification des erreurs en Java. - Exercice d'application	1h30	Laboratoire informatique
Thème n°2 : Gestion des entrées/sorties simples		
Activité	**Durée**	**Remarque(s)**
- Lecture des entrées clavier - Lecture à partir d'un fichier - Écriture sur la sortie standard "écran" - Écriture dans un fichier	1h30	Laboratoire informatique
Thème n°3 : Applications graphiques (package swing)		
Activité	**Durée**	**Remarque(s)**
- Le schéma Modèle-Vue-Contrôleur - Composants graphiques - Composants - Containers - Exemple d'interface - Contrôleurs d'événements - Événements - Interface Listener - Exercice d'évaluation	2h	Laboratoire informatique

Journée IV

Thème n°1 : Java et la programmation événementielle		
Activité	**Durée**	**Remarque(s)**
- La première fenêtre à créer - Gestion des boutons - Interaction avec des boutons	1h30	Laboratoire informatique
Thème n°2 : Gestion des Formulaires et des menus		
- Gestion des Formulaires - Gestion des menus	1h30	Laboratoire informatique
Thème n°3 : Interactions avec les bases de données		
Activité	**Durée**	**Remarque(s)**
- Connexion à une base de données - Exécutions de requêtes SQL - Parcours de resultSet, Requêtes	2h	Laboratoire informatique

Journée V

Thème n°1 : Interactions avec les bases de données (suite)		
Activité	**Durée**	**Remarque(s)**
Exercice d'application (gestion des cours) permettant de créer des interfaces graphiques d'ajout et de suppression d'un cours (utilisation des boutons, formulaires, menus…)	2h	Laboratoire informatique
- Connexion avec la base de données - Interaction avec la base de données (Ajout et suppression des données…)	2h30	
Thème n°2 : Évaluation de la formation		
Distribution d'une fiche d'évaluation à remplir par les participants	20mn	
Clôture de la formation	10mn	

Journée VI

Thème n°1 : Java et la programmation événementielle		
Activité	**Durée**	**Remarque(s)**
- La première fenêtre à créer - Gestion des boutons - Interaction avec des boutons	1h30	Laboratoire informatique
Thème n°2 : Gestion des Formulaires et des menus		
Activité	**Durée**	**Remarque(s)**
- Gestion des Formulaires - Gestion des menus	1h30	Laboratoire informatique
Thème n°3 : Interactions avec les bases de données		
Activité	**Durée**	**Remarque(s)**
- Connexion à une base de données - Exécutions de requêtes SQL - Parcours de resultSet, Requêtes	2h	Laboratoire informatique

Programmation orientée objet

1. Introduction

La programmation orientée objet existe depuis 1967 (date d'apparition du langage Simula '67). Cependant, elle n'est vraiment devenue un des paradigmes de la programmation qu'au milieu des années 1980. La programmation orientée objet résout de nombreux problèmes inhérents à cette conception en mettant dans une même entité les attributs et les opérations. Cela est plus proche du monde réel, dans lequel tous les objets disposent d'attributs auxquels sont associés des activités. Java est un pur langage orienté objet, ce qui signifie que le niveau le plus externe de la structure des données est l'objet. Il n'y a pas de constante, de variable ni de fonction indépendante en Java. On accède à toute chose via les classes et les objets. C'est un des aspects les plus agréables de Java.

2. Le concept de classe

Une classe est un modèle décrivant les caractéristiques communes et les comportements d'un ensemble d'objets. En programmation orientée objet, une classe déclare des attributs représentant l'état des objets et des méthodes représentant leur comportement.

Une Classe est caractérisée par :

- Un nom
- Une composante statique : des **champs** (ou **attributs**). Ils caractérisent l'état des objets pendant l'exécution du programme
- Une composante dynamique : des **méthodes** représentant le comportement des objets de cette classe. Elles manipulent les champs des objets et caractérisent les actions pouvant être effectuées par les objets.

a. Déclaration d'une Classe

Une classe se déclare par le mot-clé **class**.

Exemple :

modificateurs nom_classe [extends classe_mere]

```
[implements interface]
{
// insérer ici les champs et les méthodes
}
```

La déclaration d'une instance d'une classe se fait d'une façon similaire à la déclaration d'une variable :

```
nomClasse nomInstance ;
Exemple :
Surface s1 ;
```

- ✓ Les modificateurs de classe sont :

Modificateur	Rôle
Abstract	la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.
Final	la classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
Private	la classe n'est accessible qu'à partir du fichier où elle est définie
Public	La classe est accessible partout

b. Déclaration des attributs et des méthodes

- ❖ **Déclaration des attributs :**

```
Type nomAttribut ;
```

L'accès aux valeurs des attributs d'une instance de nom nomInstance se fait via la notation pointée :

```
nomInstance.nomAttribut
```

Exemple : la valeur de l'attribut largeur d'une instance r de la classe rectangle sera référencée par l'expression : **r.Largeur**

- ❖ **Déclaration d'une méthode :**

Les méthodes sont des fonctions qui implémentent les traitements de la classe.

La déclaration d'une méthode d'une classe est la suivante :

```
modificateurs type_retourné nom_méthode ( arg1, ... ) {
...
}
```

Le type retourné peut-être élémentaire ou peut correspondre à un objet. Si la méthode ne retourne rien, alors on utilise **void.**

Les modificateurs de méthodes sont :

Modificateur	Rôle
Public	la méthode est accessible aux méthodes des autres classes
Private	l'usage de la méthode est réservé aux autres méthodes de la même classe
Protected	la méthode ne peut être invoquée que par des méthodes de la classe ou de ses sous classes
Final	la méthode ne peut être modifiée (redéfinition lors de l'héritage interdite)
Static	la méthode appartient simultanément à tous les objets de la classe (comme une constante déclarée à l'intérieur de la classe). Il est inutile d'instancier la classe pour appeler la méthode mais la méthode ne peut pas manipuler de variable d'instance. Elle ne peut utiliser que des variables de classes.
synchronized	la méthode fait partie d'un thread. Lorsqu'elle est appelée, elle barre l'accès à son instance. L'instance est à nouveau libérée à la fin de son exécution.
Native	le code source de la méthode est écrit dans un autre langage

Sans modificateur, la méthode peut être appelée par toutes autres méthodes des classes du package auquel appartient la classe.

La valeur de retour de la méthode doit être transmise par l'instruction return. Elle indique la valeur que prend la méthode et termine celle-ci : toutes les instructions qui suivent return sont donc ignorées.

Exemple :

```
public class PremierProgramme {

    public static void main(String[] args) {
        // TODO code application logic here
    System.out.println("Bienvenue, Formation JAVA!");
    Rectangle r = new Rectangle();
    r.largeur =3.0;
    r.longueur =5.0;
  System.out.println("La largeur est: "+r.largeur+"La largueur est: "+r.longueur+"La surface est: "+surface(r.largeur,r.longueur));
    }

  static   class Rectangle
    {
    double largeur;
    double longueur;
    }


static double surface(double a,double b)
{

return (a*b);
}
}
```

❖ Transmission de paramètres

En Java, il n'existe que le passage par valeur : une méthode utilise toujours une copie de la valeur qui lui est passée en paramètre.

❖ Surcharge de méthodes

La surcharge d'une méthode ou d'un constructeur permet de définir plusieurs fois une même méthode/constructeur avec des arguments différents. Dans ce cas le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments.
Ce mécanisme, appelé *surcharge* des méthodes, est très utile pour écrire des méthodes « sensibles » au type de leurs arguments, c'est-à-dire des méthodes correspondant à des traitements de même nature mais s'appliquant à des entités de types différents et pour simplifier l'interface des classes vis-à-vis des autres classes.
Une méthode est surchargée lorsqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.

Exemple :

```
class affiche{
public void afficheValeur(int i) {
System.out.pintln("nombre entier =" + i);
}
public void afficheValeur(float f) {
System.out.println("nombre flottant =" + f);
}
}
```

❖ Constructeurs

Un constructeur est une méthode qui porte le nom de la classe et qui est chargé d'initialiser les variables de l'objet au moment de sa création. Le constructeur suit la définition des autres méthodes excepté que son nom doit obligatoirement correspondre à celui de la classe et qu'il n'est pas typé, pas même void, donc il ne peut pas y avoir d'instruction return dans un constructeur.
Il existe plusieurs manières de définir un constructeur :

```
NomClasse(liste_paramètres)
{
 /* initialisation des attributs
    en utilisant liste_paramètres */
}
```

Exemple

```
class Rectangle
  {
  private double largeur;
  private double longueur;
      public Rectangle(double a, double b)
      {
      largeur=a;
      longueur=b;
       }
      public Rectangle()
      {
          this(0.0,0.0);
      }
```

3. Les objets

Les objets contiennent des attributs et des méthodes. Les attributs sont des variables ou des objets nécessaires au fonctionnement de l'objet. En java, une application est un objet. La classe est la description d'un objet. Un objet est une instance d'une classe. Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.

La création d'un objet : instancier une classe

La déclaration est de la forme :

Nom_classe nom_variable

Exemple : MaClasse m;

L'opérateur **new** permet de créer une instance de la classe et de l'associer à la variable.

Exemple : m = new MaClasse();

Il est possible de tout réunir en une seule déclaration.

Exemple : MaClasse m = new MaClasse();

Remarque : sur les objets de type String : Un objet String est automatiquement créé lors de l'utilisation d'une constante chaîne de caractères sauf si celle-ci est déjà utilisée dans la classe. Ceci permet une simplification dans l'écriture des programmes. Exemple :

String chaine = "bonjour" et String chaine = new String("bonjour") sont équivalents.

a. Cycle de vie d'un objet

Le cycle de vie d’un objet passe par les étapes suivantes :

- Déclaration et initialisation (allocation de la mémoire et initialisation)
- Utilisation (éventuellement recopié)
- Destruction (Il est détruit et la mémoire est libérée : Fin de vie de l’objet)

b. Les références et la comparaison d'objets

Les variables déclarées de type objet ne contiennent pas un objet mais une référence vers cet objet. Lorsque l'on écrive c1 = c2 (c1 et c2 sont des objets), on copie la référence de l'objet c2 dans c1 : c1 et c2 réfèrent au même objet (ils pointent sur le même objet). L'opérateur == compare ces références.

Exemples :

Rectangle r1 = new Rectangle(100,50);

Rectangle r2 = new Rectangle(100,50); if (r1 == r1) { ... } // vrai

c. L'objet null

L'objet null est utilisable partout. Il n'appartient pas à une classe mais il peut être utilisé à la place d'un objet de n'importe quelle classe ou comme paramètre. Null ne peut pas être utilisé comme un objet normal : il n'y a pas d'appel de méthodes et aucune classe ne puisse en hériter.

d. Variable this

this est un objet qui est égal à l'instance de l'objet dans lequel il est utilisé.

Exemple :

```
class Rectangle
  {
  private double largeur;
  private double longueur;
      public Rectangle(double a, double b)
      {
      largeur=a;
      longueur=b;
       }
      public Rectangle()
      {
          this(0.0,0.0);
      }
```

e. Modificateurs d'accès

Les modificateurs d'accès se placent avant ou après le type de l'objet mais la convention veut qu'ils soient placés avant. Ils s'appliquent aux classes et/ou aux méthodes et/ou aux attributs. Ils ne peuvent pas être utilisés pour qualifier des variables locales : seules les variables d'instances et de classes peuvent en profiter.

Ils assurent le contrôle des conditions d'héritage, d'accès aux éléments et de modification de données par les autres objets.

Visibilité des attributs et des méthodes

De nombreux langages orientés objet introduisent des attributs de visibilité pour réglementer l'accès aux classes et aux objets, aux méthodes et aux données.

Il existe 3 modificateurs qui peuvent être utilisés pour définir les attributs de visibilité des entités (classes, méthodes ou attributs) :

- **Public** : une variable, une méthode ou classe déclarée public est visible même à l'extérieur de son package (les attributs public sont accessibles à tous).
- **Private** : les attributs, les méthodes privées sont accessibles seulement par la classe elle-même. C'est le niveau de protection le plus fort.
- **Protected** : les attributs, les méthodes protégées sont accessibles seulement à la classe elle-même et aux classes dérivées.
- **Par défaut** : Si aucun modificateur n'est indiqué, l'élément n'est accessible que depuis les classes faisant partie du même *package* : on dit que leur portée est default.

Mot clé static

Le mot clé **static** s'applique aux variables et aux méthodes pour indiquer que la valeur d'une variable (méthode) est partagée entre toutes les instances de la classe.

Exemple :

```
class Abc {
    public static void main(String[] args) {
        A.c++;
        A v1;
        v1 = new A();
        v1.modifier();
    }
}
class A {
    int b = 1;          // variable d'instance
    static int c = 10; // variable de classe
    void modifier() {
        b++;
        c++;
    }
}
```

Mot clé final

Le mot clé final s'applique aux variables, aux méthodes et aux classes.

Une variable qualifiée de **final** signifie que la variable est **constante**. On ne peut déclarer de variables **final** locales à une méthode. Les constantes sont qualifiées de **final et static**.

Exemple :

```
public static final float PI = 3.1416f;
```

f. Package

Un package est un regroupement de classes et de méthodes, il peut être considéré comme une bibliothèque des classes : il permet de regrouper. Les packages peuvent être organisés d'une manière hiérarchique en sous-packages et ainsi de suite.

Les Atouts de l'utilisation des packages sont :

- Faciliter le développement des bibliothèques et des modules autonomes.
- Les classes d'un même package travaillent conjointement sur un même domaine.
- Faciliter la réutilisabilité.

Pour utiliser un package ainsi créé, on l'importe dans le fichier :

import nomPackage.*;

Introduction et Installation

1. De la programmation classique vers la programmation orientée objet

La programmation classique ou procédurale définit un programme comme étant un ensemble de données sur lesquelles agissent des procédures et des fonctions. Les données constituent la partie passive du programme. Les procédures et les fonctions constituent la partie active.

Dans ce paradigme de programmation, les données et les procédures sont traitées indépendamment les unes des autres sans tenir compte des relations étroites qui les unissent. En effet, les données et les opérations sur les données sont séparées, les structures de données sont donc envoyées aux procédures et fonctions qui les utilisent.

Dans ce cas, la programmation classique consiste à :

- Définir un certain nombre de variables (structures, tableaux…),
- Écrire des procédures pour les manipuler sans associer explicitement les unes aux autres,
- Un paradigme qui se fonde sur le concept d'appel procédural,
- Séparer les données et les traitements,
- Mettre l'accent sur les étapes pour réaliser une tâche,
- Convertir le mieux aux applications ne nécessitant pas ou peu d'interaction avec les usagers,
- La maintenance et l'ajout de nouvelles fonctionnalités demandent de modifier ou d'insérer des séquences dans ce qui existe déjà.

Au contraire de la programmation structurée traditionnelle, la programmation orientée objet met dans une même et unique structure les données et les opérations qui leurs sont associées.

Cependant, La programmation orientée objet consiste à **:**

- Mettre l'accent sur les objets requis pour résoudre un problème,
- Définir un programme comme un ensemble d'objets et des interactions entre ces objets,
- Regrouper des traitements et des données en une seule entité.

Les atouts de la programmation orientée objet sont :

- Robustesse : Par rapport au changement ou à l'extension (possibilité de ne pas réécrire tous) et par rapport aux erreurs de manipulation des données
- Modularité,
- Lisibilité,
- Réutilisabilité : Réutilisation de code en permettant l'extensibilité des objets existants (héritage) qui peuvent alors être manipulés avec les mêmes fonctions (polymorphisme).

2. Introduction

Java est un environnement de **programmation orientée objet** adapté à la **distribution d'applications** sur Internet et s'intégrant au Web.

Java incorpore des fonctionnalités qui facilitent grandement certaines tâches de programmation avancées comme la **gestion des réseaux**, la **connectivité des bases de données** ou le **développement d'applications multitâches**. Il se compose de 4 éléments :

- Un langage de programmation
- Une machine virtuelle (JVM)
- Un ensemble de classes standards réparties dans différentes API
- Un ensemble d'outils (jdb, javadoc, ...)

3. Les techniques de base de la programmation en Java

a. Environnement de JAVA

- **Java Application Programming Interface**

L'API Java est structurée en librairies (packages) qui comprennent des ensembles fonctionnels de composants (classes).

Le noyau (core) de l'API Java (incluse dans toute implémentation complète de la plateforme Java) comprend notamment :

- ✓ Essentials (data types, objects, string, array, vector, I/O,date,…)
- ✓ Applet
- ✓ Abstract Windowing Toolkit (AWT)
- ✓ Basic Networking (URL, Socket –TCP or UDP-,IP)
- ✓ Security

- **Java Virtual Machine (JVM)**

La machine virtuelle Java (en anglais *Java virtual machine*, ou JVM) est un environnement informatique fictif qui exécute des programmes compilés sous forme de bytecode Java.
Cet environnement est simulé par un logiciel spécifique à chaque plate-forme ou couple (machine/système d'exploitation) qui permet aux applications Java compilées en bytecode de produire les mêmes résultats quelle que soit la plate-forme, tant que celle-ci est pourvue de la machine virtuelle Java adéquate.
Une machine virtuelle java permet d'exécuter d'autres langages que le langage java, du moment qu'un compilateur transforme les expressions de cet autre langage en une série d'octets respectant le format binaire d'une classe java.
La machine virtuelle exécute les instructions données par le bytecode contenu dans la classe java en suivant le modèle de la pile : chaque niveau de pile contient les données propres à chaque opération.

- **Java Runtime Environment (JRE)**

Trois tâches principales :

- ✓ Charger le code (class loader)
- ✓ Vérifier le code (bytecode verifier)
- ✓ Exécuter le code (runtime interpreter)

c. Caractéristiques de JAVA

Java est un langage très récent, puisqu'il a été introduit par Sun en 1995. Son immense succès est dû au fait qu'il possède de nombreuses caractéristiques intéressantes :

- Il est portable sur beaucoup de plates-formes.
- Le code produit (pseudo-code ou byte-code) est indépendant de la machine.
- C'est un langage qui permet de développer des applications professionnelles de grande taille.
- Il emprunte sa syntaxe, en grande partie, aux langages C et C++.
- Il intègre une interface graphique de haut niveau.
- N'importe quel éditeur de texte peut être utilisé pour éditer un fichier source Java.
- Il est nécessaire de compiler la source pour le transformer en J-code ou byte-code Java qui sera exécuté par la machine virtuelle.
- Il est préférable de définir une classe par fichier. Le nom de la classe publique et le fichier qui la contient doivent être identiques (il faut respecter la casse).

- Pour être compilé, le programme doit être enregistré au format de caractères Unicode : une conversion automatique est faite par le JDK si nécessaire.

4. Installation sous Windows

Pour utiliser Java, il est nécessaire d'avoir :

- Un environnement d'exécution Java (Oracle JDK pour pouvoir exécuter du code).
- Installer JDK.
- Un éditeur Java (Netbeans).
- Démarrer Netbeans, et écrire une première application.

Créer un premier fichier : PremierProgramme.java

```
public class PremierProgramme
{

public static void  main(String[]args)
            {

System.out.println(« Bienvenue, Formation
JAVA!"");

            }
          }
```

La première ligne du programme doit être la déclaration de la classe

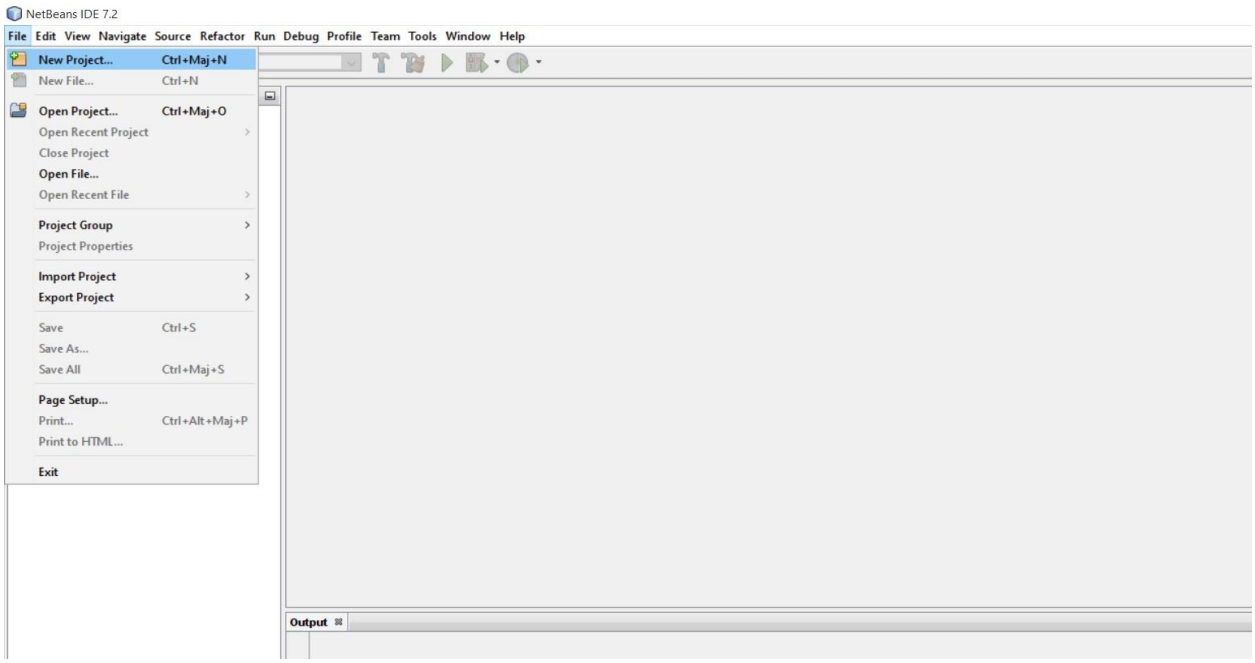
Tout programme doit contenir une méthode *main*

C'est le point d'entrée de toute application Java.

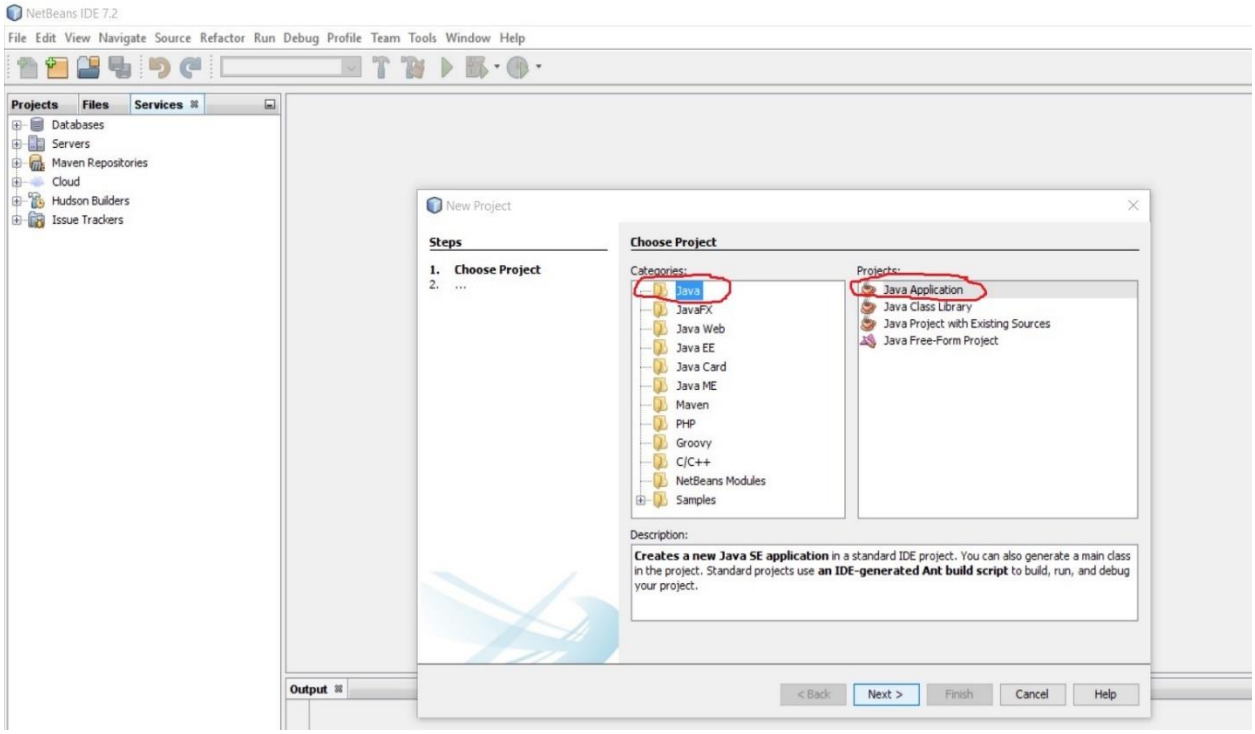
Affiche à l'écran "Bienvenue, Formation JAVA!"

Fermer les accolades

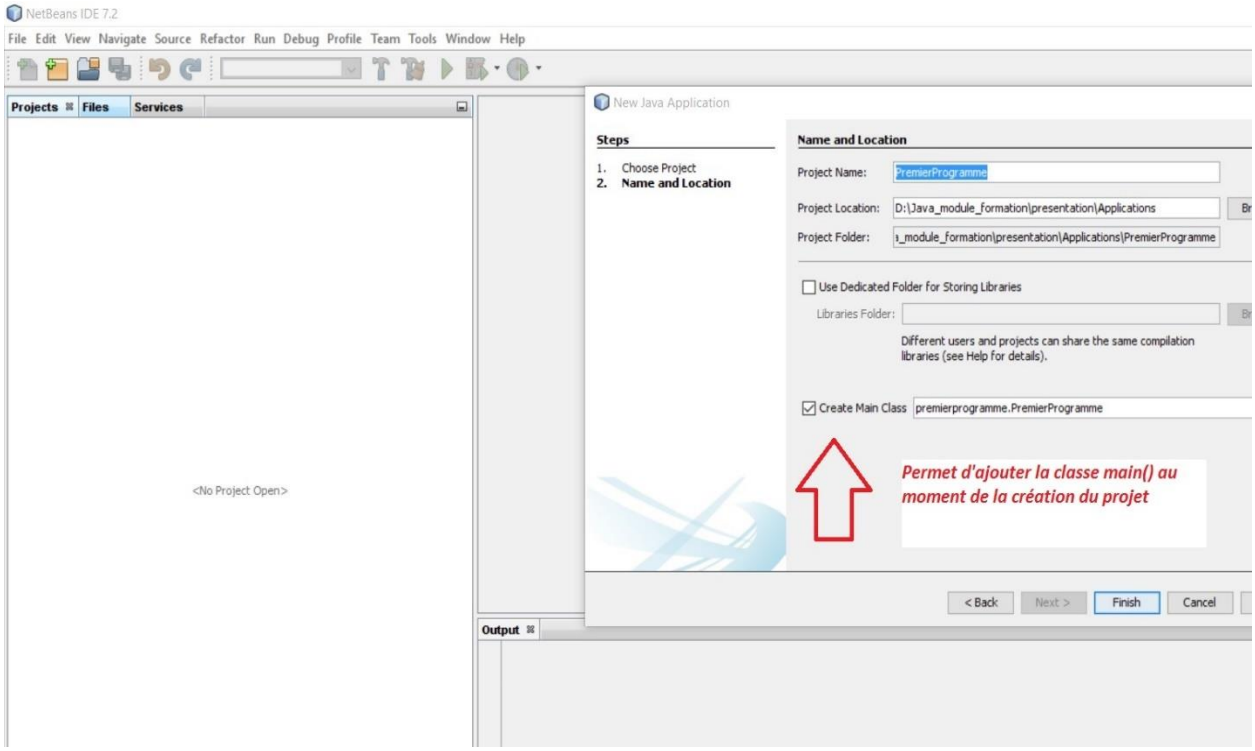
g. Créer un projet sous Netbeans – Étape 1



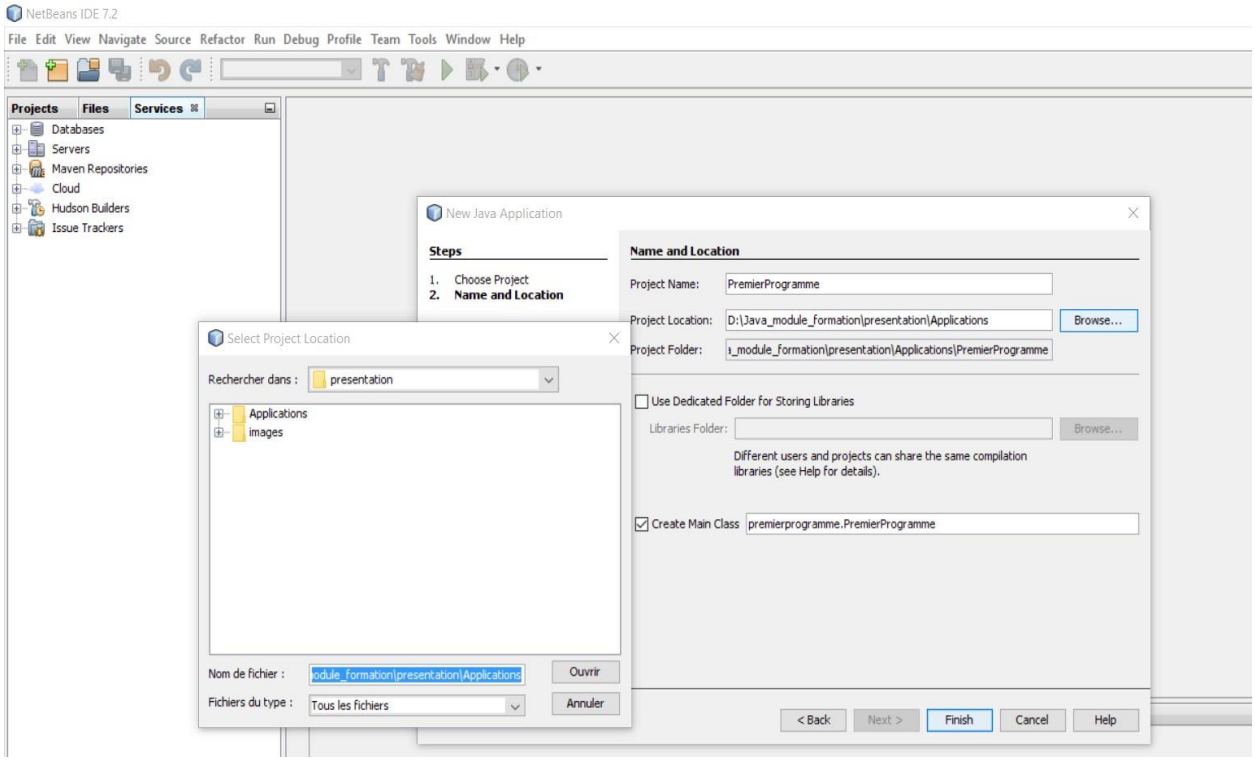
b. Créer un projet sous Netbeans – Étape 2



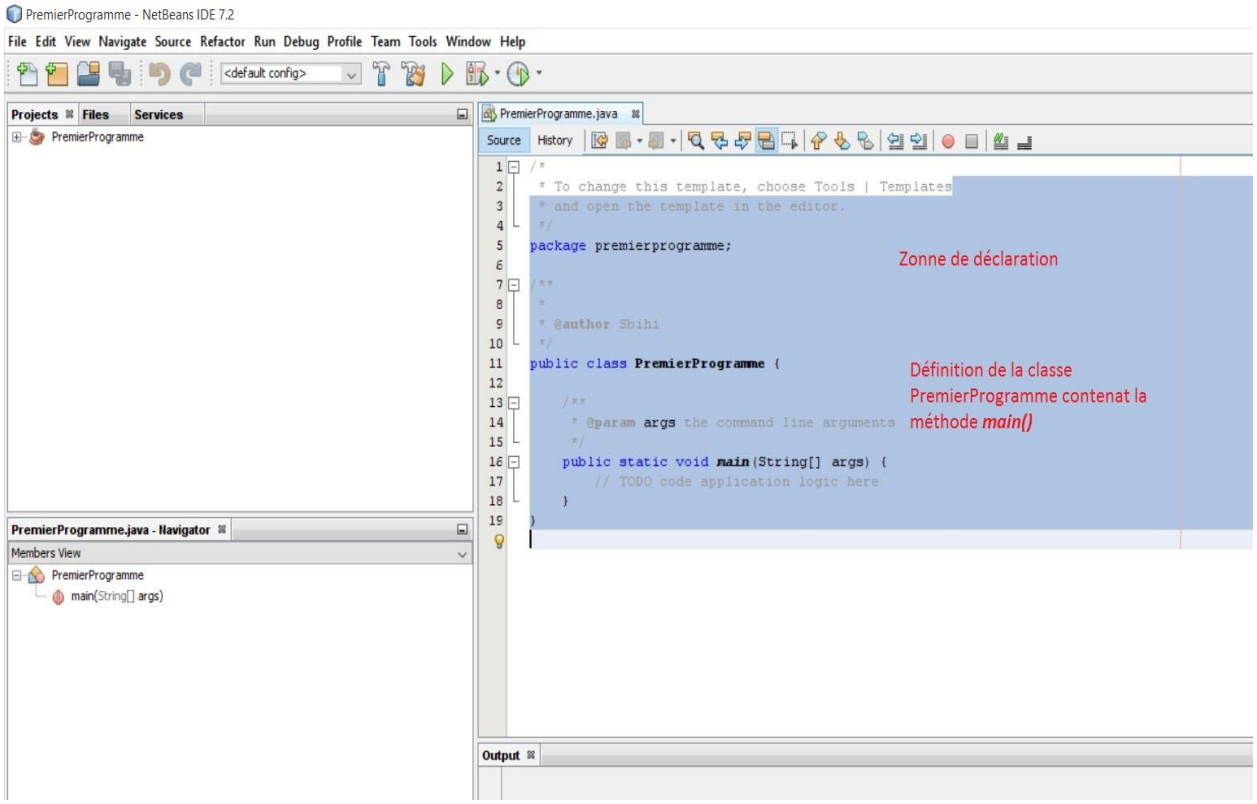
c. Créer un projet sous Netbeans – Étape 3



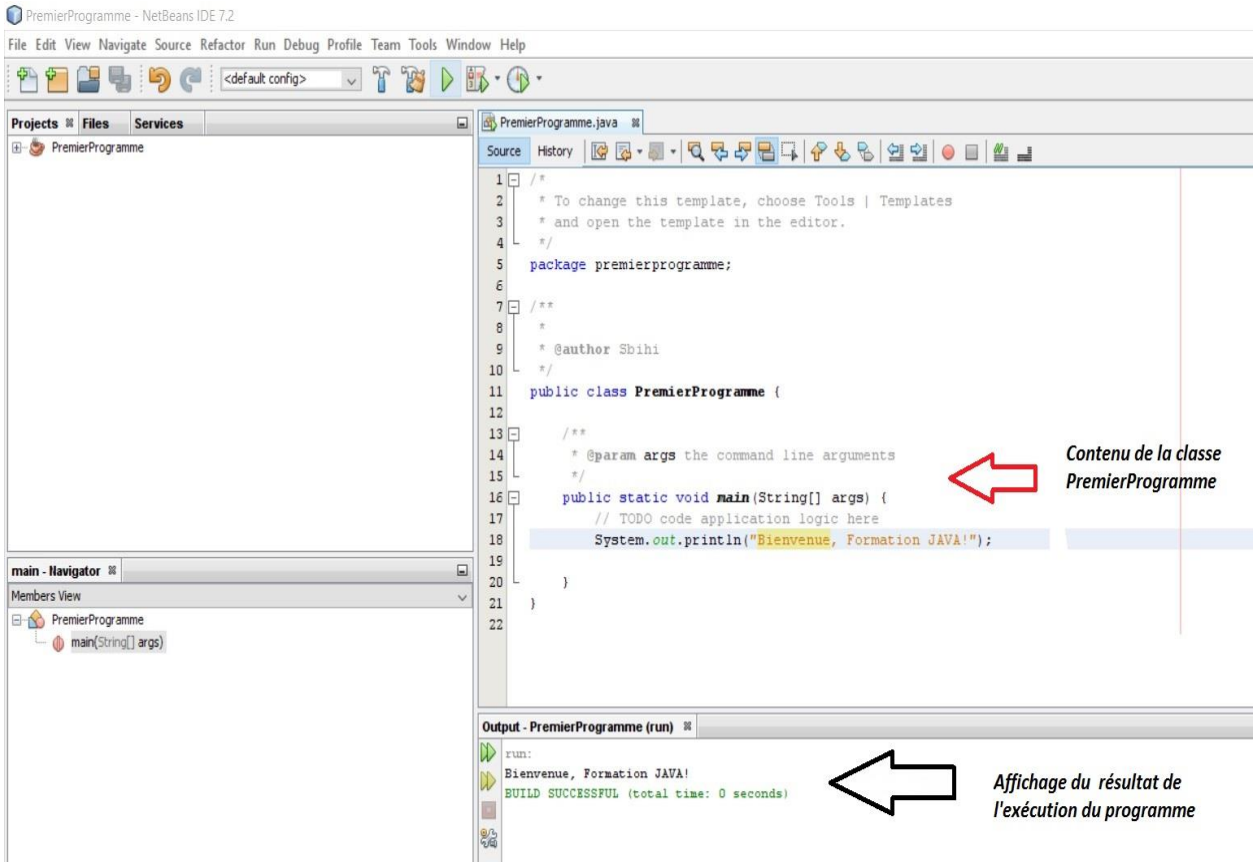
d. Créer un projet sous Netbeans – Étape 4



e. Créer un projet sous Netbeans – Étape 5



f. Créer un projet sous Netbeans – Étape 6



Exemple

❖ Les commentaires

Les commentaires se présentent sous deux formes :

- ✓ Ceux commençant par /* et se terminant par */
- ✓ Ceux commençant par // et se terminant à la fin de la ligne.

❖ Les blocs

Les blocs sont délimités par les accolades **{** et **}** (Begin et End, en Pascal). Ces blocs peuvent correspondre à une classe, une fonction ou un groupe d'instructions.

❖ La notion de classe

La **classe** est l'élément fondamental contenant les éléments de programmation. Une fonction, une instruction ou une déclaration de donnée, en programmation Java, appartiennent nécessairement à une classe qui est elle-même stockée dans un fichier. Une classe est constituée de données et de code opérant sur ces données.

Les données s'appellent **champs** ou **variables**, on trouve également les vocables de **données membres** ou **attributs**.

Le code est constitué de **méthodes** ou **fonctions**, appelées également **fonctions membres**.

Les champs et les méthodes sont parfois réunis sous le vocable de **membres**.

Dans l'exemple précédent, **main** est la seule méthode de la classe Ex1 qui, par ailleurs, n'a pas de champ (il s'agit de l'exemple le plus simple possible).

Une classe est **déclarée** par le mot-clé **class.**

❖ Fichier source

Un fichier source ne peut pas contenir plus d'une classe publique. On donne au fichier source, le nom de la classe publique qu'il contient et on le munit de l'extension .java. De plus, le nom du fichier source et celui de la classe doivent avoir la même casse (caractère minuscule/majuscule).

Un fichier peut contenir plusieurs classes à condition qu'une seule, au plus, soit publique.

Quand il y a plusieurs classes non publiques dans un fichier et qu'il y a une méthode main, c'est la classe qui contient main qui donne généralement son nom au fichier.

Exemple :

La classe **PremierProgramme** est stockée dans le fichier **PremierProgramme.java**.

NB :
Il est préférable de créer un fichier source par classe.

❖ La méthode main

En Java, il est possible de créer deux types de programmes : **les applications**, qui s'exécutent grâce à la machine virtuelle Java (c'est le cas de l'exemple précédent) et **les applets** destinées à être utilisées par un navigateur Web.

La méthode ***main*** est obligatoire pour créer une application, c'est elle qui sera exécutée en premier, elle peut, faire appel à d'autres méthodes. La méthode main correspond au programme principal en Pascal.

En Java, la méthode main possède un en-tête obligatoire, fixé une fois pour toutes :

```
public static void main (String[] args)
{
 déclaration de données et instructions
}
```

Les mots-clés **public**, **static** et **void** sont des **modificateurs** obligatoires pour main :

- ✓ **Public :** signifie que la méthode main sera connue partout en dehors de la classe Ex1,
- ✓ **Static :** signifie qu'elle ne pourra faire appel qu'aux champs statiques de la classe Ex1,
- ✓ **Void :** signifie qu'elle ne retourne pas de valeur (fonction vide).

Les Structures de données en JAVA

1. Les identificateurs

Un identificateur (*identifier*) permet de désigner une classe, une méthode, une variable...

En Java le nom de l'identificateur doit respecter les règles suivantes :

- ✓ Interdiction d'utiliser les mots-clés
- ✓ Ne peut pas contenir :
  - D'espaces
  - Des caractères internationaux (accents, etc.) (techniquement possible mais déconseillé).
- ✓ Commence par :
  - Une lettre, Un « $ »
  - Un « _ » (underscore)
- ✓ Ne commence pas par :
  - Un chiffre
  - Un signe de ponctuation autre que « $ » ou « _ »

2. Variables

- ✓ Une variable est caractérisée par un nom et un contenu.
- ✓ Le contenu d'une variable correspond à un certain type.
- ✓ Une variable Java est conçue pour un type particulier de donnée.
- ✓ La déclaration d'une variable est comme suit:

```
1 <Type de la variable> <Nom de la variable> ;
```

a. Les types primitifs

Les types primitifs ont une taille fixe quel que soit la plate-forme d'exécution : c'est un des éléments qui permettent à java d'être indépendant de la plate-forme sur lequel le code s'exécute.

Les *types primitifs*, sont des nombres entiers, des nombres réels, des booléens ou encore des caractères.

Pour utiliser une variable il faut :

- ✓ Déclaration
- ✓ Initialisation

➔ Peut se faire en même temps que la déclaration

Nom	Taille (octets)	Valeurs possibles
char	2	0 à 65535
byte	1	--128 à 127
short	2	-- 32 768 à 32 767
int	4	--2148483648 à 2147483647
long	8	--/+ 9,2 milliard de milliard
float	4	--/+ $3,4*10^{38}$
double	8	--/+ $1,7*10^{308}$
boolean	1	true ou false

b. Déclaration et initialisation d'une variable

Déclaration : Type nom_variable;

Initialisation : Nom_variable= valeur;

Déclaration et initialisation : Type nom_variable=valeur;

Exemples :

```
int  x=12;
short     x= 32;
long        x= 200L;
byte     byte x=012;
double double x=23.2323;
float     x= 23.233F;
Char caractere; Caractere ='H';
boolean a=true;
```

Il existe des opérateurs qui permettent de simplifier l'écriture d'une opération d'affectation associée à un opérateur mathématique :

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10
+=	a+=10	équivalent à : a = a + 10
-=	a-=	équivalent à : a = a - 10
*=	a*=	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<<=	a<<=10	équivalent à : a = a << 10 a est complété par des zéros à droite
>>=	a>>=10	équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

- **Les comparaisons**

Java propose des opérateurs pour toutes les comparaisons :

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<=	a <= 10	inférieur ou égal
==	a == 10	Egalité
!=	a != 10	diffèrent de
&	a & b	ET binaire
^	a ^ b	OU exclusif binaire
|	a | b	OU binaire
&&	a && b	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
||	a || b	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
? :	a ? b : c	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

❖ Les opérations arithmétiques

Les opérateurs arithmétiques se notent + (addition), - (soustraction), * (multiplication), / (division) et % (reste de la division). Ils peuvent se combiner à l'opérateur d'affectation.

***Exemple* :** nombre += 10 ;

❖ Les opérations logiques

Opérateur	Dénomination	Effet	Syntaxe
||	OU logique	Retourne *true* si au moins une des deux conditions vaut *true* (ou *false* sinon)	condition1 || condition2
&&	ET logique	Retourne *true* si les deux conditions valent *true*(ou *false* sinon)	condition1 && condition2
!	NON logique	Retourne *true* si la variable vaut *false*, et *false* si elle vaut *true*	!condition

c. Type objets

❖ Chaine de caractères

La déclaration d'une variable "objet" ne crée pas d'emplacement en mémoire pour stocker l'objet mais crée un emplacement pour stocker une *référence* vers un objet.

En Java, les chaînes de caractères sont des objets. Ce sont des instances de la classe *String*.

Exemple :

```
//Première méthode de déclaration
String ch1;
ch1="Initiation au langage JAVA";
//Deixième Méthode de déclaration
String ch2= new String();
ch2="Initiation au langage JAVA";
//Troisième Méthode de déclaration
String ch3= "Initiation au langage JAVA";
//Quaterième Méthode de déclaration
String ch4= new String("Initiation au langage JAVA");
```

Ils sont dérivés de la classe **Object** : il faut utiliser des méthodes pour y accéder dont font partie des messages de **Object** tels que **equals()** ou **getClass()**. Le premier élément possède l'indice 0.

- ✓ **La déclaration des tableaux**

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

Exemple :

```
int tableau[] = new int[50]; // déclaration et allocation
OU
int[] tableau = new int[50]; OU int tab[]; // déclaration tab
= new int[50]; //allocation
float  tableau[][] = new float[10][10];
```

La taille du tableau de la seconde dimension peut ne pas être identique pour chaque occurrence.

Exemple :

```
int dim1[][] = new
int[3][]; dim1[0] = new
int[4]; dim1[1] = new
int[9]; dim1[2] = new
int[2];
```

Chaque élément du tableau est initialisé selon son type par l'instruction new : 0 pour les numériques, \0 pour les caractères, false pour les booléens et nil pour les chaînes de caractères et les autres objets.

Exemple :

```
int tableau[5] =
{10,20,30,40,50}; int
tableau[3][2] =
{{5,1},{6,2},{7,3}};
```

La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

- ❖ **Tableaux dynamiques : ArrayList**

Un tableau dynamique, est une collection de données, dont le nombre peut changer au cours du déroulement du programme (ajoute ou retire des éléments au/du tableau).

Les tableaux dynamiques sont définis en Java par le biais du type ArrayList. On importe les définitions associées à l'aide de la directive suivante : import java.util.ArrayList;

- ✓ Pour ajouter une donnée au tableau : monTableau.add(3);
- ✓ Pour récupérer un élément en position k : Int elem = monTableau.get(k);

Une variable correspondant à un tableau dynamique se déclare de la façon suivante :

ArrayList type identificateur; où identificateur est le nom du tableau et type correspond au type des éléments du tableau. Le type des éléments doit nécessairement correspondre à un type évolué.

Exemple :

```
ArrayList  <String> tableau;
```

Un tableau dynamique initialement vide (sans aucun élément) s'initialise comme suit :

```
ArrayList identificateur = new ArrayList();
```

Où identificateur est le nom du tableau et type correspond au type des éléments du tableau.

3. Les opérations d'entrées/sorties

a. Écriture

- ✓ Affichage du contenu d'une variable :

```
System.out.print(nom_variable);
```

- ✓ Affichage de valeur d'une expression:

```
System.out.print(" la surface du carré de coté:"  +n+"   est  :" +n*n);
```

b. Lecture à partir du clavier

Importation de la classe ***Scanner,***

Création d'un objet *Scanner*:

Scanner Lect = new Scanner (System.in); *Lect* est une variable utilisée pour introduire des valeurs au clavier.

Le clavier est représenté par ***system.in***

Exemple :

Scanner Lect = new Scanner (System.in);

String s = *Lect.nextLine();* // ***nextLine*** pour lire la ligne suivante.

System.out.println(" La ligne tapée est:" +s);

4. Les structures de contrôle conditionnelles

- ***IF – THEN***

```
if (condition)
  {          //instructions;
  }
```

- ***IF – – ELSE***

```
if (condition)
  {   //instructions
  }
  else
    {   //instructions
      }
```

- ***SWITCH – CASE***

```
switch(selecteur)
{         case 1 : //instructions; A réaliser si elle vaut 1
          case 2 : //instructions; break; Si elle vaut 2
          default : //instructions; Dans tous les autres cas
}
}
```

5. Les structures de contrôle itératives

a. Boucle For

Syntaxe:

```
for (initialisation; condition; mise à jour de valeurs)
{
        // instructions
}
```

Exemple:

```
for (int i=0 ; i<10 ; i++)
{
  System.out.println(« La valeur affichée est: " + i);
}
```

Syntaxe:

```
while (CONDITION)
{
        // Instructions
}
```

Exemple:

```
int i = 0;
while(i<10) {
  System.out.println(« La valeur de i est : " + i);
  i++;
}
```

b. Boucle Do While

Syntaxe:

```
 do
{
        // Instructions
} while(CONDITION)
```

6. Les débranchements

- **break** : permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot.
- **continue** : s'utilise dans une boucle pour passer directement à l'itération suivante. **Break** et **continue** peuvent s'exécuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break. Une étiquette est un nom suivi de " ;" qui définit le début d'une instruction.

```
for (int i=0; i<10 ;i++)
{
        if (i==5) continue; // Si i=5, on passe à 6  sans afficher
        if (i==7) break;          // Si i=7, on sort de la boucle
        System.out.println(« La valeur de i est : " + i);
}
```

7. Les exceptions en Java

La cause de mauvais fonctionnement d'un programme peut être de deux types d'erreurs : une erreur de programmation ou une erreur pendant l'exécution du programme. Si le programmeur commet une erreur de programmation, le programme ne se compilera généralement pas, mais il existe des erreurs de programmation qui surviennent après la compilation, à l'appel de la machine virtuelle par **java.exe.** Toutes ces erreurs doivent être corrigées par le programmeur. Dans les deux cas précédents, le programme ne démarre pas. Le second type d'erreur survient pendant l'exécution du programme. Ce sont en quelques sortes des "bugs". On distingue deux catégories :

- Les erreurs de machine virtuelle (on ne peut rien y faire),
- Les **exceptions qu'on peut gérer**. Ces exceptions peuvent se produire dans plusieurs cas, elles ont cependant plus de chance de survenir lors d'un transfert de données **try…catch** *try…catch* peut être expliqué comme : « essaye ce bout de code (try), si une exception survient, attrape-la (catch) et exécute le code de remplacement (s'il y en a un) »

```
try {
// Code susceptible de produire des exceptions
}
catch (Exception e){
// Code de remplacement, se limite souvent à
System.out.println("Une
// erreur : " + e) ; }
```

8. Fonctions/méthodes

Une fonction est un objet caractérisé par :

- ✓ Un corps
- ✓ Un nom
- ✓ Des paramètres
- ✓ Un type et une valeur de retour
- ✓ L'utilisation de la fonction dans une autre partie du programme se nomme un appel de la fonction.

❖ Portée d'une variable

- ✓ *Variables locales :* Elles ne sont accessibles qu'à l'intérieur du bloc.
- ✓ *Variables globales* : Elles sont accessibles dans toute la classe.

❖ Syntaxe

```
type nom ( liste de paramètres )
{
    instructions du corps de la méthode;
    return expression;
}
```

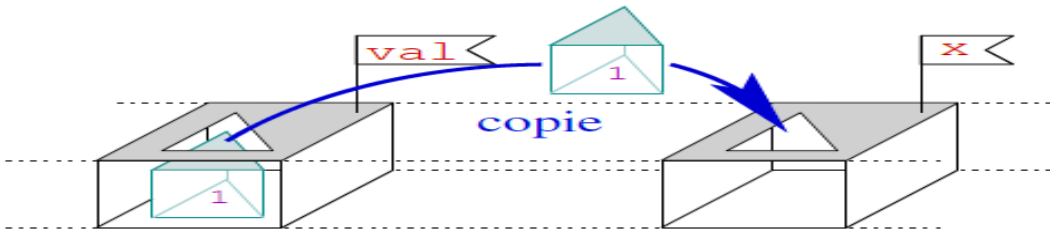
❖ Exemple

```
public class Rectangle {
 private static Scanner clavier = new Scanner(System.in);
public static void main (String[] args)
{
    double larg =0.0;
    double longu =0.0;
    System.out.println("Entrer la largeur");
    larg=clavier.nextDouble();
     System.out.println("Entrer la longueur");
    longu=clavier.nextDouble();
    System.out.println("La surface est  : "+surface(larg,longu));
    }
static double surface(double a,double b)
{

return (a*b);
}
}
```

❖ Passage de paramètres

En Java, il n'existe que le passage par valeur : une méthode utilise toujours une copie de la valeur qui lui est passée en paramètre.



Une procédure est une méthode qui ne fournit aucun résultat. On utilise alors le type particulier *void* comme type de retour.

```
static void afficheRacine(double a)
{
  if (a < 0.0) {
    return; /* Grâce à ce return, on quitte la fonction avant *
             * de calculer sqrt(a) si a est négatif.          */
  }
  System.out.println(Math.sqrt(a));
  // il n'est pas nécessaire de mettre un return ici
}
```

9. La surcharge des méthodes

La surcharge d'une méthode ou d'un constructeur permet de définir plusieurs fois une même méthode/constructeur avec des arguments différents. Ce mécanisme, appelé surcharge des méthodes, est très utile pour écrire des méthodes « sensibles » au type de leurs arguments, c'est-à-dire des méthodes correspondant à des traitements de même nature mais s'appliquant à des entités de types différents.

❖ Conventions

En Java il y a des conventions à respecter :

- ✓ Les noms de classes doivent commencer par une majuscule.
- ✓ Les noms de variables doivent commencer par une minuscule.
- ✓ Si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation.
- ✓ Pas d'accents.

Gestion des entrées sorties en Java

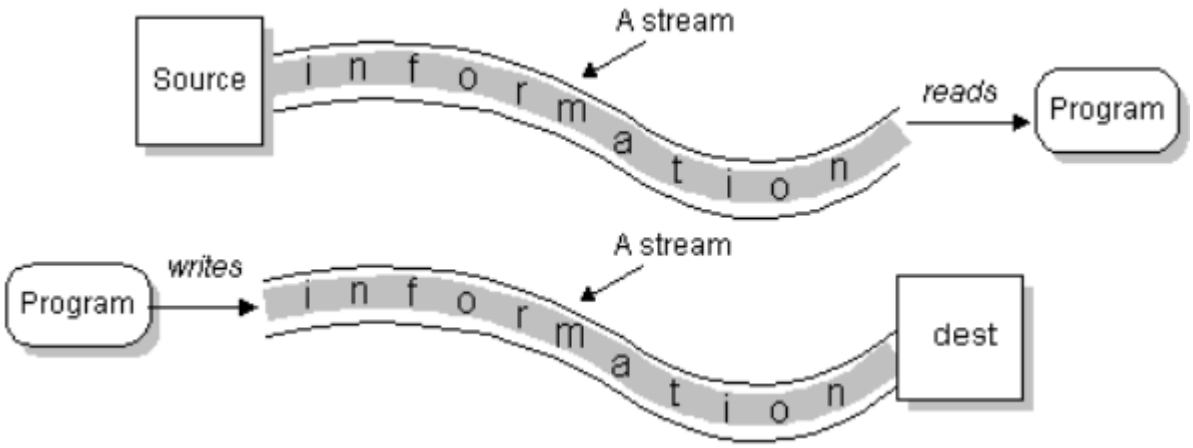
1. Introduction

Les opérations d'entrées-sorties concernant la lecture et l'écriture de données à travers différents types de flux permettent à un programme de communiquer avec certains périphériques.
Les opérations d'entrées-sorties de base sont gérées par les classes du package ***java.io***

2. Flux (Stream)

La notion de flux (Stream) caractérise un chemin (une voie) de communication entre une source d'information et sa destination. L'accès aux informations s'effectue de manière séquentielle.
Les librairies Java distinguent deux genres de flux :

- Les flux binaires (Byte Stream) qui peuvent représenter des données quelconques (nombres, données structurées, programmes, sons, images, …)
- Les flux de caractères (Character Stream) qui représentent des textes (chaînes de caractères) au format Unicod



La communication avec un flux comprend trois phases :

- ✓ L'ouverture du flux, en entrée (pour la lecture) ou en sortie (pour l'écriture)
- ✓ La répétition de la lecture ou de l'écriture des données (généralement des bytes ou des caractères)
- ✓ La fermeture du flux

3. Lecture et écriture des flux

- ✓ Pour lire un fichier texte, il faut ouvrir un flux d'entrée
  - La classe FileReader constitue une des couches de base et comporte les méthodes élémentaires pour lire le flux depuis le système de fichiers
- ✓ Pour écrire un fichier texte, il faut ouvrir un flux de sortie
  - La classe FileWriter constitue une des couches de base et comporte les méthodes élémentaires pour écrire le flux sur le système de fichiers

4. Écriture d'un fichier texte

Pour écrire séquentiellement dans un fichier texte on peut utiliser la classe PrintWriter de la manière suivante :

- ☐ **Ouverture**:

```
String filename = "C:\Formation\Ecrirure.txt";
PrintWriter p = new PrintWriter( new BufferedWriter( new FileWriter(filename)));
```

- ☐ ***Écriture*** *:*

```
p.print(« Bonjour!");
p.println(" Le résultat est : " + 3*2);
p.println("Fin du fichier " + filename);
```

- ☐ ***Fermeture***

```
p.close();
```

En plus des méthodes print() et println(), la classe PrintWriter comporte également la méthode printf(), qui permet de formater le texte écrit dans le fichier:*printf(String format, Object... args)*

Exemple:

```
p.printf("%.2f", resultScore);
p.printf("Nombre %d  Prix %d Total %d\n", p1,   pri,    pri*p1);
p.printf("%5d%8.3f %n", i, v);
```

5. Lecture d'un fichier texte

Pour lire séquentiellement le contenu d'un fichier texte on peut utiliser la classe BufferedReader de la manière suivante :

- ***Ouverture***

```
String filename = "C:\Formation\Test.txt";
BufferedReader r = new        BufferedReader( new FileReader(filename));
```

- ***Lecture* (ligne par ligne)**

```
String s;
s = r.readLine();
 while (s != null)
 {   //--- Tant qu'il y a  encore des lignes...
System.out.println(s); s =r.readLine();
}
```

- ***Fermeture***

```
r.close();
```

6. Entrées/Sorties standard (console)

- ✓ Les flux suivants sont prédéfinis dans la classe System.
- ✓ System.in : Entrée standard (lecture du clavier) [InputStream]
- ✓ System.out : Sortie standard (affichage à l'écran) [PrintStream]
- ✓ System.err : Sortie des messages d'erreur [PrintStream] (souvent identique à out par défaut)

Écriture sur la console :

- ✓ System.out.print("Résultats : ");
- ✓ System.out.println(5*4 + ", " + 6);
- ✓ System.err.println("Erreur : Nom de fichier incorrect");

7. Les flux binaires de sortie et d'entrée

❖ Les flux binaires de sortie

Les FBS permettent d'écrire des suites d'octets, de valeurs de type primitif ou d'objets dans des fichiers, en mémoire, à travers de sockets ou depuis un périphérique de communication.

❖ Les flux binaires d'entrée

Les FBE permettent de lire des suites d'octets, de valeurs de type primitif ou d'objets dans des fichiers, en mémoire, à travers de sockets ou depuis un périphérique de communication.

Les Concepts de base de la programmation orientée objet en JAVA

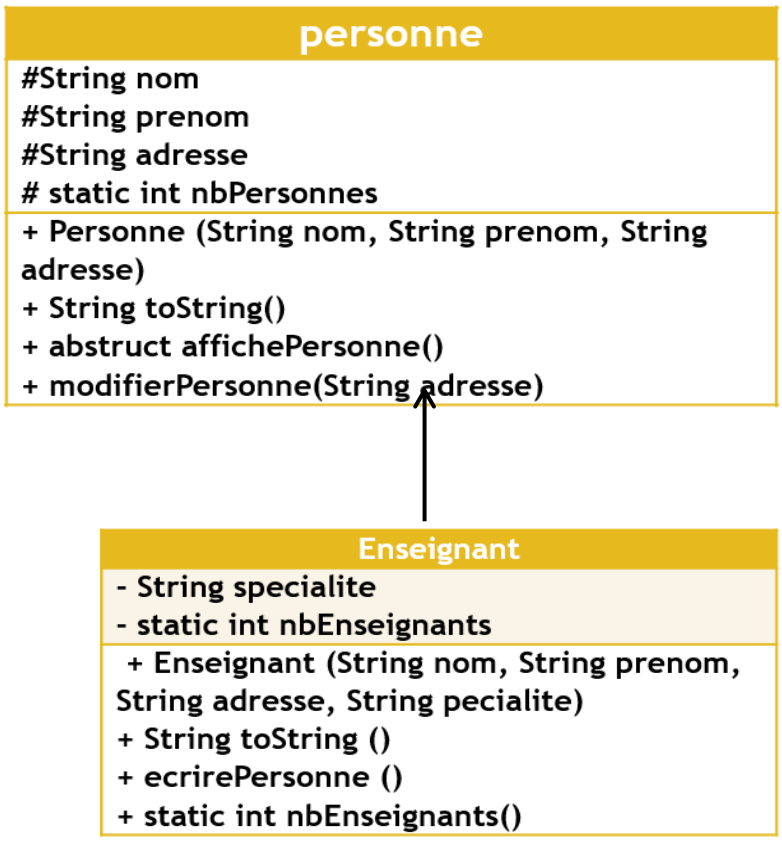
I. Héritage

L'héritage est un mécanisme qui facilite la réutilisation du code et la gestion de son évolution. Grâce à l'héritage, les objets d'une classe ont accès aux données et aux méthodes de la classe parent et peuvent les étendre. Il permet de créer des classes plus spécialisées, appelées sous-classes, à partir de classes plus générales déjà existantes, appelées super-classes.
Les sous-classes peuvent redéfinir les variables et les méthodes héritées. Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent. Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge. L'héritage successif de classes permet de définir une hiérarchie de classes qui se compose de super-classes et de sous-classes.

a. Règles de l'héritage

- ✓ **super() :** Indique que le constructeur en cours d'exécution passe la main au constructeur de la classe parente pour commencer à initialiser les attributs définis dans cette dernière. Ensuite le premier constructeur continuera son exécution.
- ✓ **this() :** indique que le constructeur considéré passe la main à un autre constructeur de la classe considérée.
- ✓ Si la première instruction d'un constructeur ne commence pas par le mot clé super le constructeur par défaut de la classe mère est appelé. Dans ce cas, l'appel à super() est implicite.
- ✓ L'appel de constructeur de la classe mère peut uniquement se faire qu'en première instruction d'une définition de constructeur. Une conséquence évidente est qu'on ne peut utiliser qu'un seul appel au constructeur de la classe mère.
- ✓ En Java le mot réservé de l'héritage est **extends**

Exemple :



la classe personne:

```
abstract class personne {

protected String nom;
protected String prenom;
protected String adresse;
protected static  int nbPersonnes = 0;  // nombre de Personne
// constructeur de Personne
personne(String nom, String prenom, String adresse) {
this.nom = nom; // voir this page 38
this.prenom = prenom;
this.adresse = adresse;
nbPersonnes++;
}
public String toString() {
return nom + " " + prenom + " " + adresse;
}
abstract void ecrirePersonne();
static void nbPersonnes () {
System.out.println ("\nNombre d'employés : " + nbPersonnes +"\nNombre de secrétaires : " + Secretaire.nbSecretaires()
 +"\nNombre d'enseignants : " + Enseignant.nbEnseignants() +"\nNombre d'étudiants : " + Etudiant.nbEtudiants());
}
void modifierPersonne(String adresse) {
this.adresse = adresse;
ecrirePersonne(); // de la classe dérivée de l'objet
}
}
```

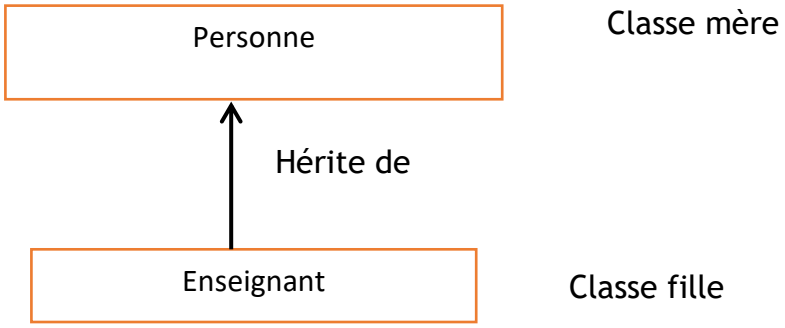
la classe Enseignant :

```
class Enseignant extends personne{ // héritage de classe
private String specialite;
private static  int nbEnseignants = 0; // nombre d'Enseignant
Enseignant(String nom, String prenom, String adresse, String specialite) {
super(nom, prenom, adresse); // appel du constructeur Personne
this.specialite = specialite;
nbEnseignants++;
}
public String toString () {
return super.toString()+ "\n spécialité : " + specialite;
}
void ecrirePersonne() {
System.out.println ("Enseignant : " + toString());
}
static int nbEnseignants() { return nbEnseignants; }
} // Enseignant
```

❖ **Définition d'une sous-classe**

```
class NomSousClasse extends NomSuperClasse
{
 /* Déclaration des attributs et méthodes
    spécifiques à la sous-classe */
}
```

Exemple :



```
class Enseignant extends personne{ // héritage de classe
private String specialite;
private static  int nbEnseignants = 0; // nombre d'Enseignant
Enseignant(String nom, String prenom, String adresse, String specialite) {
super(nom, prenom, adresse); // appel du constructeur Personne
this.specialite = specialite;
nbEnseignants++;
}
```

❖ Constructeurs et héritage

Lors de l'instanciation d'une sous-classe, il faut initialiser :

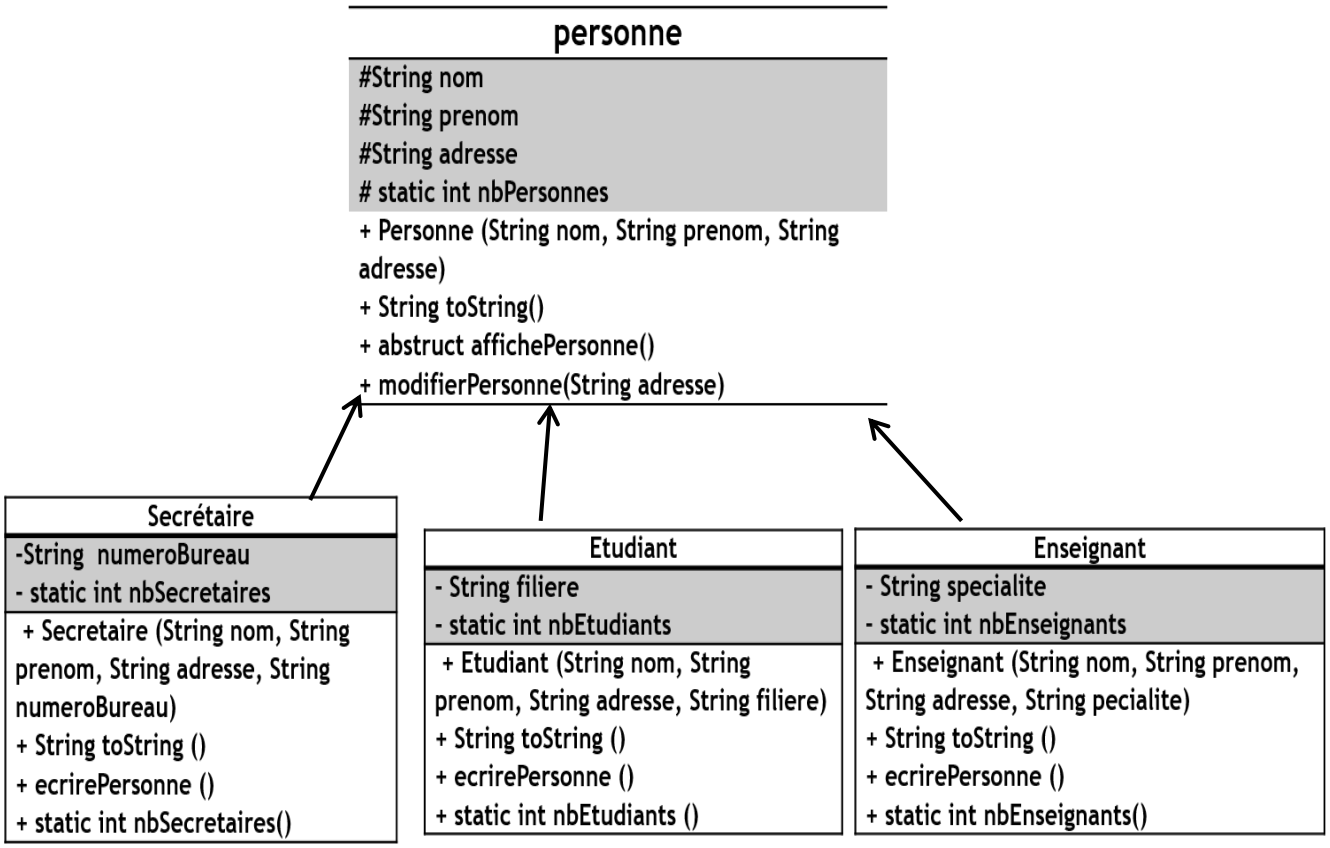
- ✓ Les attributs propres à la sous-classe
- ✓ Les attributs hérités des super-classes

L'invocation du constructeur de la super-classe se fait au début du corps du constructeur au moyen du mot réservé super.

Syntaxe :

```
SousClasse(liste de paramètres)
{
      /* Arguments : liste d'arguments attendus par
       * un des constructeurs de la super-classe de SousClasse
       */
     super(Arguments);
     // initialisation des attributs de SousClasse ici
}
```

Exercice : Soit le diagramme de classe suivant



En utilisant le diagramme décrit ci-dessus, développez les méthodes nécessaires pour obtenir le résultat suivant :

```
Output - Initiation_Java (run)
run:
Secrétaire : Abidi Souhir rue des Palmiers
 N° bureau : A123
Enseignant : Inoubli  Maha BP 256 Tunis 1100
 spécialité : Maths
Etudiant : Sbihi AMEN ALLAH Rue de pris
 Diplome en cours : informatique industriel
Etudiant : Guizani Akrem rue des palmiers Jendouba
 Diplome en cours : sécurité informatique

Nombre d'employés : 4
Nombre de secrétaires : 1
Nombre d'enseignants : 1
Nombre d'étudiants : 2


Après modification :
Secrétaire : Abidi Souhir rue des sorbiers
 N° bureau : A123
Enseignant : Inoubli  Maha rue des lilas
 spécialité : Maths
BUILD SUCCESSFUL (total time: 0 seconds)
```

Correction

```
 abstract class personne {

protected String nom;
protected String prenom;
protected String adresse;
protected static  int nbPersonnes = 0;  // nombre de Personne
// constructeur de Personne
personne(String nom, String prenom, String adresse) {
this.nom = nom; // voir this page 38
this.prenom = prenom;
this.adresse = adresse;
nbPersonnes++;
}
public String toString() {
return nom + " " + prenom + " " + adresse;
}
abstract void ecrirePersonne();
static void nbPersonnes () {
System.out.println ("\nNombre d'employés : " + nbPersonnes +"\nNombre de secrétaires : " + Secretaire.nbSecretaires()
 +"\nNombre d'enseignants : " + Enseignant.nbEnseignants() +"\nNombre d'étudiants : " + Etudiant.nbEtudiants());
}
void modifierPersonne(String adresse) {
this.adresse = adresse;
ecrirePersonne(); // de la classe dérivée de l'objet
}
}
```

```java
class Secretaire extends personne{ // héritage de classe
private String numeroBureau;
private static int nbSecretaires = 0; // nombre de Secretaire
Secretaire(String nom, String prenom, String adresse, String numeroBureau) {
// le constructeur de la super-classe
super(nom, prenom, adresse); // appel du constructeur Personne
this.numeroBureau = numeroBureau;
nbSecretaires++;
}
public String toString () {
// super.toString() : toString() de la super-classe
return super.toString()+ "\n N° bureau : " + numeroBureau;
}
void ecrirePersonne() {
System.out.println ("Secrétaire : " + toString());
}
static int nbSecretaires() { return nbSecretaires; }
} // Secretaire
```

```java
class Etudiant extends personne{ // héritage de classe
private String filiere;
private static  int nbEtudiants = 0; // nombre d'Etudiant
Etudiant(String nom, String prenom, String adresse, String filiere) {
super(nom, prenom, adresse); // appel du constructeur Personne
this.filiere = filiere;
nbEtudiants++;
}
public String toString () {
return super.toString()+ "\n Diplome en cours : "
+ filiere;
}
void ecrirePersonne() {
System.out.println ("Etudiant : " + toString());
}
public String diplomeEnCours() {
return filiere;
}
static int nbEtudiants() { return nbEtudiants; }

}
```

```
*/
abstract class personne {
public static void main(String[] args) {
Secretaire s1 = new Secretaire("Abidi", "Souhir", "rue des Palmiers ","A123");
s1.ecrirePersonne();
Enseignant p1 = new Enseignant("Inoubli ", "Maha","BP 256 Tunis 1100","Maths");
p1.ecrirePersonne();
Etudiant e1 = new Etudiant("Sbihi", "AMEN ALLAH","Rue de pris ","informatique industriel ");
e1.ecrirePersonne();
Etudiant e2 = new Etudiant("Guizani", "Akrem","rue des palmiers Jendouba", "sécurité informatique");
e2.ecrirePersonne();
personne.nbPersonnes(); // méthode static
System.out.println ("\n\nAprès modification :");
s1.modifierPersonne ("rue des sorbiers");
p1.modifierPersonne ("rue des lilas");
} // main
```

II. Polymorphisme

Le polymorphisme est la capacité pour une entité de prendre plusieurs formes (l'eau qui se trouve à l'état solide, liquide ou gazeux). C'est la faculté attribuée à un objet d'être une instance de plusieurs classes.

Un même code peut s'adapter automatiquement au type des données auxquelles il s'applique.

Un langage orienté objet est dit polymorphique, s'il offre la possibilité de pouvoir percevoir un objet en tant qu'instance de classes variées, selon les besoins.

Le polymorphisme en Java, se traduit par :

- ✓ La compatibilité entre un type dérivé et un type ascendant.
- ✓ La ligature dynamique des méthodes, dans le sens où il permet d'obtenir le comportement adapté à chaque type d'objet.
- ✓ Un objet a comme type non seulement sa classe mais aussi n'importe quelle classe dérivée.

Exemple :

La méthode modifierPersonne modifie l'adresse d'une Personne quelle que soit sa catégorie. Elle se trouve dans la classe commune Personne. Elle peut être appelée pour un objet Secrétaire, Enseignant ou Etudiant.

```
void modifierPersonne(String adresse) {
this. adresse = adresse;
ecrirePersonne();
}
```

Si chaque sous-classe redéfinit la méthode, celle de la super-classe n'est jamais exécutée et peut être déclarée abstraite. Elle sert de prototype lors de la compilation de modifierPersonne() de la classe Personne. Si la méthode est déclarée abstraite, les sous-classes doivent la redéfinir sinon, il y a un message d'erreur à la compilation.

Les instructions suivantes appellent la méthode modifierPersonne() de Personne pour deux objets différents dont la classe est dérivée de Personne.

```
System.out.println ("\n\nAprès modification :");
s1.modifierPersonne ("rue de La Liberté");
p1.modifierPersonne ("rue des Nelson Mondela");
} // main
```

Pour l'objet s1, la méthode modifierPersonne() appelle la méthode ecrirePersonne() de Secrétaire. Pour l'objet p1, la méthode modifierPersonne() appelle la méthode ecrirePersonne() de Enseignant. Donc, le même appel de la méthode ecrirePersonne() de modifierPersonne() déclenche des méthodes redéfinies différentes (portant le même nom) de différentes classes dérivées, on parle alors de polymorphisme.

Le polymorphisme découle de la notion de redéfinition de méthodes entre une super-classe et une sous-classe.

III. Interface

Une interface est constituée d'un ensemble de déclarations de propriétés ou de méthodes sans implantation. Les interfaces sont toujours abstract, sans qu'il soit nécessaire de l'indiquer explicitement. De la même façon, il n'est pas nécessaire de déclarer leurs méthodes abstract.

Les interfaces obéissent à certaines règles supplémentaires.

- ✓ Elles ne peuvent contenir que des variables **static** et **final**.
- ✓ Elles peuvent être étendues comme les autres classes, avec une différence majeure : une interface peut dériver de plusieurs autres interfaces. En revanche, une classe ne peut pas dériver uniquement d'une ou de plusieurs interfaces. Une classe dérive toujours d'une autre classe.

Une classe qui décide d'implémenter une interface s'engage à fournir une implémentation de toutes les méthodes définies dans l'interface.

a. Définition

```
Interface nom_interface {
constantes ou méthodes abstraite
}
Exemple:
public interface Motorise {
public void faisLePlein() ;
}
```

Attribution d'une interface à une classe :

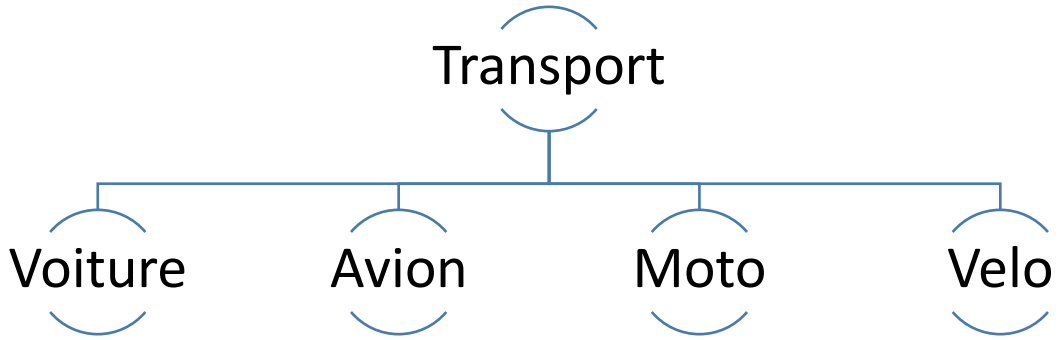
```
Syntaxe:
class nom_classe implements interface1, interface2,…, interfaceN
{
……
}
Exemple:
public interface Motorise {
public void faisLePlein() ;
}
public class Voiture extends Transport implements Motorise {
public void conduit() {}
public void faisLePlein() {}
}
```

Une classe peut implémenter plusieurs interfaces (mais étendre une seule classe). Séparer les interfaces par des virgules. On peut déclarer une hiérarchie d'interfaces : Mot-clé extends

La classe qui implémente une interface reçoit aussi le type des super-interfaces

Exemple :

Soit la hiérarchie de classes suivante :



Soit une classe station-service qui possède une méthode fairePlein(…) ayant comme paramètre le moyen de transport.

```
Soit la définition des classes suivantes:
public class Transport
 {
        public void roule() ;
 }
 public class Voiture extends Transport
 {
    public void conduit() ;
  }
 public class Avion extends Transport
{
   public void vole() ;
}
 public class Moto extends Transport
{
  public void seFaufile() ;
}
 public class Velo extends Transport
{
  public void pedale() ;
 }
```

Soit la classe stationService possédant une méthode faireLePlein :

```
public class StationService
```

```
        {
                public void faireLePlein(Transport transport)
                        {
                                if (transport instanceof Velo)
                                { // ne pas faire le plein }
                                        else { // faire le plein }
                                                }
 }
```

Écrire l'interface Motorise :

```
public interface Motorise {
public void faisLePlein() ;
}
public class Transport {
public void roule() {}
}
public class Voiture extends Transport implements Motorise {
public void conduit() {}
public void faisLePlein() { //,,,,,}
}
public class Avion extends Transport implements Motorise {
public void vole() ;
public void faisLePlein() {//,,,,,,}
}
public class Moto extends Transport implements Motorise {
public void seFaufile() ;
public void faisLePlein() {//,,,,,,}
}
public class Velo extends Transport { // ne sait pas faire le plein public void pedale() ; }
```

Le nouveau code de la classe StationService est le suivant :

```
public class StationService
{
        public void faireLePlein(Motorise motorise)
        {
        motorise.faisLePlein() ;
        }
}
```

IV. Encapsulation

L’encapsulation est l'idée de protéger l'information contenue dans un objet et de ne proposer que des méthodes de manipulation de cet objet. L’encapsulation permet de définir deux niveaux de perception des objets :

- ✓ Niveau externe : partie visible (par les programmeurs-utilisateurs),
- ✓ Niveau interne : détails d’implémentation : corps

L’encapsulation est la possibilité de ne montrer de l’objet que ce qui est nécessaire à son utilisation.

L’encapsulation permet d’offrir aux utilisateurs d’une classe la liste des méthodes et éventuellement des attributs utilisables depuis l’extérieur. Cette liste de services exportables est appelée l’interface de la classe et elle est composée d’un ensemble des méthodes et d’attributs dits publics (Public).

Les méthodes et attributs réservés à l’implémentation des comportements internes à l’objet sont dits privés (Private). Leur utilisation est exclusivement réservée aux méthodes définies dans la classe courante.

L'objet est ainsi vu de l'extérieur comme une boîte noire ayant certaines propriétés et ayant un comportement spécifié.

Les avantages de l’encapsulation sont :

- ✓ Simplification de l’utilisation des objets,
- ✓ Meilleure robustesse du programme,
- ✓ Simplification de la maintenance globale de l’application.

V.Abstraction

L’abstraction est le processus qui consiste à représenter des objets qui appartiennent au monde réel dans le monde du programme que l’on écrit. L’abstraction consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l’on souhaite manipuler, et à les placer dans un modèle informatique convenable.

C’est le faite de se concentrer sur les caractéristiques importantes d’un objet selon le point de vue de l’observateur.

L’abstraction est un principe qui consiste à ignorer certains aspects d’un sujet qui ne sont pas importants pour le problème, dans le but de se concentrer sur ceux qui le sont.

Exemple : Rectangles

La notion d'objet rectangle n'est intéressante que si l'on peut lui associer des propriétés et/ou mécanismes généraux (valables pour l'ensemble des rectangles). Les notions de largeur et hauteur sont des propriétés générales des rectangles (attributs).

Le mécanisme permettant de calculer la surface d'un rectangle (surface = largeur × hauteur) est commun à tous les rectangles (méthodes).

❖ Classes abstraites

Une classe est définie comme abstraite avec le mot clé **abstract**. Les classes abstraites sont à utiliser lorsqu'une classe mère ne doit pas être instanciée.

❖ Méthodes abstraites

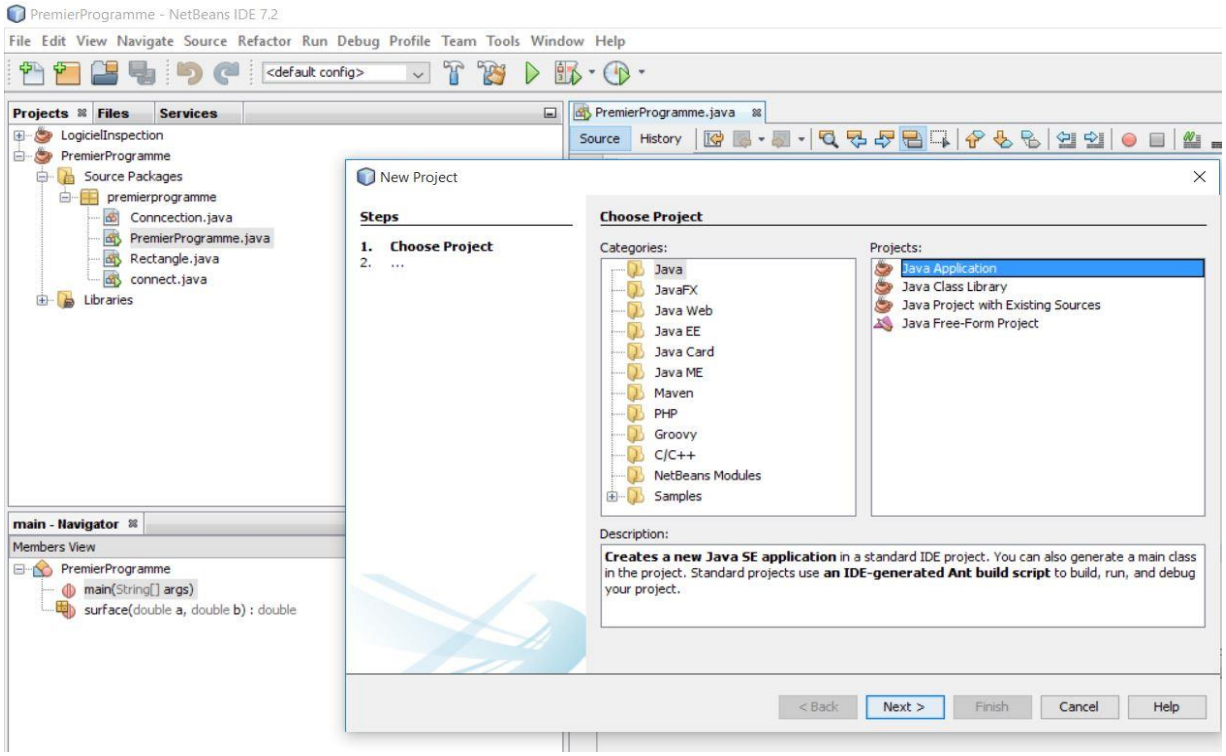
Une méthode abstraite, est incomplètement spécifiée :

- ✓ Elle n'a pas de définition dans la classe où elle est introduite (pas de corps)
- ✓ Elle sert à imposer aux sous-classes (non abstraites) qu'elles doivent définir la méthode abstraite héritée
- ✓ Elle doit être contenue dans une classe abstraite.

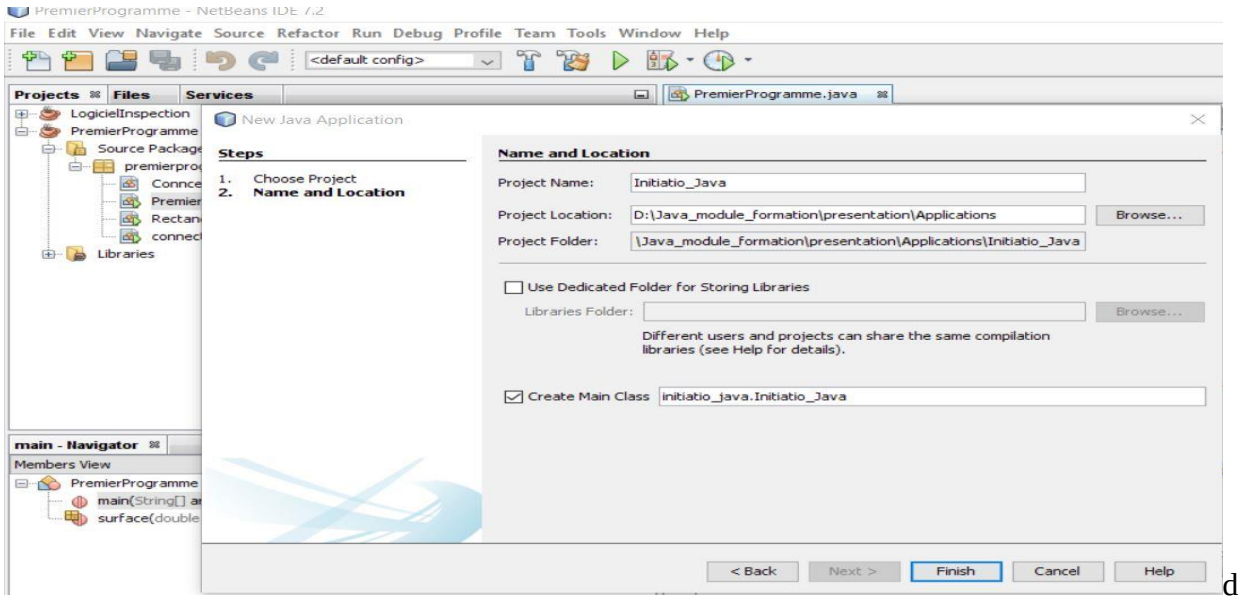
Réalisation d'interfaces graphiques Avec NetBeans

1. Création d'un projet

Dans le menu File, créez un projet en choisissant : New Project, puis choisissez Java Application.



Vérifiez ensuite le chemin

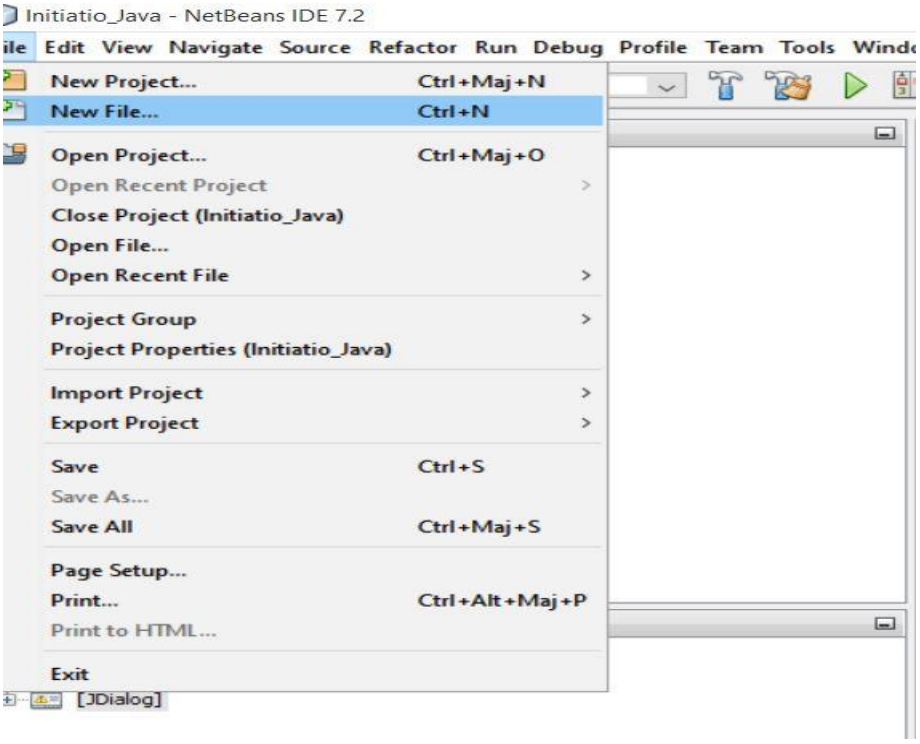


d

Sauvegardez du projet (Champs Project Location) et validez.

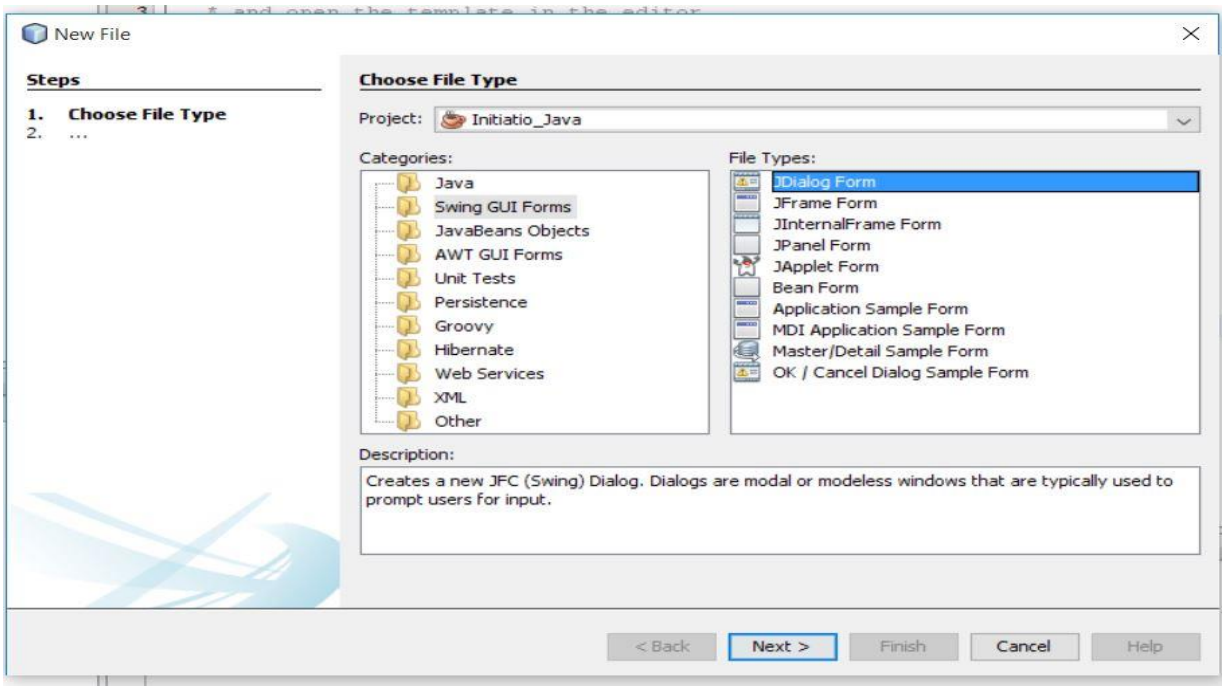
2. Ajout d'une interface dans le projet

Le projet initial se compose uniquement d'un fichier Main.java. Il s'agit par défaut d'une application console. Nous allons rapidement faire mieux !



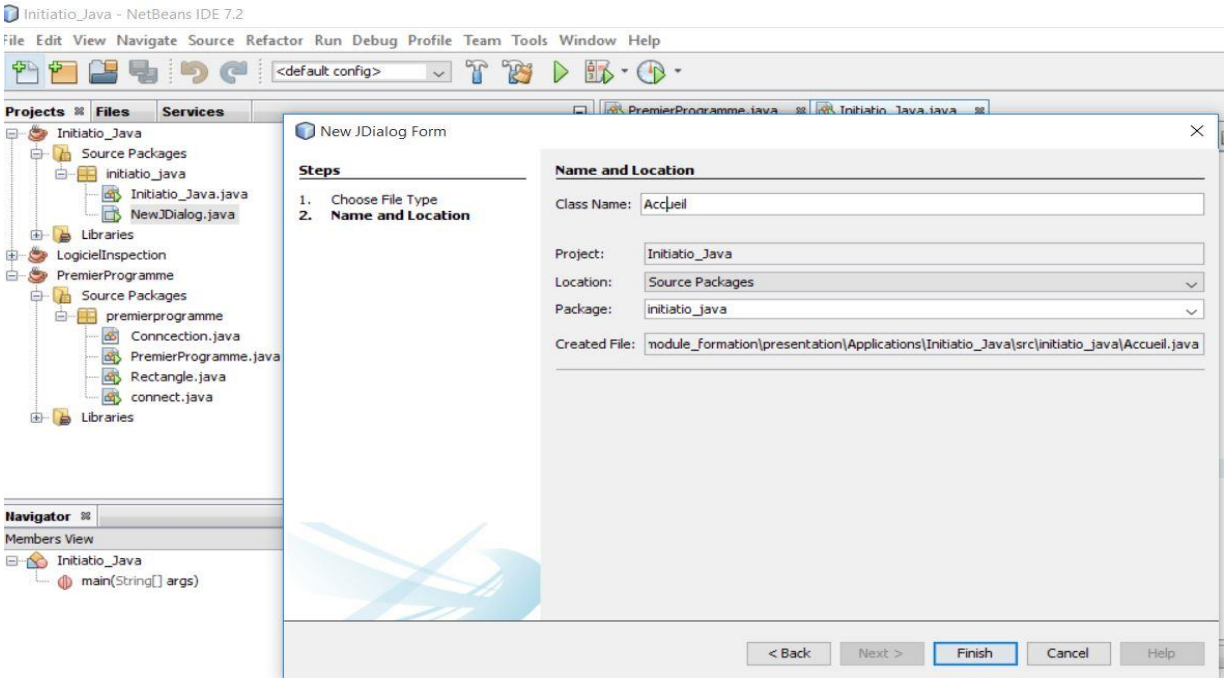
a. Ajouter un composant visuel à votre projet.

Pour cela utilisez le sous menu New File dans le menu File.



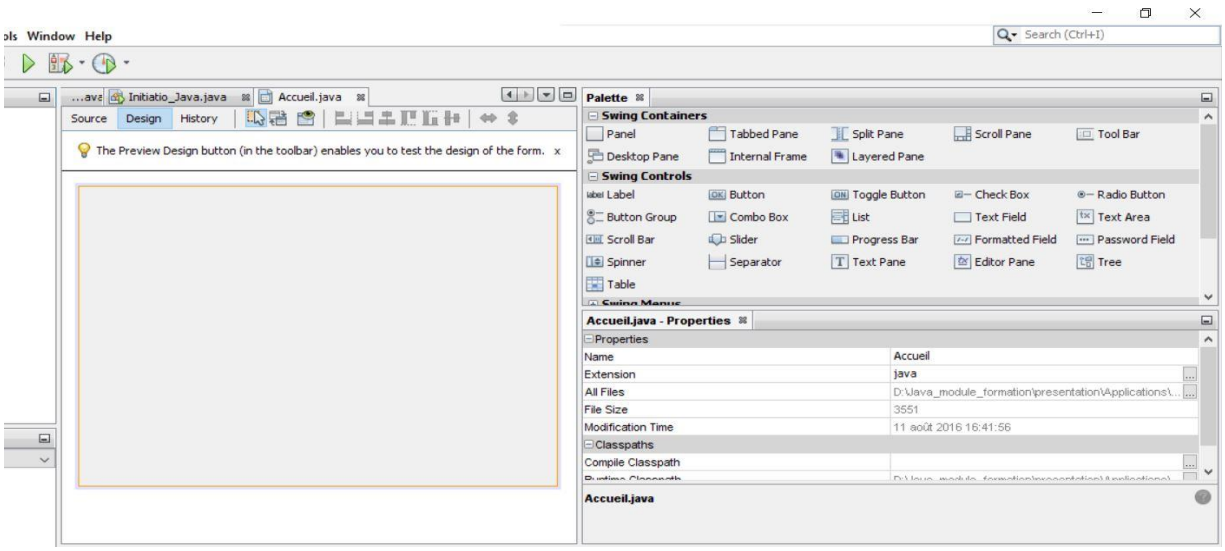
Les fichiers permettant d'obtenir des éléments graphiques sont dans Java SWING Gui Forms.

Il y a de nombreux types différents. Nous utiliserons un composant simple : **JDialog Form.**



Choisissez ensuite un nom adapté par exemple **Accueil**.

Le projet initiation_Java possède un fichier accueil.Java.



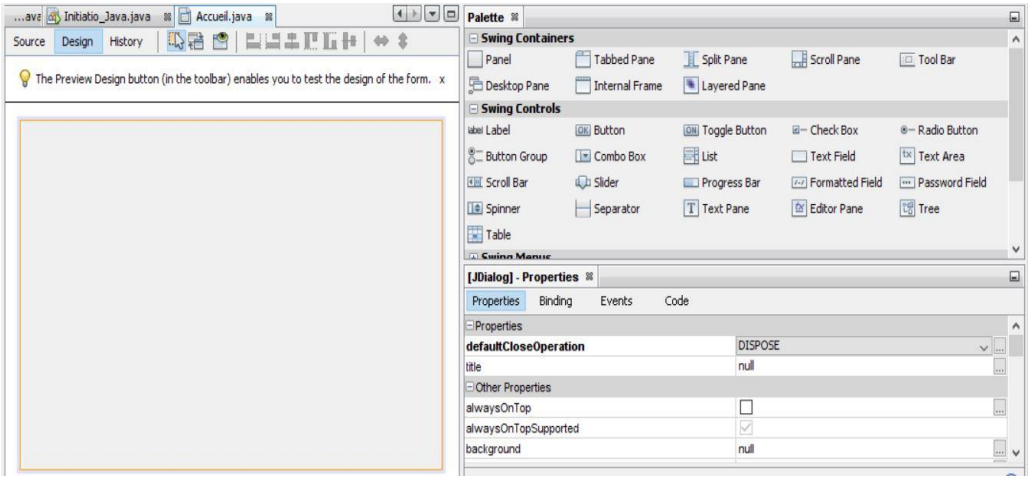
b. Compiler le projet

En cliquant sur , de la barre d'outil de NetBeans ou menu run puis choisissez la commande run projet ou appuyez sur la touche F6 du clavier.

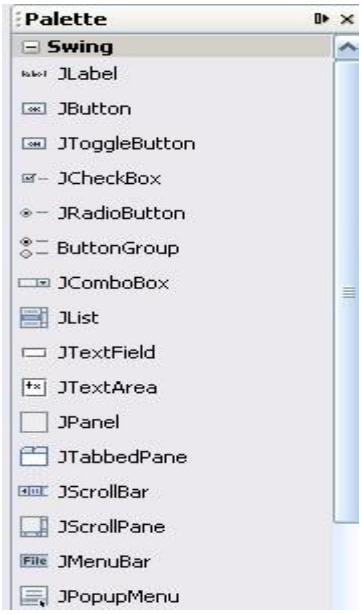
c. Les éléments de base d'une fenêtre

❖ Utilisation des objets de l'interface

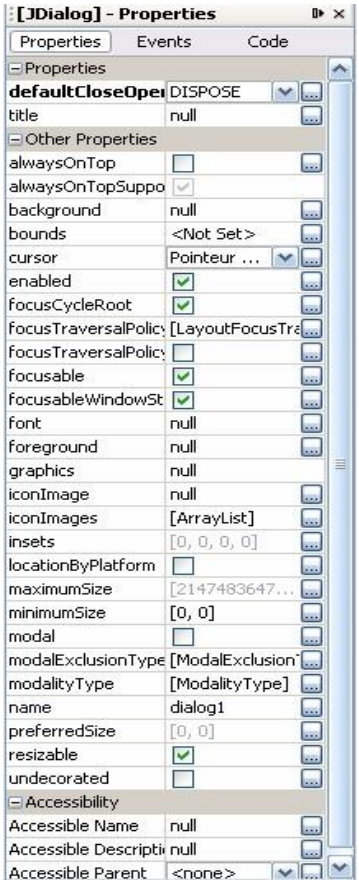
Une fenêtre graphique peut se manipuler sous forme graphique (utilisez l'onglet Design) ou sous forme textuelle (onglet Source). La construction d'une interface se fait en mode Design.



Tous les objets graphiques nécessaires à une interface sont regroupés dans l'onglet Palette.

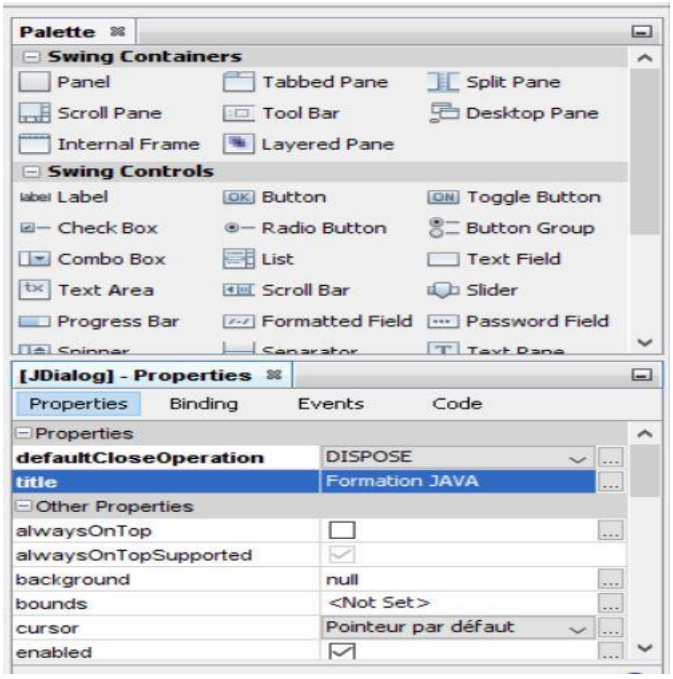


Lorsqu'un objet graphique est sélectionné par la souris, l'onglet Properties permet de manipuler ses propriétés (couleur, position, forme,….).

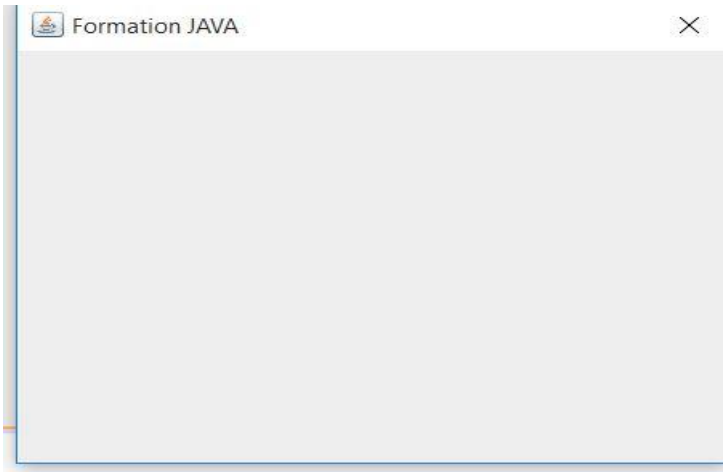


❖ Renseigner les éléments de base de la fenêtre

Pour donner un nom à cette fenêtre en l'appelant : Formation JAVA, il suffit d'ouvrir l'onglet **Properties**, et remplir le champ : **Title** par le nom désiré.



Ci-dessous, ce que vous obtenez après compilation :



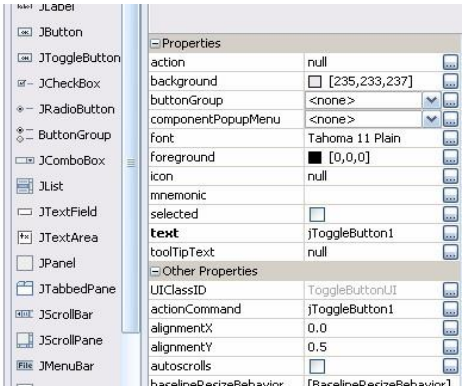
❖ Ajouter un bouton Quitter

Dans la liste des composants, sélectionnez **JToggleButton** et dessinez ensuite un bouton sur la fenêtre.

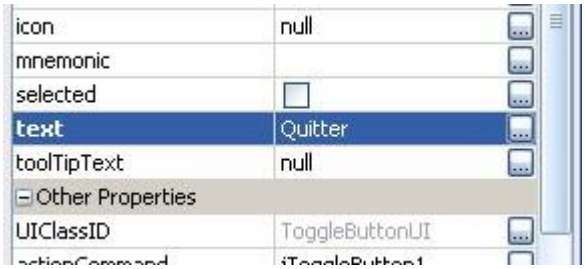
Vous devriez obtenir une fenêtre similaire à celle-ci :

Nous allons changer l'apparence de ce bouton en faisant apparaître Quitter (ne faisons pas comme Windows, ne l'appelons pas Démarrer ☺).

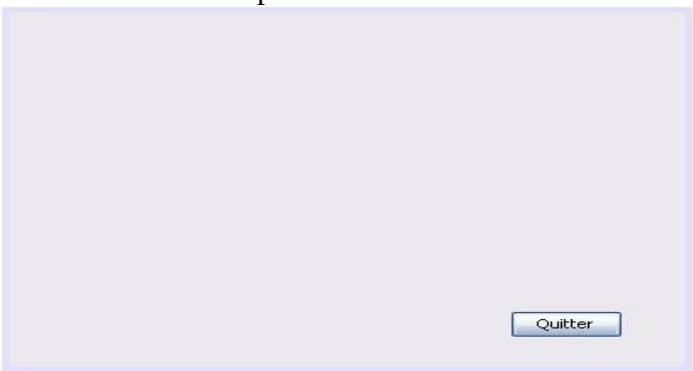
Sélectionnez le bouton à la souris, et examinez le panneau **Properties**. Modifiez ensuite le champ **Text**.



Après avoir saisi « Quitter » et une fois validé par la touche Entrée, vous obtiendrez :

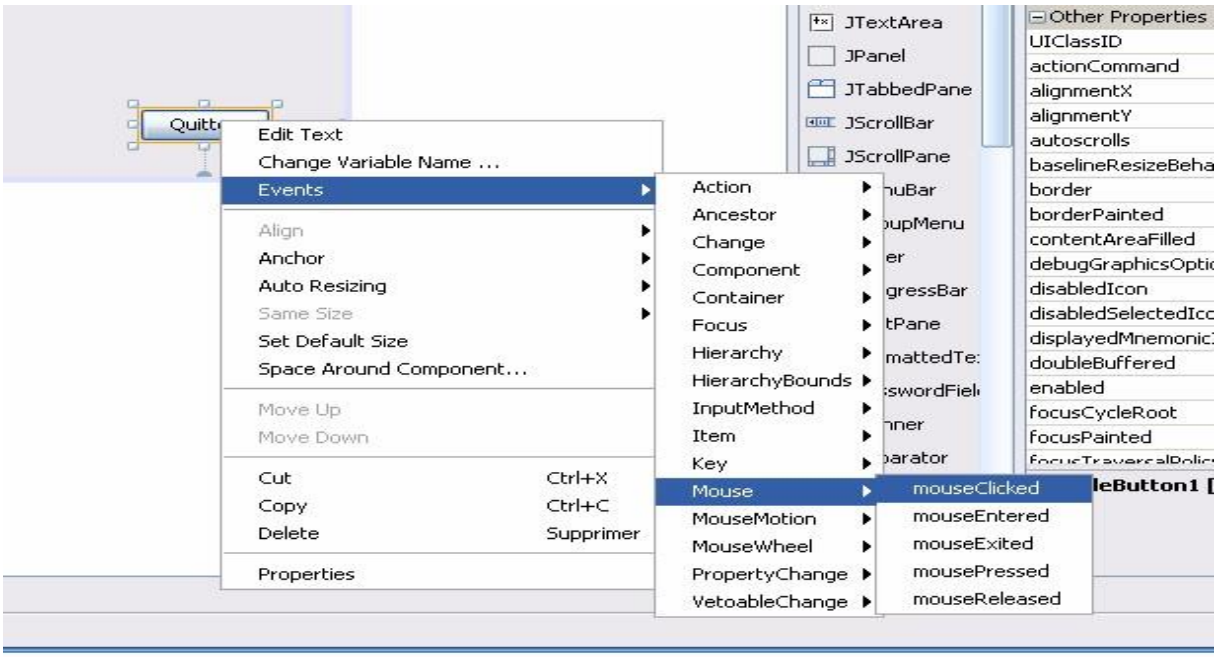


Et la fenêtre en cours de construction se présente comme suit :



❖ Attacher une action au bouton Quitter

Faire un click droit sur le bouton pour faire apparaître le menu contextuel. Et choisir ensuite : **Events/Mouse/mouseClicked.** Cela signifie que nous désirons attacher un événement lorsque l'évènement click se produira sur le bouton.



Automatiquement, NetBeans vous propose de remplir le code qui sera associé à l'événement Click sur le bouton. Le nom de la procédure est claire de ce point de vue.

```
Generated Code

private void jToggleButton1MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:
}
```

```
private void jToggleButton1MouseClicked(java.awt.event.MouseEvent evt) {
// TODO add your handling code here:

        System.exit(0);
}
```

Ce code permettra de terminer l'application sur l'événement Click.

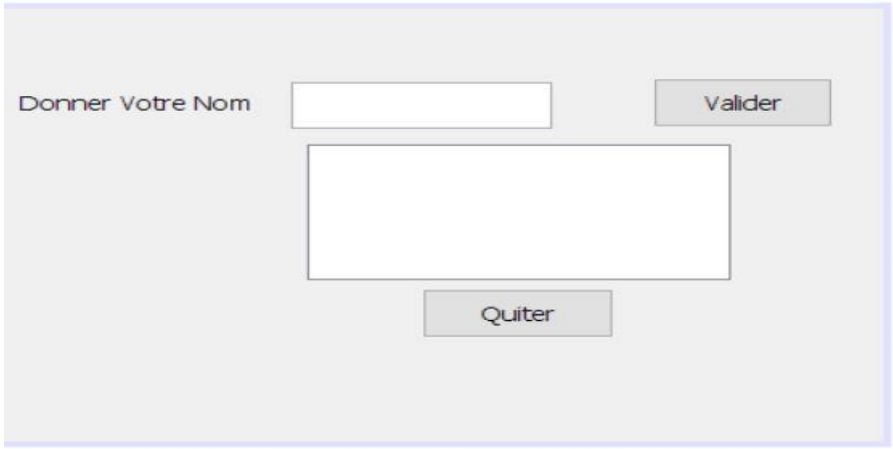
d. Communication avec l'utilisateur

❖ Lecture les données saisies et affichage du résultat

Pour lire et afficher des données saisies, Netbeans offre plusieurs composants dans l'onglet Palette comme : Jlabel, JTextField, Jlist, JTextArea …

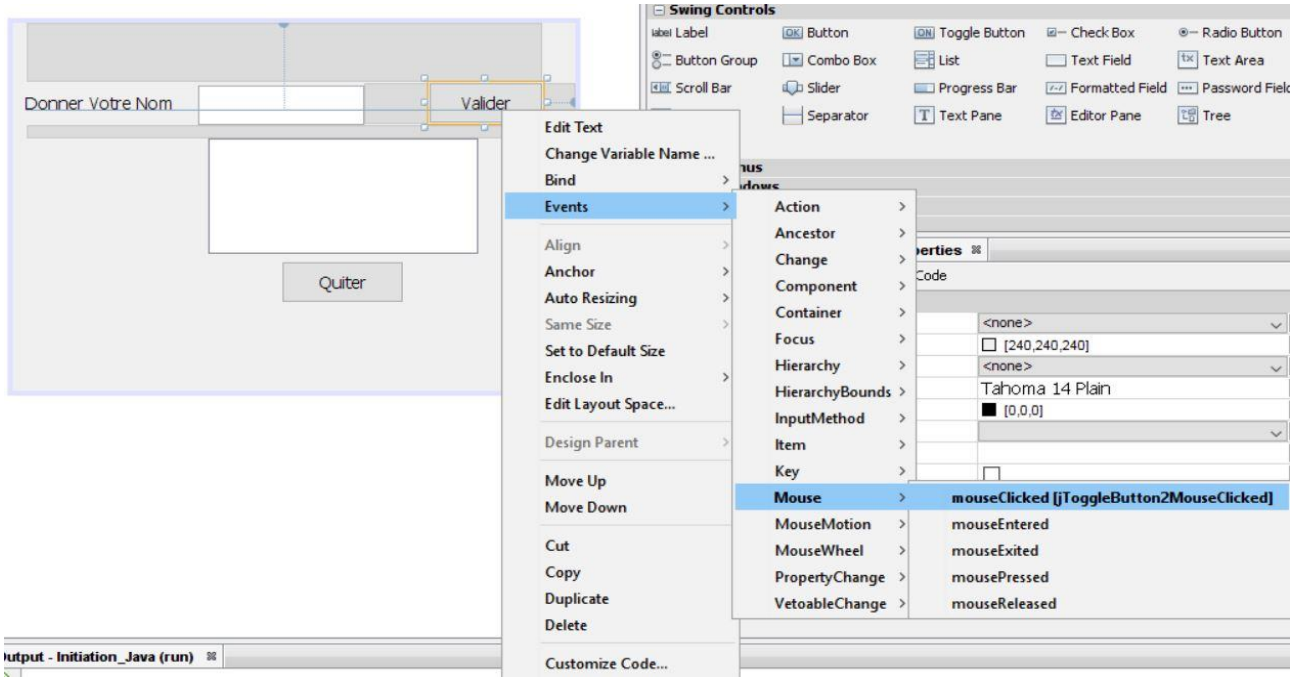
Exercice N°1:

Créer l'interface suivant :



- Écrire le code du bouton **Quitter**
- Écrire le code du bouton **Valider** sachant que lorsque l'utilisateur « clique » **sur** le bouton **Valider**, il s'affiche dans le JTextArea: « Bonjour Monsieur « contenu de JTextField»».

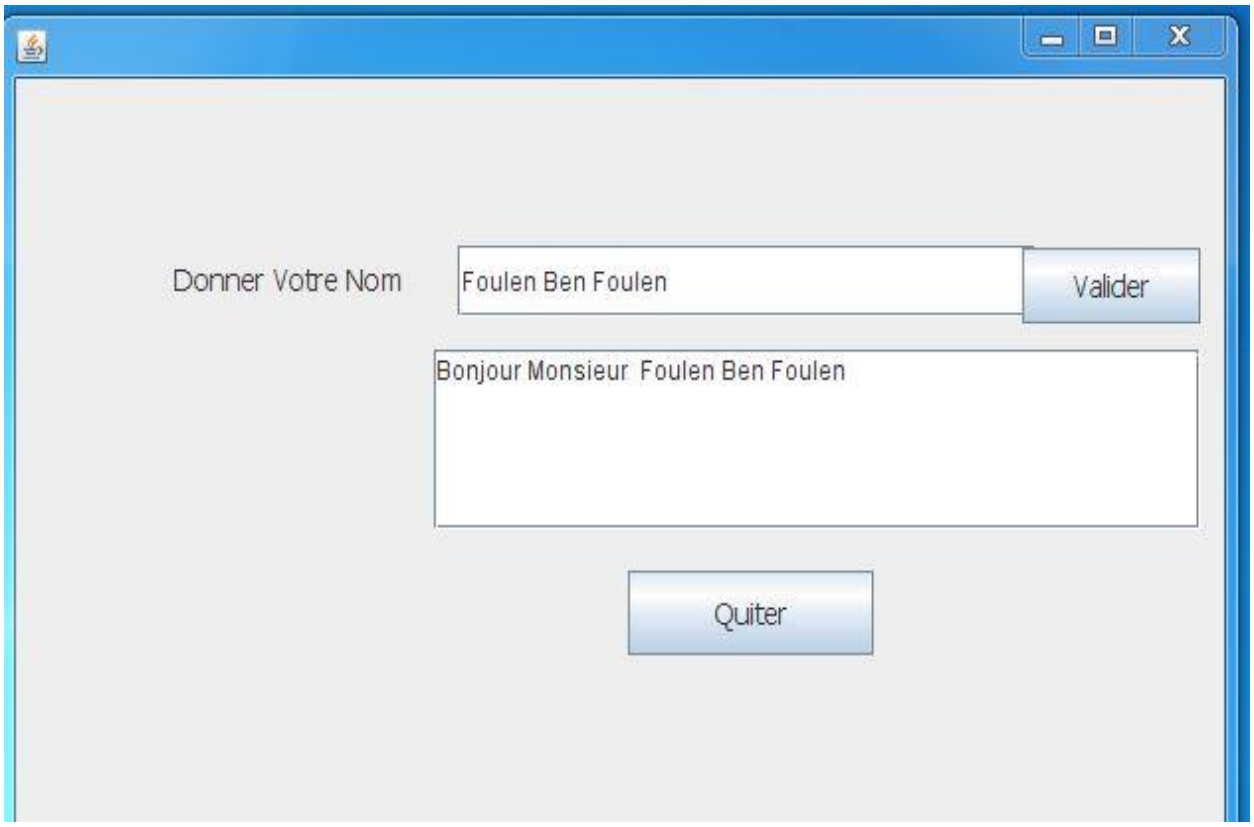
Correction



Le code du bouton Valider est le suivant :

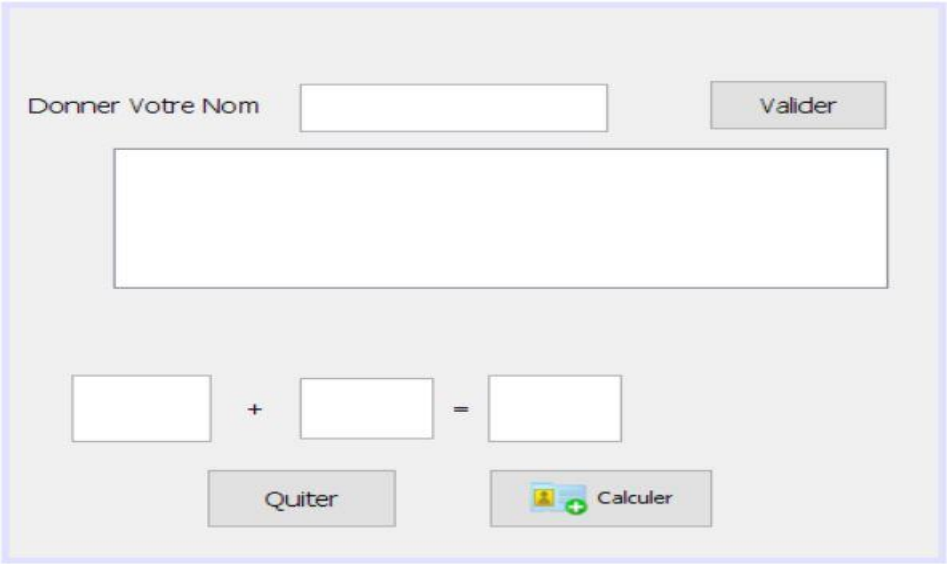
```
    private void jToggleButton2MouseClicked(java.awt.event.MouseEvent evt) {
String chaine;
chaine=jTextField1.getText();
jTextArea1.setText("Bonjour Monsieur  "+chaine);
        // TODO add your handling code here:
    }
```

Le résultat de l'exécution du fichier acueil.java en entrant Foulen Ben Foulen comme un **nom** et en cliquant sur le bouton « **Valider** » est le suivant



Exercice N°2

Dans le but d'améliorer cette fenêtre en ajoutant la possibilité de sommer deux nombres, modifier la fenêtre précédente pour obtenir l'interface suivant :

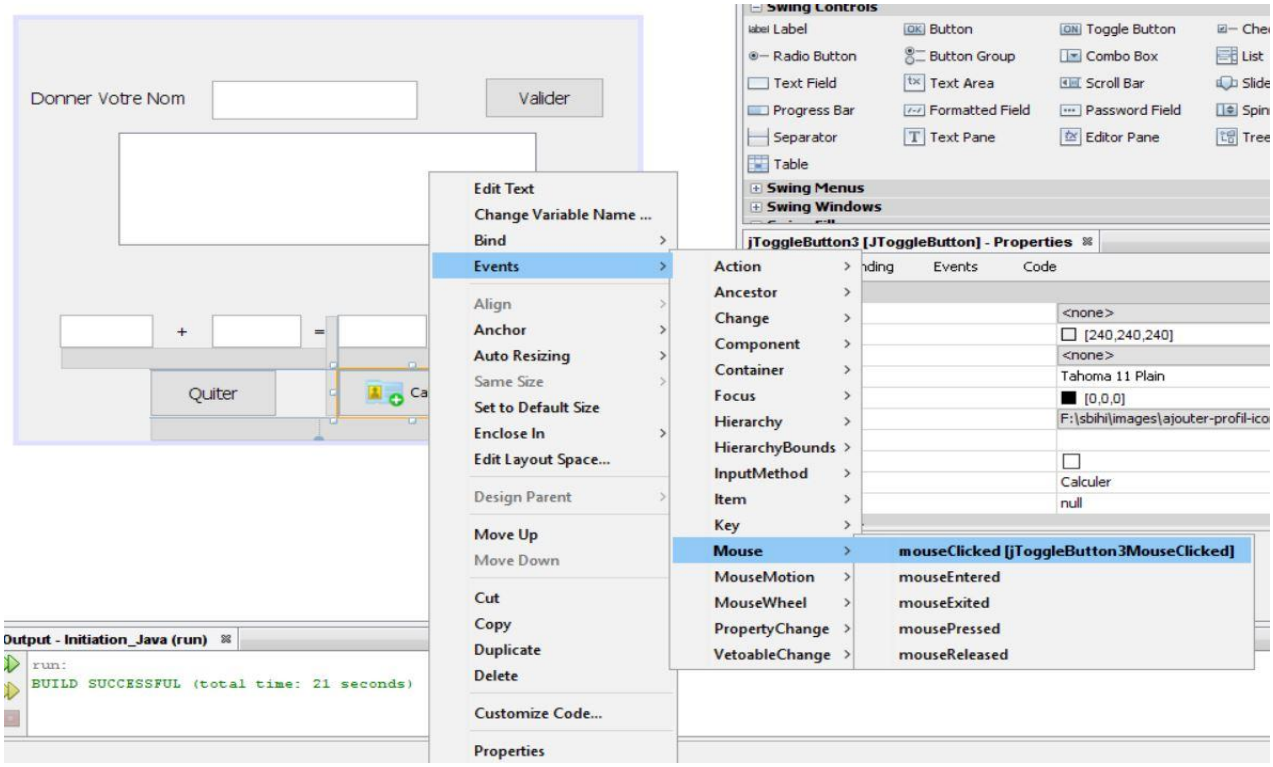


Correction

Ajouter les composants suivants :

- 3 Zones de texte,
- 2 labels (+ et =),
- Un bouton (Calculer),

Déclarez un événement sur le bouton Calculer :



Le code correspondant au bouton Calculer est le suivant :

```
    private void jToggleButton3MouseClicked(java.awt.event.MouseEvent evt) {
int i,j, result;
String ch1, ch2,ch3;
ch1=jTextField2.getText();
ch2=jTextField3.getText();
i=Integer.parseInt(ch1);
j=Integer.parseInt(ch2);
result=i+j;
Integer rr = new Integer(result);
ch3=rr.toString();
jTextField4.setText(ch3);
        // TODO add your handling code here:
    }
```

Connexion à une Base de données (JDBC)

I. Introduction

JDBC: Java DataBase Connectivity. C'est l'API standard pour permettre un accès à une base de données relationnelles dans un programme Java indépendamment du SGBD utilisé (mySQL, Oracle, Postgres ...). C'est une spécification qui définit des interfaces pour se connecter et interagir avec la base de données (exécution de requêtes ou de procédures stockées, parcourt des résultats des requêtes de sélection, ...). L'implémentation de ces spécifications est fournie par des tiers, et en particulier les fournisseurs de base de données, sous la forme de Driver.
Packages : java.sql et javax.sql

II. Étapes de connexion à une BD

1. Chargement de driver à utiliser (Driver permet de gérer l'accès à un type particulier de SGBD)
2. Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant la base utilisée
3. A partir de la connexion, créer un « statement » (état) correspondant à une requête particulière
4. Exécuter ce statement au niveau du SGBD
5. Fermer le statement
6. Se déconnecter de la base en fermant la connexion

a. Chargement du pilote

Le chargement du driver se fera avec le code suivant :
Class.forName("jdbc.DriverXXX");
Les pilotes sont chargés dynamiquement par la méthode statique forName() de la classe Class
Exemple : ***Class.forName("com.mysql.jdbc.Driver");***

Exemple

b. Connexion au SGBD

- Classe java.sql.DriverManager : permet la gestion du contrôle et de la connexion au SGBD
- *static void registerDriver(Driver driver)* : permet d'enregistrer le driver (objet driver) pour un type de SGBD particulier (Le driver est dépendant du SGBD utilisé)
- *static Connection getConnection(String url, String user, String password):* consiste à crée une connexion à une BD tel que:
    - url : identification de la base considérée sur le SGBD
    - user : nom de l'utilisateur qui se connecte à la base
    - password : mot de passe de l'utilisateur
- Connexion sans information de sécurité

Connection con = DriverManager.getConnection ("jdbc:odbc:epicerie");

- Connexion avec informations de sécurité

Connection con = DriverManager.getConnection ("jdbc:odbc:epicerie", user, password);

Exemple :

```
String Url = "jdbc:mysql://localhost:3306/";
String dbName = "Formation";
String encodage ="?&characterEncoding=UTF-8";
String driver = "com.mysql.jdbc.Driver";
String User="root";
String Password="";
```

```
Class.forName("com.mysql.jdbc.Driver");
 System.out.println("Le pilote JDBC MySQL a été chargé");
 Connection  conn = DriverManager.getConnection(Url + dbName + encodage, User, Password);
 System.out.println("Connexion effectuée  !");
```

Résultat de l'exécution du code:

```
Output
Java DB Database Process   Debugger Console   PremierProgramme (run)
run:
Le pilote JDBC MySQL a été chargé
Connexion effectuée  !
BUILD SUCCESSFUL (total time: 0 seconds)
```

c. Création de l'objet Statement

L'objet Statement permet d'exécuter des instructions SQL (select, create, insert, update, delete). Le résultat d'une requête est stocké dans l'objet ResultSet, grâce auquel on peut parcourir les lignes de résultats et les afficher.

Exemple : avec executeQuery()

```
Statement stmt =  (Statement) conn.createStatement();
 String req =" select * from direction ";
  ResultSet res = stmt.executeQuery(req);
```

- Création d'un PreparedStatement :

```
PreparedStatement ps = conn.prepareStatement(
    "insert into Marins (nom, prenom, age) values (?, ?, ?)") ;
```

- Fixer les paramètres d'un PreparedStatement :

```
PreparedStatement ps = conn.prepareStatement(
    "insert into Marins (nom, prenom, age) values (?, ?, ?)") ;

ps.setString(1,  "Surcouf") ;
ps.setString(2,  "Robert") ;
ps.setInt(3,  32) ;
```

On notera que la numérotation des paramètres commence à 1. Bien sûr, tous les paramètres doivent avoir une valeur fixée avant que la requête soit exécutée.

Exemple :

Soit une base de données intitulée « Formation » contenant les deux tables suivantes :

- ✓ Forum (ref, nom, rubrique, date, titre, email, message)
- ✓ Forum2 (refa, ref #,nom, rubrique, date, titre, email, message)

Écrire le code nécessaire pour se connecter à la base de données Formation et afficher le contenu de la table Forum.

Réponse

- Chargement du driver et Création de la connexion à la base de données :

```
❖ String Url = "jdbc:mysql://localhost:3306/";
❖ String dbName = "Formation";
❖ String driver = "com.mysql.jdbc.Driver";
❖ String User="root";
❖ String Password="";
❖ Class.forName("com.mysql.jdbc.Driver");
```

```
System.out.println("Le pilote JDBC MySQL a été chargé");
Connection  conn = DriverManager.getConnection(Url + dbName + encodage, User, Password);
System.out.println("Connexion effectuée  !");
```

Exécution d'une requête de sélection:

```
Statement stmt =  (Statement) conn.createStatement();
int ref;
String nom, rubrique, datee, titre, email, message;
        String req =" select * from forum ";
            ResultSet res = stmt.executeQuery(req);
            while (res.next()) {
            ref=res.getInt(1);
            nom=res.getString(2);
            rubrique=res.getString(3);
            datee=res.getString(4);
            titre=res.getString(5);
            email=res.getString(6);
            message=res.getString(7);
    System.out.println("nom:"+nom+","+rubrique+","+titre+","+email+","+message);
            }
```

Connexion Affichage

```java
public class connect {
  public static void main ( String[] args) throws ClassNotFoundException, SQLException  {
                String Url = "jdbc:mysql://localhost:3306/";
                String dbName = "Formation";
                String encodage ="?&characterEncoding=UTF-8";
                String driver = "com.mysql.jdbc.Driver";
                String User="root";
                String Password="";
                int ref;
                String nom, rubrique, datee, titre, email, message;

                      Class.forName("com.mysql.jdbc.Driver");
                       System.out.println("Le pilote JDBC MySQL a été chargé");
                       Connection  conn = DriverManager.getConnection(Url + dbName + encodage, User, Password);
                       System.out.println("Connexion effectuée  !");
                     Statement stmt =  (Statement) conn.createStatement();
                      String req =" select * from forum ";
                       ResultSet res = stmt.executeQuery(req);
                       while (res.next()) {
                       ref=res.getInt(1);
                       nom=res.getString(2);
                       rubrique=res.getString(3);
                       datee=res.getString(4);
                       titre=res.getString(5);
                       email=res.getString(6);
                       message=res.getString(7);
      System.out.println("nom:"+nom+","+rubrique+","+titre+","+email+","+message);
                       }
```

Résultat de l’exécution de la requête de sélection :

```
Output - PremierProgramme (run)
run:
Le pilote JDBC MySQL a été chargé
Connexion effectuée  !
nom:sbihi,algorithmique,Problème avec les adresses des variables,sbihih@voila.fr,Comment peut on savoir l'adresse d'une variable déclarée dans le programme principa
nom:toto,algorithmique,Arbre binaire,toueiti@yahoo.fr,Qui peut m'expliquer le rôle d'un arbre bianire de recherche.
merci d'avance !!
@bientot
nom:ISSAM,BD,Comment afficher sur l'ecran avec plsql,hassoun@google.fr,SVP
pouvez vous m'aider a afficher des données sur l'ecran avec plsql
j'ai utiliser DBMS_OUTPUT.putline mais ca n'a pas marcher
svp une réponse tres vite
@bientot
nom:zoro,algorithmique,tableau,zoro@voila.fr,comment declarer un tableau de deux dimension en pascal et en C
je veux des codes SVP
..
nom:Slouma,TIC,Bienvenu,toueiti@yahoo.fr,Qui peut m'expliquer le principe de AJAX ????,
nom:Sdiri,réseaux,technologie,sdiri@yahoo.com,quelles sont les dernieres technologies de la sécurité des réseaux???
nom:MedAli,BD,Affichage impossible avec plsql !!,med@hotmail.com,J'ai un code en plsql qui s'exécute sans problème de syntaxe ou autre mais il n'affiche rien sur l'
j'utilise l'outil SQL+ d'oracle!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Annexes

Série d'exercices

EXERCICE N°1

Écrire un programme en java intitulé somme qui permet de calculer et d'afficher la somme des 100 premiers entiers.

EXERCICE N°2

- Créez un nouveau projet sous Netbeans nommé TP**1**
- Créez un nouveau paquetage dans le projet **TP1** nommé **étudiant**
- Complétez le code de cette classe en lui ajoutant :
  - Un attribut privé de type String nommé nom ;
  - Un constructeur public servant à initialiser le nom de l'étudiant ;
  - Une méthode publique qui ne renvoie rien, nommée **travailler**, qui affiche à l'écran, si l'étudiant a pour nom Sbihi :
    Sbihi se met au travail !
  - Une méthode publique nommée seReposer, qui affiche à l'écran, si le nom de l'étudiant Sbihi se repose
  - Créez parallèlement à la classe Etudiant une classe TestEtudiant (aussi dans le paquetage etudiant) contenant une méthode main qui :
    - Crée un étudiant (instance de la classe Etudiant) en lui donnant un nom écrit directement dans le fichier source ;
    - Invoque la méthode travailler de l'étudiant créé ;
    - Invoque la méthode seReposer de l'étudiant créé.

EXERCICE N°3

Écrivez un programme intitulé Scalaire.java qui permet de calculer et d'afficher le produit scalaire de deux vecteurs.

NB :

Soient A(a1,a2,a3,……an) et B(b1,b2 ;b3,,,,,bn)

Le produit scalaire de *a* par *b* est : $a.b = a_1*b_1 + a_2*b_2 + ... + a_n*b_n$

Exemple : a = (5, 3, -1) b = (2, 1, 2) $a \cdot b = 11$

EXERCICE N°4

- Créez une classe Liquide contenant seulement une méthode imprimer() qui affiche : ”je suis un liquide”
- Créez 2 classes dérivées de la classe Liquide, les classes Cafe et Lait, dont les méthodes imprimer() affichent respectivement ”je suis un Café”, ”je suis du Lait”
- Enfin créerez une classe Tasse ayant un attribut l de la classe Liquide et une méthode AjouterLiquide (Liquide li) et une méthode imprimer ()
- Testez l’utilisation : créez un tableau de Tasses qui contient différents liquides.

EXERCICE N°5

Chaque animal a une race, un nom, et fait quelque chose comme bruit (un chien fait vau, vau par exemple).

1. Créez la classe Animal.
2. Créez une méthode pour la classe Animal qui écrit sur l'écran la présentation d'un animal : son nom, sa race et ce qu'il fait comme bruit.
3. Créez les classes dérivées pour les différentes races : chien, chat, majom, etc...
4. Créez une classe ZooMain qui contient la méthode main.
5. Créez les animaux du zoo dans une variable static de la classe ZooMain en forme d'un tableau. Les cases du tableau correspondent aux cages du zoo.
6. Créez un chien et deux chats dans votre zoo et faites-les présenter.
7. Ajoutez une interface pour la méthode de présentation et modifiez le code.

EXERCICE N°6

Étant donné un fichier texte de la forme suivante :

```
rouge;#FF0000
vert;#008000
bleu;#0000FF
```

On souhaite ouvrir le fichier, le parcourir ligne par ligne et afficher son contenu à l’écran de la manière suivante :

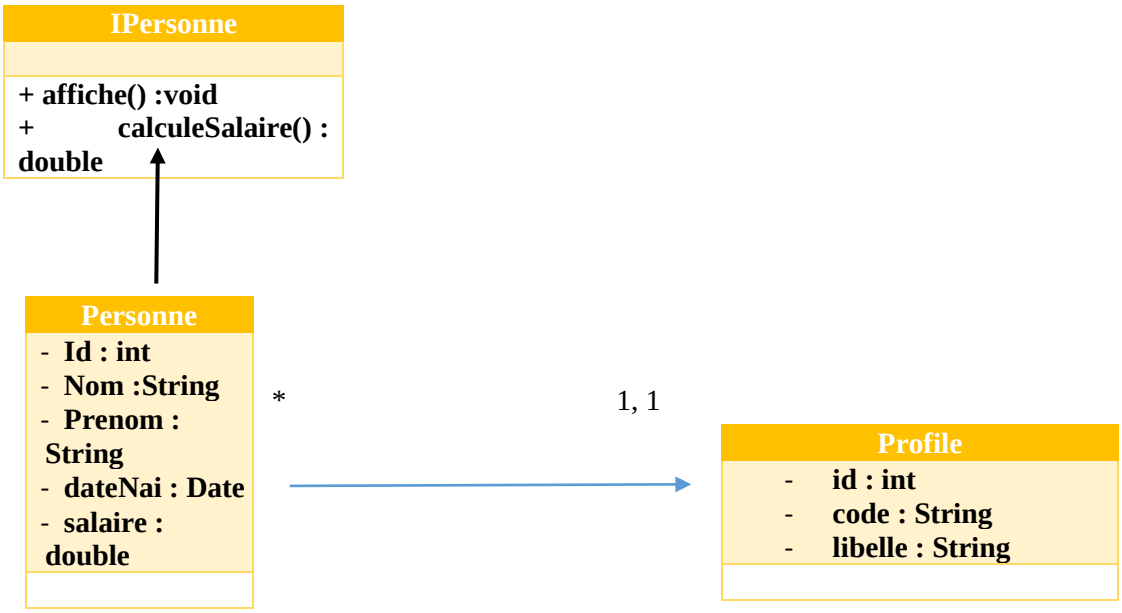
```
Ligne 1 / champs 1 : rouge
Ligne 1 / champs 2 : #FF0000
Ligne 2 / champs 1 : vert
Ligne 2 / champs 2 : #008000
Ligne 3 / champs 1 : bleu
Ligne 3 / champs 2 : #0000FF
```

EXERCICE N°7

Écrire un programme en java qui calcule et affiche le factoriel d’un entier positif n (en utilisant le schéma récursif).

EXERCICE N°8

Soit le diagramme de classe suivant :



Travail demandé :

1. Créer l'interface **IPersonne** dans le package **«Exercices».**
2. Créer les classes **Personne** et **Profil** dans le package **« Exercices »**.

Sans redéfinir les méthodes de l'interface, qu'est ce que vous remarquez ?

3. Redéfinir les méthodes **affiche ()** et **calculerSalaire ()** dans la classe **Personne.**

Sachant que :

- La méthode **affiche ()** affiche une chaine de caractère sous la forme :

Je suis le directeur Sbihi Ammen Allah Karim né le 04 juin 1970 mon salaire est

20 000dt,

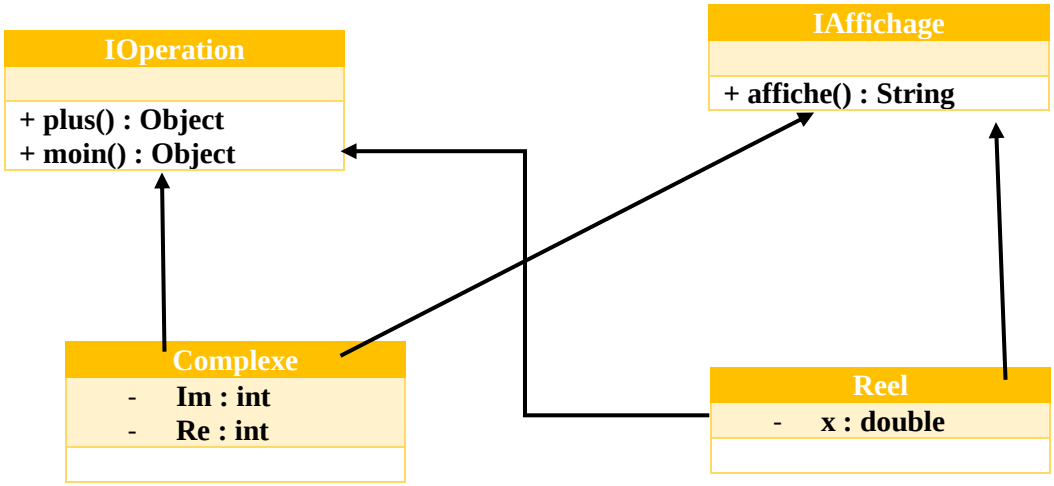
- Le directeur aura une augmentation de 20% par rapport à son salaire normal,

- Les autres employés auront une augmentation de 10%.

4. Écrire un programme de test.

EXERCICE N°9

Soit le diagramme de classe suivant :



1. Créer l'interface **IOperation** et **IAffichage** dans le package **«Exercices».**
2. Créer les classes **Complexe** et **Reel** dans le package **« Exercices».**
3. Redéfinir les méthodes **moins (), plus ()** et **affiche ()** dans la classe complexe sachant que:

Un nombre complexe est un nombre qui comporte deux parties : une partie réelle et une partie imaginaire.

Un nombre complexe est généralement écrit sous la forme : 4+3i (partie réelle = 4 et partie imaginaire = 3).

L'addition (la soustraction) de deux nombres complexes consiste à additionner (soustraire) les parties réelles de ces deux nombres pour obtenir la partie réelle du résultat et à additionner (soustraire) les deux parties imaginaires pour obtenir la partie imaginaire du résultat.

4. Surcharger les méthodes **plus ()** et **mois ()** dans la classe **Reel**.

Nouvelle signature :

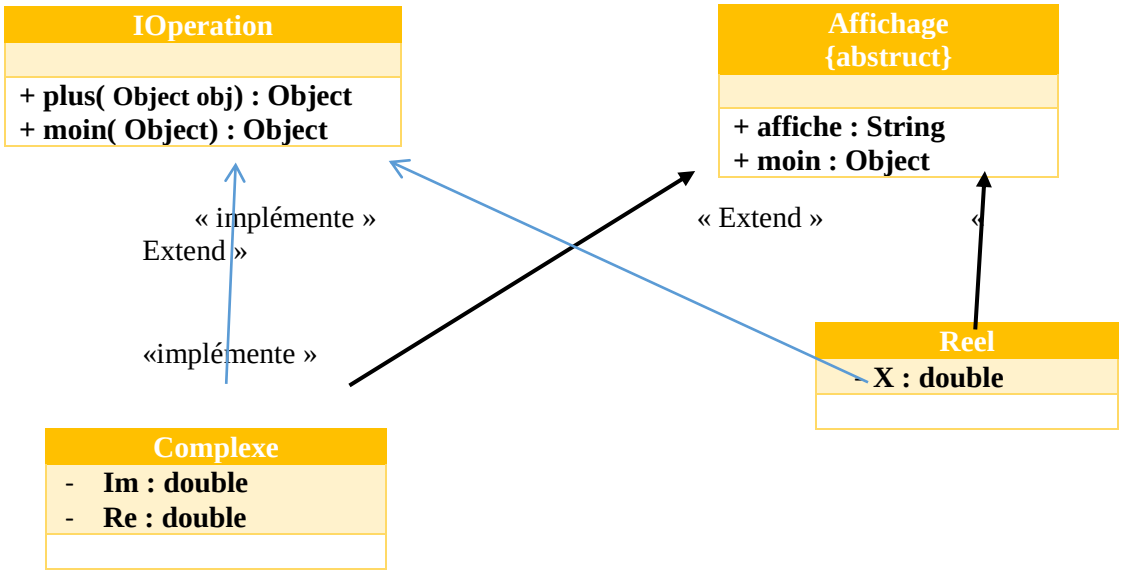
- double plus (double…x)
- double moins (double x)

Qu'est-ce que vous remarquez ?

5. Créer un programme de test.

EXERCICE N°10

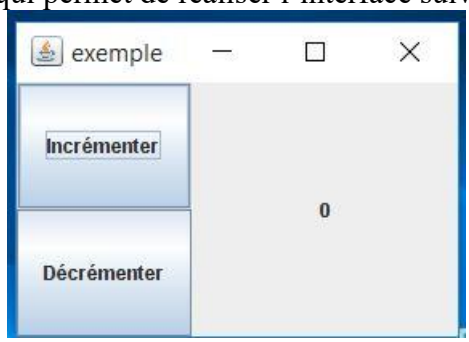
Soit le diagramme de classe suivant :



1. Créer l'interface **IOperation** dans le package **« Exercices ».**
2. Créer la classe abstraite **Affichage** dans le package **« Exercices».**
   Qu'est ce que vous remarquez ?
3. Créer la classe **Complexe** et la classe **Reel** dans le package **« Exercices».**
4. Créer un programme de test.

EXERCICE N°11

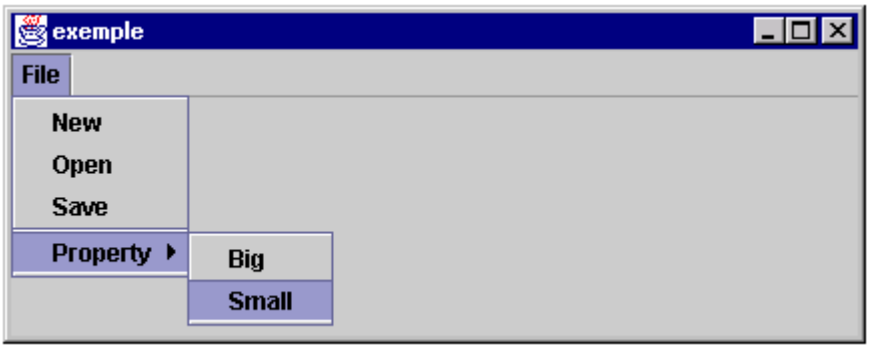
Écrire un programme en java qui permet de réaliser l'interface suivant :



- En cliquant sur le bouton Incrémenter, la valeur affichée (0) est incrémentée de 1
- En cliquant sur le bouton Décrémenter, la valeur affichée (0) est décrémentée de 1

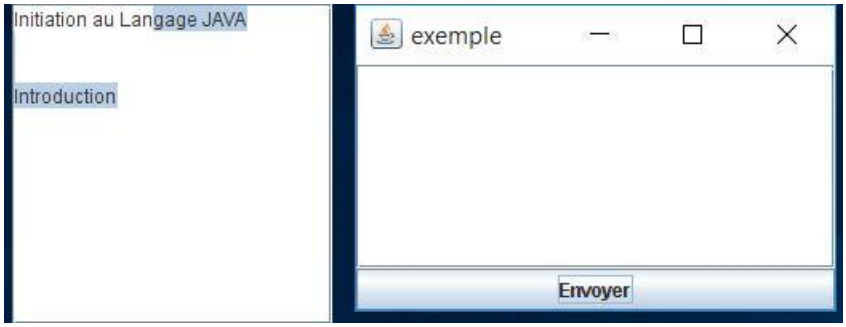
EXERCICE N°12

Écrire un programme permettant d'afficher l'interface suivant :



EXERCICE N°13

Écrire un programme permettant de créer l'interface suivant :



Cette interface est composée d'une fenêtre principale où l'on tape du texte, et une fenêtre secondaire où le texte est envoyé, ligne par ligne. La fenêtre principale contiendra une zone de texte éditable où l'utilisateur tapera la ligne à envoyer, ainsi qu'un bouton d'envoi.
La fenêtre secondaire contiendra une zone de texte non éditable où le texte reçu sera écrit.

- Pour régler la taille de la fenêtre : *setSize(200, 200)*
- Pour positionner la fenêtre : *setLocation(400, 0)*
- Pour créer un panel scrollable : *scroll = new JScrollPane(panel); panel2.add(scroll);*

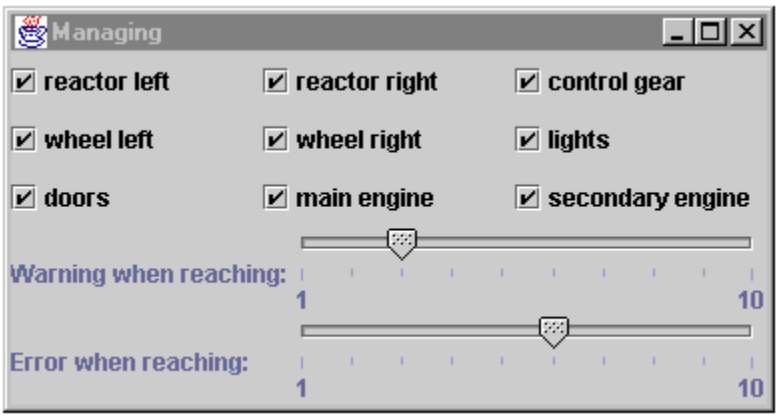
EXERCICE N°14

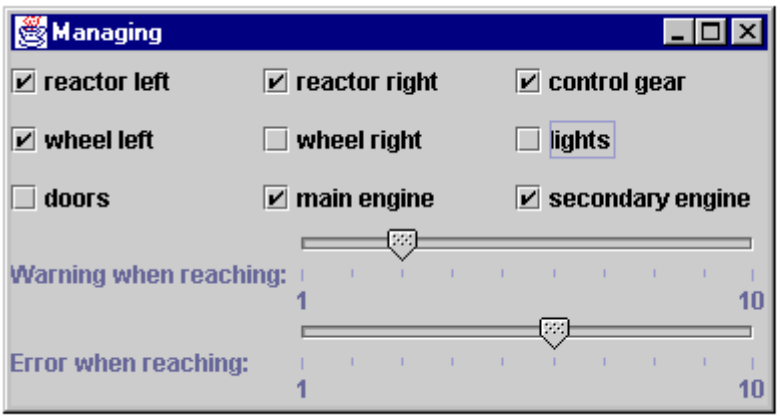
Écrire un programme qui affiche une ligne de texte *configurable.*

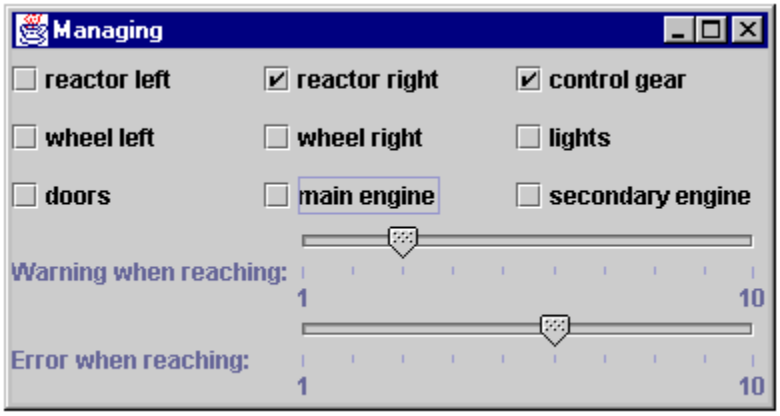
- Le panel du haut sera constitué de trois onglets (rechercher dans le tutorial comment faire)
- Le premier permettra de configurer le texte.
- Le bouton "reset" efface le texte tapé. Le bouton "update" met à jour le texte.
- Le second panel permettra de configurer la couleur, avec ses trois composantes (rouge, vert, bleu) dont la valeur varie entre 0 et 255.
- Le troisième panel permettra de configurer l'alignement du texte.
- Le panel du bas contiendra le texte configuré (ici "Default text" en violet).

EXERCICE N°15

Écrire un programme qui avertit de l'état d'un vaisseau. Une première fenêtre permettra de décocher les parties du vaisseau qui sont en panne. Dans une seconde fenêtre, l'état du vaisseau sera marqué par "Ok", "Warning" ou "Error".
De plus, dans la première fenêtre, deux "sliders" permettront de régler le nombre d'erreurs à partir duquel l'état passe de Ok à Warning, puis de Warning à Error. Le taux pour le passage de Warning à Error devra toujours être supérieur à celui pour le passage de Ok à Warning.







EXERCICE N°16

On veut écrire un programme qui permet d'afficher et de déplacer des figures géométriques colorées dans un repère en deux dimensions.
Un graphique est un ensemble de figures affichables et déplaçables par translation. On veut afficher des segments, des triangles, et des rectangles. Ces trois éléments graphiques sont définis à l'aide de « **points** ».
Après une première analyse, on décide de modéliser l'application de la manière suivante :

- Une classe **Point** représentera les coordonnées qui serviront à créer des figures
- Une classe Segment sera définie par deux points et une couleur, codée par un entier positif.
- Une classe Triangle sera définie par trois points et une couleur, codée par un entier positif.
- Une classe Rectangle sera définie par quatre points et une couleur, codée par un entier positif.
- La classe Graphique est la classe principale et permettra d'afficher un ensemble d'objets graphiques.

Comme nous pensons devoir ajouter ultérieurement de nouveaux types de figures géométriques, nous décidons d'utiliser des interfaces pour décrire leurs comportements communs.
1. Après avoir étudié les différences et les points communs entre les quatre premières classes que nous avons distinguées, décrivez la ou les **interfaces** utiles et précisez les classes qui les implémenteront.
2. Décrivez la méthode static de la classe Graphique qui permet d'afficher un tableau de figures géométriques.
3. Vous avez déjà étudié une classe Point. Sans écrire toutes les méthodes que vous utiliserez pour celle-ci, donnez les constructeurs que vous utiliserez le plus vraisemblablement dans le reste du programme, la méthode toString et le squelette de cette classe.
4. Donnez le constructeur de la classe Segment, et la méthode qui effectuera une translation.
5. Décrivez entièrement la classe Triangle.

EXERCICE N°17

Le but du TD sera de réaliser un formateur de texte fonctionnant sur le même principe que le formateur de texte fmt sous Unix.

Un objet, le parseur, prend un texte, dont il lit les mots un à un. Un deuxième objet, le formateur, accumule ces mots en listes de lignes dont chacune contient des mots séparés par des espaces.
Le formateur imprime ensuite la liste de ces lignes. Ainsi, le programme supprime les espaces redondants entre deux mots, les lignes vides redondantes entre deux paragraphes, ou les passages à la ligne non nécessaires.
On modélise le problème de la façon suivante : on introduit le concept de Boite qui représente les objets composant le texte formaté.

- Une Boite est un élément du texte formaté qui a une taille et peut être affiché.
- Une Boite espace est un élément du texte formaté qui séparera les mots. Ce sera un seul espace. Dans le cas où le texte n'est pas justifié, et potentiellement plusieurs espaces sinon.
- Une Boite mot est un élément du texte formaté qui représentera un mot.
- Une Boite composite représentera une ligne de texte.
- Le Formateur utilise le Parseur et les objets précédemment décrits pour construire le texte formaté.

La classe Parseur vous est donnée. Cette classe a un constructeur public Parseur (String filename), qui construit un parseur qui lit à partir du fichier nommé, et une méthode publique suivant() de type String, qui retourne le mot suivant, la chaine vide à la fin d'un paragraphe, et l'objet null à la fin du fichier.
Les objets implémentant l'interface Boite représenteront une unité de texte d'au plus une ligne, soit un mot, un espace, ou une ligne.
Trois classes implémenteront cette interface :

- BoiteEspace : un objet de cette classe représente un espace ;
- BoiteMot : un objet de cette classe représente un mot ;
- BoiteComposite : un objet de cette classe représente une suite horizontale de boites ; nous nous en servirons pour représenter les lignes.

1. Décrivez la déclaration de l'interface Boite, qui contient les déclarations de deux méthodes publiques : largeur(), de type entier, toString(), de type String.
2. Décrivez ensuite les définitions de deux classes qui implémentent cette interface : BoiteEspace et BoiteMot. Une BoiteEspace a une longueur de 1, et se convertit en la chaine réduite à un espace " " (`à ne pas confondre avec la chaine vide !'). Une BoiteMot représente une chaine arbitraire, sa méthode toString retourne cette chaine, et la méthode largeur() retourne sa longueur.
3. Écrivez maintenant une nouvelle classe BoiteComposite qui implémente l'interface Boite. Une boite composite contient une suite de boites ; sa largeur est la somme des largeurs des boites qu'elle contient, et sa représentation sous forme de chaine est la concaténation des représentations des boites qu'elle contient.

Remarque : Les objets de la classe BoiteComposite stockent des objets qui sont soit des mots, soit des espaces, soit eux-mêmes des boites composites. On réfléchira soigneusement au type de structure qu'il faudra utiliser.
En plus des méthodes de l'interface Boite, la classe BoiteComposite implémente une méthode publique vide qui détermine si une boite composite est vide, et une méthode publique accumule qui ajoute une boite à la fin d'une boite composite. Il sera peut-être judicieux d'ajouter des méthodes auxiliaires privées.

4. Décrivez maintenant une classe Formateur qui accumule la suite des mots contenus dans un fichier texte. Le constructeur de cette classe prendra un nom de fichier sur lequel il construira un parseur.

Décrivez les deux méthodes publiques : *lit*, qui lit la suite des mots retournés par le parseur et la range dans un tableau dont les éléments sont du type de la classe BoiteMot, et *imprime*, qui imprime cette suite de mots. Tous les mots d'un paragraphe sont accumulés dans une BoiteComposite. Le formateur commence avec une boite composite vide. A chaque fois qu'il lit un nouveau mot, il y ajoute un espace et la boite représentant ce mot. Enfin, lors d'une fin de paragraphe, il garde la boite courante, et recommence avec une boite composite vide. Il sera peut-être judicieux d'utiliser une méthode privée pour la fin d'un paragraphe. On fera attention à ne pas ajouter d'espaces ni au début ni à la fin d'un paragraphe.

Correction de la Série d'exercices

EXERCICE N°1

```
public class Somme {
public static void main(String[] arg) {
int Somme = 0;
for (int i = 1; i <= 100; i++)
Somme = Somme + i;
System.out.println(Somme) ;
}
}
```

EXERCICE N°2

```
 *
 * @author Sbihi
 */
package etudiant;
public class Etudiant {
        private String nom;
        Etudiant(String nom) {
                this.nom = nom;    }
                public String getNom() {
                return this.nom;  }
                public void travailler() {
                System.out.println(this.nom + " se met au travail !");
                public void seReposer() {
                System.out.println(this.nom + " se repose");            }
}
```

```
package etudiant;

public class TestEtudiant {
        public static void main(String[] arg) {
                Etudiant etudiant = new Etudiant("sbihi");
                etudiant.travailler();
                etudiant.seReposer();            }
}
```

```
package etudiant;
public class TestEtudiantBis {
        public static void main(String[] arg) {
                Etudiant etudiant = new Etudiant(arg[0]);
                etudiant.travailler();
                etudiant.seReposer();            }
}
```

Résultat de l'exécution

```
Output - etudiant (run)
run:
sbihi se met au travail !
sbihi se repose
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCICE N°3 :

```
import java.util.Scanner;
public class Scalaire {
   private static Scanner scanner = new Scanner(System.in);
   public static void main(String[] args) {
      int nMax = 10;
      int n = 0;
      while (n < 1 || n > nMax) {
         System.out.print("Quelle taille pour vos vecteurs [entre 1 et "+ nMax + "] ? ");
         n = scanner.nextInt();
      }
      double [] v1 = new double [n];
      double [] v2 = new double [n];
      System.out.println("Saisie du premier vecteur :");
      for (int i = 0; i < n; i++) {
         System.out.print(" v1[" + i + "] = ");
         v1[i] = scanner.nextDouble();
      }
      System.out.println("Saisie du second vecteur :");
      for (int i = 0; i < n; i++) {
```

```
            System.out.print(" v2[" + i + "] = ");
            v2[i] = scanner.nextDouble();
        }
        // Calcul du produit scalaire
        double somme = 0.0;
        for (int i = 0; i < v1.length; i++) {
            somme += v1[i] * v2[i];
        }
        System.out.println("Le produit scalaire de v1 par v2 vaut " + somme);
    }
}
```

EXERCICE N°4

```
public class Liquide {
                public void imprimer(){
                System.out.println("je suis un liquide");
               }
 }
class Cafe extends Liquide {
 public void imprimer()
{ System.out.println("je suis du cafe");
}
}
 class Lait extends Liquide{
public void imprimer(){
System.out.println("je suis du lait");
}
 }
class Tasse {
private Liquide l;
public void AjouterLiquide (Liquide l)
{this.l = l;}
public void imprimer() { l.imprimer();
}
 }
public class Main {
public static void main(String[] args)
{
Tasse latasse;
latasse =new Tasse();
 latasse.AjouterLiquide(new Liquide());
 latasse.imprimer(); latasse=new Tasse();
latasse.AjouterLiquide(new Cafe());
latasse.imprimer();
Tasse[] lestasses;
 lestasses = new Tasse[3];
for (int i=0;i<3 ;i++)
lestasses[i].imprimer() ;
}
}
```

EXERCICE N°5

```
public class Animal
{
   public String nom;
    public String race;
    public String jeDis;
    public Animal(String a, String b, String c) {
        nom = a;
        race = b;
        jeDis = c;
    }
    public void sePresenter() {
        System.out.println("Bonjour! Je suis un(e) "+race+".");
        System.out.println("Je m'appelle "+nom+".");
        System.out.println("Je fais "+jeDis+".");
    }
}
 public class Chat extends Animal
{
        public Chat(String n) {
        super(n,"chat","miau, miau");
    }
}
 public class Chien extends Animal
{
    public Chien(String n) {
        super(n, "chien", "vau, vau");
    }
}
public class ZooApp
{
  public static Animal[] Zoo = new Animal[3];
    public static void main(String[] args) {
        Zoo[0] = new Chien("Lia");
        Zoo[1] = new Chat("Toto");
        Zoo[2] = new Chat("Sisi");
        for (int j=0; j<3; j++) Zoo[j].sePresenter();
        try{System.in.read();}
        catch(java.io.IOException e) {}
    }
}
```

EXERCICE N°6

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.*;

public class TxtParser {

    public static void main(String[] args) {
            String txtFile = "C:/monfichier.txt";
            String delimiter = ";";
            String line = null;
            StringTokenizer strToken = null;
            BufferedReader bufferReader;
            int lineID = 0;
            int fieldID = 0;
//Ouvrir le fichier txt
            bufferReader = new BufferedReader(new FileReader(txtFile));
//parcourir les lignes du fichier txt
            while ((line = bufferReader.readLine()) != null) {
                lineID++;
//Parcourir les champs séparés par delimiter
                strToken = new StringTokenizer(line, delimiter);
                while (strToken.hasMoreTokens()) {
                    fieldID++;
                    System.out.println("Ligne " + lineID + " / champs " + fieldID
                            + " : " + strToken.nextToken());
                }
                fieldID = 0;
        }
    }
}
```

EXERCICE N°7

```
public class Factorielle
{
        static long fact (int n)
        {
                if (n==0) return 1;
                return n*fact(n-1);
        }
        public static void main (String [] args)
        {
                System.out.println("5!="+fact(5));
        }
}
```

EXERCICE N°8

L'interface IPersonne :

```
package exercices;

/**
 *
 * @author Sbihi
 */
public interface IpPersonne {
public void affiche();
public double calculerSalaire ();
}
```

La classe Personne :

```
package exercices;
/** * @author Sbihi */
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;
public class Personne implements IpPersonne{
private int id;
private String nom;
private String prenom;
private Date dateNais;
private double salaire;
private Profile profil;
private static int comp ;
public Personne(String nom,String prenom,Date dateNais ,Double salairee,Profile profil){
comp++ ;
this.id = comp;
this.nom = nom;
this.prenom = prenom;
this.dateNais = dateNais;
this.salaire = salairee;
this.profil = profil;
}
public void affiche() {
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);
String date = df.format(dateNais);
System.out.println("Je suis le "+this.profil.getLibelle() +" "+nom+" "+prenom+ " né le "
+date+ " mon salaire est "+calculerSalairee()+ " dh") ;
}
public double calculerSalairee() {
if(this.profil.getCode().equals("DG"))return this.salaire= this.salaire +
0.2*this.salaire;
if(this.profil.getCode().equals("EMP"))return this.salaire= this.salaire +
0.1*this.salaire;
return this.salaire;
}
public double calculerSalaire() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

La classe Profile

```
package exercices;
/**
 * @author Sbihi
 */
public class Profile {
 private int id;
private String  code;
private String libelle;
private static int comp;
public Profile(String code, String libelle) {
comp++;
this.id = comp;
this.code = code;
this.libelle = libelle;
}
public String getCode() {
return code;
}
public String getLibelle() {
return libelle;
}
}
```

La classe Test :

```
package exercices;

/**
 *
 * @author Sbihi
 */
import ...
public class Test {
public static void main(String[] args) {
Profile profiles[] = new Profile[2];
profiles[0] = new Profile("EMP","Employé");
profiles[1] = new Profile("DG","Directeur Géneral");
Personne personnes[] = new Personne[3];
personnes[0] = new Personne("ELALAOUI", "Mohamed",new
Date("1986/04/04") ,9000.0, profiles[0]);
personnes[1] = new Personne("DOURID", "Raouan",new
Date("1960/03/24") , 10000.00, profiles[1] );
personnes[2] = new Personne("ALAOUI", "Sara",new
Date("1940/09/11") , 8200.00, profiles[0] );
for (Personne per : personnes){
per.affiche();
}
}
}
```

Résultat d'exécution :

```
Output
Exercices (run)   Exercices (run) #2
run:
Je suis le Employé ELALAOUI Mohamed né le 4 avril 1986 mon salaire est 9900.0 dh
Je suis le Directeur Géneral NEHDI Raouan né le 24 mars 1960 mon salaire est 12000.0 dh
Je suis le Employé SBIHI Sara né le 11 septembre 1940 mon salaire est 9020.0 dh
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCICE N°9

L'interface IOperation:

```java
package complexe;

/**
 *
 * @author Sbihi
 */
public interface IOperation {

public Object plus (Object obj);
public Object moins (Object obj);
}
```

L'interface IAffichage:

```java
package complexe;

/**
 *
 * @author Sbihi
 */
public interface IAffiche {
public String affiche ();
}
```

La classe Complexe
:

```
package complexe;

/**
 *
 * @author Sbihi
 */
public class Complexe implements IOperation,IAffiche{
private double im ;
private double re;
public Complexe ( double re , double im){
this.re=re;
this.im=im;
}
public Object plus(Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re + com.re, this.im + com.im);
}
public Object moins (Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re - com.re, this.im - com.im);
}
public String affiche() {
if(this.im > 0) return this.re +" + "+this.im+" i";
else if(this.im < 0) return this.re +" "+this.im+" i";
else return this.re+"";
}
}
```

La classe Reel :

```
package complexe;

/**...*/
public class Reel implements IOperation,IAffiche{
private double x;
public Reel(double x){
this.x=x;
}
public Object moins(Object obj) {
Reel reel = (Reel) obj;
return new Reel(this.x - reel.x);
}
public Object plus(Object obj) {
Reel reel = (Reel) obj;
return new Reel(this.x - reel.x);
}
public double plus(double ... x) {
double som = this.x;
for(Double d : x)
som += d;
return som;
}
public double moins(double ... x) {
double s = this.x;
for(Double d : x)
s -= d;
return s;
}
public String affiche() {
return (x+"");
}
}
```

La classe Test :

```
package complexe;

/**
 *
 * @author Sbihi
 */

public class Test {
public static void main (String[] args){
Complexe c1 = new Complexe(2,2) ;
Complexe c2 = new Complexe(-3,4) ;
Complexe c3, c4 ;
Reel r1=new Reel(3.4);
Reel r2=new Reel(3.89);
Reel r3;
c3 =(Complexe)(c1.plus(c2));
c4=(Complexe)(c1.moins(c2));
System.out.println("c2 + c1 = " +c3.affiche());
System.out.println("c2 - c1 = " +c4.affiche());
r3=(Reel)(r1.plus(r2));
System.out.println("r1 + r2 = " +r3.affiche());
r3 = new Reel(100);
System.out.println("{1} = "+r3.plus(1,3,4));
System.out.println("{2} = "+r3.plus(1,3,4,6,9));
System.out.println("{3} = "+r3.moins(22,3,4,6,9));
System.out.println("{4} = "+r3.moins(2,4,6,9));
}
}
```

Résultat d'exécution :

```
Output
Exercices (run)    Exercices (run) #2
run:
c2 + c1 = -1.0 + 6.0 i
c2 - c1 = 5.0 -2.0 i
r1 + r2 = -0.4900000000000002
{1} = 108.0
{2} = 123.0
{3} = 56.0
{4} = 79.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCICE N°10

L'interface IOperation:

```
package complexeab;

/**
 *
 * @author Sbihi
 */
public interface IOperation {
public Object plus (Object obj);
public Object moins (Object obj);
}
```

La classe Affichage :

```
package complexeab;

/**
 *
 * @author Sbihi
 */
public abstract  class Affichage {
public abstract String affiche();
}
```

La classe Complexe :

```
public class Complexe extends Affichage implements IOperation {
private double im ;
private double re;
public Complexe (double re , double im){
this.re=re;
this.im=im;
}
public Object plus(Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re + com.re, this.im + com.im);
}
public Object moins (Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re - com.re, this.im - com.im);
}
public String affiche() {
if(this.im > 0) return this.re +" + "+this.im+" i";
else if(this.im < 0) return this.re +" "+this.im+" i";
else return this.re+"";
}
}
```

```java
public class Complexe extends Affichage implements IOperation {
private double im ;
private double re;
public Complexe (double re , double im){
this.re=re;
this.im=im;
}
public Object plus(Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re + com.re, this.im + com.im);
}
public Object moins (Object c){
Complexe com = (Complexe) c;
return new Complexe (this.re - com.re, this.im - com.im);
}
public String affiche() {
if(this.im > 0) return this.re +" + "+this.im+" i";
else if(this.im < 0) return this.re +" "+this.im+" i";
else return this.re+"";
}
}
```

La classe Reel :

```java
package complexeab;

/**
 *
 * @author Sbihi
 */
public class Reel extends Affichage implements IOperation{
private double x;
public Reel(double x){
this.x=x;
}
public Object moins(Object obj) {
Reel reel = (Reel) obj;
return new Reel(this.x - reel.x);
}
public Object plus(Object obj) {
Reel reel = (Reel) obj;
return new Reel(this.x - reel.x);
}
public String affiche() {
return x+"";
}
}
```

La classe Test :

```
package complexeab;

/**
 *
 * @author Sbihi
 */

public class Test {
public static void main (String[] args){
Complexe c1 = new Complexe(2,2) ;
Complexe c2 = new Complexe(-3,4) ;
Complexe c3,c4 ;
Reel r1=new Reel(3.4);
Reel r2=new Reel(3.89);
Reel r3;
c3 =(Complexe)(c1.plus(c2));
c4=(Complexe)(c1.moins(c2));
System.out.println("c2 + c1 = " +c3.affiche());
System.out.println("c2 - c1 = " +c4.affiche());
r3=(Reel)(r1.plus(r2));
System.out.println("r1 + r2 = " +r3.affiche());
}
}
```

Résultat d'exécution :

```
Output - Exercices (run)
run:
c2 + c1 = -1.0 + 6.0 i
c2 - c1 = 5.0 -2.0 i
r1 + r2 = -0.4900000000000002
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCICE N°11

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import java.awt.event.*;
public class Exercice {
  private int count = 0;
  private JLabel label;
  private JButton buttonPlus;
  private JButton buttonMoins;
  public Exercice() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());
    buttonPlus = new JButton("Incrémenter");
    buttonMoins = new JButton("Décrémenter");
    buttonPlus.addActionListener(new MyActionListener());
    buttonMoins.addActionListener(new MyActionListener());
    label = new JLabel("0", SwingConstants.CENTER);
    JPanel pane = new JPanel(new BorderLayout());
    JPanel buttons = new JPanel(new GridLayout(0, 1));
    buttons.add(buttonPlus);
    buttons.add(buttonMoins);
    pane.add(buttons, BorderLayout.WEST);
    pane.add(label, BorderLayout.CENTER);
    frame.getContentPane().add(pane, BorderLayout.CENTER);
    frame.pack();
    frame.show();
  }
  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }
```

```
  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }
private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("nnnnnnnnnn");
      if(event.getSource() == buttonPlus) {
        count++; System.out.println("nnnnnnnnnn");
      } else {
        count--;
      }
      label.setText(Integer.toString(count));
    }
  }
public static void main(String[] args) {
     new Exercice();
 System.out.println("eeeeeeeee");
//System.exit(0);
}
}
```

EXERCICE N°12

```
package exercices;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JSeparator;
public class Exercice2 {
 private JMenuItem item1, item2, item3, item4, item5;
  private JLabel label;
  public Exercice2() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());
    JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("File");
    JMenu submenu = new JMenu("Property");
    item1 = new JMenuItem("New");
    item2 = new JMenuItem("Open");
    item3 = new JMenuItem("Save");
    item4 = new JMenuItem("Big");
    item5 = new JMenuItem("Small");
    MyActionListener listener = new MyActionListener();
    item1.addActionListener(listener);
    item2.addActionListener(listener);
    item3.addActionListener(listener);
    item4.addActionListener(listener);
    item5.addActionListener(listener);
    menu.add(item1);     menu.add(item2);     menu.add(item3); menu.add(new JSeparator()); menu.add(submenu);
    submenu.add(item4);     submenu.add(item5);
    menuBar.add(menu);
    label = new JLabel("-");
    frame.setJMenuBar(menuBar);
    frame.getContentPane().add(label);
    frame.pack();   frame.show();
  }
```

```
  ,
  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      label.setText(((JMenuItem)event.getSource()).getText());
    }
  }

  public static void main(String[] args) {
    new Exercice2();
  }
}
```

EXERCICE N°13

```
package exercices;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JWindow;
public class exercice3 {
private JTextArea text, area;
public exercice3() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());
    text = new JTextArea();
    text.setEditable(false);
    JWindow window = new JWindow(frame);
    JPanel panel = new JPanel(new BorderLayout());
    panel.add(text, BorderLayout.CENTER);
    JScrollPane scroll = new JScrollPane(panel);
    window.getContentPane().add(scroll);
    JPanel panel2 = new JPanel(new BorderLayout());
    area = new JTextArea();
    panel2.add(area, BorderLayout.CENTER);
    JPanel panel3 = new JPanel(new BorderLayout());
    JButton button = new JButton("Envoyer");
    panel3.add(new JScrollPane(panel2), BorderLayout.CENTER);
    panel3.add(button, BorderLayout.SOUTH);
    button.addActionListener(new MyActionListener());
    frame.getContentPane().add(panel3);
    frame.setSize(200, 200);
    window.setSize(200, 200);
    frame.setLocation(0, 0);
    window.setLocation(400, 0);
    frame.show();    window.show();
  }
```

```java
  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      text.append(area.getText() + "\n");
      area.setText("");
    }
  }

  public static void main(String[] args) {
    new exercice3();
  }
}
```

EXERCICE N°14

```java
package exercices;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class exercice4  extends JPanel {
    private JTextField textField;
    private JLabel label;
    private JSlider redSlider, greenSlider, blueSlider;
    private JRadioButton centerRadio, leftRadio, rightRadio;
    public exercice4() {
        super(new BorderLayout());
        // premier panel : pour modifier le texte
        JPanel textPanel = new JPanel(new GridLayout(2, 1));
        textField = new JTextField("text" , 20);
        JButton resetButton = new JButton("Reset");
        JButton updateButton = new JButton("Update");
        JPanel subTextPanel = new JPanel(new FlowLayout()); // contient les boutons "reset" et "update"
        subTextPanel.add(resetButton);
        subTextPanel.add(updateButton);
        textPanel.add(textField);
        textPanel.add(subTextPanel);
        resetButton.addActionListener(new ResetActionListener());
        updateButton.addActionListener(new UpdateActionListener());
        // second panel : pour modifier la couleur
        JPanel colorPanel = new JPanel(new BorderLayout());
        redSlider = new JSlider(SwingConstants.HORIZONTAL, 0, 255, 128);
        greenSlider = new JSlider(SwingConstants.HORIZONTAL, 0, 255, 128);
        blueSlider = new JSlider(SwingConstants.HORIZONTAL, 0, 255, 128);
        JPanel colorTextPanel = new JPanel(new GridLayout(3, 1));
        JPanel colorSliderPanel = new JPanel(new GridLayout(3, 1));
        colorTextPanel.add(new JLabel("Red:"));
        colorTextPanel.add(new JLabel("Green:"));
        colorTextPanel.add(new JLabel("Blue:"));
        colorSliderPanel.add(redSlider);
        colorSliderPanel.add(greenSlider);
        colorSliderPanel.add(blueSlider);
        colorPanel.add(colorTextPanel, BorderLayout.WEST);
        colorPanel.add(colorSliderPanel, BorderLayout.CENTER);
```

```
        redSlider.addChangeListener(changeListener);
        greenSlider.addChangeListener(changeListener);
        blueSlider.addChangeListener(changeListener);
        // troisieme panel : pour modifier l'alignement
        JPanel alignPanel = new JPanel(new GridLayout(3, 1));
        centerRadio = new JRadioButton("Center");
        centerRadio.setSelected(true);
        leftRadio = new JRadioButton("Left");
        rightRadio = new JRadioButton("Right");
        ButtonGroup alignGroup = new ButtonGroup();
        alignGroup.add(centerRadio);
        alignGroup.add(leftRadio);
        alignGroup.add(rightRadio);
        alignPanel.add(centerRadio);
        alignPanel.add(leftRadio);
        alignPanel.add(rightRadio);
        centerRadio.addChangeListener(changeListener);
        leftRadio.addChangeListener(changeListener);
        rightRadio.addChangeListener(changeListener);
        // tabbedPane pour les trois panels du haut
        JTabbedPane tabbedPane = new JTabbedPane();
        tabbedPane.add(textPanel, "Text");
        tabbedPane.add(colorPanel, "Color");
        tabbedPane.add(alignPanel, "Align");
        add(tabbedPane, BorderLayout.NORTH);
        // label du bas : texte en couleur
        label = new JLabel();
        add(label, BorderLayout.CENTER);
        updateCurrentColor();
        updateCurrentText();
        updateCurrentAlign();
    }
    private void updateCurrentColor() {
```

```
    private void updateCurrentColor() {
        label.setForeground(new Color(redSlider.getValue(), greenSlider.getValue(), blu
    }
    private void updateCurrentText() {
        label.setText(textField.getText());
    }
    private void updateCurrentAlign() {
        int align;
        if(centerRadio.isSelected()) align = SwingConstants.CENTER;
        else if(leftRadio.isSelected()) align = SwingConstants.LEFT;
        else if(rightRadio.isSelected()) align = SwingConstants.RIGHT;
        else align = SwingConstants.CENTER;

        label.setHorizontalAlignment(align);
    }
    private class ResetActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            textField.setText("");
        }
    }
     private class UpdateActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            updateCurrentText();
        }
    }
    private class AllChangeListener implements ChangeListener {
        public void stateChanged(ChangeEvent e) {
            updateCurrentColor();
            updateCurrentAlign();
        }
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Exemple");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new BorderLayout());
        frame.getContentPane().add(new exercice4(), BorderLayout.CENTER);
        frame.pack();
        frame.show();
    }
```

EXERCICE N°15

```
package exercices;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
public class exercice5 {
  private static final String[] checks = new String[] {
        "reactor left", "reactor right", "control gear",
        "wheel left", "wheel right", "lights",
        "doors", "main engine", "secondary engine"
    };
    private JPanel managingPanel, viewingPanel;
    private JSlider toWarningSlider, toErrorSlider;
    private JLabel okLabel, warningLabel, errorLabel;
    private JCheckBox[] boxes;
    private int toWarning;
    private int toError;

    public exercice5() {
        toWarning = 3;
        toError = 6;
        viewingPanel = new JPanel(new GridLayout(3, 1));
        okLabel = new JLabel("Ok");
        warningLabel = new JLabel("Warning");
        errorLabel = new JLabel("Error");
        viewingPanel.add(errorLabel);
        viewingPanel.add(warningLabel);
        viewingPanel.add(okLabel);
        managingPanel = new JPanel(new GridLayout(2, 1));
        JPanel checkPanel = new JPanel(new GridLayout(0, 3));
        boxes = new JCheckBox[checks.length];
        for(int i = 0; i < checks.length; i++) {
            JCheckBox box = new JCheckBox(checks[i]);
            boxes[i] = box;
            box.setSelected(true);
            checkPanel.add(box);
        }
        toWarningSlider = new JSlider(SwingConstants.HORIZONTAL, 1, checks.length + 1, 1);
        toWarningSlider.setSnapToTicks(true);
```

```
    toWarningSlider = new JSlider(SwingConstants.HORIZONTAL, 1, checks.length + 1,
    toWarningSlider.setSnapToTicks(true);
    toWarningSlider.setPaintTicks(true);
    toWarningSlider.setMajorTickSpacing(checks.length);
    toWarningSlider.setMinorTickSpacing(1);
    toWarningSlider.setPaintLabels(true);
    toErrorSlider = new JSlider(SwingConstants.HORIZONTAL, 1, checks.length + 1, 1)
    toErrorSlider.setSnapToTicks(true);
    toErrorSlider.setPaintTicks(true);
    toErrorSlider.setMajorTickSpacing(checks.length);
    toErrorSlider.setMinorTickSpacing(1);
    toErrorSlider.setPaintLabels(true);
    toWarningSlider.setValue(toWarning);
    toErrorSlider.setValue(toError);
    JPanel slidersPanel = new JPanel(new GridLayout(2, 1));
    slidersPanel.add(toWarningSlider);
    slidersPanel.add(toErrorSlider);
    JPanel slidersTextPanel = new JPanel(new GridLayout(2, 1));
    slidersTextPanel.add(new JLabel("Warning when reaching:"));
    slidersTextPanel.add(new JLabel("Error when reaching:"));
    JPanel downPanel = new JPanel(new BorderLayout());
    downPanel.add(slidersTextPanel, BorderLayout.WEST);
    downPanel.add(slidersPanel, BorderLayout.CENTER);
    managingPanel.add(checkPanel);
    managingPanel.add(downPanel);
    ChangeListener checkChangeListener = new CheckChangeListener();
    for(int i = 0; i < checks.length; i++) {
        boxes[i].addChangeListener(checkChangeListener);
    }
    ChangeListener sliderChangeListener = new SliderChangeListener();
    toWarningSlider.addChangeListener(sliderChangeListener);
    toErrorSlider.addChangeListener(sliderChangeListener);
    updateLabels();
}

public JPanel getManagingPanel() {
    return managingPanel;
}
```

```java
public JPanel getManagingPanel() {
    return managingPanel;
}
public JPanel getViewingPanel() {
    return viewingPanel;
}
private void updateLabels() {
    int count = 0;
    for(int i = 0; i < boxes.length; i++) {
        if(!boxes[i].isSelected()) count++;
    }
    if(count < toWarning) {
        okLabel.setForeground(Color.green);
        warningLabel.setForeground(Color.lightGray);
        errorLabel.setForeground(Color.lightGray);
    } else if(count < toError) {
        okLabel.setForeground(Color.lightGray);
        warningLabel.setForeground(Color.orange);
        errorLabel.setForeground(Color.lightGray);
    } else {
        okLabel.setForeground(Color.lightGray);
        warningLabel.setForeground(Color.lightGray);
        errorLabel.setForeground(Color.red);
    }
}
private class CheckChangeListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        updateLabels();
    }
}
```

```java
    private class SliderChangeListener implements ChangeListener {
        public void stateChanged(ChangeEvent e) {
            JSlider slider = (JSlider)e.getSource();
            if(slider == toWarningSlider) {
                toWarning = slider.getValue();
                if(toWarning > toError) {
                    toError = toWarning;
                    toErrorSlider.setValue(toError);
                }
            } else {
                toError = slider.getValue();
                if(toError < toWarning) {
                    toWarning = toError;
                    toWarningSlider.setValue(toWarning);
                }
            }
            updateLabels();
        }
    }
    public static void main(String[] args) {
        exercice5 exo = new exercice5();
        JFrame managingFrame = new JFrame("Managing");
        managingFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        managingFrame.getContentPane().setLayout(new BorderLayout());
        managingFrame.getContentPane().add(exo.getManagingPanel(), BorderLayout.CENTER);
        JFrame viewingFrame = new JFrame("Viewing");
        viewingFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        viewingFrame.getContentPane().setLayout(new BorderLayout());
        viewingFrame.getContentPane().add(exo.getViewingPanel(), BorderLayout.CENTER);
        managingFrame.pack();
        managingFrame.show();
        viewingFrame.pack();
        viewingFrame.show();
    }
}
```

EXERCICE N°16

1. Création des interfaces

```java
package formeGeo;
/**
 * @author Sbihi
 */
public interface FormGeo {
// choisis une couleur
void setColor( int color) ;
// retourner la couleur
int getColor() ;
// dessiner
void draw() ;
}
```

2. Classe graphique

```
package formeGeo;
public class Triangle implements FormGeo , deplacement{
private Segment segment1 ;
private Segment segment2 ;
private Segment segment3 ;
private int aColor ;
Triangle( Point p1 , Point p2 , Point p3 , int aColor ){
this.segment1 = new Segment( p1 , p2 , aColor ) ;
this.segment2 = new Segment( p2 , p3 , aColor ) ;
this.segment3 = new Segment( p3 , p1 , aColor ) ;
this.aColor = aColor ;
}
public void setColor( int color){
this.aColor = aColor ;
}
public int getColor(){
return aColor ;
}
public void draw(){
segment1.draw() ;
segment2.draw() ;
segment3.draw() ;
}
public void translate( int dx , int dy ){
segment1.translate( dx , dy ) ;
segment2.translate( dx , dy ) ;
segment3.translate( dx , dy ) ;
}
}
```

3. Classe Point

```
package formeGeo;

public class Point implements deplacement{
private int x ;
private int y ;
public Point( int x , int y ){
this.x = x ;
this.y = y ;
}
public Point( Point p ){
this.x = p.x ;
this.y = p.y ;
}
public int getX(){
return x ;
}
public int getY(){
return y ;
}
public void setX( int x ){
this.x = x ;
}
public void setY( int y ){
this.y = y ;
}
public String toString(){
return( "(" + x + "," + y + ")" ) ;
}
public void translate( int dx , int dy ){
x = x + dx ;
y = y + dy ;
}
}
```

4. Classe Segment

```java
package formeGeo;

/**
 *
 * @author Sbihi
 */
public class Segment implements FormGeo , deplacement{
private Point beginning ;
private Point ending ;
private int aColor ;
Segment( Point beginning , Point ending , int aColor){
this.beginning = new Point( beginning ) ;
this.ending = new Point( ending ) ;
this.aColor = aColor ;
}

    Segment(java.awt.Point p3, java.awt.Point p4, int i) {
        throw new UnsupportedOperationException("Not yet implemented");
    }
public int getColor(){
return aColor ;
}
public void setColor(int aColor){
this.aColor = aColor ;
}
public void translate ( int dx , int dy ){
beginning.translate ( dx , dy ) ;
ending.translate ( dx , dy ) ;
}
public void draw(){
System.out.println( aColor + " debut = " + beginning + " fin = " + ending ) ;
}
}
```

5. Classe Triangle

```
package formeGeo;
public class Triangle implements FormGeo , deplacement{
private Segment segment1 ;
private Segment segment2 ;
private Segment segment3 ;
private int aColor ;
Triangle( Point p1 , Point p2 , Point p3 , int aColor ){
this.segment1 = new Segment( p1 , p2 , aColor ) ;
this.segment2 = new Segment( p2 , p3 , aColor ) ;
this.segment3 = new Segment( p3 , p1 , aColor ) ;
this.aColor = aColor ;
}
public void setColor( int color){
this.aColor = aColor ;
}
public int getColor(){
return aColor ;
}
public void draw(){
segment1.draw() ;
segment2.draw() ;
segment3.draw() ;
}
public void translate( int dx , int dy ){
segment1.translate( dx , dy ) ;
segment2.translate( dx , dy ) ;
segment3.translate( dx , dy ) ;
}
}
```

EXERCICE N°17

1. Création des interfaces

```
package formateur;

/**
 *
 * @author Sbihi
 */
public interface Boite {
public int largeur();
public String toString();
}
```

2. Classe Espace

```
 */
package formateur;

/**
 *
 * @author Sbihi
 */

public class BoiteEspace implements Boite{
public BoiteEspace(){
;
}
public int largeur(){
return 1;
}
public String toString(){
return " ";
}
}
```

3. Classe BoiteMot

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package formateur;

/**
 *
 * @author Sbihi
 */
public class BoiteMot implements Boite{
String mot;
public BoiteMot(String contenu){
mot = contenu;
}
public int largeur(){
return mot.length();
}
public String toString(){
return mot;
}
}
```

4. Classe BoiteComposite

```
package formateur;

/**
 *
 * @author Sbihi
 */
public class BoiteComposite implements Boite{
Boite boites[];
int num;
public BoiteComposite(){
boites = new Boite[10];
num = 0;
}
public int largeur(){
int i;
int l;
l = 0;
for(i = 0; i < num; i++)
l = l + boites[i].largeur();
return l;
}
public boolean vide(){
return num == 0;
}
private void redimensionne(int n){
Boite[] nboites;
int i;
nboites = new Boite[n];
for(i = 0; i < num; i++) {
nboites[i] = boites[i];
}
boites = nboites;
}
public void accumule(Boite b){
if(num >= boites.length)
redimensionne(2 * num);
boites[num] = b;
num = num + 1;
}
public String toString(){
int i;
StringBuffer s;
s = new StringBuffer();
for(i = 0; i < num; i++)
s.append(boites[i].toString());
return s.toString();
}
}
```

Classe Formateur

```
package formateur;
public class Formateur
{
private Parseur parseur;
private Boite[] boites;
private int num;
private BoiteComposite boite;
public Formateur(String filename){
parseur = new Parseur(filename);
num = 0;
boites = new Boite[10];
boite = new BoiteComposite();
}
public void lit(){
String s;
do {
s = parseur.suivant();
if(s == null || s.equals("")) {
nouveauParagraphe();
if(s == null)
break;
} else {
BoiteMot m;
m = new BoiteMot(s);
if(!boite.vide())
boite.accumule(new BoiteEspace());
boite.accumule(m);
}
} while(s != null);
}
public void imprime(){
for(int i = 0; i < num; i++) {
System.out.println(boites[i]) ;
}
}
```

```
private void nouveauParagraphe(){
if(!boite.vide()) {
accumule(boite);
boite = new BoiteComposite();
}
}
private void redimensionne(int n){
Boite[] nboites;
int i;
nboites = new BoiteComposite[n];
for(i = 0; i < num; i++) {
nboites[i] = boites[i];
}
boites = nboites;
}
private void accumule(Boite b){
if(num >= boites.length)
redimensionne(2 * num);
boites[num] = b;
num = num + 1;
}
}
```

Bibliographie

- Lemay L, Le Programmeur JAVA 2, Campus Press.
- Bailly C, Chaline J.F., Ferry H.C & al, Les langages orientés objets, Cepadues édi
- Vincent Granet, 20

Webographie

- https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-flux-d-entree-sortie
- http://blog.paumard.org/cours/java/chap07-heritage-interface-interface.html
- http://cedric.cnam.fr/~farinone/IAGL/EntSort.pdf
- http://jacques.bapst.home.hefr.ch/pr1/cours/pr1_08.pdf
- http://java.sun.com