# Introduction à la Programmation par Objets: pratique avec Java

Georges Edouard KOUAMOU

Ecole Nationale Supérieure Polytechnique

Université de Yaoundé I

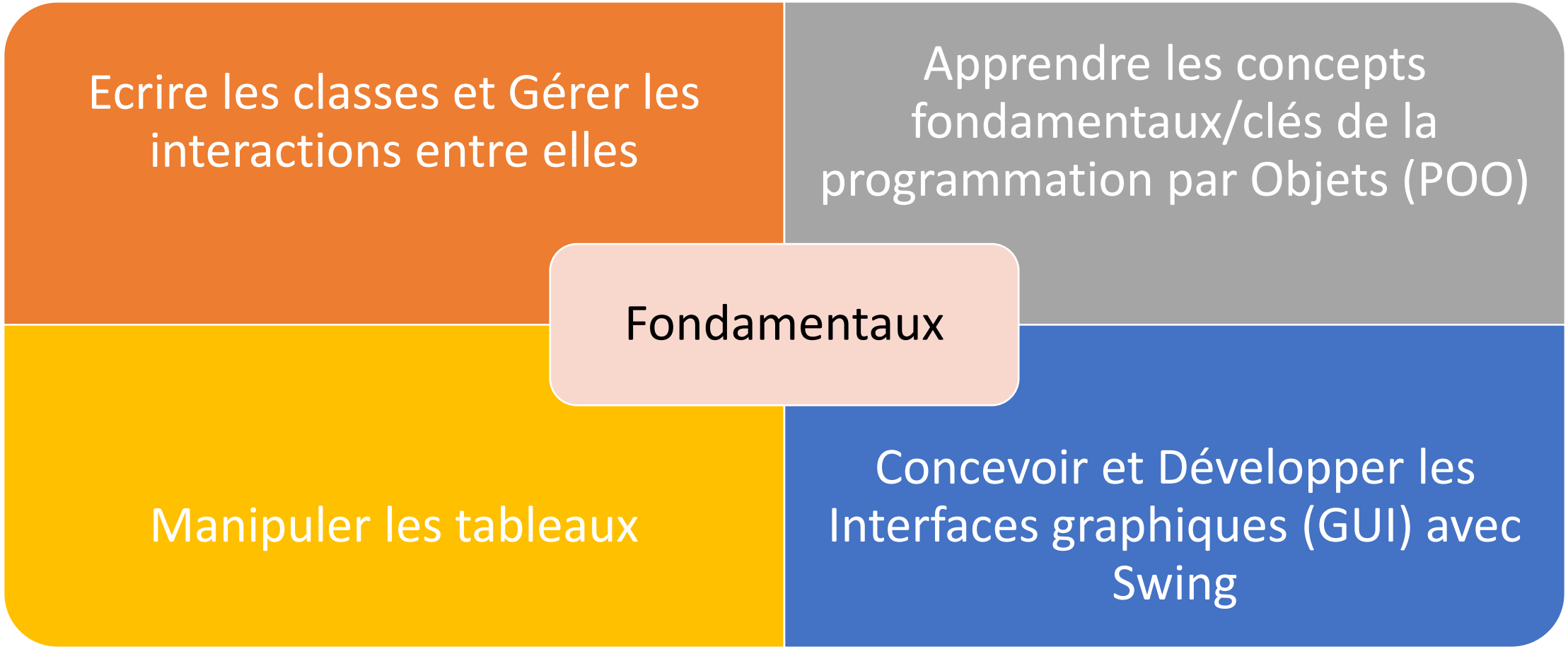
# Généralités (1)

- **Programmer**
  - Inclure dans un programme (cinéma, radio, télévision) ou Élaborer un programme.
  - Prévoir, planifier
  - Créer un programme (en informatique)
- Programme (informatique)
  - Ensemble d'opérations destinées à être exécutées par un ordinateur
  - Liste d'ordres indiquant à 1 ordinateur ce qu'il doit faire
- **Programmation** ou codage
  - Ensemble d'activités qui permettent d'écrire un programme informatique
  - Pour cela, On sert d'un langage de programmation
- **Langage de programmation**
  - Définit la manière de donner des ordres à 1 ordinateur
  - Similaires aux langues vivantes => **Vocabulaire**, **grammaire**, règles **sémantiques**

# Généralités (2)

- Programmation par objets
  - Paradigme de programmation dont les briques de base sont les objets
  - Pionier: Ole-Johan Dahl et Kristen Nygaard (1960)
- Objets
  - Une chose tangible/physique, une abstraction
- Exemple:
  - abstraction: un nombre complexe, une vecteur
  - Physique: une chaise, un véhicule

# Objectifs du cours



# Contenu

- Concepts fondamentaux de la POO
- Présentation de Java
- Notion de classes et d’objets
- Types de données, Opérateurs et instructions de controle
- Héritage, packages et interfaces
- Les exceptions
- Introduction à Swing

# Références

- Java tutorials
  - Available at https://docs.oracle.com/javase/tutorial/java
- Herbert Schildt. Java ™: A Beginner’s Guide, Sixth Edition. Oracle

# Concepts de la programmation par objets

- Objet
  - Un objet est une entité logicielle dotée d'états et de comportements associés.
  - Les objets logiciels sont souvent utilisés pour modéliser les objets du monde réel que vous trouvez dans la vie quotidienne.
- Classe
  - Une classe est un plan directeur ou un prototype à partir duquel des objets sont créés. Introduire le principe de l'encapsulation
- Polymorphisme
  - Généralement, la capacité de différentes classes d'objets à répondre au même message de différentes manières, spécifiques à une classe.
  - On utilise des méthodes polymorphes qui ont un nom mais différentes implémentations pour différentes classes.

# Concepts de la programmation par objets

- Héritage
  - L'héritage est le processus par lequel un objet peut acquérir les propriétés d'un autre objet.
  - L'héritage fournit un mécanisme puissant et naturel pour organiser et structurer votre logiciel.
- Interface
  - Une interface est un contrat entre une classe et le monde extérieur.
  - Lorsqu'une classe implémente une interface, elle promet de fournir le comportement publié par cette interface.
- Paquettage (package)
  - Un paquet est un espace de noms pour organiser les classes et les interfaces de manière logique.
  - Le fait de placer votre code dans des packages facilite la gestion de grands projets logiciels.

# Bénéfices des objets

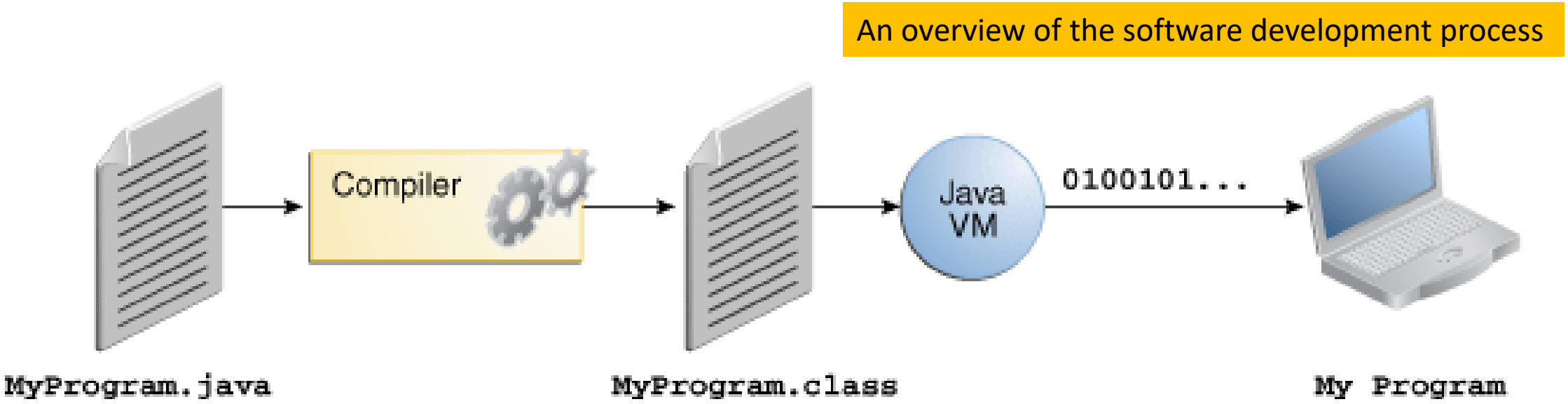
- Les **objets** du monde réel partagent deux caractéristiques: ils ont tous un état et un comportement
- La **modularité**
  - Le code source d'un objet peut être écrit et géré indépendamment du code source d'autres objets. Une fois créé, un objet peut facilement être passé à l'intérieur du système.
- **Masquage des informations**
  - En n'interagissant qu'avec les méthodes d'un objet, les détails de son implémentation interne restent cachés du monde extérieur.
- **Réutilisation de code**
  - Si un objet existe déjà (peut-être écrit par un autre développeur de logiciel), vous pouvez utiliser cet objet dans votre programme. Cela permet aux spécialistes d'implémenter / tester / déboguer des objets complexes, spécifiques à une tâche, que vous pouvez ensuite faire confiance pour exécuter dans votre propre code.
- Pluggability et facilité de débogage
  - Si un objet particulier s'avère problématique, vous pouvez simplement le supprimer de votre application et brancher un autre objet en remplacement. Ceci est analogue à la résolution de problèmes mécaniques dans le monde réel. Si un boulon se casse, vous le remplacez, pas toute la machine.

# Prise en main de l’environnement

- Installation de la JDK
- Installation de NetBeans

# A propos de JAVA

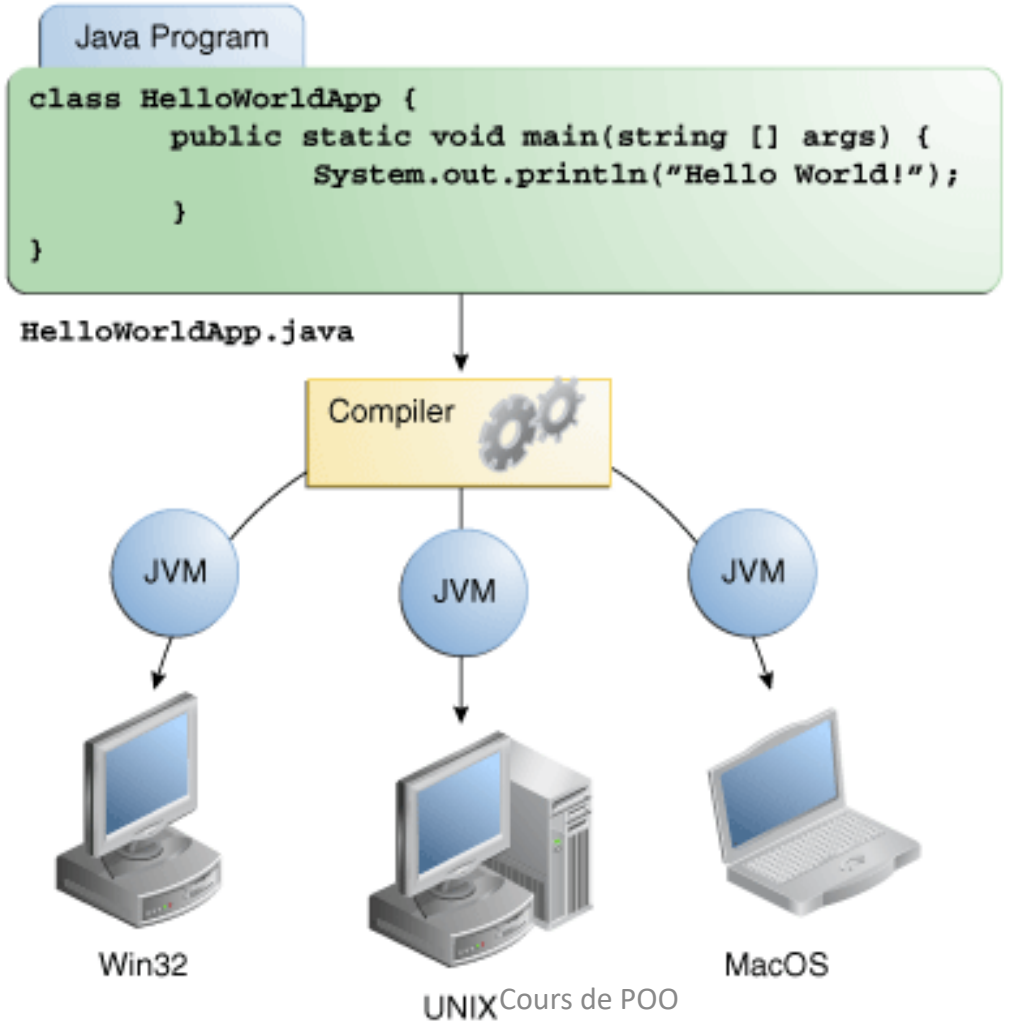
- Java est une technologie comprenant
  - Un **langage de programmation** et une **plateforme**



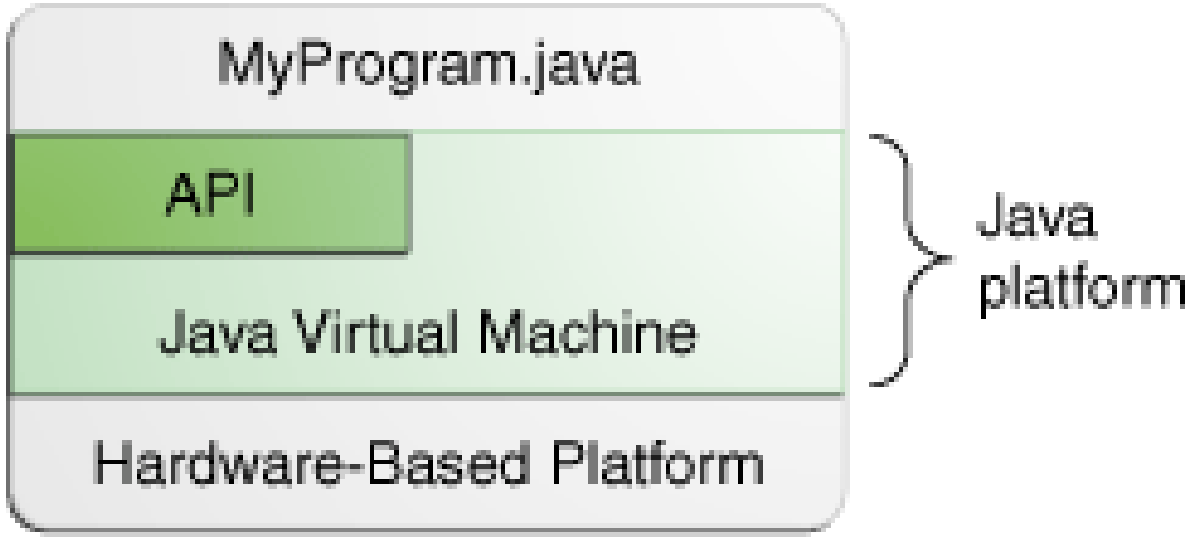
# Le langage Java

- Créé en 1995 par Sun Microsystems
  - Version actuelle Java 8, maintenu actuellement par Oracle
- est orienté objet
- est fortement typé
  - Toute variable doit être déclarée avec un type
  - Le compilateur vérifie que les utilisations des variables sont compatibles avec leur ty pe (notamment via un sous typage correct)
  - Les types sont d'une part fournis par le langage, mais également par la définition des classes
- est compilé
  - En bytecode, i.e., code intermédiaire indépendant de la machine
- est interprété
  - Le bytecode est interpété par une machine virtuelle Java

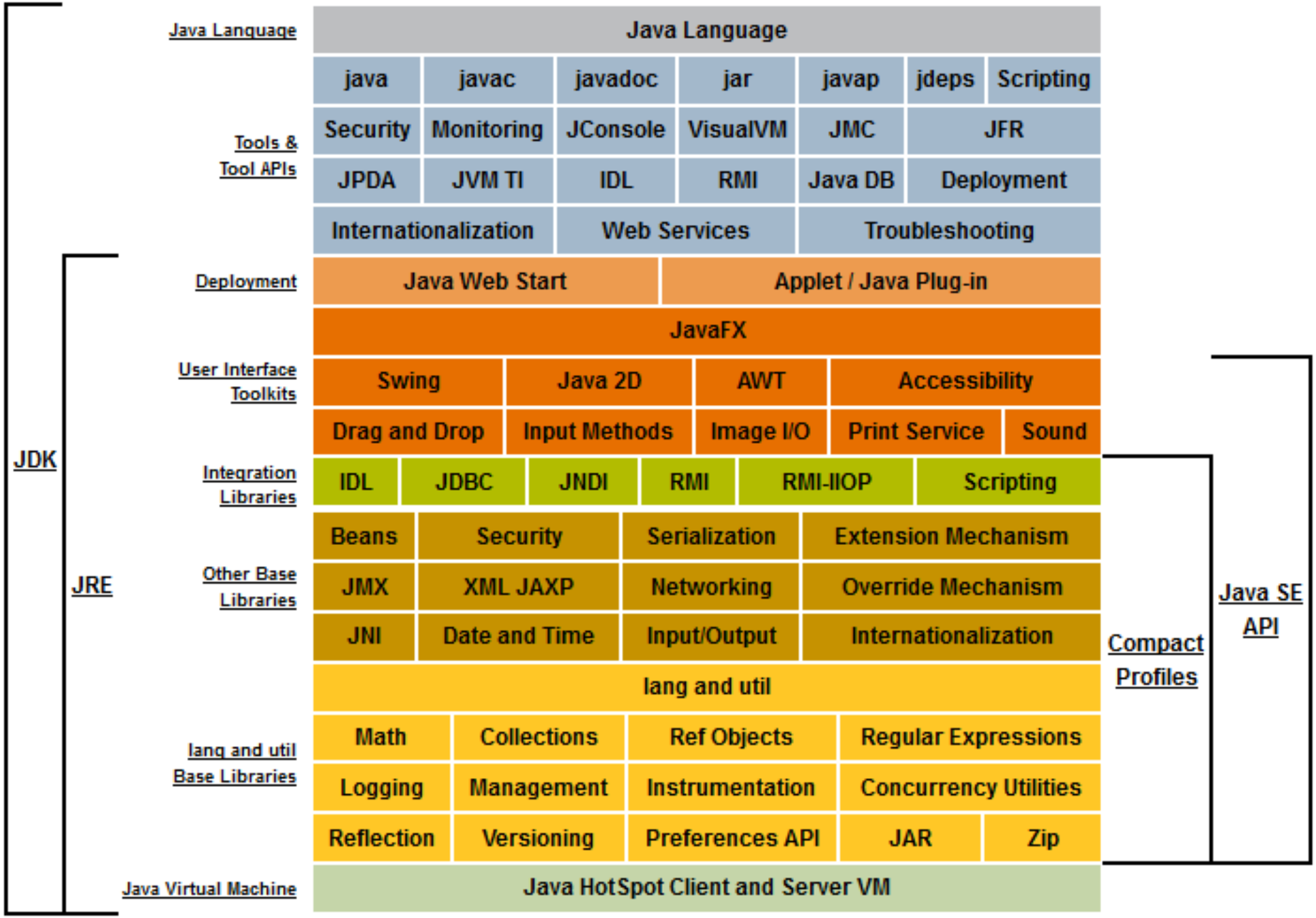
# Portabilité: Java Virtual Machine



# Java platform




- The Java platform has two components:
  - The *Java Virtual Machine*
  - The *Java Application Programming Interface* (API)
- The API is a large collection of ready-made software components that provide many useful capabilities.
  - It is grouped into libraries of related classes and interfaces; these libraries are known as packages


Java Language
Java Language
java
javac
javadoc
jar
javap
jdeps
Scripting
Tools & Tool APIs
Security
Monitoring
JConsole
VisualVM
JMC
JFR
JPDA
JVM TI
IDL
RMI
Java DB
Deployment
Internationalization
Web Services
Troubleshooting
Deployment
Java Web Start
Applet / Java Plug-in
JavaFX
User Interface Toolkits
Swing
Java 2D
AWT
Accessibility
Drag and Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC
JNDI
RMI
RMI-IIOP
Scripting
Other Base Libraries
Beans
Security
Serialization
Extension Mechanism
JMX
XML JAXP
Networking
Override Mechanism
JNI
Date and Time
Input/Output
Internationalization
lang and util
lang and util Base Libraries
Math
Collections
Ref Objects
Regular Expressions
Logging
Management
Instrumentation
Concurrency Utilities
Reflection
Versioning
Preferences API
JAR
Zip
Java Virtual Machine
Java HotSpot Client and Server VM
JDK
JRE
Compact Profiles
Java SE API

# Java

- Dans la technologie Java, on a donc besoin
  - Du **langage de programmation** et du compilateur
    - Et plein de commandes bien utiles: jar, javap, javadoc, etc
  - De la **JVM** et des **APIs** (Application Programming Interfaces) regroupées dans une « plateforme »:
    - Java SE (Java Platform, Standard Edition): Java SE 6 pour applications classiques, desktop
    - Java EE (Java Platform, Enterprise Edition): Java EE 6 pour développer et déployer des applications serveur, Web Services, etc.
    - Java ME (Java Platform, Micro Edition): J2ME pour les applications embarquées PDA, Téléphone, etc.
- Si on veut juste exécuter, il suffit du **JRE** (Java Runtime Environment) par opposition au **JDK** (Java Development Kit)

# Setting up your environment

- Getting JDK 8 or later
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- IDE
  - Netbeans: https://www.netbeans.org/index.html
  - Eclipse: https://www.eclipse.org/
- Exercise 1
  - Set up your environment, and make sure that everything is well configured
  - Try this sample code

```
public class MyFirstJavaProgram {
       public static void main(String [ ] args) {
              System.out.println("Java and the Web");
       }
}
```

# Premier exemple

- Dans un fichier de nom MyFirstJavaProgram.java
  - ***Règle: toute classe publique doit être dans un fichier qui a le même nom que la classe***
  - ***Règle: tout code doit être à l'intérieur d'une classe***

```
public class MyFirstJavaProgram {
        public static void main(String [ ] args) {
                System.out.println("Java and the Web");
        }
}
```

- Ça définit une classe, qui est une unité de compilation
- Comme il y a une méthode main, cette classe est « exécutable »

# Bases du langage

- Les variables, les opérateurs, les expressions, les instructions, blocs, contrôle de flot sont très proches de ceux du C
  - Les types primitifs ont une taille et une représentation normée
  - S'y ajoutent des spécificités syntaxiques liées à la programmation objet, aux cl asses, à l'héritage...
- Un **style de nommage** (très fortement) conseillé
  - Style « chameau » (CamelCase) pour les indentificateurs
  - Première majuscule pour les classes (class MyFirstProgram)
  - Première minuscule pour les variables/champs et les fonctions/méthodes (rayon, getRayon())
  - Tout en majuscule pour les constantes (MAX_SIZE)

# TP 1: la classe Carré

- Le carré est une figure géométrique caractérisée par
  - Le coté (seul attribut)
- Les opérations/méthodes applicables sont
  - Superficie
  - Périmètre
  - On ajoutera des méthodes spécifiques
    - Constructeur (on verra plus loin): méthode qui permet de créer les objets carré

# Notion de classe

- Une classe représente
  - une unité de compilation: la compilation d'un programme produit autant de fichier **.class** que de classes qu'il contient
  - la définition d'un type : peut servir a déclarer des variables
  - Un moule pour fabriquer/créer les objets du type ainsi défini
    - Les champs (Fields/attributs) décrivent l'état d'un objet de ce type et un ensemble d'opérations (méthodes) définit son comportement ou ses fonctionnalités
- Chaque objet de la classe
  - Dispose de **son propre état** qui sont les valeurs de ses champs
  - Répond au **même comportement** (utilise les méthodes de la classe)

# Structure d’une classe

- Une classe est définie par son nom et son package d’appartenance (espace de nom)
  - En l’absence de directive, les classes sont dans un package par défaut
- Une classe peut contenir
  - Des attributs ou champs
  - Des méthodes (de différentes natures)
    - Constructeurs
    - mutateurs
  - Des classes internes (déconseillé)
- Les membres statiques (ayant le modificateur **static**) sont des membres de classe: ils sont définis sur la classe et non sur l’objet
- Les membres non statiques (dits **d’instance**) ne peuvent exister sans un objet)

# exemple

```
public class Carre {
        //Ceci est le seul attribut
        double cote;
//Les méthodes ici
        double perimetre(){
            double p = cote*4;
             return p;
        }
        double surface() {
            return cote*cote;
        }
}
```

# Variables

- Les variables (champs des classes, variables locales aux méthodes) peuvent être:
- De type primitif. La déclaration de la variables réserves la place mémoire pour stocker sa valeur (qui dépend de son type)
- Une référence, i.e de type « objet »
  - Dans ce cas la déclaration réserve la place d'une référence (un pointeur) qui permettra d'accéder à l'endroit en mémoire où est effectivement stocké l'objet
  - Elle vaut **null** si la référence est inconnue

# Types primitifs

- Types entiers signés (représentation en complément à 2)
  - byte (octet) sur 8 bits: [-128 .. 127]
  - short sur 16 bits [-32768 .. 32767]
  - int sur 32 bits [-2147483648 .. 2147483647] (défaut pour entiers)
  - long sur 64 bits [-9223372036854775808 .. 9223372036854775807]
- Type caractère non signé (unités de code UTF16)
  - char sur 16 bits ['\u0000' .. '\uffff']
- Types à virgule flottante (représentation IEEE 754)
  - float sur 32 bits
  - double sur 64 bits (défaut pour flottants)
- Type booléen:  boolean (true ou false)

# Exercice Pratique

- Créer un projet « LicPro »
- Créer un package « figure »
- Créer une classe pour chaque figure géométrique en précisant leur caractéristiques (attributs et méthodes)
  - Carre
  - Rectangle
  - Triangle
  - Cercle
- Créer une classe Test ayant une fonction main pour valider les classes créées

# Leçon apprise

- Disposer d'un éditeur de code pour écrire les programmes
- Toute classe dans Netbeans se crée dans un projet
- Connaissance des classes d'affichage et des mots
  - System.out
  - void, class
- Déclaration des variables/attributs
  - Le type précède le nom de l'attribut
- Déclaration et écriture du corps des méthodes/opérations
- Si une méthode est déclarée avec **void**, elle ne retourne rien
- L'exécution d'un programme commence toujours dans une méthode main
- Note
  - Toujours créer une classe de Test contenant la méthode main
  - On peut déclarer plusieurs méthodes main dans un programme Java

# Leçon apprise

- L'opérateur d'instanciation : new
- Chaque classe est stocké dans un fichier  qui porte le même nom que la classe
  - Ce fichier a pour extension .java
- La structure d'une classe
  - Attribut + opérations
- Appliquer la méthode sur un objet
  - *Nom_objet*.*nom_methode*(*paramètres s'il y a lieu*)
- Note
  - Déconseiller d'écrire une classe dans une autre

# Autres types

- Tous les autres types sont des « objets » et ils sont manipulés par des références
- Ils sont définis
  - Soit dans les APIs
    - Java.lang.Object, java.lang.String, etc
  - Soit des types cachés
    - Tableau => peut contenir des types primitifs ou d'autres types d'objets
    - int[ ] tableau
    - String [ ] args
  - Soit par le programmeur

# Chaîne de caractères

- On utilise une classe particulière, nommée String, fournie dans le package java.lang.
- l'initialisation d'une chaîne de caractères s'écrit :
  - `String s = new String();` //pour une chaine vide
  - `String s2 = new String("hello world");` // pour une chaîne de valeur "hello world"
- Une fois qu'une variable est initialisée, sa valeur ne peut pas être modifiée.
- on utilise l'opérateur + pour concaténer deux chaînes de caractères
  - `String s1 = "hello" ;`
  - `String s2 = "world" ;`
  - `String s3 = s1 + " " + s2 ;` //Après ces instructions s3 vaut "hello world"
- Un ensemble de méthodes de la classe java.lang.String permettent d'effectuer des opérations ou des tests sur une chaîne de caractères

# Quelques opérations sur les String

# Tableau

- Déclaration : même syntaxe qu'en C
  - Il y'en a deux qui sont acceptées
  - int[]  mon_tableau ;
  - int  mon_tableau2[];
- Un tableau a toujours une taille fixe qui doit être précisée avant l'affectation de valeurs à ses indices
  - int[]  tab  =  new  int[20]; //tab réservera la place pour contenir 20 entiers
- la taille de ce tableau est disponible dans une variable length appartenant au tableau et accessible pour l'exemple par *tab.length*
- on accède aux éléments d'un tableau en précisant un indice entre crochets
  - tab[3] est le quatrième entier du tableau
  - un tableau de taille n stocke ses éléments à des indices allant de 0 à n-1.

# Pratique 2

- Ajouter une méthode affiche dans chacune des classes du TP1.
  - Cette méthode devra restituer la forme (carre, rectangle, triangle, cercle) suivi de chaque propriété sous le format <nom: valeur>
- Dans la classe Test, ajouter un champ tableau qui contiendra des carre (définir la taille par vous même)
  - Créer les carrés et insérer dans le tableau
  - Parcourir le tableau et afficher ses éléments

# Rappel

- **Déclaration d'une classe**

```
<modificateur> class <nom de la classe>{
   Corps de la classe
   // Les attributs
   // Les méthodes
}
```

- **Déclaration d'un attribut**

```
<modificateur> type <nom de l'attribut>;
```

# Méthodes

- Les méthodes définissent les actions, fonctionnalités, les comportements acceptés par les objets de la classe
- Chaque méthode a 2 parties
  - Sa signature est constituée de <**modificateur**> <type retour> <nom_methode>(paramètres)
  - son corps (code): constitué de blocs imbriqués

# Modificateurs

- Public
- Private
- Protected
- Final
- Static
- On peut combiner final et static avec les 3 précédents
- Les modificateurs définissent l'accessibilité à une classe, où un membre de la classe

# Accessibilité des membres

- Private: le membre (attribut/methode) est accessible uniquement dans la classe
- Protected: le membre est accessible depuis toutes les classes qui sont dans le même package et également les classes qui héritent de la classe en question
- Public: accessible depuis n'importe où
- Par défaut (pas de modificateur): le membre est accessible depuis toutes les classes qui sont dans le package de la classe en question
- Final
  - Classe: la classe ne peut être étendue, pas d'héritage possible
  - Attribut: il doit avoir une affectation unique, c'est une constante
- Static
  - Propriété de classe, pas d'objet

# Les membres statiques (static)

- Déclarés avec le mot clé static
- Attribut
  - Sa valeur est la même pour tous les objets
  - Les objets se partagent le même emplacement mémoire destiné à l'attribut
  - L'accès se fait à partir du nom de la classe ou de la référence à n'importe quel objet de la classe
- Méthodes
  - Son code ne dépend pas d'un objet de la classe
  - Son code peut être exécuté même si aucun objet existe (e.g main)
- Classes internes
  - Tout code utilisé dans les membres statiques ne peut pas faire référence à l'instance courante (this)

# Typologie des méthodes

- On distingues des méthodes particulières: constructeurs, accesseurs, mutateurs
  - Les autres méthodes dépendent des opérations à faire sur les objets en fonction de la nature du problème à traiter
- Constructeurs
  - Rôle: sert à initialiser un objet
  - **Porte le même nom que la classe**, pas de type retour même pas void
  - Son modificateur est toujours **public**, il peut y en avoir autant que nécessaire
  - *En l'absence de constructeur explicitement défini, le compilateur ajoute un constructeur par défaut qui est « public » et sans paramètre*.
- Accesseurs (getter)
  - permettent de récupérer la valeur des attributs d'une classe
- Mutateurs (setter)
  - permettent de modifier la valeur des attributs d'une classe

# TP

- Ecrire des constructeurs pour chaque classe (Carre, Rectangle, Cercle, …)
  - Les paramètres de ces constructeurs doivent permettre d'initialiser la valeur des attributs
  - Reecrire le constructeur par défaut
- Ajouter les mutateurs et les accesseurs dans chacune des classes
- Modifier le code de la classe Test pour valider vos modifications.

# Passage de paramètres

- Les arguments sont toujours passés par valeur lors d'un appel de méthode
- Pour les types primitifs, la valeur de l'argument qui est recopiée dans le paramètre de la méthode
  - Les modifications sur le paramètre dans la méthode sont sans effet sur l'argument transmis
- Dans les autres cas, c'est la référence de l'objet qui est transmise à la méthode
  - Les modifications effectuées sur l'emplacement mémoire référencé sont répercutées/visibles sur l'argument
  - Par contre, la modification de la référence elle-même est sans effet sur l'argument

# Gestion des Entrées/Sortie

- La classe Scanner
  - Présente dans le package java.util
- Etapes d'utilisation
  - Ouvrir le flux et le connecter sur un objet Scanner
    - Scanner sc = new Scanner(System.in); // par défaut sur l'entrée standard (clavier)
  - Lecture des éléments en entrée avec les méthodes
    - Sc.nextInt() : recupère l'élément sur le flux et le retourne en int
    - Sc.nextDouble():
    - …
  - Test de fin de flux
    - Sc.hasNext() : vrai ou faux selon qu'il y a ou non des données positionnées en entrée

# Structure de contrôle

- Instructions conditionnelles (2 syntaxes possibles)
  - if (<condition>) <bloc1> [else <bloc2>]
  - <condition>?<instruction1>:<instruction2>;
  - Dans tous les cas <condition> doit renvoyer une valeur booléenne
- Instructions itératives
  - Syntaxe : while (<condition>) <bloc>
    1. la condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on passe à l'étape 2, sinon on passe à l'étape 4 ;
    2. le bloc est exécuté ;
    3. retour à l'étape 1 ;
    4. la boucle est terminée et le programme continue son exécution en interprétant les instruction suivant le bloc.

# Structure de contrôle

- Syntaxe : do <bloc> while (<condition>);
  - le bloc est exécuté ;
  - la condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on retourne à l'étape 1 , sinon on passe à l'étape 3;
  - la boucle est terminée et le programme continue son exécution en interprétant les instruction suivant le bloc.
- Syntaxe : for (<init>;<condition>;<instr_post_itération>) <bloc>
  1. les initialisations sont effectuées ;
  2. la condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on passe à l'étape □, sinon on passe à l'étape □ ;
  3. le bloc principal est exécuté ;
  4. les instructions à exécuter après chaque itération sont exécutées ;
  5. retour à l'étape 2 ;
  6. la boucle est terminée et le programme continue son exécution

# Instructions de contrôle et tableau

- **Exercice 1: tableau d'entiers**
- Dans cet exercice, on va travailler avec un tableau d'entiers initialisé: int [ ] tab = {12 , 15 , 13 , 10 , 8 , 9 , 13 , 1 4 } ;
  - **Question 1:** Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau(réponse de type oui/non).
  - **Question 2:** Ecrire un programme qui saisit un entier au clavier et qui recherche si cet entier appartient au tableau. Au cas où la réponse est positive, l'indice de cet entier dans le tableau est affiché. S'il y a plusieurs occurrences, le dernier indice est affiché.
  - **Question 3:** Ecrire un programme qui saisit deux indices et échange les valeurs contenues dans le tableau à ces deux indices. Le programme affichera le contenu du tableau avant et après cette transformation.
- **Exercice 2: tableau dont les valeurs sont saisies au clavier**
  - **Question 1 :** Ecrire un programme qui saisit 6 entiers et les stocke dans un tableau, puis recherche et affiche le plus grand élément du tableau.
  - **Question 2 :** Ecrire un programme qui saisit un tableau de 6 entiers puis calcule la moyenne de ces six entiers. Attention, la moyenne des entiers n'est pas un entier.
  - **Question 3:** Ecrire un programme qui saisit d'abord un nombre n, puis ensuite saisit n entiers et les place dans un tableau. Recherche le max de ces n entiers puis calcule leur moyenne. Attention, la moyenne des entiers n'est pas un entier.

# Exercice: classe et objet

- Réaliser une classe **Point** permettant de représenter un point sur un axe. Chaque point sera caractérisé par un nom (de type char) et une abscisse et une ordonnée (de type double). On prévoira :
  - un constructeur recevant en arguments le nom, l'abscisse et l'ordonnée d'un point,
  - une méthode affiche imprimant (en fenêtre console) le nom du point et son abscisse,
  - une méthode translate effectuant une translation définie par la valeur de son argument.
- Écrire une classe **Test** utilisant cette classe pour créer un point, en afficher les caractéristiques, le déplacer et en afficher à nouveau les caractéristiques.

# Exercice: affectation et comparaison

- On donne la classe Entier ci-dessous

```
class Entier {
   public Entier(int nn) {
      n = nn;
   }
   public void incr(int dn) {
      n += dn;
   }
   public void imprime() {
      System.out.println(n);
   }
   private int n;
}
```

- Que fournit le programme suivant ?

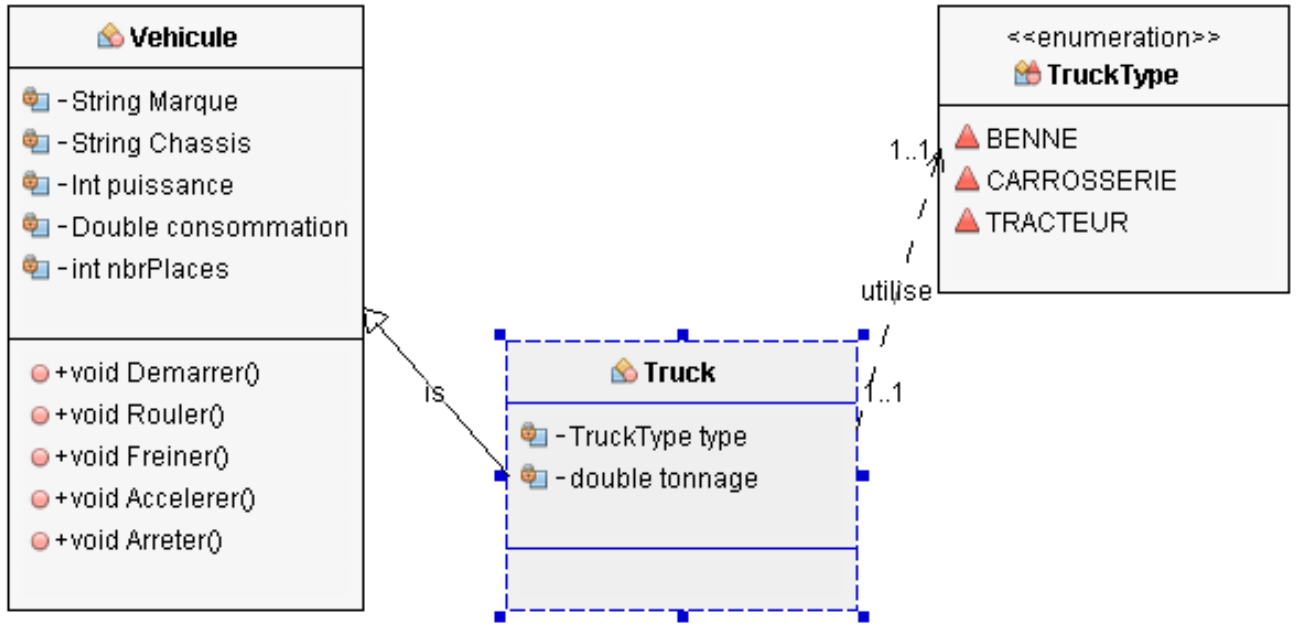
```
public class TestEnt{
      public static void main (String args[]){
            Entier n1 = new Entier (2) ; System.out.print ("n1 = ") ;
            n1.imprime() ;
            Entier n2 = new Entier (5) ; System.out.print ("n1 = ") ;
            n2.imprime() ;
            n1.incr(3) ; System.out.print ("n1 = ") ; n1.imprime() ;
            System.out.println ("n1 == n2 est " + (n1 == n2)) ;
            n1 = n2 ;
            n2.incr(12) ;
            System.out.print ("n2 = ") ; n2.imprime() ;
            System.out.print ("n1 = ") ; n1.imprime() ;
            System.out.println ("n1 == n2 est " + (n1 == n2)) ;
      }
}
```

# Héritage

Mot clé **extends**

# Définition

- Consiste à définir une classe (dite classe fille ou classe dérivée) à partir d'une autre classe (dite classe mère ou classe de base).
- Tous les membres (attributs et méthodes) de la classe de base sont récupérés automatiquement dans la classe fille
- La classe **java.lang.Object** est la super-classe de toutes les classes Java, directement ou indirectement.



# Sous-classes (classes derivées)

- Peut ajouter de nouveaux champs et méthodes
- Peut redéfinir (remplacer) une méthode de la superclasse
- Doit fournir ses propres constructeurs, mais appelle les constructeurs de la superclasse
- N'a pas **un accès direct aux champs privés** de sa superclasse
  - Seules les méthodes et les variables déclarées public ou protected peuvent être utilisées dans une classe héritée
  - le compilateur rejette votre demande lorsque vous tentez d'accéder à des ressources privées d'une classe mère

# Mot clé Super

- Utilisée en java pour référencer un objet de classe parent immédiat
  - super peut être utilisé pour référencer la variable d'instance de la classe parent immédiate
  - super peut être utilisé pour invoquer la méthode de la classe parent immédiate
  - super() peut être utilisé pour appeler le constructeur de la classe parent immédiate
- La syntaxe pour appeler un constructeur de superclasse est
  - super() : le constructeur de la superclasse sans argument est appelé
  - super (liste de paramètres) : le constructeur de superclasse avec une liste de paramètres correspondante est appelé

# Remarques

- Ne pas confondre la redéfinition avec la surcharge :
  - **Redéfinition** : même type de retour et mêmes paramètres.
  - **Surcharge** : type de retour et paramètres différents, les paramètres au moins doivent être différents.
- Une classe définie avec le modificateur d'accès **final** ne peut pas être dérivée.
- Une **méthode** définie avec le modificateur d'accès **final** ne peut pas être redéfinie dans les classes dérivées. Ceci peut se faire pour des raisons :
  - d'efficacité : le compilateur peur générer du code inline, sans faire appel à la méthode.
  - de sécurité : le mécanisme de polymorphisme dynamique fait qu'on ne sait pas quelle méthode va être appelée.

# Exercice de programmation

- Ecrire une classe **Personne**
  - Attribut: nom, age, genre (M/F), adresse
  - Constructeur, accesseurs
  - toString() : String => retourne une chaine contenant attribut: valeur par ligne
- Ecrire une classe **Etudiant** (hérite de Personne)
  - Attributs spécifiques: matricule, filiere, niveau
  - Methodes:
    - Inscrire(filiere, niveau) => affecte filiere et niveau à l'étudiant
    - Preinscrire(matricule) => fixe le matricule de l'étudiant

# Exercice héritage : les employés

- Une entreprise a un certain nombre d'employés. Un employé est connu par son nom, son matricule (qui l'identifie de façon unique) et son indice salarial. Le salaire est calculé en multipliant cet indice par une certaine valeur qui peut changer en cas d'augmentation générale des salaires, mais qui est la même pour tous les employés.
- **Question 1**. Ecrivez la classe des employés avec les informations utiles et des méthodes pour afficher les caractéristiques d'un employé et pour calculer son salaire
- **Question 2**. Certains employés ont des responsabilités hiérarchiques. Ils ont sous leurs ordres d'autres employés. Ecrivez une sous-classe des employés qui représente ces responsables en enregistrant leurs inférieurs hiérarchiques directs dans un ArrayList. Ecrivez une méthode qui affiche les inférieurs directs (placés directement sous leurs ordres) et une autre qui affiche les employés inférieurs directs ou indirects (c'est à dire les subordonnés des subordonnés). On suppose que la hiérarchie de l'entreprise en pyramidale.
- **Question 3.** Les commerciaux ont un salaire composé d'un fixe et d'un intéressement proportionnel à leurs ventes. Ecrivez une sous-classe des commerciaux qui contient l'information sur leurs ventes du dernier mois, une méthode pour mettre à jour cette information et redéfinissez la méthode de calcul de leurs salaires.
- **Question 4.** Ecrivez une classe représentant tout le personnel de l'entreprise, avec une méthode calculant la somme des salaires à verser.

# Leçons

- Stocker des objets autre que des types primitifs dan un tableau
- Polymorphisme
- Utiliser une bibliothèque java: ArrayList
- Établir les  liens de composition entre les objets
- Tester la restriction de l'accès aux champs de la super-classe
- Utilisation de l'opérateur **instanceof**
- **Cast** : conversion de type

# Gestion des Exceptions

Comment se prémunir des bugs/erreurs?

# Exemple

```
public class TestException {
   public static void main(java.lang.String[] args){
      int i = 3;
      int j = 0;
      System.out.println("résultat = " + (i / j));
//toute instruction à la suite ne peut s'exécuter
   }
}
```

- Remarque: une exception levée à l'exécution non capturée

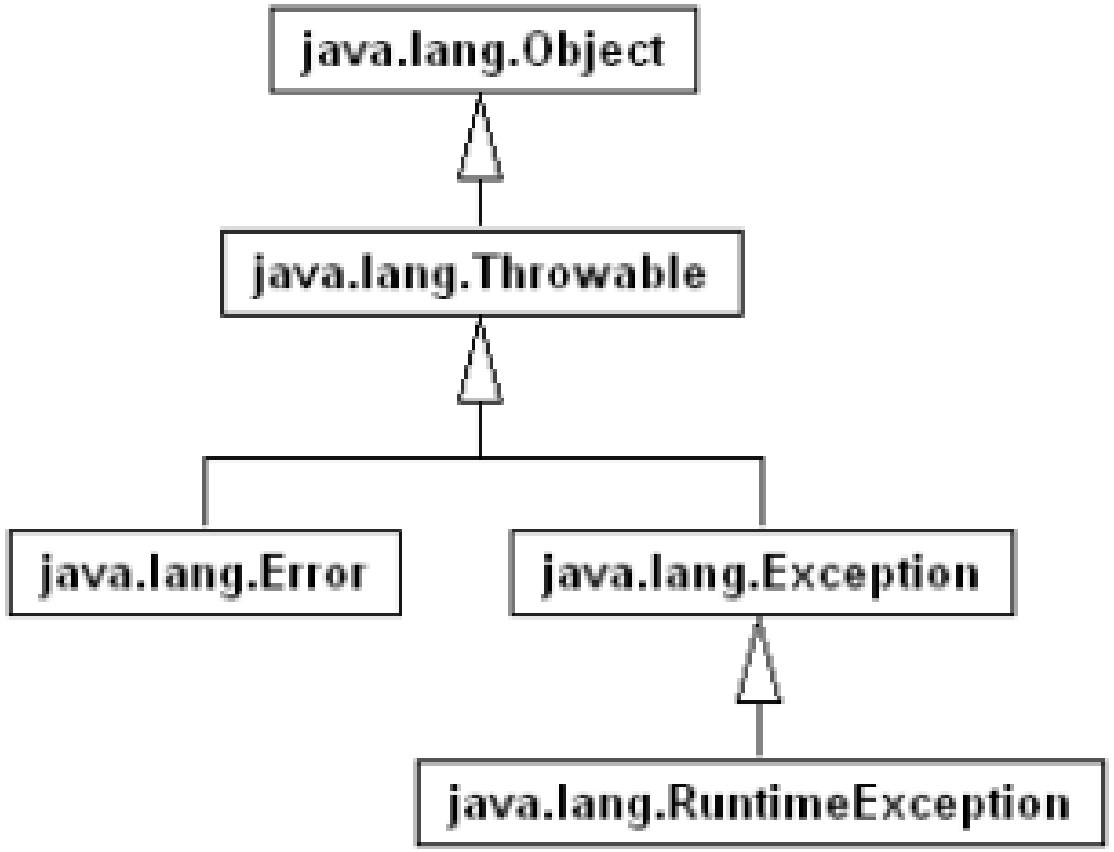
# Définition

- Les **exceptions** représentent le mécanisme de gestion des erreurs intégré au langage Java.
- Il se compose:
  - d'objets représentant les erreurs
  - d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs ( try, catch et finally ) et de les lever ou les propager (throw et throws).
- **Throws** permet de déléguer la responsabilité des erreurs vers la méthode appelante.
- Ces mécanismes permettent de renforcer la sécurité du code Java.

# Exemple modifié

```
public class TestException {
    public static void main(java.lang.String[] args) {
        // Insert code to start the application here.
        int i = 3;
        int j = 0;
        try {
            System.out.println("résultat = " + (i / j));
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

# Exception, RunTimeException et Error




- ces 3 classes descendent de Throwable
  - en fait, toutes les exceptions dérivent de la classe Throwable
- La classe **Error** représente une erreur grave intervenue dans la machine virtuelle Java ou dans un sous système Java.
  - L'application Java s'arrête instantanément dès l'apparition d'une exception de la classe Error.
- La classe **Exception** représente des erreurs moins graves.
  - Les exceptions héritant de classe RuntimeException n'ont pas besoin d'être détectées impérativement par des blocs try/catch.

# Les méthodes de la classe Trowable

- String getMessage( )
  - lecture du message
- void printStackTrace( )
  - affiche l'exception et l'état de la pile d'exécution au moment de son appel
- void printStackTrace(PrintStream s)
  - Idem mais envoie le résultat dans un flux

# Typologie des exceptions

| Nature | Description |
|---|---|
| ArithmeticException | Erreur arithmétique (division par 0) |
| IOException | Erreurs d'entrée/sortie |
| Exception | Classe générique pour capturer tout type d'exception |
| ArrayIndexOutOfBoundsException | Débordement des limites de tableau |
| Exception personnalisée | Définie par le programmeur |

# Propagation des Exceptions

- Pour générer une exception, il suffit d'utiliser le mot clé **throw**, suivi d'un objet dont la classe dérive de **Throwable**.
  - Si l'on veut générer une exception dans une méthode avec throw, il faut l'indiquer dans la déclaration de la méthode, en utilisant le mot clé throws.
- En cas de nécessité, on peut créer ses propres exceptions.
  - Elles descendent des classes Exception ou RunTimeException
  - Pas de la classe Error
  - Il est préférable (*par convention*) d'inclure le mot «**Exception**» dans le nom de la nouvelle classe.
- **Les méthodes pouvant lever des exceptions doivent inclure une clause** *throws* nom_exception **dans leur en tête**.

# Exemple: pile avec Exception

```
public class Stack{
   private int[] content;
   private int _top;
   private final int MAX = 100;
   public Stack() { //Init a stack
      this._top = 0;
      this.content =  new int[this.MAX];
   }
   public int pop() throws Exception {
      if (!this.isEmpty()) {
         return this.content[--this._top];
      } else {
         throw new Exception("Stack is empty");
      }
   }
```

```
   public void push(int e) throws Exception {
      if (!this.isFull()) {
         this.content[this._top++] = e;
      } else {
         throw new Exception("Stack is full");
      }
   }

   public boolean isEmpty() {
      return this._top == 0;
   }

   public boolean isFull() {
      return this._top == this.MAX;
   }
```

# Pile avec exception (suite)

```
    public int top() throws Exception {
        if (!this.isEmpty()) {
            return this.content[this._top-1];
        } else {
            throw new Exception("Stack is empty");
        }
    }
}
```

```
public class StackDemo {
    public static void main(String[] args) {
        Stack intStack = new Stack(); // Créer une pile d'entiers ici
        try {
            for (int i = 0; i < 10; i++) {
                intStack.push(i);
            }
            System.out.println("Contenu de la pile: ");
            while (!intStack.isEmpty()) {
                System.out.print(intStack.pop() + "  ");
            }
            System.out.println("\nSommet de pile: " + intStack.top());
        } catch (Exception ex) {
            System.out.println("\n" + ex.getMessage());
        }
    }
}
```

# Interface graphique: GUI

# Java et les GUI

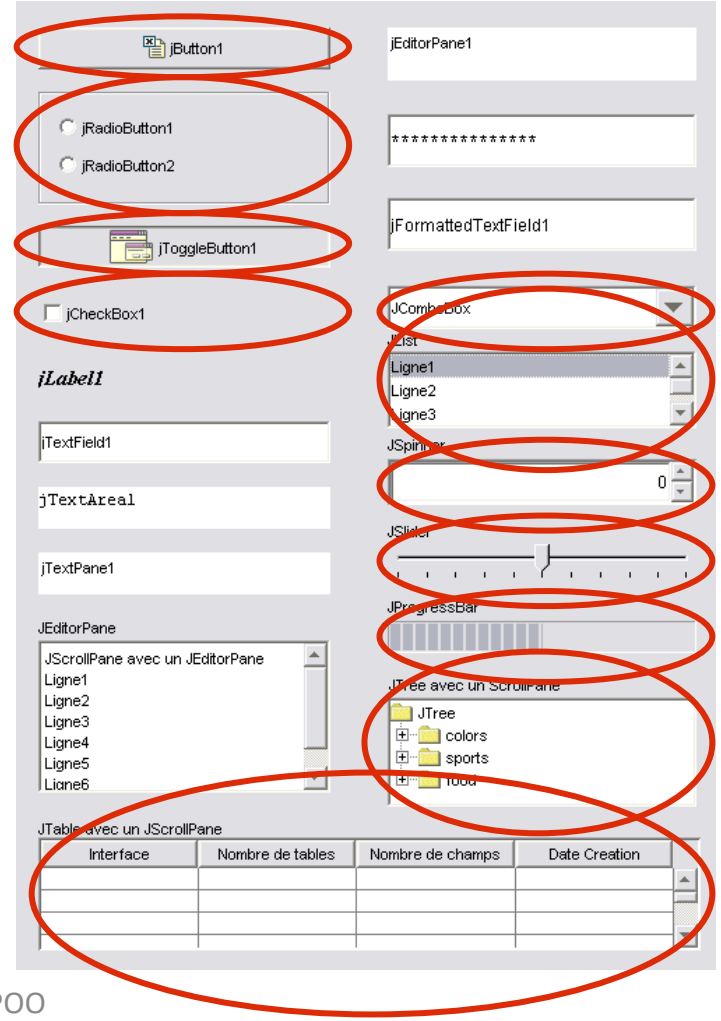
- Java fournit plusieurs packages pour la construction des interfaces graphiques
  - AWT (Abstract Window Toolkit) (quasi obselète)
    - Utilise les composants graphiques natifs de la plate-forme
    - Peut avoir un comportement différent suivant la plate-forme
    - Limité aux caractéristiques communes à toutes les plates-formes cibles
    - Exécution assez rapide.
  - SWING
    - Bibliothèque écrite en 100% pure Java ;
    - Bibliothèque très riche proposant des composants évolués (arbres, tables,...)
    - Construite au dessus de la partie portable de AWT ;
    - Look & Feel modifiable (application du patron MVC) ;
    - Exécution assez lente.
  - JFX
- Dans ce cours nous mettrons l'accent sur **javax.swing**

# Construire un GUI en pratique: MVC

- Définir le Modèle de données de l'application
- Définir l'ergonomie du GUI : la Vue
  - Construire une fenêtre de base (JFrame, JDialog, JApplet…)
  - Construire un composant intermédiaire : Jpanel, JScrollPane, JSplitPane,…
  - Ajouter des objets graphiques ou d'autres composants intermédiaires dans le composant intermédiaire : méthode **add** du composant intermédiaire
  - Ajouter le composant intermédiaire à la fenêtre de base : methode *add* de la fenêtre principale
  - Positionner et dimensionner les composants méthode pack() de la fenêtre principale
  - Visualiser la fenêtre de base : methode **setVisible(true)** de la fenêtre principale
- Définir la réaction aux actions de l'utilisateur sur les éléments du GUI : le Contrôleur

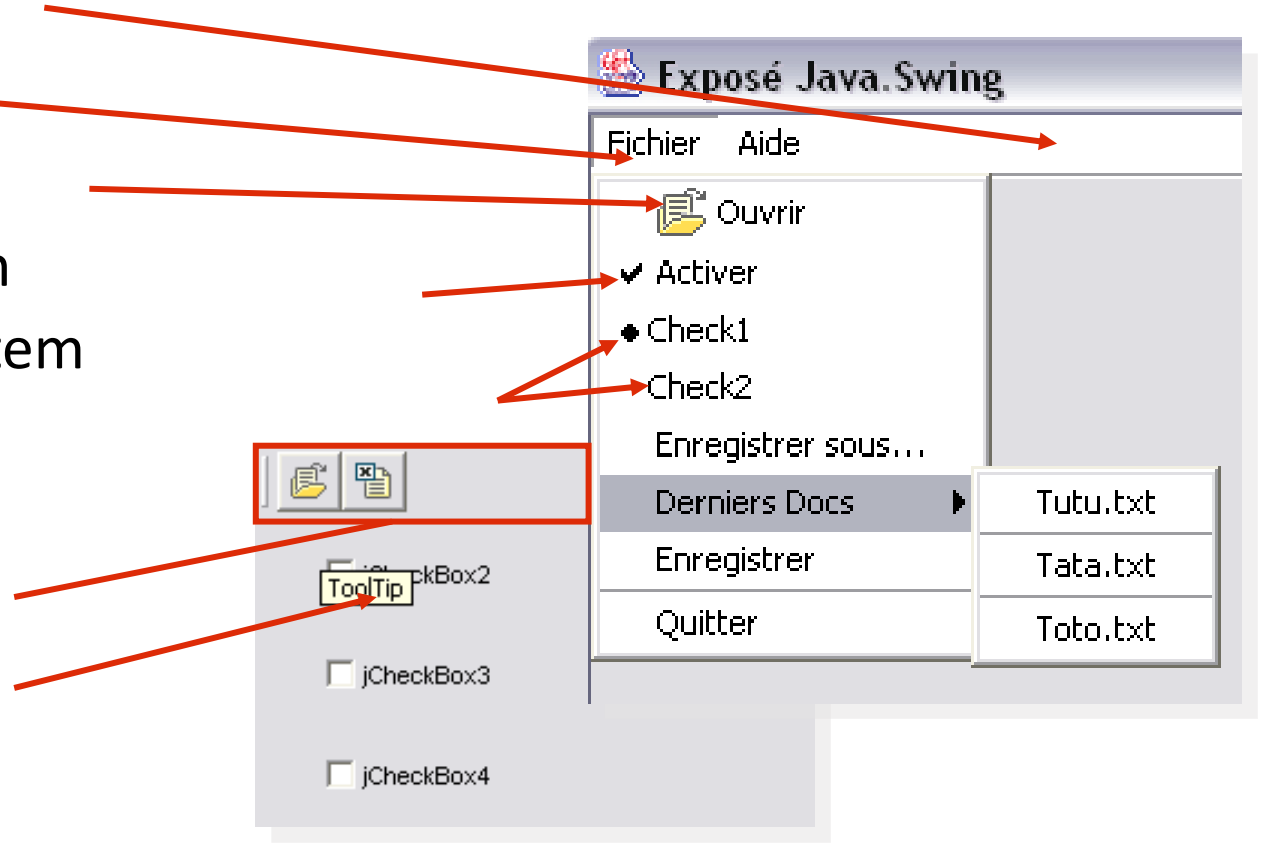
# Les Composants Swing

- Composants Basiques
  - JApplet
  - JButton
  - JCheckBox
  - JRadioButton
  - JToggleButton
  - JComboBox
  - JList
  - JSlider
  - JTable
  - JTree
  - JProgessBar
  - JSpinner



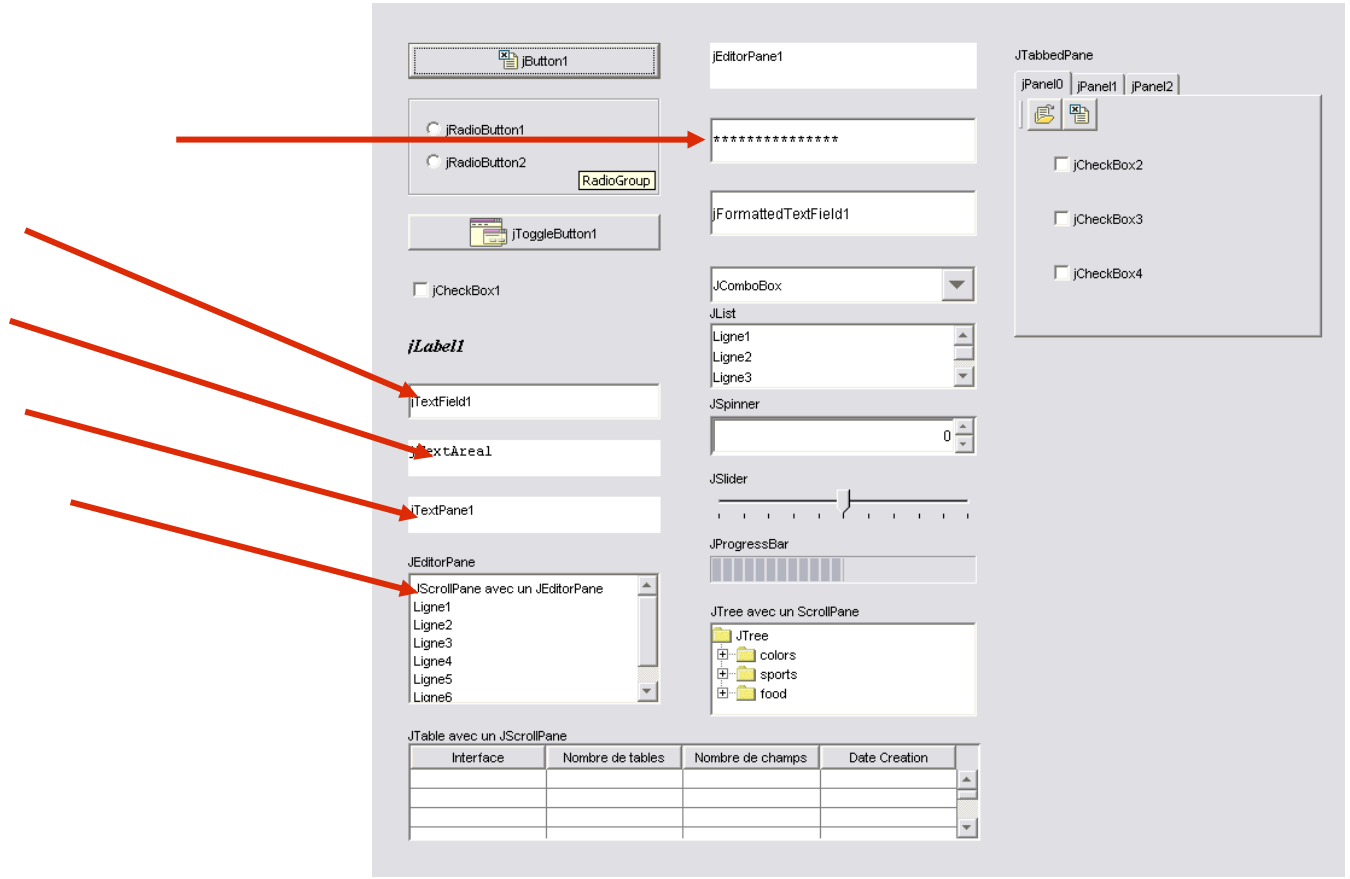
# Les Composants Swing

- Menus, Bar d'outils et ToolTips
  - JMenuBar
  - JMenu
  - JMenuItem
  - JCheckBoxMenuItem
  - JRadioButtonMenuItem
  - JPopupMenu
  - JToolBar
  - JToolTip



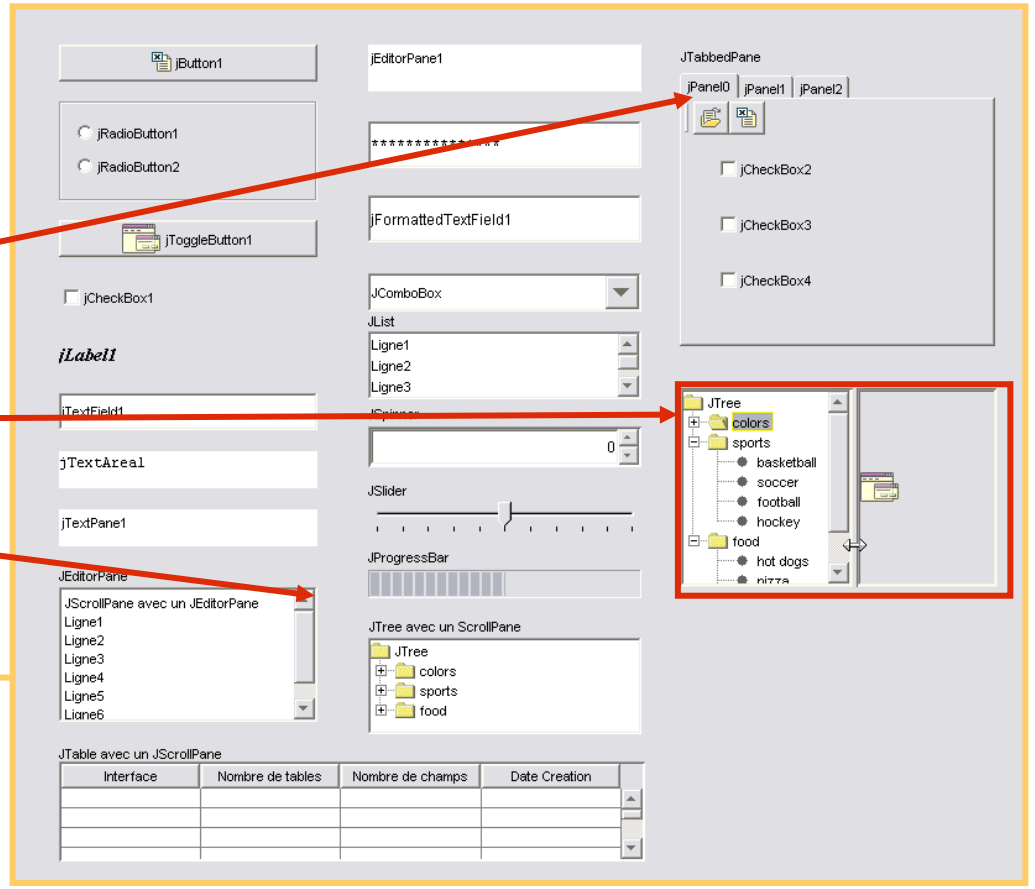
# Les Composants Swing

- Composants Textes
  - JPasswordField
  - JTextField
  - JTextArea
  - JTextPane
  - JEditorPane



# Les Composants Swing

- Containers
  - JOptionPane
  - JDialog
  - JTabbedPane
  - JSplitPane
  - JScrollPane
  - JFrame
  - JInternalFrame
  - JDesktopPane
  - JWindow

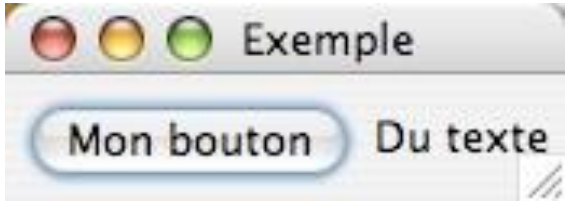


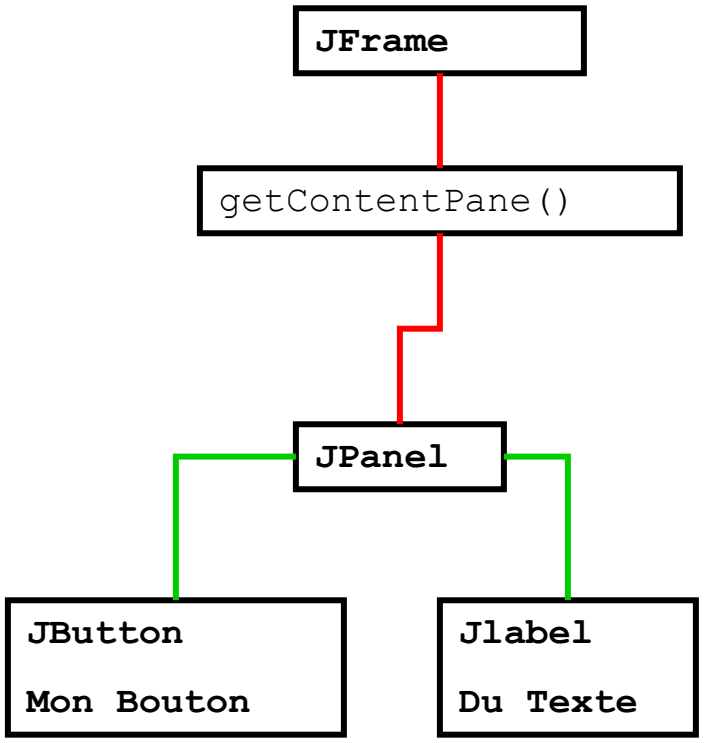
# Layout et les containers

- Pour positionner un composant, nous avons  plusieurs positions prédéfinis. Ces positions qui sont proposés par Java sont:
    - BorderLayout
    - BoxLayout
    - CardLayout
    - FlowLayout
    - GridBagLayout
    - GridLayout

# Methode de construction

- Une plaque de base:  Jframe, Jwindow, Jdialog, Objets JApplet
- On ajoute des briques prédéfinies par dessus :composants ou contrôle
  - boutons, textfield,etc.
- Le sous système graphique est objet
  - chaque composant s'affiche
  - chaque composant provoque l'affichage des composants qu'il contient
    - La fenêtre top level JFrame s'affiche
    - Le contentpane affiche un fond opaque
    - Le JPanel s'affiche (par dessus le précédent) :
      - le fond (paintComponent())
      - les bordures(paintBorder())
    - Il demande à son contenu de s'afficher(paintChildren())
      - Le JButton "Mon bouton" s'affiche par paintComponent()
        - le fond puis le texte
      - Le Jlabel "Du texte" s'affiche par paintComponent() : le texte
- Pour provoquer l'affichage, utiliser
  - repaint() : affiche tout le composant
  - repaint(int,int,int,int) : affiche le rectangle



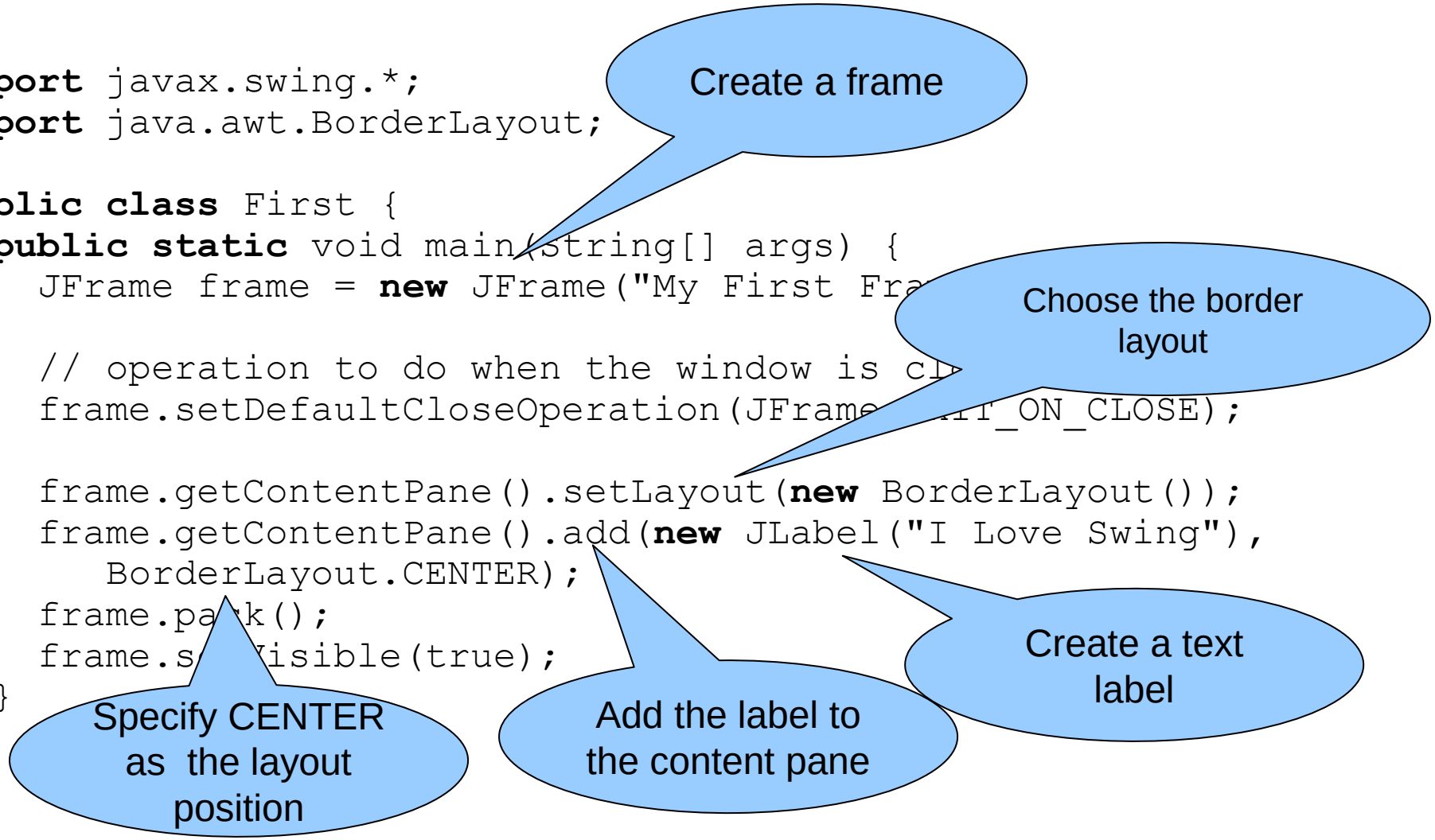



# Mon premier Programme Swing

```
import javax.swing.*;
import java.awt.BorderLayout;

public class First {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My First Frame");

    // operation to do when the window is closed.
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(new JLabel("I Love Swing"),
       BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
  }
}
```

# Mon premier programme expliqué

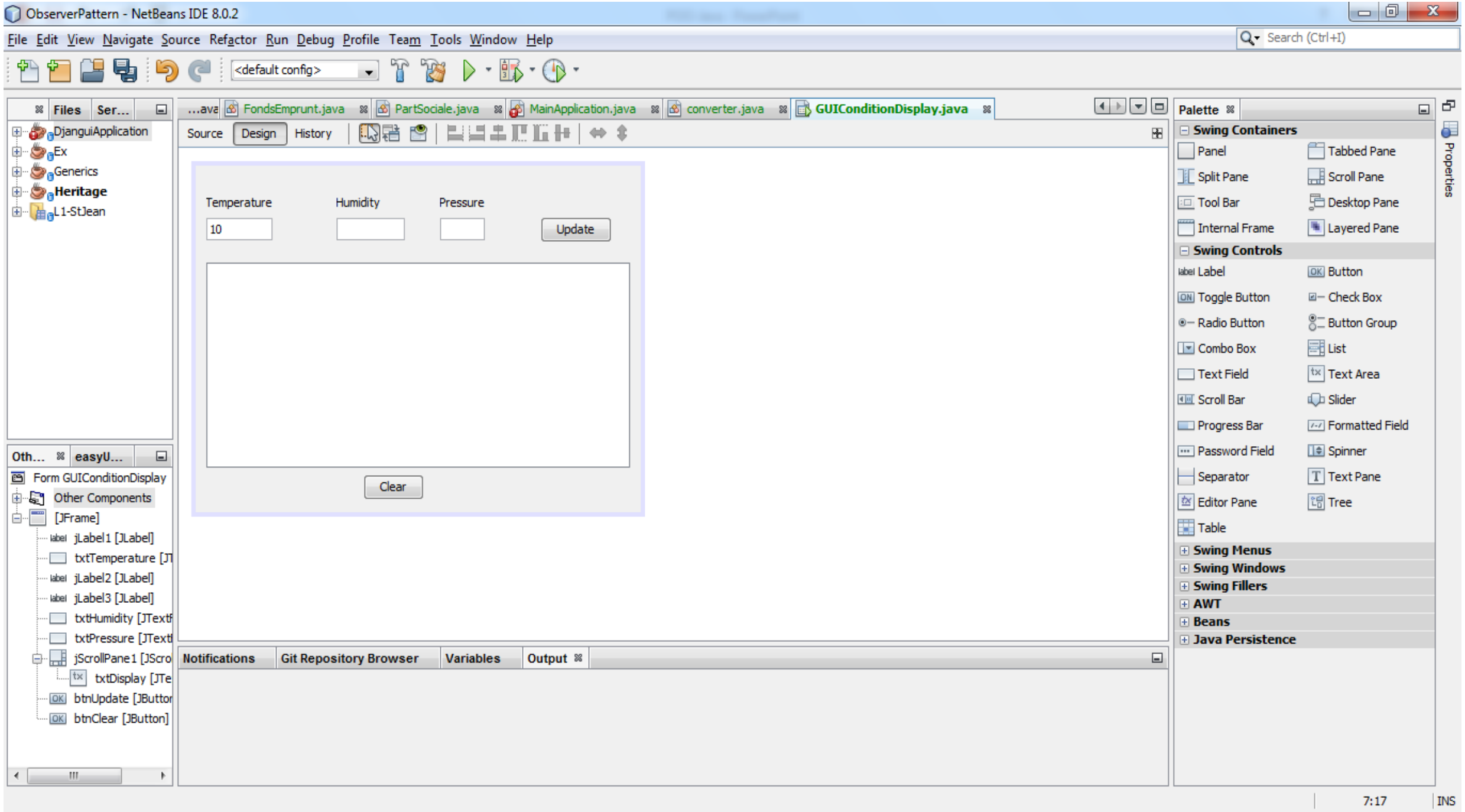
```
import javax.swing.*;
import java.awt.BorderLayout;

public class First {
  public static void main(String[] args) {
    JFrame frame = new JFrame("My First Fra

    // operation to do when the window is cl
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().setLayout(new BorderLayout());
    frame.getContentPane().add(new JLabel("I Love Swing"),
       BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
  }
}
```



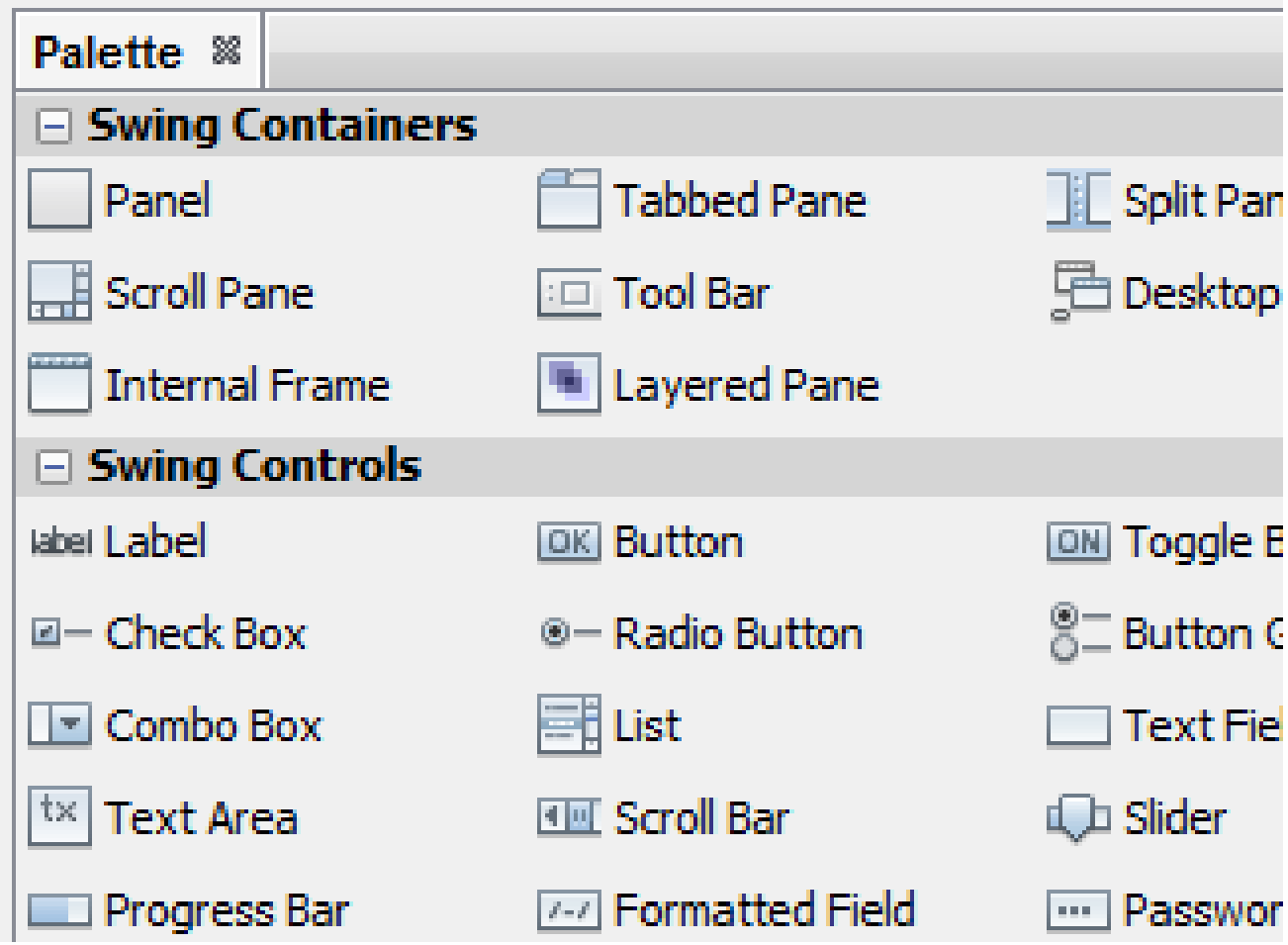
# Concevoir les GUI dans Netbeans

# GUI Builder

- Concevoir les IU Swing en déplaçant et en positionnant les composants à partir d'une palette
- Prend soin automatiquement de l'espacement correct et de l'alignement des objets
- Permet de définir et d'éditer directement les propriétés de chaque objet.
- Les compartiments de l'interface du Builder
  - Zone de dessin: fenêtre principale de création et d'édition des objets GUI
  - Navigateur: offre une représentation de tous les composants de l'application
  - Palette: liste des composants disponibles
  - Fenêtre de propriétés : affiche les propriétés de l'objet sélectionné


ObserverPattern - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Files
Ser...
DjanguiApplication
Ex
Generics
Heritage
L1-StJean
...ava
FondsEmprunt.java
PartSociale.java
MainApplication.java
converter.java
GUIConditionDisplay.java
Source
Design
History
Temperature
Humidity
Pressure
10
Update
Clear
Oth...
easyU...
Form GUIConditionDisplay
Other Components
[JFrame]
jLabel1 [JLabel]
txtTemperature [JT
jLabel2 [JLabel]
jLabel3 [JLabel]
txtHumidity [JTextF
txtPressure [JTextF
jScrollPane1 [JScro
txtDisplay [JTe
btnUpdate [JButtor
btnClear [JButton]
Notifications
Git Repository Browser
Variables
Output
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
Separator
Text Pane
Editor Pane
Tree
Table
Swing Menus
Swing Windows
Swing Fillers
AWT
Beans
Java Persistence
Properties
7:17
INS

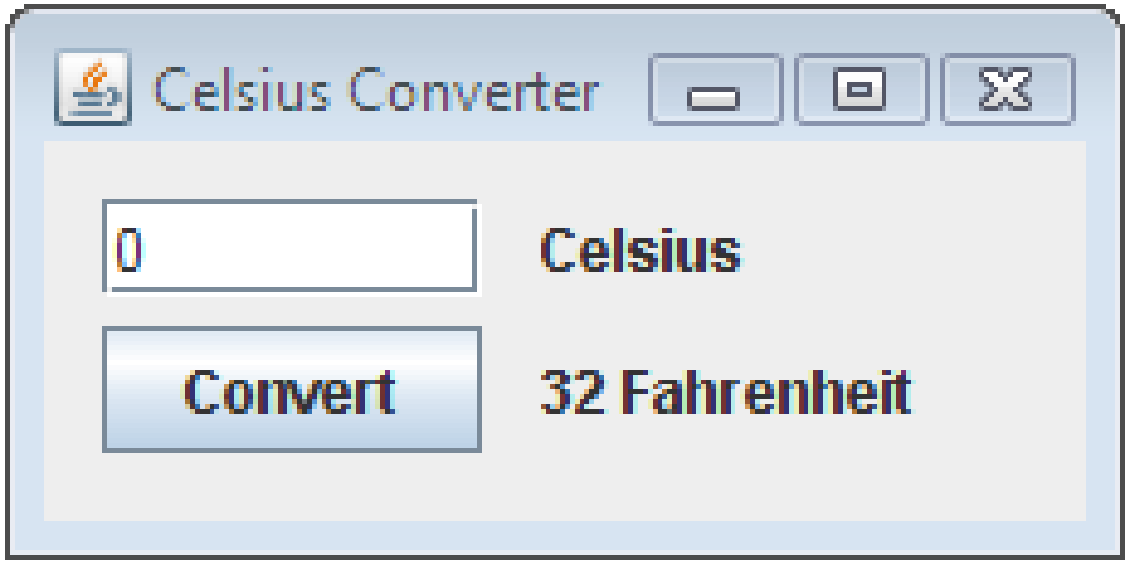
# Palette




- La palette est extensible
- À l'installation, elle contient les composants Swing et AWT

# Exemple

- Un convertisseur de température celsius en Fahrenheit



# Les événements graphiques (1)

- L'utilisateur effectue
  - ♦ une action au niveau de l'interface utilisateur (clic souris, sélection d'un item, etc)
  - ♦ alors un **événement graphique** est émis.
- Lorsqu'un événement se produit
  - ♦ il est reçu par le composant avec lequel l'utilisateur interagit (par exemple un bouton, un curseur, un champ de texte, etc.).
  - ♦ Ce composant transmet cet événement à un autre objet, un **écouteur** qui possède une méthode pour traiter l'événement
    - ➢ cette méthode reçoit l'objet événement généré de façon à traiter l'interaction de l'utilisateur.

# Les événements graphiques (2)

- La gestion des événements passe par l'utilisation d'objets "écouteurs d'événements" (les *Listener*) et d'objets sources d'événements.
  - ♦ Un objet écouteur est l'instance d'une classe implémentant l'interface ***EventListener*** (ou une interface “fille”).
  - ♦ Une source d'événements est un objet pouvant recenser des objets écouteurs et leur envoyer des objets événements.
- Lorsqu'un événement se produit,
  - ♦ la source d'événements envoie un objet événement correspondant à tous ses écouteurs.
  - ♦ Les objets écouteurs utilisent alors l'information contenue dans l'objet événement pour déterminer leur réponse.

# Les événements graphiques (4)

- Les écouteurs sont des interfaces
- Donc une même classe peut implémenter plusieurs interfaces écouteur.
  - ♦ Par exemple une classe héritant de Frame implémentera les interfaces MouseMotionListener (pour les déplacements souris) et MouseListener (pour les clics souris).
- Chaque composant de l'AWT est conçu pour être la source d'un ou plusieurs types d'événements particuliers.
  - ♦ Cela se voit notamment grâce à la présence dans la classe de composant d'une méthode nommée add**XXX**Listener().

# Les événements graphiques (5)

- L’objet événement envoyé aux écouteurs et passé en paramètres des fonctions correspondantes peut contenir des paramètres intéressants pour l'application.
  - ♦ Par exemple, getX() et getY() sur un MouseEvent retournent les coordonnées de la position du pointeur de la souris.
  - ♦ Une information généralement utile quelques soit le type d’événement est la source de cet événement que l’on obtient avec la méthode getSource().

# Catégories d'événements graphiques (1)

- Plusieurs types d'événements sont définis dans le package java.awt.event.
- Pour chaque catégorie d'événements, il existe une interface qui doit être définie par toute classe souhaitant recevoir cette catégorie d’événements.
  - ♦ Cette interface exige aussi qu'une ou plusieurs méthodes soient définies.
  - ♦ Ces méthodes sont appelées lorsque des événements particuliers surviennent.

# Catégories d'événements graphiques (2)

| Catégorie | Nom de l'interface | Méthodes |
|---|---|---|
| Action | ActionListener | actionPerformed (ActionEvent) |
| Item | ItemListener | itemStateChanged (ItemEvent) |
| Mouse | MouseMotionListener | mouseDragged (MouseEvent)<br>mouseMoved (MouseEvent) |
| Mouse | MouseListener | mousePressed (MouseEvent)<br>mouseReleased (MouseEvent)<br>mouseEntered (MouseEvent)<br>(MouseEvent)<br>mouseExited<br>mouseClicked |
| Key | KeyListener | keyPressed (KeyEvent)<br>keyReleased (KeyEvent)<br>keyTyped (KeyEvent) |
| Focus | FocusListener | focusGained (FocusEvent)<br>focusLost (FocusEvent) |

# Catégories d'événements graphiques (3)

| | | |
|---|---|---|
| Adjustment | AdjustmentListener | adjustmentValueChanged (AdjustmentEvent) |
| Component | ComponentListener | componentMoved (ComponentEvent)<br>componentHidden (ComponentEvent)<br>componentResized (ComponentEvent)<br>componentShown (ComponentEvent) |
| Window | WindowListener | windowClosing (WindowEvent)<br>windowOpened (WindowEvent)<br>windowIconified (WindowEvent)<br>windowDeiconified (WindowEvent)<br>windowClosed (WindowEvent)<br>windowActivated (WindowEvent)<br>windowDeactivated (WindowEvent) |
| Container | ContainerListener | componentAdded (ContainerEvent)<br>componentRemoved (ContainerEvent) |
| Text | TextListener | textValueChanged (TextEvent) |

# Catégories d'événements graphiques (4)

- ActionListener
  - ♦ Action (clic) sur un bouton, retour chariot dans une zone de texte, « tic d'horloge » (Objet Timer)
- WindowListener
  - ♦ Fermeture, iconisation, etc. des fenêtres
- TextListener
  - ♦ Changement de valeur dans une zone de texte
- ItemListener
  - ♦ Sélection d'un item dans une liste
- FocusListener
  - ♦ Pour savoir si un élément a le "focus"
- KeyListener
  - ♦ Pour la gestion des événements clavier

# Catégories d'événements graphiques (5)

- MouseListener
  - ♦Clic, enfoncement/relâchement des boutons de la souris, etc.
- MouseMotionListener
  - ♦Déplacement de la souris, drag&drop avec la souris, etc.
- AdjustmentListener
  - ♦Déplacement d'une échelle
- ComponentListener
  - ♦Savoir si un composant a été caché, affiché …
- ContainerListener
  - ♦Ajout d'un composant dans un Container

# Les Apparences LookAndFeel

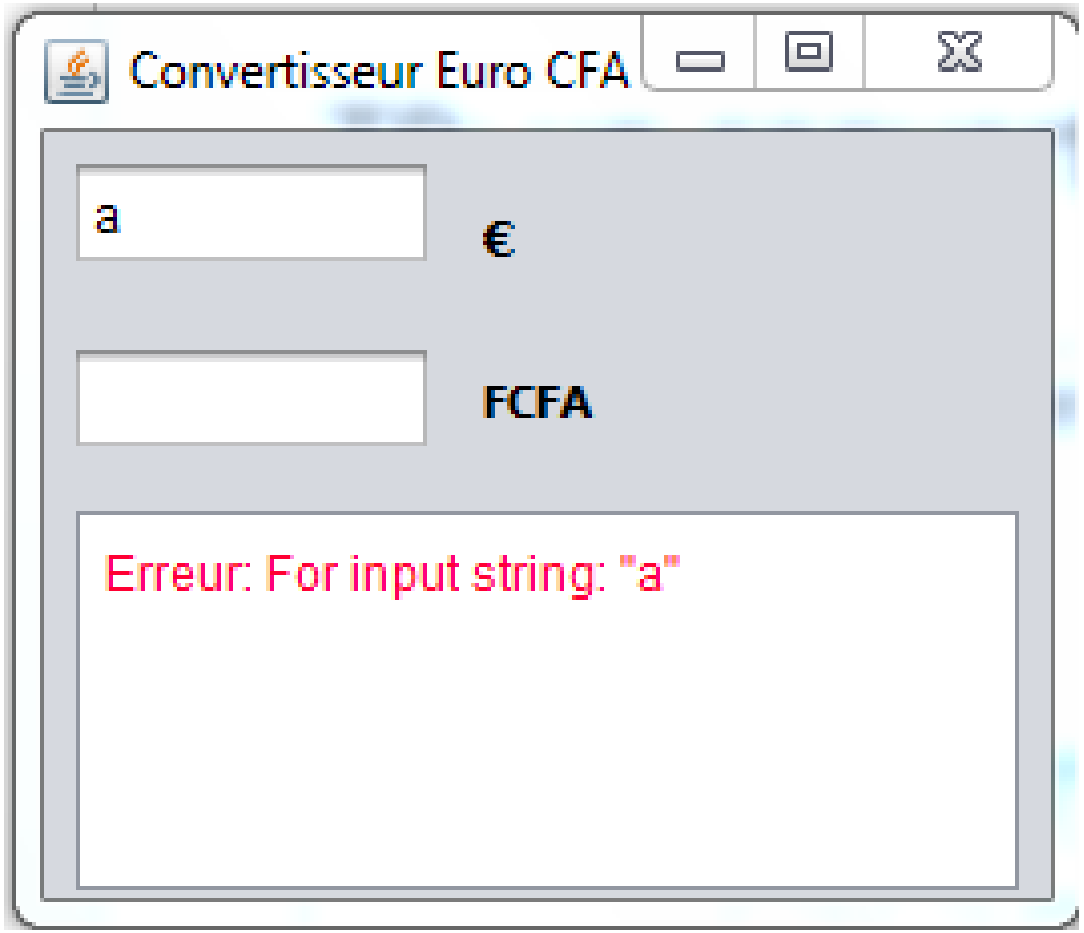
- Grâce à java, nous avons la possibilité de choisir l'apparence. Pour changer l'aspect de l'application, il suffit de mettre ce qui suit dans la méthode main.
- Pour choisir l'apparence de la plateforme où la machine virtuelle est lancée
  - UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
- Pour choisir l'apparence Motif
  - UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
- Pour choisir l'apparence Metal
  - UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
- Pour choisir l'apparence Windows
  - UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

# Apparence look and feel

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName());
    } catch (Exception e) { }

    new SwingApplication(); //Create and show the GUI.
}
```

# TP: un convertisseur Euro-CFA

- Le montant saisi dans la zone Euro est converti en CFA et le résultat est affiché dans la zone CFA
- Inversement lorsqu'on saisit un montant dans la zone CFA
- Gérer les exceptions:
  - Les erreurs sont contenues dans la zone texte en rouge



# Exercice: convertisseur de devises

- On se propose de construire une application pour **convertir les devises**. L'interface doit proposer aux choix de l'utilisateur une liste de devise source, une liste de devise destination. Elle doit offrir en outre une zone de texte pour saisir le montant à convertir. Une fois que ces données sont renseignées, le clic sur un bouton doit procéder à la conversion et afficher le résultat dans une zone de texte non modifiable (JLabel).
- **Indication**: une matrice doit stocker le taux de change entre les devises
  - *taux[i][j] contient le taux de change entre la devise en position i de la liste source et celle en position j de la liste destination*.
- **Question**
  - Proposer une Interface SWING pour l'application et la réaliser dans votre IDE préféré.

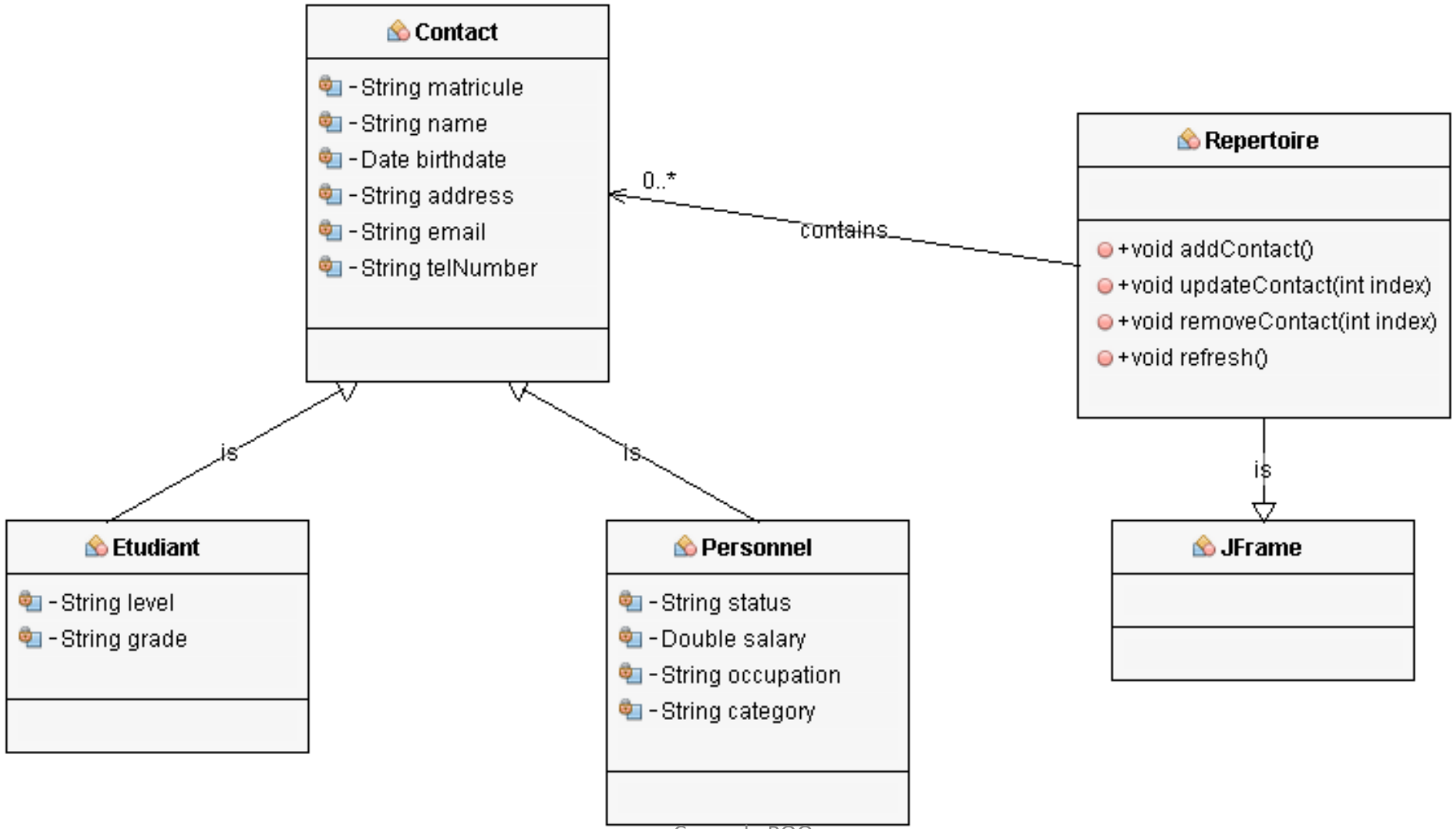
# Illustration

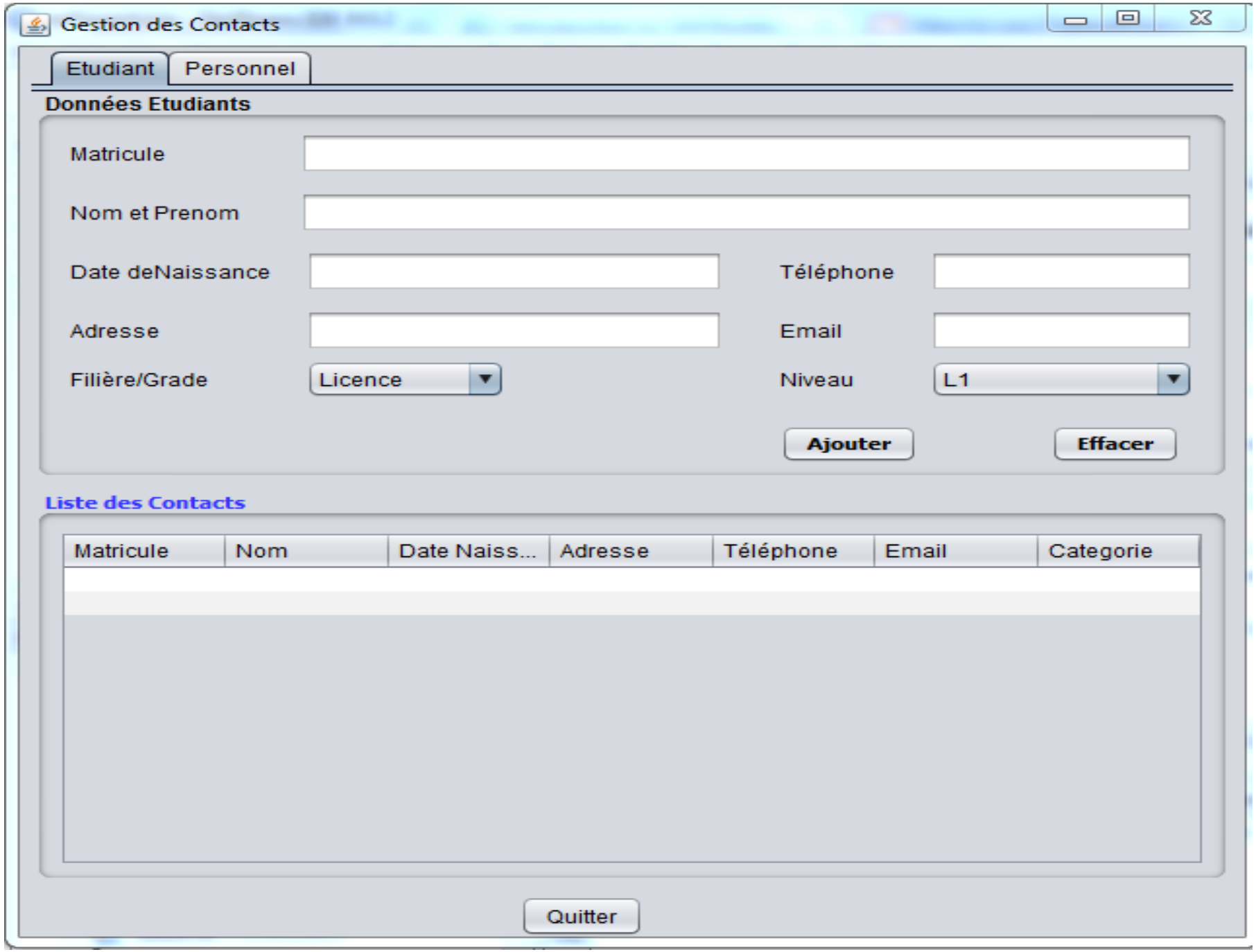
| | 0 = XAF | 1 = EUR | 2 = USD | 3 = GBP | 4= NGN | 5 = CNY |
|---|---|---|---|---|---|---|
| 0 =XAF | 1 | 0,00152 | 0,00184 | 0,00134 | 0,6616 | 0,116 |
| 1 = EUR | 655,957 | 1 | 1,2074 | 0,8765 | 434,129 | 7,6756 |
| 2 = USD | 543,400 | 0,82 | 1 | 0,7262 | 359,480 | 6,3404 |
| 3 = GBP | 748,076 | 1,1405 | 1,3773 | 1 | 495,127 | 8,72723 |
| 4 = NGN | 1,511 | 0,0023 | 0,0027 | 0,0020 | 1 | 0,0176 |
| 5 = CNY | 85,70 | 0,13077 | 0,15773 | 0,11466 | 56,698 | 1 |

# Projet: Annuaire des contacts

- On se propose de construire un annuaire de votre établissement. Pour cela on met en œuvre une application Java doté d'une GUI pour collecter les données, créer les entrées du répertoire et manipuler les contacts enregistrés.
- Services
  - Ajouter un contact (une entrée)
  - Supprimer un contact
  - Modifier (mettre à jour/update) un contact
  - Actualiser l'aperçu (Jtable) du répertoire
- But du TP
  - Utilisation de **ArrayList**, **JTabbedPane, JTable** et des composants déjà connus
  - Manipuler les référence d'objets
- Données du TP
  - Diagrammes des classes (structure de l'application)
  - Maquette de l'interface graphique (GUI)

# Diagramme de classes




Gestion des Contacts
Etudiant
Personnel
Données Etudiants
Matricule
Nom et Prenom
Date deNaissance
Téléphone
Adresse
Email
Filière/Grade
Licence
Niveau
L1
Ajouter
Effacer
Liste des Contacts
Matricule
Nom
Date Naiss...
Adresse
Téléphone
Email
Categorie
Quitter

# Révision générale

# Fondamentaux de la POO

- Considérer la classe suivante:

```
public class IdentifierMesParties {
    public static int x = 7;
    public int y = 3;
}
```

- Quelles sont les variables de classe?
- Quelles sont les variables d'instance?
- Quel résultat produit le code ci-contre?

```
IdentifierMesParties a = new IdentifierMesParties ();
IdentifierMesParties b = new IdentifierMesParties ();
a.y = 5;
b.y = 6;
a.x = 1;
b.x = 2;
System.out.println("a.y = " + a.y);
System.out.println("b.y = " + b.y);
System.out.println("a.x = " + a.x);
System.out.println("b.x = " + b.x);
System.out.println(  " X= " + IdentifierMesParties.x);
```

# Questions

- Comment un programme crée-t-il des objets?
- Comment un programme détruit-il des objets qu'il a créé?
- Quelle est l'erreur dans ce programme(*tenir compte de la classe Rectangle vue en cours*)?

```
public class SomethingIsWrong {
    public static void main(String[] args) {
        Rectangle myRect;
        myRect.longueur = 40;
        myRect.largeur = 50;
        System.out.println("la surface de mon rectangle est " + myRect.surface());
    }
}
```

- Donner la version correcte.