



JAVA

Abderrazak JEMAI

Abderrazak.Jemai@insat.rnu.tn

PLAN

- Chapitre I – Présentation générale de JAVA
- Chapitre II – Concepts de base
- Chapitre III – Composants AWT en JAVA

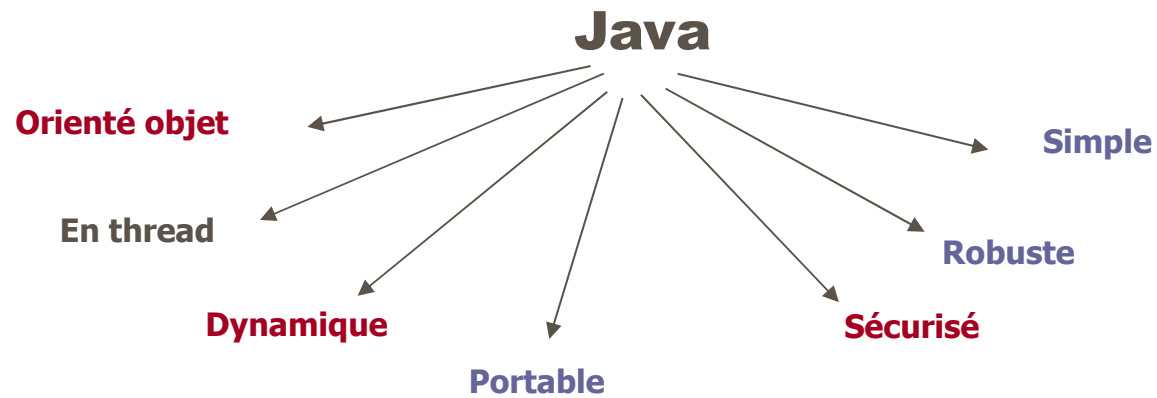
Chapitre I

Présentation générale JAVA

Présentation générale de Java

- ▶ **Java est un langage développé par Sun dans l'intention de créer un langage de programmation orienté objet dynamique (Sun).**
- ▶ **Java est un langage très proche du langage C++.** Certaines sources d'erreurs ou de confusion ont été supprimées ou contrôlées (OO).
- ▶ **Java est adapté aux applications larges échelles accessibles à distance ou via des services Web (WS).**

Les caractéristiques de JAVA



Java : un langage orienté objet

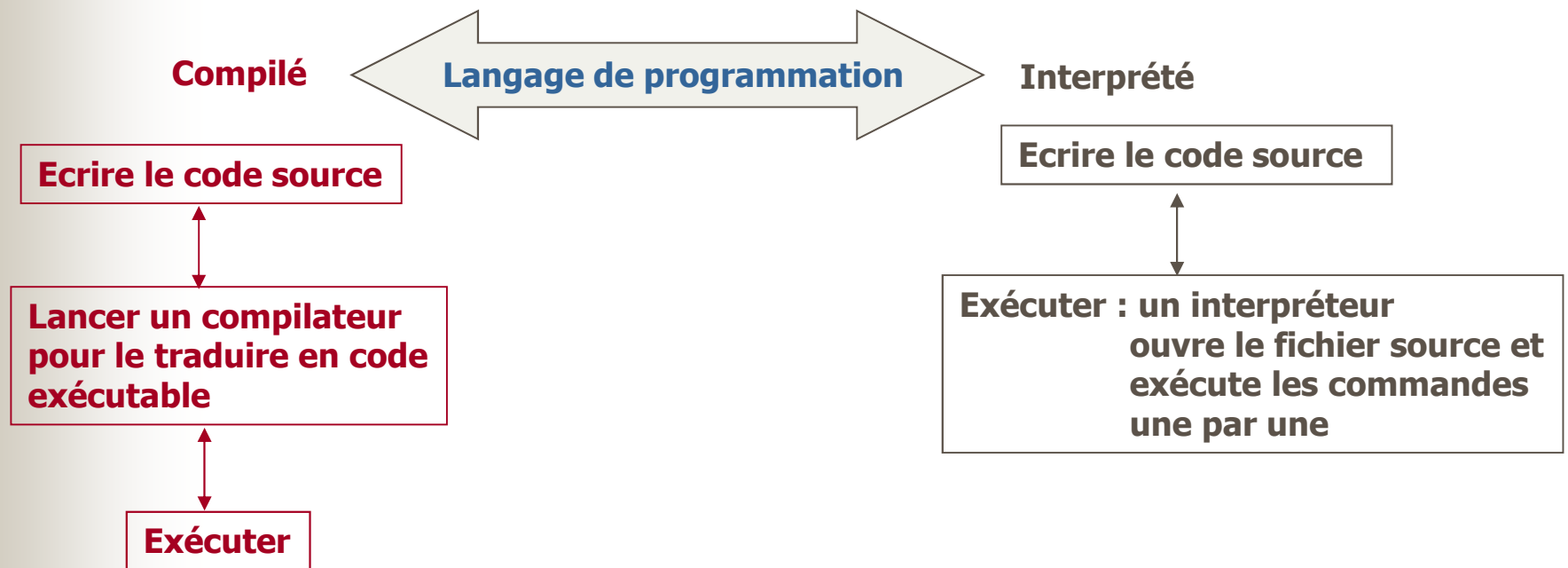
- Exemple : le langage SmallTalk est considéré comme un langage orienté objet pur.

- Java  types primitifs (char, int, float, ...)
Classes



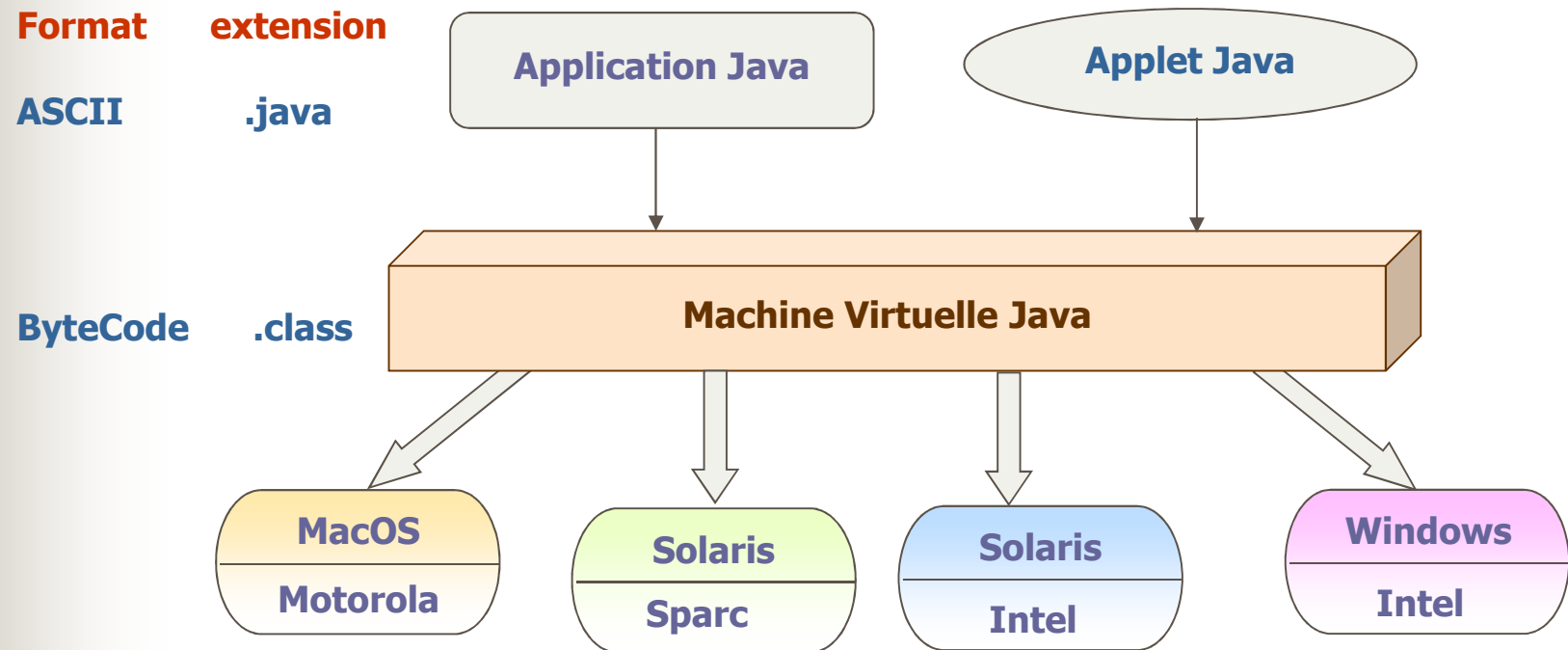
Java est un langage orienté objet pur :

Java : langage compilé ou interprété ?



JVM : Machine Virtuelle Java

- Java est un langage compilé



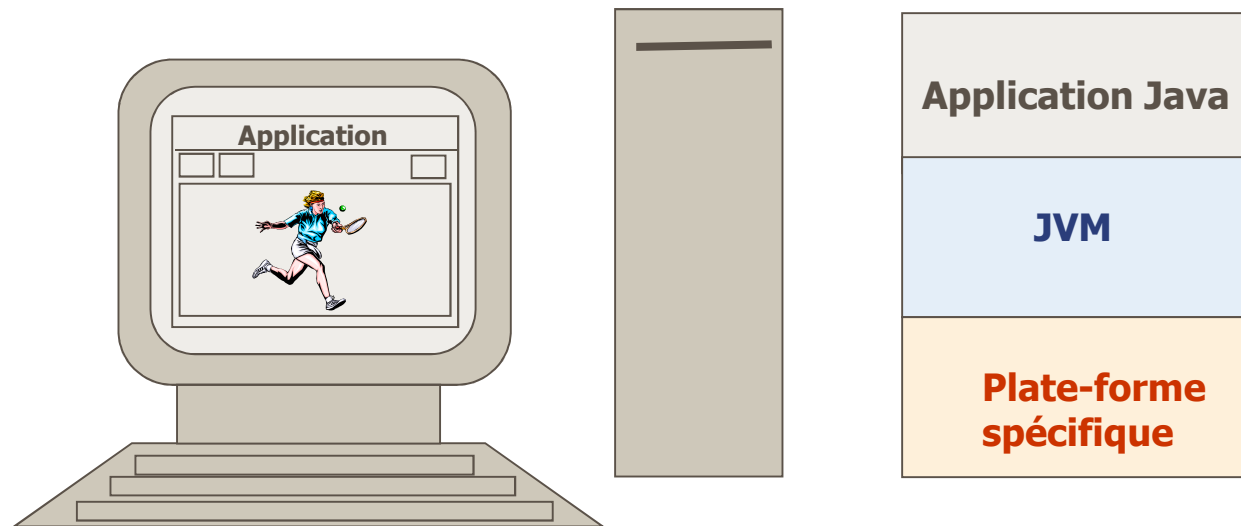
Architecture de Java

- Un programme Java est soit
 - Une Application classique s'exécutant sur une machine.
 - Une Applet accédée via un navigateur (browser) pour être exécutée.
 - Une Servlet : logée chez un serveur. Son invocation génère un fichier HTML.
- **Avantage de la JVM : Rendre les applications java indépendantes du système d'exploitation cible (*target-OS*) et de l'architecture (*target-CPU*).**
- **Le fichier source d'un programme Java est un fichier ASCII écrit avec la syntaxe du langage Java qui contient des définitions de classes.**
- **Une application peut être constituée de plusieurs fichiers sources.**
 - Un fichier source peut contenir une ou plusieurs classes.
 - Un fichier source peut contenir au maximum une classe déclarée à accès public (public class).
 - Un fichier tenant une classe publique doit avoir le même nom que le nom de la classe publique.
- **Les fichiers source ont l'extension ".java".**
- **Les fichiers exécutables ont l'extensions « .class ».**

ByteCode, JVM et API

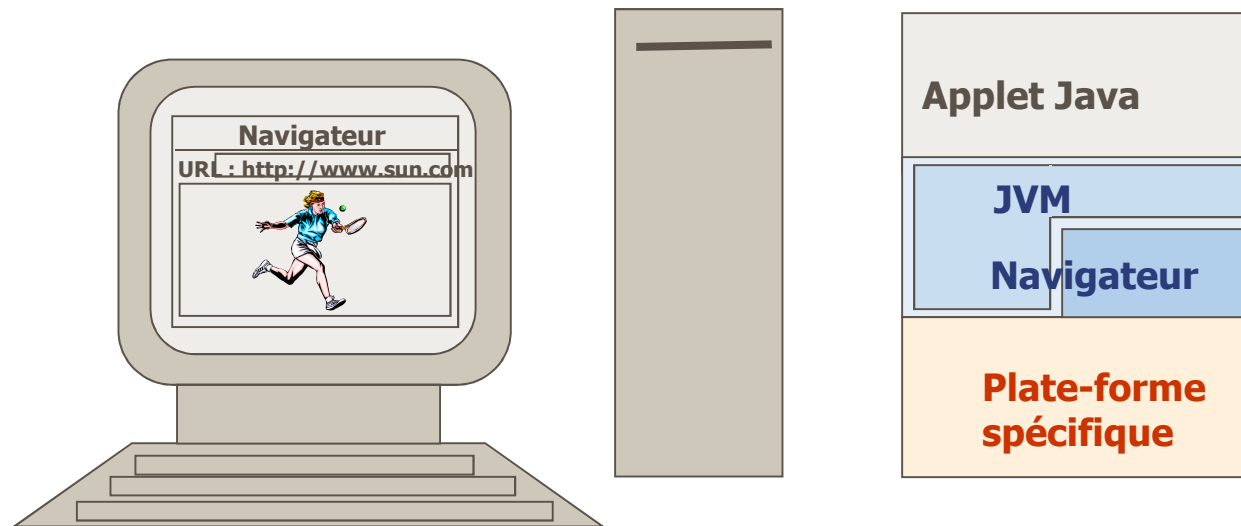
- **Les sources Java doivent être compilées afin de générer le ByteCode. Les fichiers générés ont l'extension ".class"**
- **Le ByteCode java est un langage machine dédié à la machine virtuelle appelée JVM "Machine Virtuelle Java" ou "Java Virtual Machine".**
- **La JVM est une couche logicielle écrite pour chaque machine, lui permettant d'interpréter le ByteCode Java et fournir à l'application les APIs ou services requis (dynamiquement).**

Application Java Standalone



- **C'est une application qui s'exécute sur une machine isolée du réseau et qui possède ses propres ressources (disque, CPU, OS)**
- **La JVM doit être chargée.**
- **Les ".class" et les API Java se trouvent sur la même machine locale.**

Applet Java



- **Applet est une application Java qui s'exécute via un navigateur (browser)**

Chapitre II

Concepts de base

Structure d'une classe Java

Syntaxe des classes en Java



Définition

```
[modificateur] class Nom_de_classe [extends Nom_de_classe] [implements interface] {  
    [déclaration de variables]  
    [déclaration de méthodes]  
}
```

Exemple

```
class Etudiant{  
    String nom;  
    String prenom;  
    float moyenne;  
    float calculMoyenne() { ... }  
    void imprimerReleveNote() {  
        if (....) { System.out.println("...");}  
        else { System.out.println(" ..... "); }  
    }  
}
```

Instructions et blocs

- Soit les expressions suivantes :

```
c = a/b  
k = 2+2  
i = i+1
```

- Pour que ces expressions deviennent compréhensibles par le compilateurs Java, il faut rajouter à chacune un ; à la fin, ce qui donne une **déclaration d'expression ou une instruction Java** :

```
c = a/b;  
k = 2+2;  
i = i+1;
```

- Dans un programme Java, les instructions sont organisées en **blocs**. Un bloc est reconnu par les **accolades** qui le délimitent : une accolade ouvrante { **au début** du bloc et une accolade fermante } **à la fin** du bloc. **Il n'y a pas de ; à la fin d'un bloc**

Les commentaires

- ▶ C'est du **texte** inclu dans le **code source** d'un programme Java
- ▶ Ils fournissent des **renseignements** au programmeur ou au lecteur à propos du code
 - entre **/*** et ***/**, le commentaire peut occuper **plusieurs** lignes
 - après **//** le commentaire **s'arrête** à la fin de la ligne

```
/* exemple de commentaire 1 */  
/* un exemple de commentaire  
sur plusieurs lignes  
*/  
  
//un commentaire d'une seule ligne  
  
a = a+1 ; //le code est exécuté et ce commentaire est ignoré  
  
/* un commentaire de plusieurs lignes  
a = a+1;  
la déclaration de la ligne précédente ne sera jamais exécutée  
*/
```


Les identificateurs

- Ce sont des séquences de caractères alphanumériques utilisables pour nommer une variable, une méthode ou une classe.

caractères permis :
les lettres de a-z et de A-Z
les chiffres de 0 à 9 sauf au début de l'identificateur
_ \$

- Java distingue minuscules et majuscules
- Convention : tout nom autre que le nom d'une classe en minuscules

a ≠ A
reponse ≠ Reponse
choiX ≠ choix

- quelques exemples corrects

```
int i;  
double le_perimetre  
boolean $paye  
char lettre_double
```

- quelques exemples incorrects

```
int 2k;  
char val%  
double int;
```

Les identificateurs

Liste des interdits (mots clés Java)

<code>abstract</code>	<code>for</code>	<code>protected</code>
<code>boolean</code>	<code>future</code>	<code>public</code>
<code>break</code>	<code>generic</code>	<code>rest</code>
<code>byte</code>	<code>goto</code>	<code>return</code>
<code>case</code>	<code>if</code>	<code>short</code>
<code>cast</code>	<code>implements</code>	<code>static</code>
<code>catch</code>	<code>import</code>	<code>super</code>
<code>char</code>	<code>inner</code>	<code>switch</code>
<code>class</code>	<code>instanceof</code>	<code>synchronized</code>
<code>const</code>	<code>int</code>	<code>this</code>
<code>continue</code>	<code>interface</code>	<code>throw</code>
<code>default</code>	<code>long</code>	<code>throws</code>
<code>do</code>	<code>native</code>	<code>transient</code>
<code>double</code>	<code>new</code>	<code>try</code>
<code>else</code>	<code>null</code>	<code>var</code>
<code>extends</code>	<code>operator</code>	<code>void</code>
<code>final</code>	<code>outer</code>	<code>volatile</code>
<code>finally</code>	<code>package</code>	<code>while</code>
<code>float</code>	<code>private</code>	

Les variables simples

- ▶ Les variables sont des récipients qui contiennent les données utilisées dans un programme Java
- ▶ Avant qu'une variable puisse être utilisée dans un programme Java, elle doit avoir été déclarée au préalable
- ▶ Déclarer une variable est l'action de lui donner un nom et de définir le type des données que l'on peut mettre dedans
- ▶ Un nom de variable est un **identificateur**

```
type  identificateur;
```

Les données littérales

- Une donnée littérale est la représentation la plus **explicite** d'un type de données dans un code source Java.

```
a = 2 ; // 2 est un entier littéral  
s = "Bonjour"; /* le texte entre guillemets est une chaîne de  
caractère littérale */
```

- Dans Java, chaque type de données littérales est régi par des règles d'utilisation

Les données littérales

Les nombres

- **Les valeurs numériques dans Java peuvent être des entiers, des nombres à virgule flottante et des caractères**
- **Les entiers**
 - **ce sont des nombres entiers qui peuvent être représentés en décimal, hexadécimal ou en octal.**

```
a = 2 ; // 2 est un entier littéral
```

- **Les nombres à virgule flottante**
 - **un tel nombre est n'importe quel nombre à virgule**
 - **on peut également utiliser la notation scientifique en précédant d'un E la valeur de l'exposant**

```
a = 11.23 ; // 11.23 est un littéral à virgule flottante  
b = 3.2 E -10;
```

Les données littérales

➤ Les caractères

- ce sont des valeurs 16 bits, représentées par un seul caractère entouré de guillemets simples
- certains caractères doivent être représentés par une séquence d'escape

code escape	caractère
<code>\b</code>	arrière (backspace)
<code>\f</code>	formfeed
<code>\n</code>	newline
<code>\r</code>	retour chariot
<code>\t</code>	tabulation horizontale
<code>\v</code>	tabulation verticale
<code>\\</code>	backslash
<code>\?</code>	point d'interrogation
<code>\'</code>	guillemet simple
<code>\"</code>	guillemet double
<code>\000</code>	nombre octal
<code>\xhh</code>	nombre hexadécimal

```
r = 'O';  
x = 'a';
```

Les données littérales

Les booléens

- Ils peuvent avoir la valeur true ou false

```
reponse = true;  
quitter = false;
```

Les chaînes de caractères

- Ce sont des chaînes contenant zéro ou plusieurs caractères entre des guillemets doubles
- une chaîne de caractère n'est pas accessible en tant que tableau de caractères

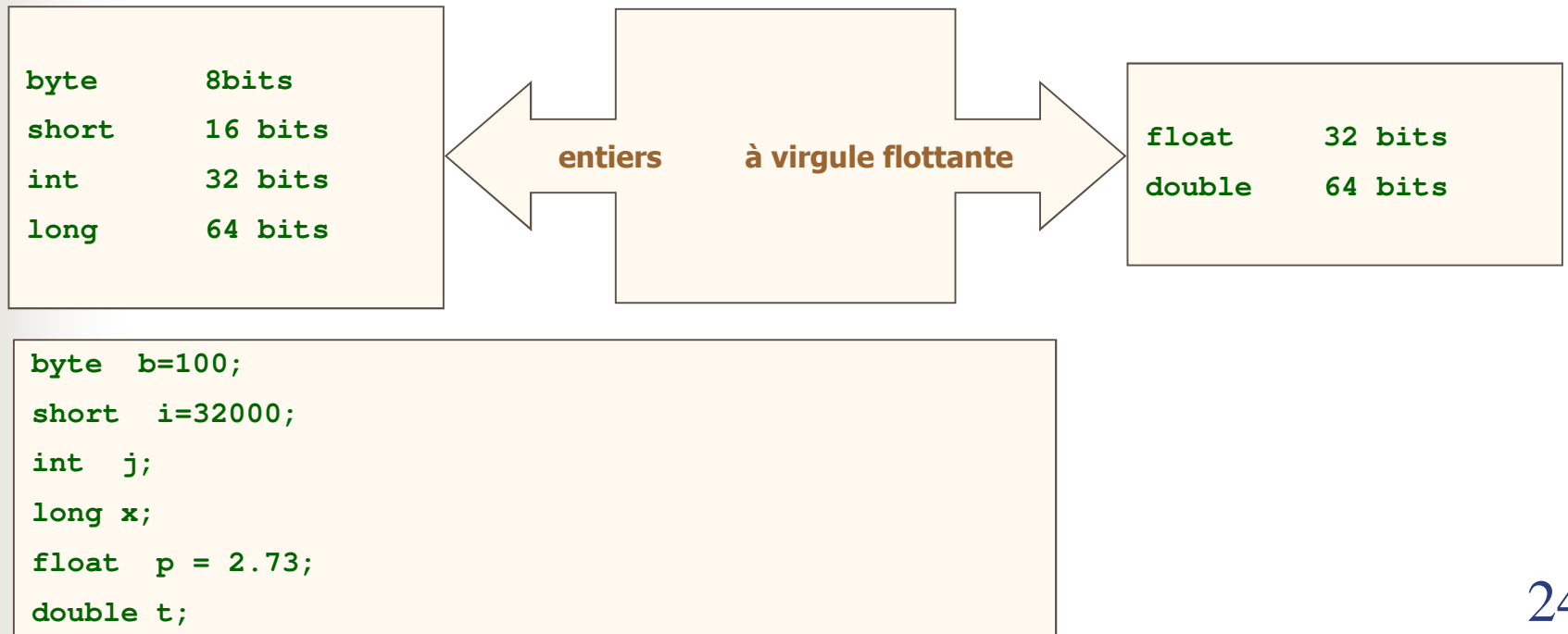
```
message = "Bienvenue à Tunis";
```

Les types de base

- ▶ Les types servent à décrire aussi bien le contenu des variables que les valeurs renvoyées par les expressions
- ▶ Ils sont principalement utilisés dans les déclarations des variables

Les nombres

- ▶ Java implémente quatre sortes de types d'entiers et deux sortes de types de nombres flottants
- ▶ tous ces types se distinguent par leur taille, ce qui signifie que chaque type correspond à un intervalle différent



Les types de base

Les caractères : char

- ▶ Ce type est défini comme étant des valeurs Unicode 16 bits non signées
- ▶ Rigoureusement parlant, un caractère est un type numérique, équivalent à une valeur entière short (16 bits)
- ▶ Ainsi, il est possible d'assigner un littéral numérique à une variable de type caractère et un littéral caractère à une variable entière

```
char caract = 'a';  
char c1 = 48;  
int code1 = 'a';
```

Les booléens : boolean

- ▶ Les variables de ce type peuvent prendre l'une des deux valeurs : **true** ou **false**

```
boolean trouve = false;  
boolean reponse = true;
```

Les affectations et les casts de types

- la valeur assignée à une variable doit être compatible avec le type de celle-ci

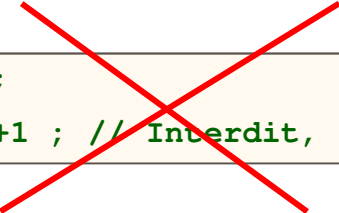
```
type variable_a = (type)variable_b;
```

```
int a = 5;  
byte b = (byte)a;
```

Les initialisations

- ▶ Avant qu'une variable puisse être utilisée, il faut qu'elle soit initialisée : une valeur doit être mise dans une variable avant de s'en servir

```
int i;  
i = i+1 ; // Interdit, car i n'a pas encore été initialisé
```



```
int i;  
i=0;  
i = i+1 ;
```

- ▶ la méthode la plus pratique est d'utiliser un **initialisateur** : c'est une affectation facultative qui fait partie de la déclaration de la variable. C'est une manière plus pratique d'être certain qu'une variable est initialisée avant son utilisation

```
int i=0;
```

Expressions et opérateurs

- ▶ une expression est tout ce qui renvoie une valeur. Elle est typiquement constituée de deux valeurs ou opérandes séparées par un opérateur
- ▶ Pour l'ordre de précedence Java garde l'ordre mathématique connu, qui peut être changé en utilisant des parenthèses

Opérateurs mathématiques

binaires :

+	addition
-	soustraction
*	multiplication
/	division
%	modulo

unaires :

-	négation unaire
++	incrément
--	décrément

```
a++;  
b+=1;  
c*=2;
```

Expressions et opérateurs

Opérateurs binaires : valeurs entières

unaires :

- complément

binaires :

& ET binaire

| OU binaire

^ OU exclusif binaire

<< décalage à gauche

>> décalage à droite arithmétique (propageant le bit de signe)

>>> décalage à droite logique (faisant rentrer des zéros à gauche)

Comparaisons

== égal

!= différent de

< inférieur à

<= inférieur ou égal à

> supérieur à

>= supérieur ou égal à

Opérateurs binaires : valeurs booléennes

! négation : NOT

& ET binaire

| OU binaire

^ OU exclusif binaire

&& ET d'arrêt

|| OU d'arrêt

Les flots de contrôle

if ... else

```
if (expression de test) instruction-vrai (ou bloc) ;  
[else instruction-faux (ou bloc) ;]
```

? :

```
expression-test ? valeur-vrai : valeur-faux
```

```
plus_petite = (j < k) ? j : k
```

```
if (j < k)  
    plus_petite = j;  
else  
    plus_petite = k;
```

switch

```
switch (expression) {  
    case valeur-littérale1 :  
        instruction1;  
        [break];  
    case valeur-littérale2 :  
        instruction2;  
        [break];  
    ...  
    default :  
        instruction-default ;  
        [break];  
}
```


while

```
while (expression_de_test) instruction_de_la_boucle (ou bloc);
```

do ... while

```
do instruction_de_la_boucle (ou bloc);  
while (expression_de_test_)
```

for

```
for (initialisation : expression_de_test : incrément)  
instruction_de_boucle;
```

break

- ▶ l'instruction break peut être utilisée dans la plupart des formes de boucle pour permettre de sortir d'une boucle avant que la condition de sortie ne soit satisfaite

continue

- ▶ l'instruction continue est utilisée pour lancer l'exécution d'une boucle sans avoir à exécuter le reste des instructions de la boucle
- ▶ cela peut être pratique lorsque le programmeur veut sauter certaines instructions dans le code d'une boucle compliquée

La programmation orientée objet

Qu'est ce qu'un objet ?



- ▶ Notre **monde réel** n'est constitué que d'objets : personnes, animaux, plantes, formes, véhicules, ordinateurs, etc.

- ▶ Ces objets ont **deux éléments (champs/attributs/membres)**

 **un état** (nom, couleur, taille, espèce, etc.)

actions/comportement (calculMontant(), isTrue(), etc.)



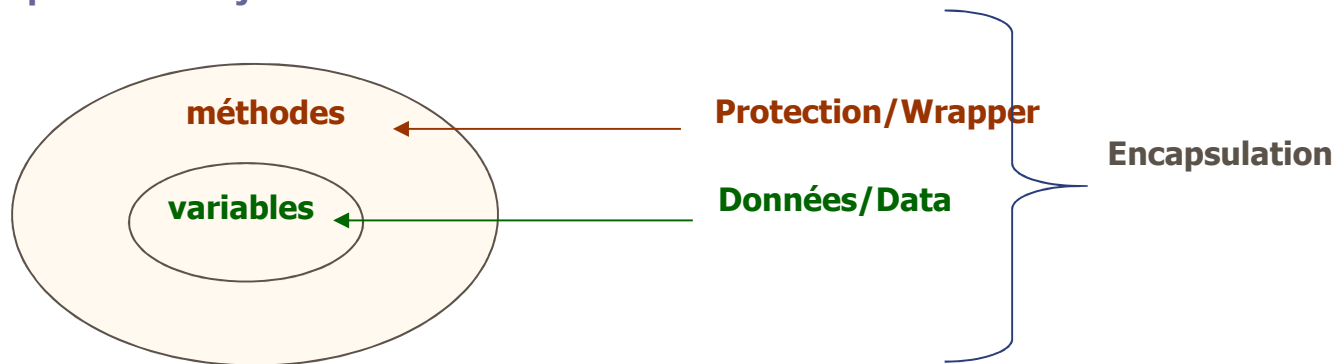
- ▶ **Exemple : frame** **état : dimension, titre, couleurFond**
actions: minimiser, agrandir, fermer,
- ▶ A chaque état est associé une variable. (variable de référence/variable d'instance, variable statique)
- ▶ A chaque comportement est associé une méthode. (méthode d'instance, méthode de classe).

Définition

Les variables et les méthodes sont le seul moyen pour exprimer l'état et le comportement d'un objet.

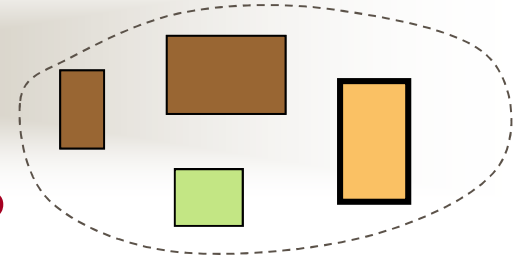
Qu'est ce qu'un objet ?

Schéma conceptuel d'un objet



```
class Etudiant
{
    int numInscription;
    int cin;
    String nom;
    float calculMoyenne()
    {
        for (i=1; i<j; i++) {...}
    }
}
```

Qu'en est-il d'une classe ?

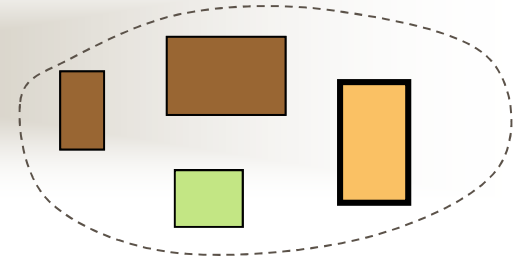


- ▶ Une classe est la définition ou la spécification.
- ▶ Un objet est **une instance** de la classe
- ▶ Une classe est **un modèle** des objets. Le modèle est appelé **classe**

Définition

Une classe est un prototype qui définit les variables et les méthodes communes à tous les objets d'un certain genre.

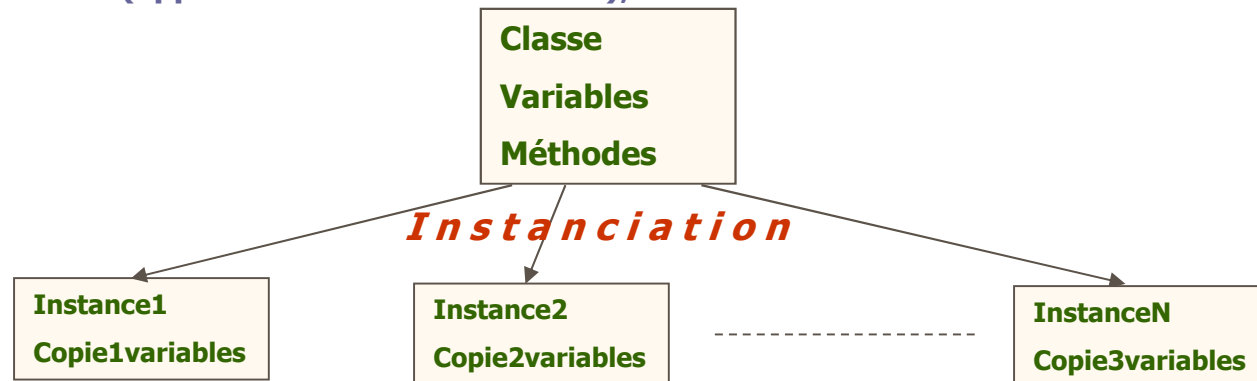
Qu'en est-il d'une classe ?



- ▶ dans notre exemple :

Classe Bicyclettes
Variables
Méthodes

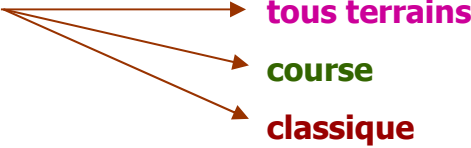
- ▶ Après la création d'une classe, le développeur peut créer **n'importe quel nombre** d'objets de cette classe
- ▶ Lorsqu'on **crée une instance** d'une classe, le système **alloue autant d'espace mémoire** qu'il est nécessaire pour cet objet et toutes ses variables. Chaque instance aura **sa propre copie** des variables (appelées variables d'instance);



- ▶ Toutefois, si la valeur d'une variable d'instance est **toujours la même** pour tous les objets de cette classe, elle sera définie comme une **variable de classe**
- ▶ On peut également définir des **méthodes de classe**

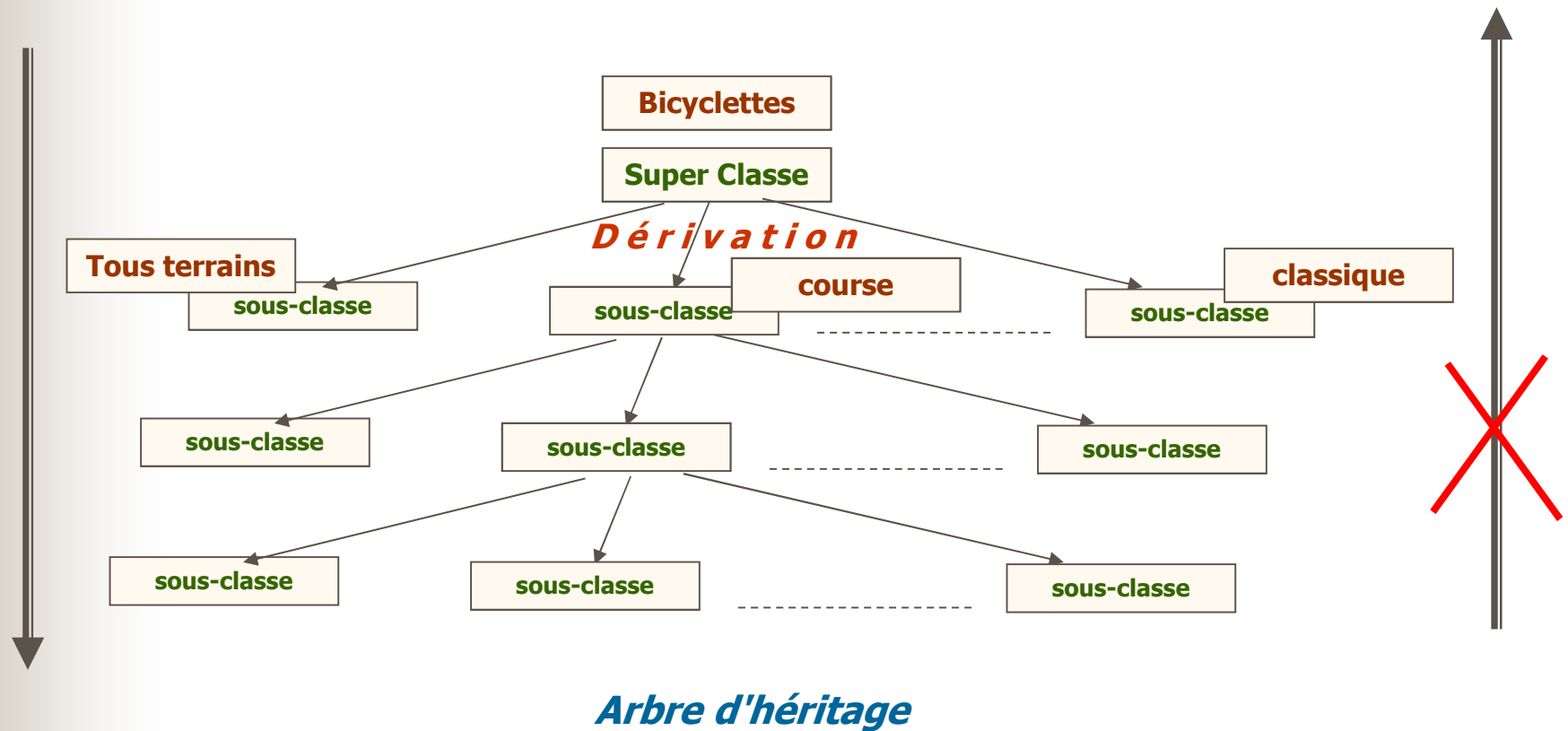


Sous-classe et Héritage ?

- ▶ Il arrive souvent que l'on veuille **organiser** les objets d'une classe en **sous-groupes** pour mieux les **gérer**
- ▶ chaque sous-groupe réunira des objets qui **se partagent** certaines caractéristiques spécifiques
- ▶ **bicyclettes** 
 - **tous terrains**
 - **course**
 - **classique**
- ▶ chaque sous-groupe est appelé **sous-classe** de la classe initiale ou **classe dérivée** ou **classe descendante**
- ▶ la classe initiale est appelée **super classe** ou **classe parente**
- ▶ chaque sous-classe **hérite** les variables et les méthodes de sa classe parente
- ▶ de plus, une sous-classe peut **rajouter** des variables et des méthodes à celle qu'elle hérite
- ▶ une sous-classe peut également **outrepasser** une méthode héritée pour la remplacer par une autre du même type plus adaptée à son comportement spécifique : **Polymorphisme**



Sous-classe et Héritage ?





Classe abstraite ?

- ▶ Une **classe abstraite** est une classe qui a des méthodes non encore définies.
- ▶ La classe qui étend cette classe abstraite doit définir les méthodes abstraites (vides). Elle peut bien entendu redéfinir (Polymorphisme) les méthodes déjà définies dans la classe parente.
- ▶ **A quoi sert une classe abstraite** : Parfois, on ne peut pas définir certaines méthodes d'une classe parente.
- ▶ Exemple :
- ▶

```
class GenieLogiciel extends Etudiant
{
    int calculMoyenne()
    {
        for (i= 1 ; i< nbMatiere) {...}
    }
}
```

```
abstract class Etudiant
{
    int cin;
    String nom;
    ...
    int calculBourse()
    {
        for (i= 1 ; i< nbMatiere)
        {...}
    }
    abstract int calculMoyenne()
}
```




Interface ?

- ▶ une interface est tout simplement une classe là où toutes ses méthodes sont non définies.
- ▶ La classe qui implémente (étend) cette interface doit définir toutes les méthodes de l'interface.
- ▶ Une interface n'admet pas de variables membre.

```
abstract class candidat
{
    abstract int calculMoyenne()
}
```

Syntaxe des classes en Java

Définition



```
[modificateur] class Nom_de_classe [extends Nom_de_classe] [implements interface] {  
    [déclaration de variables]  
    [déclaration de méthodes]  
}
```

Exemple

```
class Lampe {  
    boolean allumee;  
    void metEtat(boolean allumee) {  
        this.allumee=allumee;  
    }  
    void afficheEtat() {  
        if (allumee) { System.out.println("La lampe est allumée");}  
        else { System.out.println("La lampe est éteinte); }  
    }  
}
```

Classes

Les méthodes

Déclaration

```
[modificateur] type_de_retour  nom_de_la_méthode(liste_de_paramètres) {  
    [instructions]  
}
```

void
int
String

int x, float y, ...

Classes

Les méthodes

Méthodes surchargée

- ▶ Créer plusieurs **versions** différentes de la méthode, pour pouvoir l'appeler différemment.
- ▶ Exemple :

La méthode **print** doit pouvoir imprimer des entiers, des chaînes, etc.

```
Class MyObject {  
    float x,y,z;  
    public void print(int x) { ... }  
    public void print(float x) { ... }  
    public void print(String x) { ... }  
}
```

Classes

Les constructeurs

- ▶ Un constructeur est une fonction qui est appelée **lorsqu'un objet est créé** comme instance d'une classe
- ▶ Un constructeur est défini comme **une méthode qui a le même nom** que sa classe
- ▶ Un constructeur est une méthode qui **ne retourne aucune valeur**

```
class MaDate {  
    int jour = 1;  
    int mois = 1;  
    int annee = 2001  
  
    MaDate() {  
        annee = 2002;  
    }  
    MaDate(int nouveau_mois, int nouveau_jour, int nouvelle_annee) {  
        jour = nouveau_jour;  
        mois = nouveau_mois;  
        annee = nouvelle_annee;  
    }  
}
```

Classes

Méthodes/Variables de classe (statiques) et d'instance

- Une méthode ou une variable est dite de classe si le mot clé `static` figure dans sa **déclaration** (le main est une méthode de classe)
- Si le mot clé `static` **ne figure pas** dans la déclaration d'une méthode ou d'une variable, il s'agit alors d'une méthode ou d'une variable **d'instance**
- Les variables statiques sont utilisées lorsque le développeur veut que la variable soit **commune** à toutes les instances de la classe

```
class Alien {  
    boolean vivant;  
  
    static int nombre_alien = 0;  
  
    void Alien() {  
        vivant = true;  
        nombre_alien++;  
    }  
  
    void explode() {  
        if (vivant) {  
            vivant = false;  
            nombre_alien--;  
        }  
    }  
}
```

Classe

Variable d'instance
Ne sera modifiée que dans l'objet
Auquel elle appartiendra

Variable statique ou de classe
Partagée par tous les objets
instance de Alien

Classes

- Une méthode statique sert lorsque le développeur veut définir des **fonctions globales** qui sont associées à une classe
- Une telle méthode ne peut pas accéder aux variables d'instance de la classe

```
class OutilsMath {  
    static int met_au_carre(int x) {  
        return x*x;  
    }  
    static int moitie(int x) {  
        return x/2;  
    }  
}  
  
class Calcul_Math {  
    void resultat {  
        int i = OutilsMath.met_au_carree(4);  
        int j = OutilsMath.moitie(i);  
        System.out.println(j);  
    }  
}
```

Classes

Méthodes/variables privées et publiques

- Une méthode ou une variable est dite de type privé si le modificateur `private` figure dans sa **déclaration**
- Une variable ou méthode privée n'est visible qu'à l'intérieur de la classe où elle est définie : **on ne pourra pas** accéder directement à une telle variable ou méthode **depuis une instance** de la classe qui le contient.

Variables finales

- Lorsque le développeur veut travailler avec une variable qu'il ne souhaite pas modifier (constante), il suffit qu'il la déclare comme étant final et ce en précédant son identificateur par le mot clé `final`

Exemple

```
class Classique
{
    private String nom;
    private int nbRepetition;
    String message = "travaillons";
    Classique(String nom, int nbRepetitions)
    {
        this.nom = nom;
        this.nbRepetitions = nbRepetitions;
    }
    void repete()
    {
        for (int i = 0; i < nbRepetitions; i++)
        {
            System.out.println(message);
        }
    }
    public String toString()
    {
        return nom + " dit : " + message;
    }
    public static void main(String[] argv)
    {
        int repetition = Integer.parseInt(argv[0]);
        Classique ecrivain = new Classique("Jean", repetition);
        ecrivain.repete();
        ecrivain.message="reposons-nous";
        System.out.println(ecrivain);
    }
}
```

variables d'instance

méthodes d'instance

méthodes de classe

- ◆ Lorsque le programme est lancé, **aucune** instance de la classe Classique n'existe, ni donc évidemment aucune variable **d'une instance** de cette classe
- ◆ En revanche, les variables et les méthodes **statiques** existent dès qu'une classe est évoquée
- ◆ La méthode **main** peut donc être lancée **sans** qu'aucune instance de la classe **Classique** n'existe
- ◆ l'**argument** de la méthode main est un tableau de chaînes de caractères. Les chaînes contenues dans ce tableau sont des arguments envoyés par la ligne de commande, le nom du programme ne faisant pas partie des arguments

Objets

Références

Définition

- ▶ Une fois un objet créé, sa référence est retournée et peut être utilisée par le développeur pour accéder aux variables et aux méthodes de cet objet
- ▶ Il est **interdit** de manipuler les références des objets **elle-même** par exemples en tant que pointeurs comme dans C++

```
ObjetDate laDate;  
laDate = new ObjetDate(23,12,65);  
laDate.année=1964;
```

this

- ▶ Java fournit cette variable spéciale pour faire référence à l'objet courant.
- ▶ Elle peut être utilisée pour passer la référence de l'objet **courant** à une méthode d'un autre objet

Objets

Références

null

- ▶ Si la valeur d'une référence d'objet est égale à `null`, alors cette référence **n'est pas valide**, c'est à dire qu'elle pointe sur un objet qui **n'existe pas**
- ▶ Elle peut être utilisée pour *tester* la validité d'une référence

```
if (monAmpoule != null) {  
    System.out.println("monAmpoule existe !");  
}
```

opérateurs

<code>==</code>	égal à
<code>!=</code>	différent de

Héritage

Sous-classement

```
class Nom-de-classe extends Nom-de-classe [implements interface] {  
    [déclaration de variables]  
    [déclaration de méthodes]  
}
```

Classe parente

Classe dérivée

```
class Felins {  
    boolean affame = true;  
    void parle() {  
    }  
    void appelle() {  
        System.out.println("Ici, petit petit petit ...");  
        if (affame) parle();  
    }  
}  
class Chat extends Felins {  
    void parle() {  
        System.out.println("Miaou...");  
    }  
}  
class Tigre extends Felins {  
    void parle() {  
        System.out.println("Grrrrr...");  
    }  
}
```

Héritage

Outrepasser des méthodes (*polymorphisme*)

Outrepasser des méthodes

- ▶ Pour que des objets dans une même hiérarchie de classes répondent à des messages identiques de manière différente, il suffit d'implémenter les méthodes identiques dans les sous-classes de manière différente
- ▶ Une méthode est **outrepassée** lorsqu'une version en est implémentée dans une sous-classe et que cette version ait **les mêmes caractéristiques** (nom, type de valeur de retour, paramètres) que la méthode originale de la classe parent

```
Felins leChat;  
leChat=new Chat;  
leChat.parle();
```

- ▶ Ce mécanisme est appelé aussi *polymorphisme*

Les tableaux et les chaînes de caractères

Déclaration et allocation de tableaux

➤ Déclaration d'un tableau

```
char[] t;
```

Déclare le tableau T comme un tableau de caractères

➤ Allocation d'un tableau

```
t=new char[2]
```

Alloue le tableau T pour deux variables de type caractère

➤ Affectation de valeurs

```
class tableauA
{
public static void main(String[] argv)
{
char[] t;
t=new char[2];
t[0]='h'
t[1]='a'
}
}
```

Dimensions d'un tableau et dépassement des bornes

- Pour connaître la longueur du tableau T on peut utiliser : **T.length;**
- Length est en quelque sorte un variable du tableau que l'on peut d'ailleurs uniquement lire
- Le plus grand indice dans un tableau est T.length-1, ainsi référencer T.length provoquerait sûrement une erreur de débordement qu'il aurait fallu gérer

Tableau d'objets

- L'exemple suivant montre comment manipuler un tableau d'objets, plus spécifiquement un tableau d'Integer. A part pour cela, il n'est pas utile, pour additionner quelques entiers d'utiliser la classe d'objets Integer

```
class tableauC
{
public static void main(String[] argv)
{
Integer T[]= new Integer[4];
int somme=0;

for (int i=0; i<T.length; i++)
    T[i]=new Integer[i];
for (int i=0; i<T.length; i++)
    somme+=T[i].intValue();
}
}
```

Les chaînes de caractères

- Pour traiter les chaînes de caractères on utilise la classe **Java.lang.String**
- Lorsque le compilateur rencontre une série de caractères entre **doubles apostrophes**, il crée automatiquement **un objet String**.
- Java définit par ailleurs l'opérateur **+** de **concaténation** pour les chaînes de caractères
- Lorsque cet opérateur a une opérande qui est une chaîne de caractère et la deuxième non, alors il **convertit** cette dernière grâce à la méthode **toString** qui est définie dans **Object**
- Toutefois **toString** peut être redéfinie par l'utilisateur dans sa propre classe pour réaliser les fonctions qu'il souhaite

Chapitre III

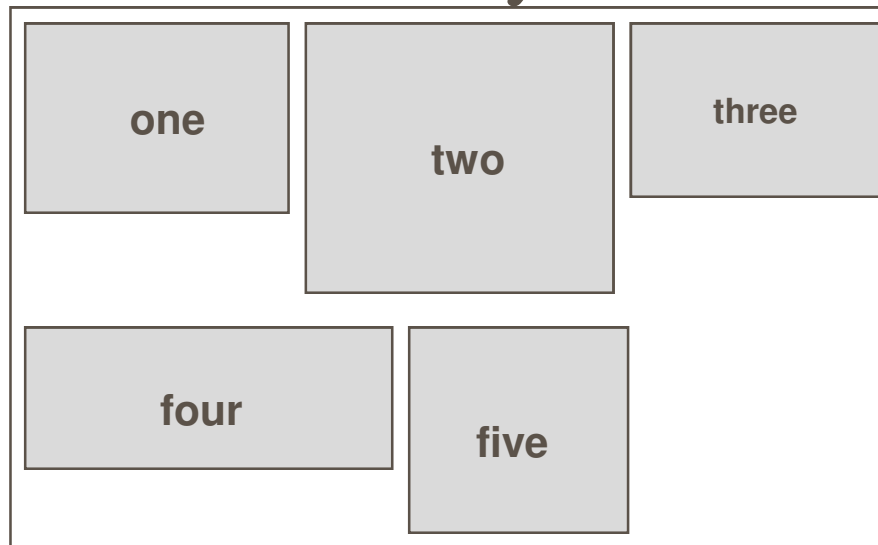
JAVA et les composants AWT

Les composants AWT

- Les composants AWT comportent :
 - Button
 - Checkbox
 - Choice
 - Label
 - MenuBar
 - MenuItem
 - TextArea
 - TextField

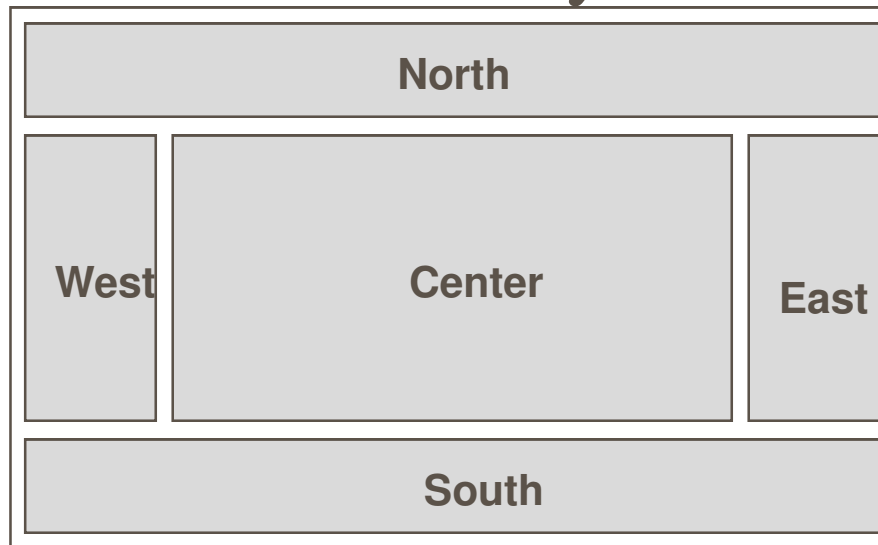
FlowLayout

■ java.awt.FlowLayout



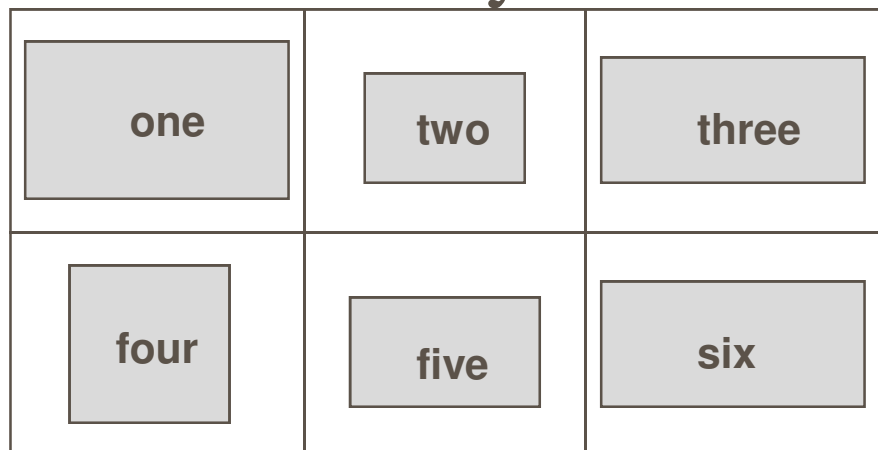
BorderLayout

- `java.awt.BorderLayout`

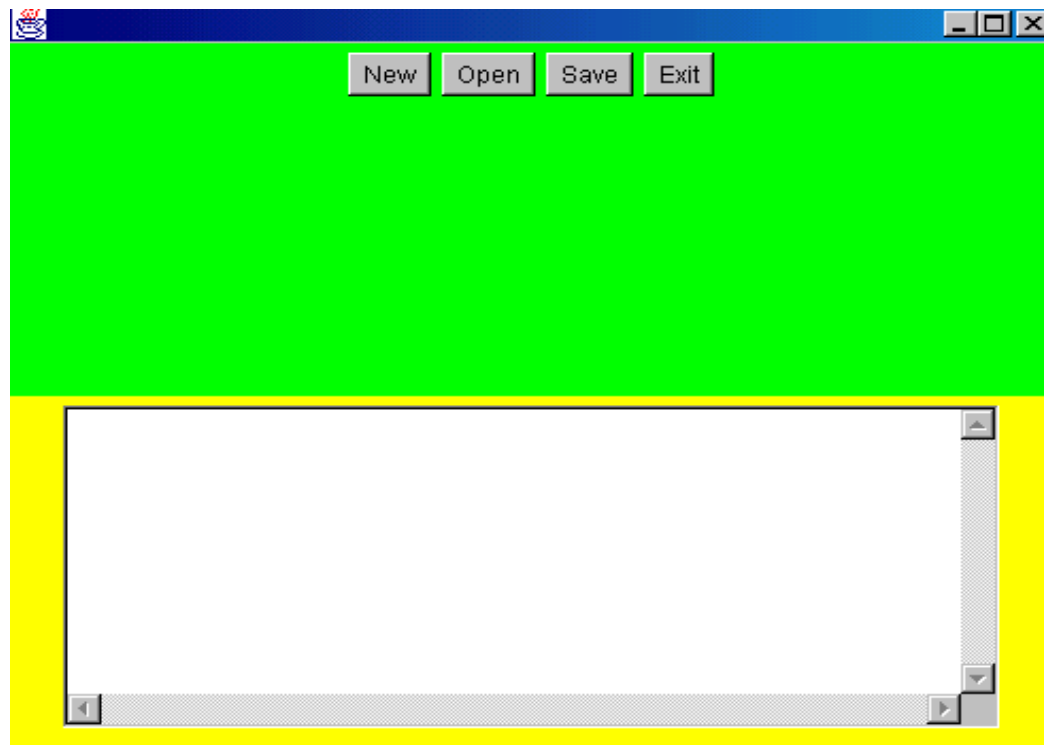


GridLayout

- `java.awt.GridLayout`



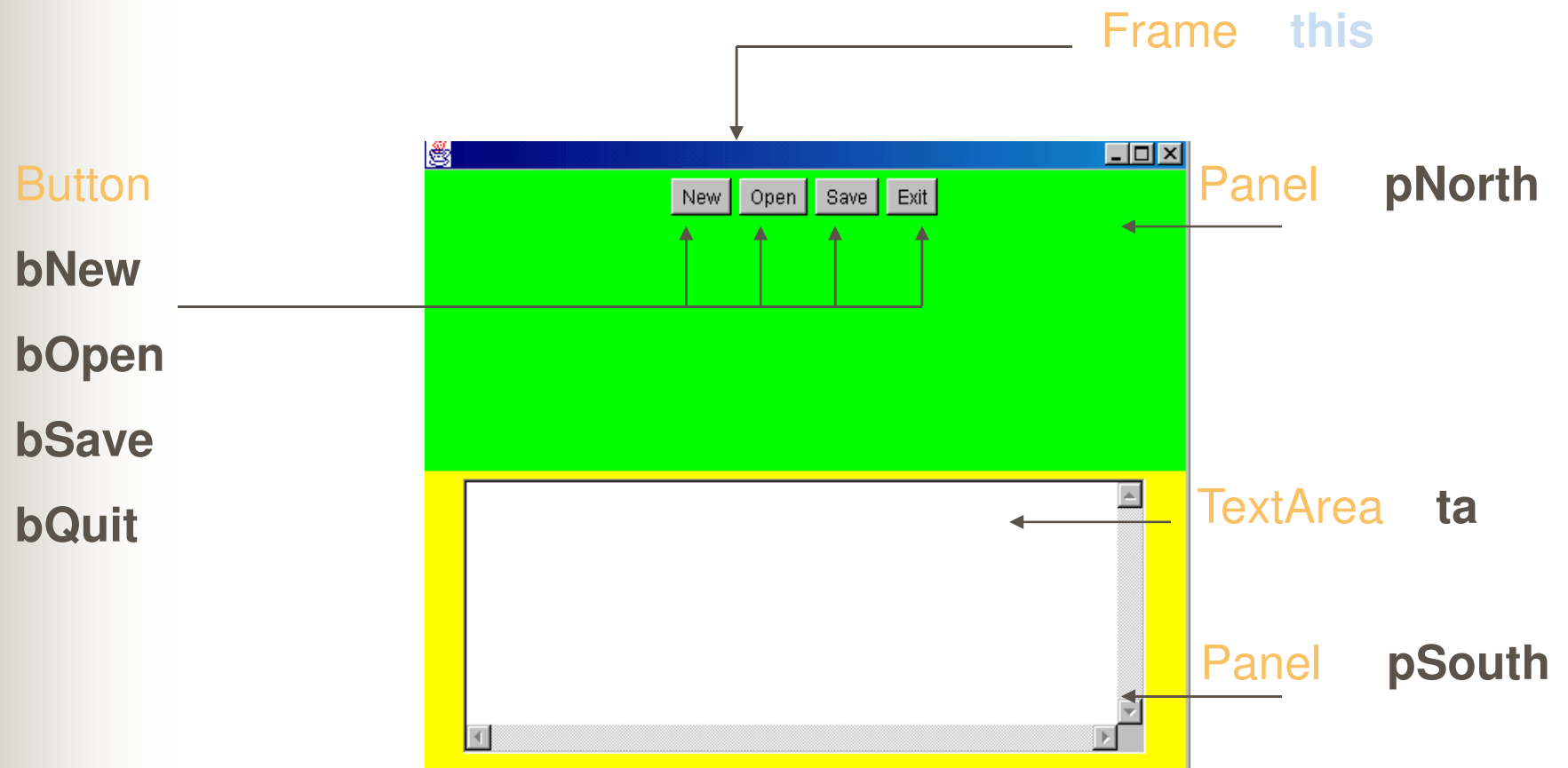
Exemple



Approche méthodologique

1. Dessiner l'interface
2. Attribuer des noms à chaque composant
3. Identifier les types de chaque composant
4. Déclarer la classe qui hérite de `java.awt.Frame`
5. Déclarez le ou les constructeurs
6. Déclarez les composants comme variables membres.
7. Rajoutez les instructions dans le constructeur.

Application



Application

1. Le dessin étant effectué
2. Attribuer des noms à chaque composant
 - `Panel pNorth = new Panel();`
 - `Button bNew = new Button("New");`
 - `Button bOpen = new Button("Open");`
 - `Button bSave = new Button("Save");`
 - `Button bQuit = new Button("Exit");`

 - `Panel pSouth = new Panel();`
 - `TextArea ta = new TextArea("");`

Ces déclarations font l'objet des variables membres de la classe `MyFrameAWT`

Classes

- Classe principale : **MyAppAWT**
- Classe Frame : **MyFrameAWT**
- Autres Classes pour la gestion des événements: **MyActionListenerForOpen**

Classes

- Classe principale : **MyAppAWT**

```
class MyAppAWT
{
    public static void main (String args [])
    {
        System.out.println("Application AWT");
        MyFrameAWT f = new MyFrameAWT();
    }
}
```

Classes

■ Classe Frame : **MyFrameAWT**

```
import java.awt.*;
class MyFrameAWT extends Frame
{
    Panel pNorth = new Panel();
    Button bNew = new Button("New");
    Button bOpen = new Button("Open");
    Button bSave = new Button("Save");
    Button bQuit = new Button("Exit");

    Panel pSouth = new Panel();
    TextArea ta = new TextArea("");
}
```

Classe MyFrameAWT - Variables Membres -

■ Classe Frame : **MyFrameAWT**

```
import java.awt.*;
class MyFrameAWT extends Frame
{
    Panel pNorth = new Panel();
    Button bNew = new Button("New");
    Button bOpen = new Button("Open");
    Button bSave = new Button("Save");
    Button bQuit = new Button("Exit");

    Panel pSouth = new Panel();
    TextArea ta = new TextArea("");
}
```


Classe MyFrameAWT - Constructeurs -

■ Classe Frame : **MyFrameAWT**

```
class MyFrameAWT extends Frame
{
... // variables membres
// Constructeurs
Public MyFrameAWT () {
    pNorth.setBackground(Color.green);
    pNorth.add(bNew);    pNorth.add(bOpen);
    pNorth.add(bSave);   pNorth.add(bQuit);

    pSouth.setBackground(Color.yellow);
    pSouth.add(ta);

    this.setBackground(Color.blue);
    this.setLayout(new GridLayout(2,1));
    this.add(pNorth); this.add(pSouth);
    this.setBounds(10,10,500,400);
    this.setVisible(true);
}
}
```

Gestion des événements

- Fermeture de la fenêtre -

- Gestion des événements : fermeture de la fenêtre
- Créer une nouvelle classe appelée MyWindowListener qui hérite de **WindowAdapter**

```
MyWindowListener x1 = new MyWindowListener()  
This.addWindowListener(x1);
```

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

Gestion des événements

- Bouton Open -

- Gestion des événements : Bouton Open
- Créer une nouvelle classe appelée `MyActionListenerForOpen` qui implémente **ActionListener**
- Rajouter dans le constructeur `MyAWTFrame()` les deux lignes suivantes :

```
MyActionListenerForOpen x2 = new MyActionListenerForOpen ()  
bOpen.addActionListener(x2);
```

```
public class MyActionListenerForOpen implements ActionListener  
{  
    public void actionPerformed(ActionEvent e)  
    {  
        System.out.println("Bouton Open actionné");  
    }  
}
```

Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Ouverture du FileDialog -

- Classe cible : `MyActionListenerForOpen`
- Méthode cible : `ActionPerformed`
- Instructions :

1. Créer une instance de `FileDialog` ayant comme argument un `Frame`.
 - Faire passer comme argument l'objet « `this` » dans `MyFrameAWT`.

```
MyActionListenerForOpen x2 = new MyActionListenerForOpen(this);  
bOpen.addActionListener(x2);
```

- Rajouter un constructeur dans `MyActionListenerForOpen` qui admet comme argument un paramètre de type `MyFrameAWT`.

```
MyFrameAWT f;  
public MyActionListenerForOpen(MyFrameAWT f)  
{  
    this.f=f;  
}
```

Entée/Sortie

- Ouverture du FileDialog (suite) -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Instructions :
 1. Créer une instance de FileDialog ayant comme argument un Frame.

```
public void actionPerformed(ActionEvent e)
{
    System.out.println("Bouton Open actionné");
    FileDialog fd = new FileDialog(f);
    fd.setVisible(true);
}
```

- N'oublier pas d'importer les paquetages

```
import java.awt.*;
```

Entée/Sortie

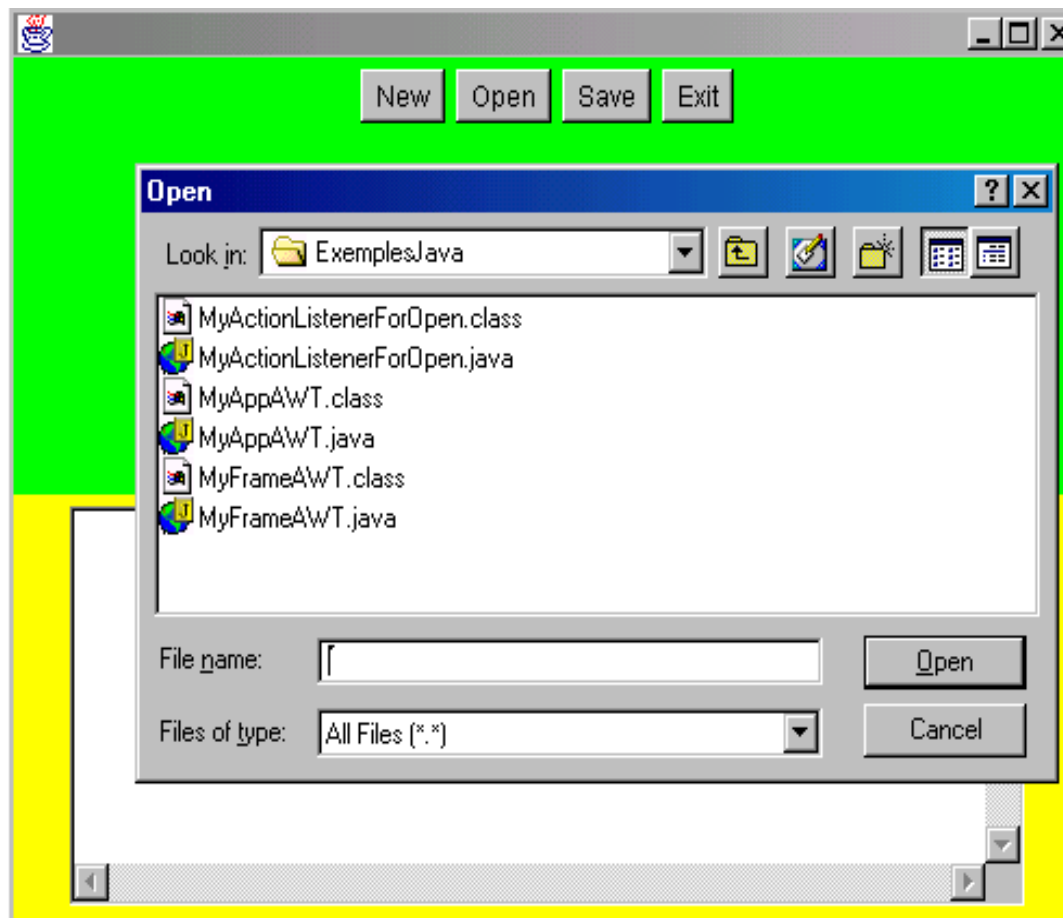
- Ouverture du FileDialog (Solution Complète) -

```
import java.awt.event.*;
import java.awt.*;

class MyActionListenerForOpen implements ActionListener
{
    MyFrameAWT f;
    public MyActionListenerForOpen(MyFrameAWT f)
    {
        this.f=f;
    }
    public void actionPerformed(ActionEvent e)
    {
        FileDialog fd = new FileDialog(f);
        fd.setVisible(true);
    }
}
```

Ouverture du FileDialog

- Résultat -



Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - **Sélectionner le fichier à lire**
 - **Afficher son répertoire**
 - **Afficher son PATH complet**
 2. Créer l'objet instance de File
 - Créer une instance de File
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
class MyActionListenerForOpen implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Bouton Open actionné");
        FileDialog fd = new FileDialog(f);
        fd.setVisible(true);

        String nomFichier = fd.getFile();
        String repFichier = fd.getDirectory();
        String nomComplet = repFichier + nomFichier;
        System.out.println("Nom Fichier : " + nomFichier);
        System.out.println("Répertoire Fichier : " + repFichier);
        System.out.println("Nom complet du Fichier : " +
            nomComplet);
    }
}
```

Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. **Créer l'objet instance de File**
 - **Créer l'objet file**
 - **Afficher la taille du fichier**
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
    // suite du code
    String nomFichier = fd.getFile();
    String repFichier = fd.getDirectory();
    String nomComplet = repFichier + nomFichier;
    System.out.println("Nom Fichier : " + nomFichier);
    System.out.println("Répertoire Fichier : " + repFichier);
    System.out.println("Nom complet du Fichier : " +
        nomComplet);
    File file = new File(nomComplet);
    int size;
    size = (int) file.length();
    System.out.println("Taille du Fichier : " + size);
}
}
```

Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. **Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream**
 4. Lire les données dans un tableau de bytes[]
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Sélection du fichier -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
    // suite du code
    File file = new File(nomComplet);
    int size;
    size = (int) file.length();
    System.out.println("Taille du Fichier : " + size);

    try
    {
        FileInputStream in = new FileInputStream(file);
    } catch (FileNotFoundException e2) {System.out.println(e2);}

}
}
```

Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. **Lire les données dans un tableau de bytes[]**
 5. Convertir le tableau en String
 6. Affecter le String dans le TextArea.

Entée/Sortie

- Lecture des données -

```
import java.awt.event.*;
import java.awt.*;
import java.io.*;
class MyActionListenerForOpen implements ActionListener
{
public void actionPerformed(ActionEvent e)
{
    // suite du code
    try
    {
        FileInputStream in = new FileInputStream(file);
        byte data[] = new byte[size];
        in.read(data);
    } catch (FileNotFoundException e2) {System.out.println(e2);}
    catch (IOException e3) {System.out.println(e3);}
}
}
```


Entée/Sortie

- Bouton Open -

- Classe cible : MyActionListenerForOpen
- Méthode cible : ActionPerformed
- Approche :
 1. Activer le FileDialog
 - Créer une instance de FileDialog ayant comme argument un Frame.
 - Sélectionner le fichier à lire
 - Afficher son répertoire
 - Afficher son PATH complet
 2. Créer l'objet instance de File
 - Créer l'objet file
 - Afficher la taille du fichier
 3. Ouvrir le fichier en mode Lecture :Créer l'objet instance de FileInputStream
 4. Lire les données dans un tableau de bytes[]
 5. **Convertir le tableau en String**
 6. **Affecter le String dans le TextArea.**

Entée/Sortie

- Sélection du fichier -

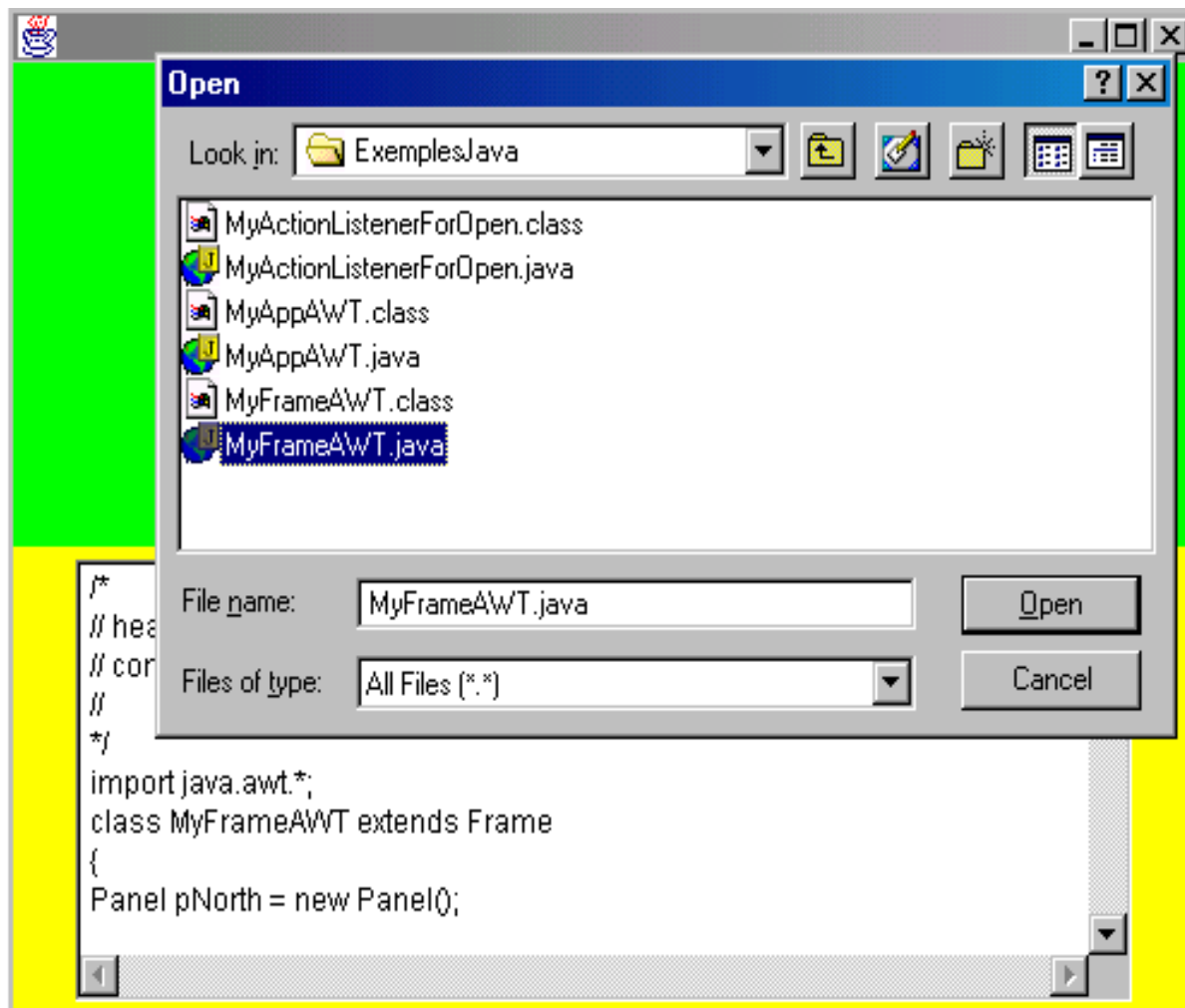
```
// import
class MyActionListenerForOpen implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        // suite du code

        try
        {
            FileInputStream in = new FileInputStream(file);
            byte data[] = new byte[size];
            in.read(data);

            String s = new String(data);
            f.ta.setText(s);

        } catch (FileNotFoundException e2) {System.out.println(e2);}
        catch (IOException e3) {System.out.println(e3);}
    }
}
```

Ouverture d'un fichier avec FileDialog



File Navigation and I/O

I/O File classes (File class)

- **File** : creates a java object (File MetaData-descriptor serving as a java driver which negotiates with the corresponding filesystem (AIX/jfs, Solaris/ufs, Windows/NTFS, etc.).
- Several methods can be invoked like :
 - delete()
 - renameTo()
 - exists()

File Reading classes

- **FileReader** This class is used to inform the filesystem that read operations would be requested. A `NoPermissionException` could be returned. Otherwise, the filesystem and JVM will prepare the suitable mechanism to serve future read access methods.
- **BufferedReader** This class is used to make `FileReader` more simple and efficient to use. Designer will find in `BufferedReader` better methods in term of flexibility and performance.

File Writing classes

- **FileWriter** This class is used to inform the filesystem that write operations would be requested. It is used to write to character files. Its write() methods allow you to write character(s) or Strings to the target file.
- **BufferedWriter** This class is similar to FileWriters and used to make some kind of write operations more efficient and easier to use.
- **PrintWriter** This class has been created to enhance mechanism and related methods of writing to a file. New methods like format(), printf(), and append() make PrintWriters very flexible and powerful.

Exercises : I/O and AWT

Exercise 1

- Write a java application called “CatFile.java” which prints on standard output a file (The file name is passed as argument).
- Rewrite “CatFile.java” into an interactive version called “CatFileFd.java” which invokes a FileDialog object if no argument is passed.

In the two versions all exception catches should be controlled and taken. For each case, give a solution with FileInputStream, an other with FileReader, and a third version with BufferedReader.

Exercise 2

- Write a java application called “WriteFile.java” which permits the construction and the writing of a new file (The file name is passed as argument). Data are written using keyboard. You can use a `Scanner(System.in)` instance to handle with keyboard input.

All exceptions should be caught. Give a solution with `FileOutputStream`, an other with `FileWriter`, a third version with `BufferedWriter` and a fourth version with `PrintWriter`.

Exercise 3

- Rewrite “WriteFile.java” into an interactive version called “WriteFileFd.java” which invokes a FileDialog object if no argument is passed.
- Write a java application called “AppendFile.java” which permits to add new data to an existing file (The file name is passed as argument).

Rewrite “AppendFile.java” into an interactive version called “AppendFileFd.java” which invokes a FileDialog object if no argument is passed.

Exercise 4

Rewrite all the previous four Java applications into new versions using Frames and TextAreas. The frame should be splitted into two Panels. The top panel (Refer to Panel class in javadoc) contains the file proprieties and the bottom panel contains the TextArea. This latter (TextArea) will contains the data of the file. In other hand, the Menubar should be used to interact with the system (for example an open MenuItem will invokes the FileDialogue to select the desired file. I Addition, a “Format” Menu attached to The Menubar with help the user to customize displayed data in TextArea space (for example a Size MenuItem will help to adjust police size “12” or “14”).