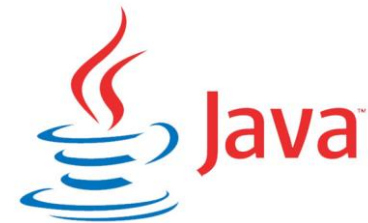


Langage Java



Promotion: élèves ingénieurs de 1^{ère} année (TIC-1)

ENSEIGNANT: SLAH BOUHARI

Plan

Notions de base du langage Java

Héritage et polymorphisme

Aspects avancés des classes

Package et classes utilitaires

Gestion des erreurs et des E/S

Partie 1:

NOTIONS DE BASE DU LANGAGE JAVA

Notions de base du langage Java

Atout de Java

Fonctionnement de la JVM

Kit Java SE

Choix de l'environnement approprié

Application indépendante

Fichier source Java

Types de variables

Les opérateurs arithmétiques et logiques

Les structures conditionnelles et répétitives

Atout de Java

Un **langage orienté objet** développé par Sun Microsystems (rachetée par Oracle).

Doté d'un ensemble de **bibliothèques (API)** très riches et très variées: interfaces graphiques, 2D, programmation multitâches, réseau,...

Sûr : il effectue de nombreuses vérifications au chargement des classes et durant leur exécution.

Adapté à Internet : architectures multi-tiers et traitement distribué .

Atout de Java

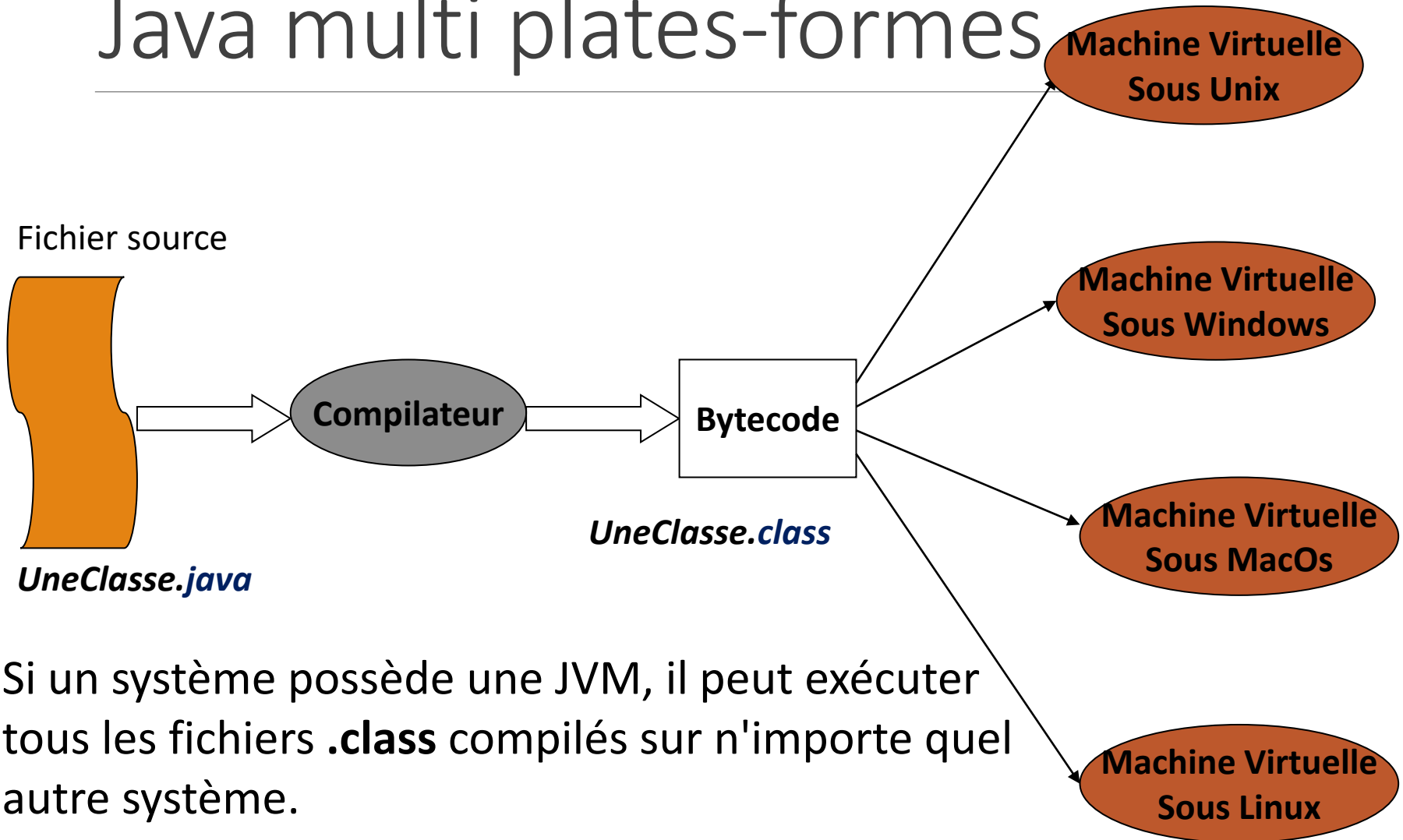
doté d'un mécanisme de gestion des erreurs (**exceptions**).

Portable : (multi plates-formes) les programmes Java s'exécutent sans aucune modification sur tout les environnements (Windows, Unix, Mac).

Grâce à l'exécution par une **machine virtuelle (JVM)** : "Write once, run everywhere".

Le code source est traduit dans un langage appelé "**bytecode**".

Java multi plates-formes



Si un système possède une JVM, il peut exécuter tous les fichiers **.class** compilés sur n'importe quel autre système.

Fonctionnement de la JVM

Le programme de chargement de classes charge toutes les classes requises.

- La JVM utilise un paramètre CLASSPATH pour localiser les fichiers de classe.

Le vérificateur JVM recherche les codes exécutables interdits

Le vérificateur JVM exécute les codes exécutables

- La JVM peut appeler un compilateur JIT(Just-in-time)

Le gestionnaire de mémoire rend au système d'exploitation la mémoire utilisée par l'objet déréférencé.

- La JVM gère le nettoyage de la mémoire

Kit Java SE (Java Standard Edition)

Le kit Java SE fournit plusieurs composants:

- Le compilateur Java (***javac***)
- Bibliothèque de classes standard (***rt.jar***)
- Débogueur (***jdb***)
- L'interpréteur de code exécutable Java (***java***)
- Générateur de documentation (***javadoc***)
- Utilitaire Java archive (***jar***)
- Autres

Choix de l'environnement approprié

Java présente trois versions:

Java SE (J2SE) : Environnement de développement complet pour Internet.

Java EE (J2EE) : Enterprise Edition qui ajoute (en plus des composants de Java SE) un serveur d'applications et des outils de prototypage.

Java ME (J2ME) : Micro Edition, version pour écrire des programmes embarqués (carte à puce/Java card, téléphone portable,...)

Application indépendante

Une application indépendante :

- est exécutée uniquement par la JVM,
- possède un point d'entrée unique, la méthode main(), de prototype :

```
public static void main(String argv[])
```

Premier programme Java

```
class HelloWorld {  
  
public static void main(String[] args)  
{  
    System.out.println("Hello world");  
}  
  
}
```

Fichier source Java

Un fichier source Java peut contenir trois structures de premier niveau:

- Un seul mot clé **package** par fichier, suivi du nom du package (première ligne du fichier)
- D'éventuelles instructions **import**, suivies de noms de classes entièrement qualifiés ou du caractère "*" qualifié par le nom d'un package
- Une ou plusieurs définitions **class** ou **interface** suivies d'un nom et d'un bloc.

Un fichier doit présenter le même nom que la classe publique ou l'interface publique.

Fichier source Java : Lecture.java

```
package input;
import java.util.*;

public class Lecture
{
    public static void main(String[] args)
    {
        String nom = ""; int age = 0;
        Scanner Sc = new Scanner(System.in);
        System.out.println("Entrez votre nom:");
        nom = Sc.next();
        System.out.println("Entrez votre age:");
        age = Sc.nextInt();
        System.out.println("votre nom est "+ nom +" votre age
est "+ age);
    }
}
```

Déclaration des variables

Modificateur d'accès	Niveau d'accès de la variable
<i>static</i>	La variable est une variable de classe
final	La variable est une constante
transient	S'il ne faut pas sauvegarder la variable dans un fichier
volatile	Pour indiquer au compilateur de ne pas optimiser cette variable
type nomvariable	Type et nom de la variable
= initialisation	Initialisation de la variable

Types de variables

Les Entiers

On distingue quatre types d'entiers :

- byte
- short
- int
- long

	byte	short	int	long
Taille (oct)	1	2	4	8
Etendue	-128 .. 127	-32768 .. 32767	$-2^{31} .. 2^{31}-1$	$-2^{63} .. 2^{63}-1$

Types de variables

Les décimaux:

On distingue deux types de décimaux :

- float
- double

	float	double
Taille (oct)	4	8
Etendue	1.4E-45 .. 3.4E+ 38	4.9E-324 .. 1.8E+ 308

Le type **boolean** accepte seulement deux états:

- true
- false

Le Type caractère

Une variable de type caractère est introduite par le mot clé **char**.

Un caractère (char) est codé sur 2 octets par le codage Unicode (et pas ASCII).

Un caractère Unicode entouré par « ' » 'A' ,'\t'

CR et LF interdits (caractères de fin de ligne)

'\u20ac' (\u suivi du code hexadécimal d'un caractère Unicode ; représente €)

Opérateurs arithmétiques

Opérateur	Utilisation
+	op1 + op2
-	op1 - op2
*	op1 * op2
/	op1 / op2
%	op1 % op2

Opérateurs relationnels

Opérateur	
>	Strictement inférieur
<	Strictement supérieur
>=	Supérieur ou égal
<=	Inférieur ou égal
==	égal
!=	différent

Opérateurs logiques

Opérateur	Utilisation	
&&	<code>exp1 && exp2</code>	Si <code>exp1</code> vrai évaluer <code>exp2</code>
 	<code>Exp1 exp2</code>	Si <code>exp1</code> faux évaluer <code>exp2</code>
&	<code>exp1 & exp2</code>	« et » logique
 	<code>exp1 exp2</code>	« ou » inclusif
^	<code>exp1 ^ exp2</code>	« ou » exclusif
!	<code>! exp1</code>	négation

Opérateurs d'affectation

Opérateur	Utilisation	
=	<code>op1 = op2</code>	Affectation de valeur
+=	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
-=	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
*=	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
/=	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
%=	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>
&=	<code>op1 &= op2</code>	<code>op1 = op1 & op2</code>
 =	<code>op1 = op2</code>	<code>op1 = op1 op2</code>
^=	<code>op1 ^= op2</code>	<code>op1 = op1 ^ op2</code>

L'opérateur conditionnel

`exp1 ? op2 : op3`

est équivalent à

```
if (exp1)
    op2
else
    op3
```

Structures conditionnelles

if (*expression Booléenne*)
 bloc-instructions ou instruction
else
 bloc-instructions ou instruction

if (*expression Booléenne*)
 bloc-instructions ou instruction
else
 if(*expression Booléenne*)
 bloc-instructions
 else
 bloc-instructions

```
if (a>b)
{
    if (a>c)
        {max = a;}
    else
        {max = c;}
}
else
{
    if (b>c)
        {max =b;}
    else
        {max = c;}
}
```


Structure "switch"

- ▶ **switch**(*expression*) {
 case *val1*: *instructions*;
 break;
 ...
 case *valn*: *instructions*;
 break;
 default: *instructions*;
}
- ▶ *expression* est de type *char*,
 byte, *short*, ou *int*

```
char lettre;  
int nbVoyelles = 0,  
    nbA = 0, nbT = 0,  
    nbAutre = 0;  
  
...  
switch (lettre) {  
  case 'a' : nbA++;  
  case 'e' : // pas  
            d'instruction !  
  case 'i' : nbVoyelles++;  
            break;  
  case 't' : nbT++;  
            break;  
  default : nbAutre++;  
}
```

Structures répétitives

```
while(expressionBooléenne)  
{  
    bloc-instructions ou  
    instruction  
}  
do  
{  
    bloc-instructions ou  
    instruction  
}  
while(expressionBooléenne);
```

```
for(init ; test ; incrément)  
{  
    instructions;  
}  
est équivalent à  
  
    init;  
    while (test) {  
        instructions;  
        incrément;  
    }
```

L'instruction `break`

Elle permet de quitter une boucle ou une instruction `switch`.

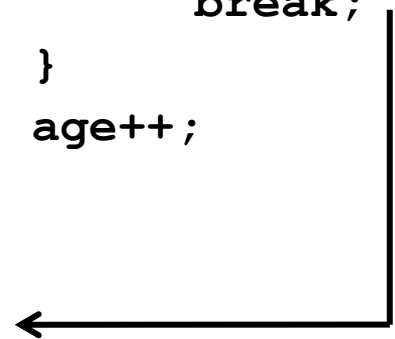
Elle transfère le contrôle à la première instruction après le corps de la boucle ou l'instruction `switch`.

Elle permet de simplifier le code, mais elle doit être utilisée avec parcimonie

Example

...

```
while (age <= 65)
{
    balance = (balance + payment) * (1 + interest) ;
    if ( balance >= 250000 )
    {
        break;
    }
    age++;
}
```



...

L'instruction continue

Elle ignore l'itération d'une boucle

Elle passe à l'instruction suivante

Exemple:

```
...  
for (int i = 0; i<=10; i++)  
{  
    //dépasser les nombres impaires  
    if ( i % 2 != 0)  
    {  
        continue;  
    }  
    System.out.print(i + " ");  
}  
...
```

CRÉER DES CLASSES ET DES OBJETS

Créer des classes et des objets

Déclaration d'une classe

Création des objets

Les variables d'instance

Les variables de classe

Les méthodes d'une classe

Récupérer la mémoire

La classe String

Les Tableaux

Déclaration d'une classe

public	Modificateur d'accès à la classe
abstract	La classe ne peut être instanciée
final	La classe ne peut être héritée
class nomDeLaClasse	Nom de la classe (obligatoire)
extends superClasse	La classe mère dont la classe hérite
implements nomInterfaces	Interfaces implémentées par la classe
{ /* attributs (variables et	méthodes) de la classe */ }

Création des objets

Les objets sont généralement créés à l'aide de l'opérateur `new`.

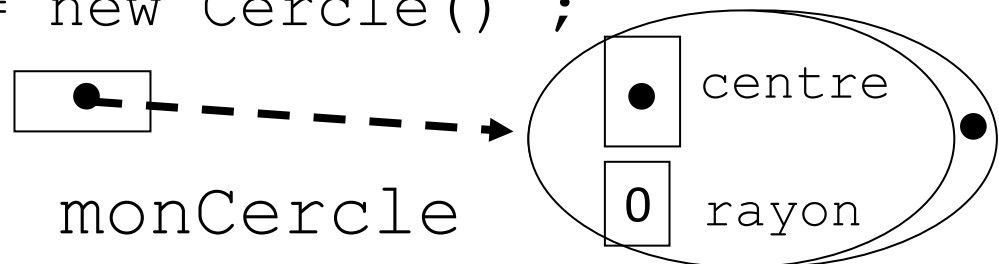
Rôle de l'opérateur `new`:

1. Allocation de la mémoire nécessaire à l'état de cet objet.
2. Appel de la méthode d'initialisation spéciale dans la classe (constructeur).
3. Envoi de la référence au nouvel objet

Example

```
class Cercle
{
    Point    centre;
    double   rayon;
    ...
}
```

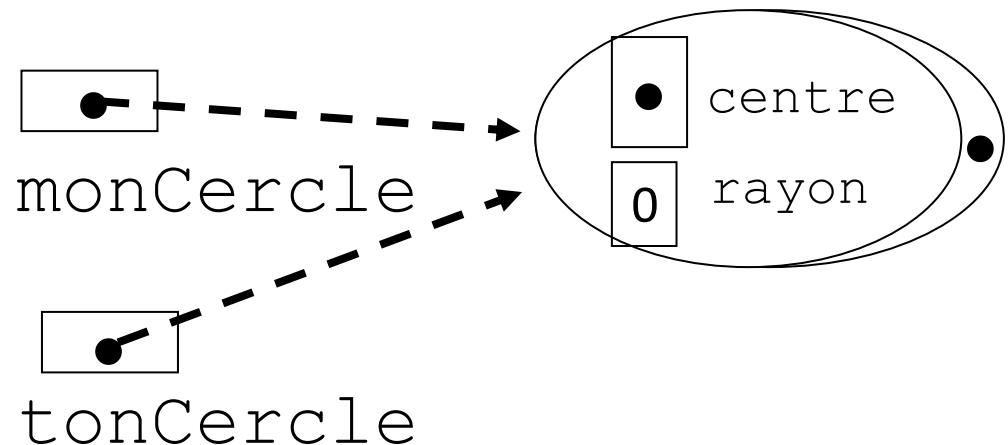
Cercle monCercle = new Cercle() ;



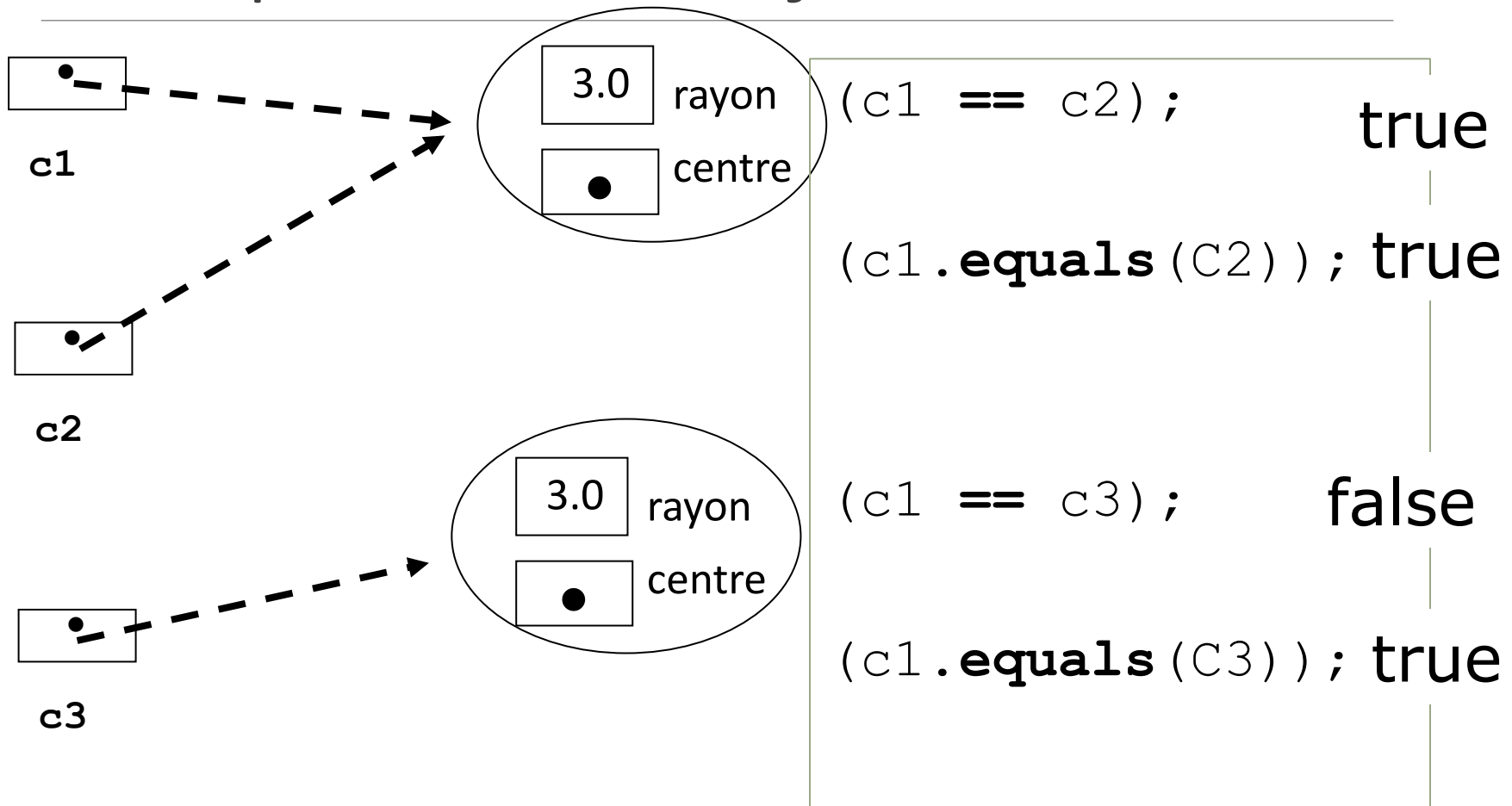
Affecter des références

L'affectation d'une référence à une autre entraîne deux références au même objet

```
Cercle monCercle = new Cercle() ;  
Cercle tonCercle;  
tonCercle = monCercle;
```



Comparer des objets



Les variables d'instance

Les variables d'instance sont des variables globales pour toutes les méthodes de cette classe.

Les variables d'instance est une structure implicite qui sera créée pour chaque objet de la classe.

Les variables d'instance peuvent êtres:

- Variable de primitive: déclarée avec un type primitif de Java et initialisée avec la valeur par défaut ou la valeur donnée.
- Variable objet: déclarée à base d'un autre objet et initialisée avec la référence **null**.

Les valeurs par défaut

Type	Valeur initiale par défaut
boolean	false
char	\u0000
byte	(byte)0
short	(short)0
int	0
long	0L
float	+0.0f
double	+0.0d
Reference	null

Accès aux variables d'instance

L'opérateur '.' permet d'accéder aux variables d'instance (si le modificateur d'accès le permet)

```
class Cercle
{
    Point    centre;
    double   rayon;
    ...
}
```

```
Cercle monCercle = new Cercle() ;
monCercle.rayon = 2.5;
```

Les variables de classe

Appartiennent à une classe et sont communes à toutes les instances de cette classe.

Sont déclarées comme étant `static` dans les définitions de classe.

Peuvent être initialisées au moment de la déclaration.

```
class Cercle
{
    Point    centre;
    double   rayon;
    static Color couleur;
}
```


Les méthodes d'une classe

Les constructeurs

Le destructeur

La méthode main

Les méthodes (membres)

Les accesseurs

Les constructeurs

Un constructeur est une méthode servant à définir l'état initial des objets instanciés.

Un constructeur:

- n'a pas de type de retour
- a le même nom que la classe dans laquelle il est défini.
- généralement déclaré comme `public`

Les constructeurs acceptent la surcharge.

2 règles sur les constructeurs

Si aucun constructeur n'est spécifié, dans la définition de la classe, un constructeur par défaut est obligatoirement fournie sans paramètres.

Si vous en définissez au moins un, le constructeur par défaut n'est plus fourni. Si vous en avez utilisé il vous faudra alors le définir explicitement.

Surcharge du constructeur

```
class Cercle {  
    ...  
    public Cercle() {  
        centre = new Point();  
        rayon   = 0;  
    }  
    public Cercle(Point p, double r) {  
        centre = p;  
        rayon  = r;  
    }  
}
```

Récupérer la mémoire

Lorsque toutes les références à un objet sont perdu, l'objet est marqué comme disponible pour un nettoyage de la mémoire.

Le nettoyage de la mémoire (récupération) est automatique : le Garbage Collector est chargé de cette fonction.

Le garbage collector peut être appelé à la demande en invoquant la méthode `gc()` de la classe `System`.

Le Garbage Collector

Avant de supprimer un objet, le garbage collector exécute la méthode `finalize()` associée à cet objet.

Chaque classe peut redéfinir la méthode `finalize()` qui sera donc appelée automatiquement avant que l'espace réservé par les objets de la classe soit libéré.

La méthode `System.runFinalization()` peut être utilisée pour forcer l'exécution immédiate des méthodes `finalize()` des objets qui ne sont plus référencés.

Déclaration des méthodes

niveau d'accès	Niveau d'accès de la méthode
<code>static</code>	Méthode de classe
<code>abstract</code>	Méthode non implémentée
<code>final</code>	La méthode ne peut être redéfinie
<code>native</code>	Méthode implémentée en c ou en c++
<code>synchronized</code>	Liée au mécanisme de thread
typeRésultat nomMéthode	Type du résultat et nom de la méthode
(paramListe)	Liste des types et noms des paramètres formels
<code>throws exceptions</code>	Lié au mécanisme de gestion des exceptions

Signature des méthodes

La signature comprend :

- le nom de la méthode
- le nombre et type des paramètres dans l'ordre

Tous les paramètres sont transmis par valeur, il s'en suit que :

- Un argument objet est passé par **référence**.
- Un tableau objet est passé par **référence**.
- Les variables de type simple sont passées par **copie de valeur**.

Surcharger une méthode

Dans une classe, plusieurs méthodes peuvent porter le même nom.

Les méthodes doivent posséder différentes signatures.

```
...  
void deplacer(double dx ,double dy) {  
    centre.x = centre.x + dx ;  
    centre.y = centre.y + dy ;  
}  
void deplacer(double x) {  
    centre.x = centre.x + x ;  
}  
void deplacer() {  
    centre.x = centre.x + 1.0 ;  
    centre.y = centre.y + 1.0 ;  
}
```

Méthodes accesseurs

Ils permettent l'accès individualisé à chaque variable de classe soit en lecture soit en mise à jour.

```
class Point {  
    double x, y;  
  
    public int    getX()        { return x; }  
    public int    getY()        { return y; }  
    public void    setX(double x1) { x= x1; }  
    public void    setY(double y1) { y= y1; }  
    ...  
}
```

La classe String

String est une **classe** de Java qui décrit des objets qui contiennent des chaînes de caractères **constantes**.

```
String chaine = "Bonjour";
```

new force la création d'une nouvelle chaîne

```
chaine1 = "Bonjour";
```

```
chaine2 = new String("Bonjour");
```

La classe String

```
String chaine = "Bonjour";  
chaine = "Hello";
```

La dernière instruction correspond aux étapes suivantes :

- 1) Une nouvelle valeur (*Hello*) est créée
- 2) La variable **chaine** référence la nouvelle chaîne *Hello* (et plus l'ancienne chaîne *Bonjour*)
- 3) La place occupée par la chaîne *Bonjour* pourra être récupérée à un moment ultérieur par le ramasse-miette.

La méthode **equals()** teste si 2 instances de **String** contiennent la même valeur

Quelques méthodes de String

Les caractères d'une **String** sont numérotés de 0 à **length() - 1**.

on emploie la méthode **char charAt(int i)** pour extraire le *i*^{ème} caractère.

startsWith, **endsWith**, **trim** (enlève les espaces de début et de fin),

toUpperCase, **toLowerCase**, **valueOf** (conversions en **String** de types primitifs, tableaux de caractères)

Les Tableaux

En Java les tableaux sont considérés comme des objets (donc la classe hérite de **Object**) :

- les variables de type tableau contiennent des références aux tableaux,
- les tableaux sont créés par l'opérateur **new**,
- ils ont une variable d'instance (**final**) : **final int length**
- ils héritent des méthodes d'instance de **Object**

Déclaration et création des tableaux

Déclaration : la taille n'est pas fixée

```
int[] tabEntiers;
```

tabEntiers

Création : on doit donner la taille

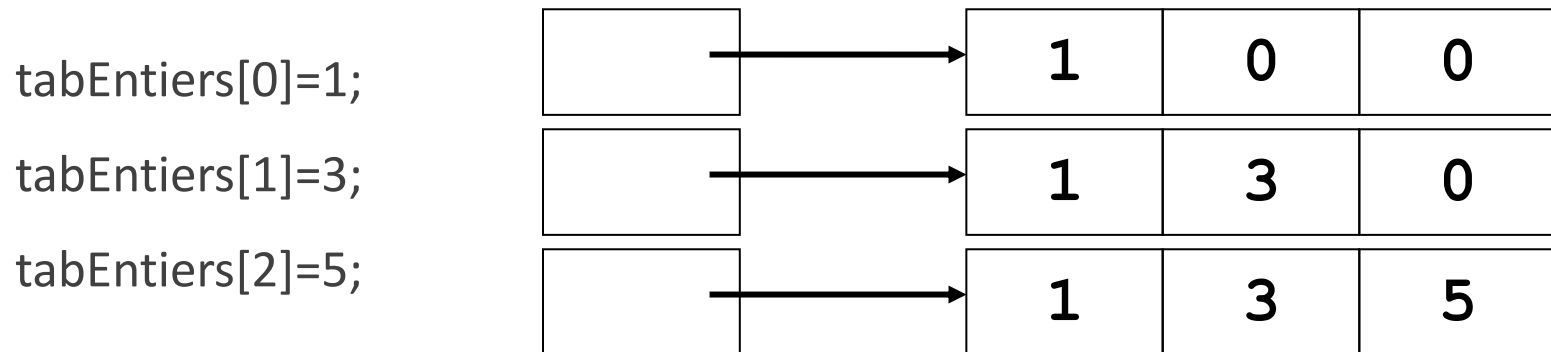
```
tabEntiers = new int[3];
```



Chaque élément du tableau reçoit la valeur par défaut du type de base du tableau. La taille ne pourra plus être modifiée par la suite

Affectation des éléments

Affectation des éléments ; l'indice commence à 0 et se termine à `tabEntiers.length - 1`



Taille du tableau

```
int l = tabEntiers.length;
```