PROGRAMMATION OBJET-JAVA

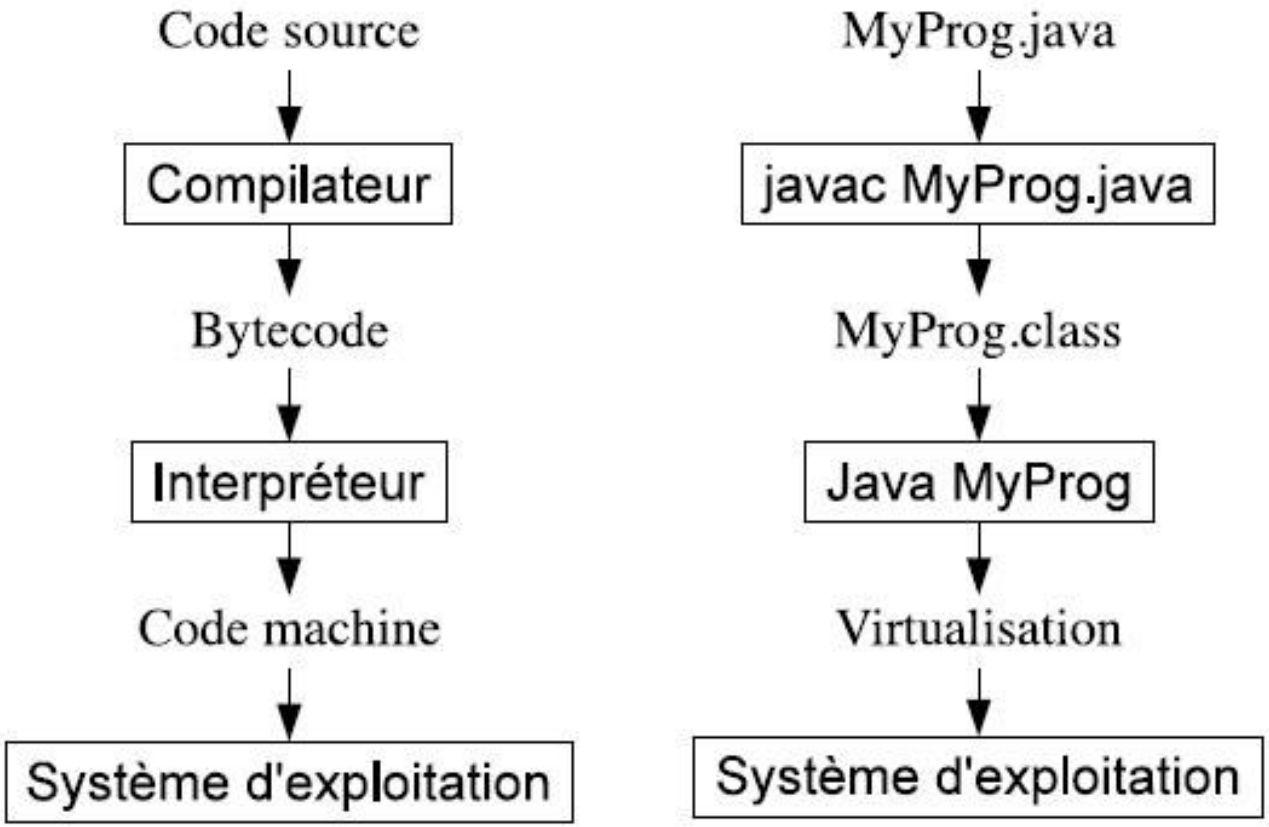
SAMIA CHELBI
MAITRE TECHNOLOGUE
ENTREPREUNEUR CREATIF
PRESIDENTE CREATEC ASSOCIATION

INTRODUCTION

- JAVA est un langage de programmation
  - **Orienté objet**
  - **Modulaire**
  - **Rigoureux**
  - **Portable**
  - **Lent à l'exécution**

Environnement Java

Bonus Java: ByteCode + Machine virtuelle



Programmation orientée-objet

- **Langage C**:programmation **procédurale** car il suppose que le programmeur s'intéresse en priorité aux **traitements** que son programme devra effectuer.
- **Programmation orientée-objet** :méthodologie centrée **données**
- Programmeur Java doit identifier un ensemble **d'objets**
- **Objet= {données , méthodes }**
- Un objet= est une variable donc associée à un type.
- **Type= classe**
- **Classe est une structure de données**

Exemple:

```
class Rectangle {
//données
int longueur ;
int largeur ;
int origine_x ;
int origine_y ;
//méthodes
void deplace(int x, int y) {
this.origine_x = this.origine_x + x ;
this.origine_y = this.origine_y + y ;
}
int surface() {
return this.longueur * this.largeur ;
}
}
```

Encapsulation

- Penser Objets c'est:
  - identifier les objets et leurs données
  - **Mais aussi les droits d'accès qu'ont les autres objets sur ces données.**

- **L'encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme.**
- Une donnée peut être déclarée en accès :
- **public : permission d'accès en lecture & modification**
- **privé : protection contre l'accès externe et directe aux données mais indirectement par des méthodes (public) de l'objet concerné**

Méthode constructeur

- Chaque classe doit définir une ou plusieurs méthodes particulières appelées des **constructeurs.**
- Un **constructeur est une méthode invoquée lors de la création d'un objet.**
- Cette méthode, qui peut être vide, effectue les opérations nécessaires à l'initialisation d'un objet.
- Chaque constructeur doit avoir **le même nom que la classe** où il est défini et n'a aucune valeur de retour (c'est l'objet créé qui est renvoyé).

Exemple constructeur

- class Rectangle {

```
...
Rectangle(int lon, int lar) {
        this.longueur = lon ;
        this.largeur = lar ;
        this.origine_x = 0 ;
        this.origine_y = 0 ;
}
...
}
```

Objet

- **Instance d'une classe**
- Exemple de création d'un objet de type Rectangle:
  - **Déclaration:**
    - **Rectangle r1;**
  - **Instanciation:**
    - **r1= new Rectangle();**
    - **r1=new Rectangle(15,5);**

Accès aux variables et aux méthodes

- **int temp = r1.longueur ;**
- **r1.deplace(10,-3);**

this

- Pour référencer l'objet "courant"
- Exemple

```
class Carre {
int cote ;
int origine_x ;
int origine_y ;
Carre(int cote, int x, int y) {
        this.cote = cote ;
        this.origine_x = x ;
        this.origine_y = y ;
}
Carre(int cote) {
        this(cote, 0, 0);
}
}
```

Sythaxe du Langage

- le caractère de fin d'une instruction est " **;**"
- **int a ;** ***// ce commentaire tient sur une ligne***
- */*Ce commentaire nécessite*

2 lignes/*

- identificateurs de variables ou de méthodes: **{a..z}, {A..Z}, $, _, {'0'..'9'}** (~~4iset~~).

Types de données

- **Types primitifs:**

Type	Classe éq.	Valeurs	Portée	Défaut
`boolean`	`Boolean`	`true` ou `false`	N/A	`false`
`byte`	`Byte`	entier signé	{-128..128}	o
`char`	`Character`	caractère	{\uoooo..\uFFFF}	\uoooo
`short`	`Short`	entier signé	{-32768..32767}	0
`int`	`Integer`	entier signé	{-2147483648..2147483647}	0
`long`	`Long`	entier signé	$\{-2^{31}..2^{31}-1\}$	0
`float`	`Float`	réel signé	$\{-3,4028234^{38}..3,4028234^{38}\}$ $\{-1,40239846^{-45}..1,40239846^{-45}\}$	0.0
`double`	`Double`	réel signé	$\{-1,797693134^{308}..1,797693134^{308}\}$ $\{-4,94065645^{-324}..4,94065645^{-324}\}$	0.0

Classe liée au type primitive

- type primitif possède une classe qui encapsule un attribut du type primitif
- Exemple **Integer**
- **Java est un langage fortement typé**
- **Exemples**

- **int a ;**
  double b = 5.0 ;
  a = b ;
- **int a ;**
  double b = 5.0 ;
  a = (int)b ; (cast)

Tableaux et matrices

- int[] mon_tableau ;
- int mon_tableau2[];
- int[] mon_tableau = new int[20];
- mon_tableau.length
- int[][] ma_matrice;

Chaînes de caractères

- classe **String** (java.lang)
- Les variables de type String:
  - **leur valeur ne peut pas être modifiée**
  - **+ pour concaténer deux chaînes de caractères**
- **Exemples:**
- String s1 = ”hello” ;
  String s2 = ”world” ;
  String s3 = s1 + ” ” + s2 ;
- **String s = new String(); *//pour une chaine vide***
  String s2 = new String(”hello world”); *// pour une chaîne de valeur ”hello world”*

Opérateurs

ordre de priorité décroissante

<ari> valeur arithmétique
<boo> valeur booléenne
<cla> classe
<ent> valeur entière
<ins> instruction
<str> chaîne de caractères (String)
<val> valeur quelconque
<var> variable

Pr.	Opérateur	Syntaxe	Résultat	Signification
1	++	++<ari>	<ari>	pré incrémentation
		<ari>++	<ari>	post incrémentation
	–	–<ari>	<ari>	pré décrémentation
		<ari>–	<ari>	post décrémentation
	+	+<ari>	<ari>	signe positif
	-	-<ari>	<ari>	signe négatif
	!	!<boo>	<boo>	complément logique
	(*type*)	(*type*)<val>	<val>	changement de type
2	*	<ari>*<ari>	<ari>	multiplication
	/	<ari>/<ari>	<ari>	division
	%	<ari>%<ari>	<ari>	reste de la division
3	+	<ari>+<ari>	<ari>	addition
	-	<ari>-<ari>	<ari>	soustraction
	+	<str>+<str>	<str>	concaténation
4	<<	<ent> << <ent>	<ent>	décalage de bits à gauche
	>>	<ent> >> <ent>	<ent>	décalage de bits à droite
5	<	<ari> < <ari>	<boo>	inférieur à
	<=	<ari> <= <ari>	<boo>	inférieur ou égal à
	>	<ari> > <ari>	<boo>	supérieur à
	>=	<ari> >= <ari>	<boo>	supérieur ou égal à
	instanceof	<val>instanceof<cla>	<boo>	test de type
6	==	<val>==<val>	<boo>	égal à
	!=	<val>!=<val>	<boo>	différent de
7	&	<ent>&<ent>	<ent>	ET bit à bit
		<boo>&<boo>	<boo>	ET booléen
8	^	<ent>^<ent>	<ent>	OU exclusif bit à bit
		<boo>^<boo>	<boo>	OU exclusif booléen
9	|	<ent>|<ent>	<ent>	OU bit à bit
		<boo>|<boo>	<boo>	OU booléen
10	&&	<boo>&&<boo>	<boo>	ET logique
11	||	<boo>||<boo>	<boo>	OU logique
12	?:	<boo>?<ins>:<ins>	<ins>	si...alors...sinon
13	=	<var>=<val>	<val>	assignation

Structures de contrôle

- **if (<condition>) <bloc1> [else <bloc2>]**
- **<condition>?<instruction1>:<instruction2>**
- **Exemple**

```
if (a == b) {
        a = 50 ;
        b = 0 ;
} else {
        a = a - 1 ;
}
```

Instructions itératives

- **while (<condition>) <bloc>**
- Exemple :
  - **while (a != b) a++;**
- **do <bloc> while (<condition>);**
- Exemple :

```
do a++
while (a != b);
```

Instructions itératives

- **for (<init>;<condition>;<instr_post_itération>) <bloc>**
- Exemple :

```
for (int i = 0, j = 49 ; (i < 25) && (j >= 25); i++, j--) {
        if (tab[i] > tab[j]) {
        int tampon = tab[j];
```