ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

M.2.5.1. Paradigme Objet

ENSIAS

Plan

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

- **Classes Java**
- **Objets Java**

Classes Java

```
public class Compte {
    private int code;
    private String nom;
    private double solde;

    public void deposer(double montant) {
        solde += montant;
    }
    public int getSolde(){return solde; }
}
```

Compte

Champs/Variables
code, nom, solde

Méthodes
deposer()
retirer()
getSolde()

Contrôle d'accès

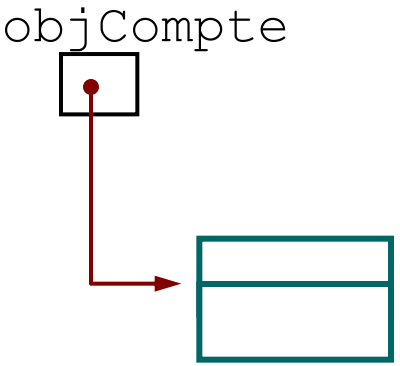
public	*Accessible par tout le monde*
abstract	*Ne peut être instanciée*
final	*Ne peut être héritée*
<défaut>	*Accessible par les classes du package*

Objets : création

Création

⇨ *en 3 temps*

```
Compte objCompte = new Compte();
```



1. Déclarer l'objet (référence)
2. Créer l'instance (allocation …)
3. Affecter la référence

Accès aux champs

```
objCompte.solde += montant ;
objCompte.débiter(4500.0);
```

```
new Compte().débiter(4500.0); // C'est possible
```

Objets : constructeurs

Constructeurs ⇨ Création de l'instance

Compte.java

```
public class Compte {
  private int code;
  private String nom;
  private double solde;
  public Compte(){//par défaut
    this.nom = null; this.solde=0.0
  public Compte(String nom){
    this.nom = nom; this.solde=0.0;}
  public Compte(String nom, double solde){
    this(nom);this.solde = solde;}
}
```

A partir du moment où un constructeur est déclaré, celui par défaut n'existe plus

this : *référence à l'objet*

this() : *constructeur de la classe en cours*

Objets : constructeurs

Exercice

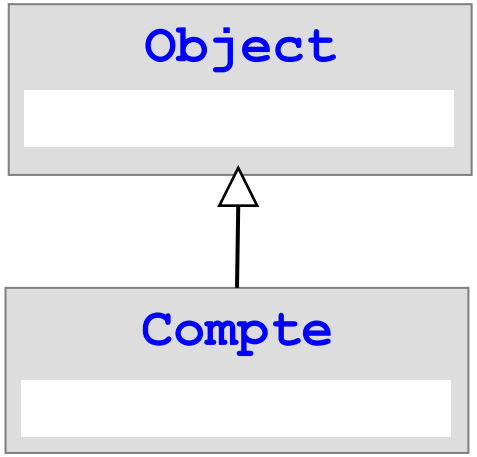
Quel constructeur va-t-il être appelé lorsqu'on exécute :

a) `new Ex(new Compte())`

b) `new Ex(null)`

```
public Ex(Compte d) { }
public Ex(Object d) { }
```

Car plus spécifique

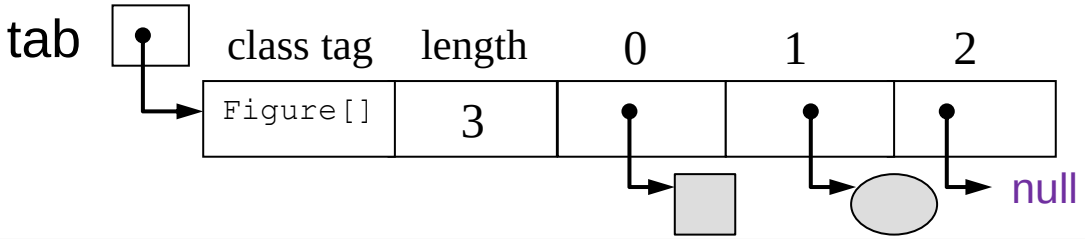


Objets : constructeurs

Tableaux ... sont des objets

1. **Déclaration** `Figure [] tab;`
2. **Dimensionnement** `tab = new Figure[3];`
3. **Initialisation**

```
tab[0] = new Carré(12);
tab[1] = new Cercle(9);
```




```
Figure [] tab = {// Pour les tableaux de petite taille
                 new Carré(12), new Cercle(9), null
                 };
```

Objets : constructeurs

Exercice

Quel constructeur va-t-il être appelé lorsqu'on exécute :

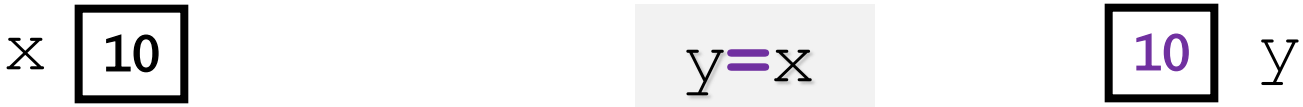
```
new   Ex(null)
```

```
public Ex(double[] d) { }
public Ex(Object d) { }
```

Objets : constructeurs

Affectation

1. x et y de type primitif (`int, double, …`)

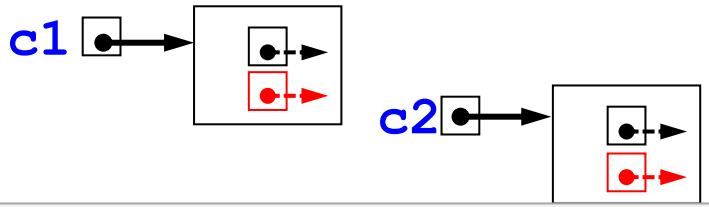


2. a et b des objets

Objets : constructeurs

Constructeur par copie

```
Compte c1 = new Compte("Owner",new Date());
Compte c2 = new Compte(c1);
```

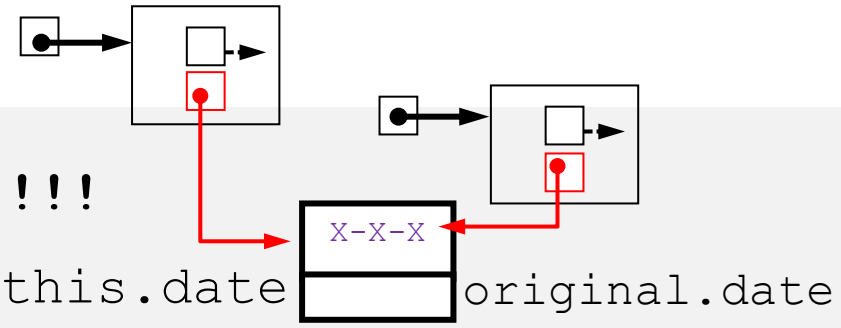



```
public class Compte {
   private String nom;//propriétaire du compte
   private Date date; //date de création du compte
   …
   public Compte(Compte original){
      //…
   }
}
```

Objets : constructeurs

Constructeur par copie

```
public Compte(Compte original){
    if(original == null ) {
        System.out.println("Erreur fatale.");
        System.exit(0);
    }
    this.nom = original.nom;
    this.date = original.date; // Not Good
    this.date = new Date(original.date); // Good
}
}
```



this.setDate(newDate);!!!

Même objet Date

Objets : constructeurs

Constructeur par copie

```
        this.nom = original.nom; // Good
//
        this.date = new Date(original.date);
```

Alors pourquoi ça marche pour `this.nom` ?

- La classe `String` est **immutable**

  Ne contient pas de méthodes qui risquent de la changer

- La classe `Date` est **mutable**

Objets : attributs

Variables d'instance

Chaque objet à ses propres valeurs

Compte.java

```
public class Compte {
     private int code;
     protected String nom;
     protected double solde;
     // ...
}
```

Contrôle d'accès

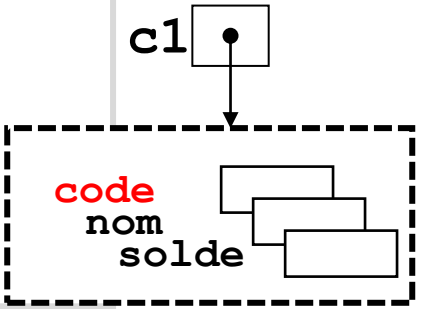
private: *classe uniquement*

public: *visible partout*

protected: *sous-classes/package*

final : *Ne peut être modifié*

```
public class Test {
   public static void main(String[] args){
     Compte c1= new Compte();
      c1.code = 2134;  // illegal
     return;
   }
```



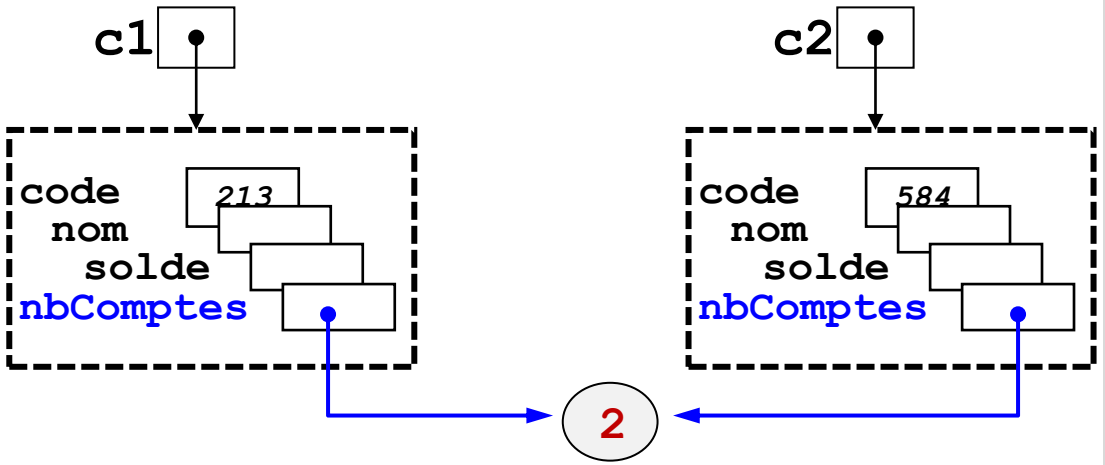
Objets : attributs

Variables statiques (de classe)

référence commune à tous les objets

Compte.java

```
public class Compte {
  private double solde; //...
  private static int nbComptes;
  //...
```



Inutile d'instancier pour y accéder

```
System.out.println(Compte.nbComptes);
```

Objets : méthodes

Méthodes d'instance

Déclaration et contrôle d'accès

```
modificateur typeRetour méthode(listeArgs)
                                throws Exception{ … }
```

- **public**
- **protected**
- **private**
- **accès de paquetage**

- **void**
- **primitif (int, …)**
- **type d'objet (String, …)**
- **type d'un tableau**

Il existe d'autres modificateurs spéciaux

Objets : méthodes

Méthodes statiques

→ **Des méthodes n'ont pas besoin d'instance spécifique**

- `java.lang.System.exit(0);`
- `java.lang.Math.sqrt(4); //retourne 2.0`

```
public static double sqrt(double argument)
```

→ **Les méthodes statiques agissent sur des variables statiques**

Objets : méthodes

Méthodes statiques

Agissent sur des variables statiques

Compte.java

```
public class Compte {
  private double solde; //...
  private static int nbComptes;
  public static int getNbComptes(){ return nbComptes;  }

  public static void main(String args[]){
    this.solde = 1000.0;
    Compte obj = new Compte();
    obj.solde = 1000.0
    Compte.nbComptes ++ ;
    System.out.println(Compte.getNbComptes());
}}
```

Non static

```
Compte.solde = 1000.0;
```

Objets : méthodes

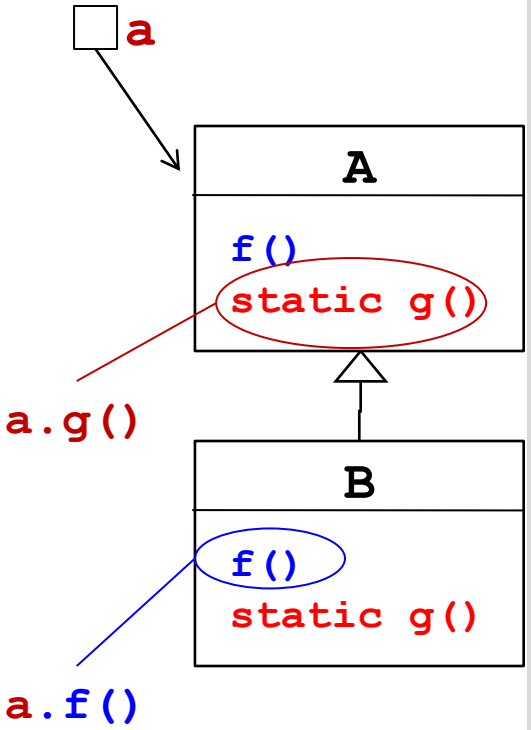
Méthodes statiques

la version utilisée est déterminée **par le compilateur** en fonction **du type de la référence**

Méthodes non statiques

la version utilisée est déterminée **dynamiquement** (à l'exécution par la JVM) en fonction **du type de l'objet référencé**

```
A a = new B();
```



`a` fait référence à une instance de `B`

Objets : méthodes

Exercice

```
class A {

    public static int g(){
        return (5);
    }
}
```

```
class B extends A {

    public static int g(){
        return (3);
    }
}
```

Qu'affiche le code suivant ?

```
A a = new B();

System.out.println(a.g());
```

(a) 5
(b) 3
(c) Erreur de compilation
(d) Erreur d'exécution

ENSIAS

Objets : méthodes

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Exercice

```
class A {
    public int f(){
        return (2);
    }
    public static int g(){
        return (5);
    }
}
```

```
class B extends A {
    public int f(){
        return (4);
    }
    public static int g(){
        return (3);
    }
}
```

```
B a1 = new B();
A a2 = new A();
A a3 = new B();
```

Qu'affiche chacun des codes suivant ?

```
System.out.println(a2.f()+" "+ a1.f());
    □ 2 3 □ 5 3 □ 2 4 □ 2 5
System.out.println(a2.g()+" "+ a3.g());
    □ 5 3 □ 5 5 □ 3 5 □ 3 3
System.out.println(a3.f()+" "+ a1.g());
    □ 2 5 □ 2 3 □ 4 5 □ 4 3
```

Objets : méthodes

Exercice

```
class A {
    public int f(){
        return (2);
    }
    public static int g(){
        return (5);
    }
}
```

```
class B extends A {
    public int f(){
        return (4);
    }
    public static int g(){
        return (3);
    }
}
```

```
A a4 = (A) new B();
```

```
System.out.println(a4.f()+" "+ a4.g());
```

□ 4 5 □ 2 3 □ 2 5 □ 4 3

Qu'affiche chacun des codes suivant ?

Objets : wrappers

type primitif ⇩ *Objets d'emballage*

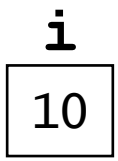
int ⇩ **java.lang.Integer**

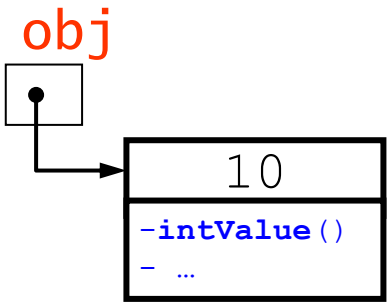
java.lang.Float

java.lang.Double

```
Integer obj = new Integer (10);

int i = obj.intValue();
```





Objets : wrappers

→ **Double rôle pour les wrappers**

1. Représentation objet des types primitifs
2. **Dépôt de constantes et de méthodes statiques utiles**

`Integer.MAX_VALUE`	Le plus grand entier
`Double.parseDouble("24.5")`	Renvoie la valeur `double` d'une chaîne
`Double.toString(24.5)`	Sens inverse
`Character.isLetterOrDigit('&')`	Renvoie false

→ **N'ont pas de constructeur sans argument**

```
new Integer(10)
```

Objets : construction alternative

➔ Méthodes statiques de fabrication

⇨ À la place des constructeurs

A P I	public final class Integer ...
static Integer valueOf(int i)	Returns an Integer instance representing the specified int value.
static Integer valueOf(String s)	Returns an Integer object holding the value of the specified String.

```
Integer obj1 = new Integer(10);
Integer obj2 = Integer.valueOf("80"); //static factory method
System.out.println("obj1:"+obj1+"  obj2:"+obj2);
//obj1:10  obj2:80
```

Objets : construction alternative

➔ Singleton

- Une classe instanciée **exactement une fois**
- **Exemple** : Besoin d'un objet pour interagir avec le **système d'exploitation** natif. Une seule instance de cet objet système est nécessaire

Objets : construction alternative

➔ Singleton : implémentation

- Garder le constructeur privé et exporter un membre **statique public** pour permettre l'accès à l'instance unique

```
class Single{//Singleton with static factory
        private static final Single INSTANCE = new Single();
        private Single(){System.out.println("Object Created");}
        public static Single getInstance(){
                return INSTANCE;
        }
}
Single c = Single.getInstance(); Single d = Single.getInstance();
System.out.println("c:"+c.hashCode());
System.out.println("d:"+d.hashCode());
```

```
Object Created
c:1311053135
d:1311053135
```

Classes utiles : java.lang.System

```
public final class java.lang.System                                    {
    // Fields
    public static PrintStream err;
    public static InputStream in; //clavier
    public static PrintStream out; //écran

    // Methods

    public static long currentTimeMillis();

    public static void exit(int status);

    public static void gc();

    public static Properties getProperties();

    public static void setProperties(Properties pps);

    public static SecurityManager getSecurityManager();

    public static void setSecurityManager(SecurityManager)
```

ENSIAS

Classes utiles : java.lang.System

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

```
import java.io.*;
import java.util.Properties;
public class VarEnvironnement {
    static public void main(String[] args){
        Properties properties =System.getProperties();
        System.out.println(properties);
    }}
```

```
-- listing properties --
java.home=/elfaker/java/SUNJWS/JDK/bin/..
...
```

ENSIAS

Classes utiles : java.lang.Scanner

→ **Travaille avec des tokens (chaines) séparés par un délimiteur**

```
public class InputDemo{

  public static void main(String[] args){

   Scanner kbd = new Scanner(System.in) ;

    while( kbd.hasNext()) //reste-t-il un token à lire ?

        System.out.println(scan.nextLine()) ;
```

- `scan.nextLine()`: la ligne entière (pas de `tokens` !)
- `scan.nextInt()`: "convertir" le prochain `token` en un `int`

ENSIAS

Classes utiles : java.util.Scanner

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

→ **S'interface avec des flux pour une lecture puissante**

`new Scanner(source)`	**Un "wrapper" pour la `source`**

`InputStream, FileInputStream, String, …`

```
import java.io.*;
InputStream fis;
fis = new FileInputStream("stuf.txt");
Scanner scan = new Scanner(fis) ;
```

A partir de 1.5

ENSIAS

Classes utiles : java.util.Scanner

Université Mohammed V de Rabat

→ **S'interface avec des flux pour une lecture puissante**

`new Scanner(source)`	**Un "wrapper" pour la `source`**

`InputStream`, `FileInputStream`, `String`, …

```
String chaine = "Azzouzi 29";
Scanner scan = new Scanner(chaine);
```

```
scan.next();   //nom (Azzouzi)
scan.nextInt(); //age (29)
```

ENSIAS

M.2.5.1. Paradigme Objet

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat