IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

# Les objets en Java

F. Dagnat, J. Mallet & T. Ledoux

Diaporama – UE-IR-S6 –
Session 1– 1er semestre 2026

# Plan


1 Java

2 Compiler et exécuter en Java

3 Instanciation d'objets

4 Manipulations des références

5 Un exemple complet

# Avancement

1 Java

2 Compiler et exécuter en Java

3 Instanciation d'objets

4 Manipulations des références

5 Un exemple complet

# Les principes

- Langage objet
- Typage fort
  - Les objets, variables sont déclarés avec un type
  - et leur utilisation est vérifiée par le compilateur
- Pas de mélange entre programmation objet et procédurale
  Un calcul est réalisé par un objet (ou une classe - *static*)
- Support pour la documentation : Javadoc
- De très nombreuses bibliothèques
  - Apprendre un langage objet, c'est apprendre sa syntaxe...
  - et connaître ses bibliothèques de code

# Définition d'une classe

```
class BankAccount {
  // Des attributs
  String firstName;
  String lastName;
  float balance;
  // Un constructeur
  BankAccount(String firstName,String lastName,float balance) {
    ...
  }
  // Des méthodes
  float getBalance() {
    ...
  }
  void deposit(float amount) {
    ...
  }
}
```

# Aperçu de la syntaxe Java

- déclaration de variable et affectation : `type variable = value;`
- déclarations de méthodes

```
<accessModifier> <returnType> <functionName>(<parameterType1> <parameterName1>, ...){
  // Corps de la méthode
}
```

⚠ **Document des bases de la syntaxe java** :
https://hub.imt-atlantique.fr/ueinfo-fise1a/fr/s6/prog/ressources/

# Encapsulation

- **Rappel S5** : L'encapsulation assure que l'état d'un objet (attributs) n'est accessible que depuis l'objet lui-même (ses méthodes)
- En Java, chaque attribut ou méthode peut contenir un modifieur de visibilité :

| Java | accessible par |
|---|---|
| **`private`** | les objets de même classe |
| **`public`** | tous (pas d'encapsulation) |

- Il faut des règles de bonne pratique :
  - Utiliser au maximum des attributs **`private`**
  - Ne mettre **`public`** que les méthodes qui doivent être accédées de l'extérieur

# Attribut et méthode de classe

Quand un calcul ne dépend pas d'un objet (fonction mathématique ou constante, par exemple.)

- Modifieur **`static`**
- Par ex : `Math.PI`
- Par ex : `System.exit(0);`, `Math.random(); ...`
- Par ex :

```
public class BankAccount {
  // Des attributs
  ...
  private static float minBalance;
  // Des méthodes
  public static void setMinBalance(float min) {
    BankAccount.minBalance = min;
  }
}
```

# Avancement


1 Java

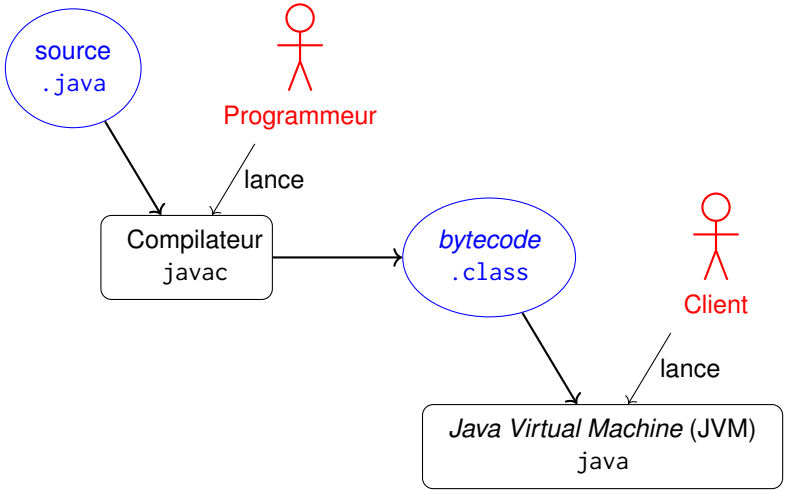
2 Compiler et exécuter en Java

3 Instanciation d'objets

4 Manipulations des références

5 Un exemple complet

# Schéma général



# Exécution

- Une classe publique peut être exécutée
- La JVM lance alors l'exécution de sa méthode `main`
- Cette méthode doit suivre la signature suivante

```
public static void main(String[] args){
  ...
}
```

Seul le nom de l'argument peut être modifié

# Avancement


1 Java

2 Compiler et exécuter en Java

3 Instanciation d'objets

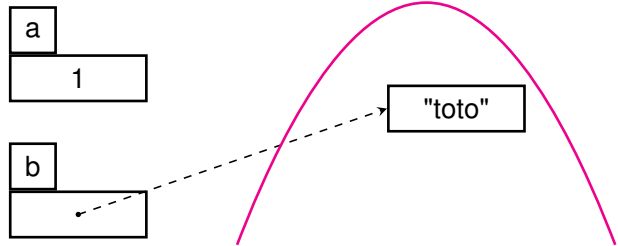
4 Manipulations des références

5 Un exemple complet

# Manipulation des objets

- Les objets ne sont jamais manipulés directement :
  - ils sont créés et stockés dans une zone mémoire spécifique (le tas)
  - on y accède par des références (des adresses)
- Les types primitifs (**`int`**, **`boolean`**, ...) sont manipulés directement.
- Les tableaux sont manipulés également par référence

```
int a = 1;
String b = "toto";
```



# La création

- Des méthodes spécifiques : les constructeurs
- Même nom que la classe et pas de type de retour :

```
public NomClasse(Type1 nomVar1,Type2 nomVar2) {
  ...
}
```

- Appelés uniquement à la création d'une nouvelle instance à l'aide de l'opérateur **new**
- But : initialiser les attributs du nouvel objet

# L'instanciation

```java
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```

```java
public class BankAccount {
  private String firstName;
  private String lastName;
  private float balance;

  public BankAccount(String f,String l,float bal) { ... }
  public BankAccount(String f,String l) { ... }

  ...
}
```

# L'instanciation

```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```

```
public class BankAccount {
  private String firstName;
  private String lastName;
  private float balance;

  public BankAccount(String f,String l,float bal) { ... }
  public BankAccount(String f,String l) { ... }

  ...
}
```

# L'instanciation

```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```

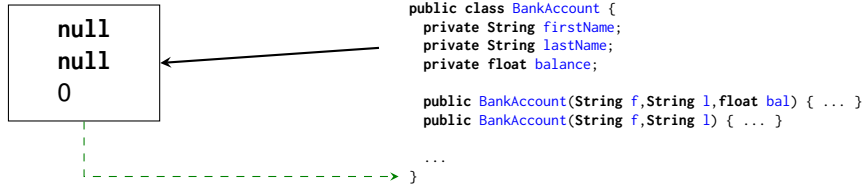
```
public class BankAccount {
  private String firstName;
  private String lastName;
  private float balance;

  public BankAccount(String f,String l,float bal) { ... }
  public BankAccount(String f,String l) { ... }

  ...
}
```

L'objet conserve un lien sur sa classe

1 création de la structure en mémoire (tas), allocation

# L'instanciation

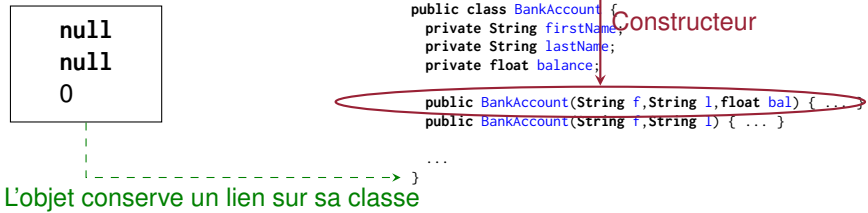
```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```




1. création de la structure en mémoire (tas), allocation
2. initialisation des attributs

# L'instanciation

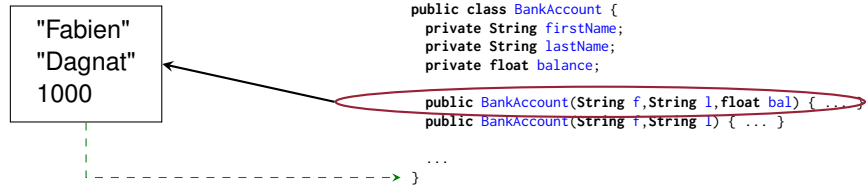
```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```




1. création de la structure en mémoire (tas), allocation
2. initialisation des attributs
3. exécution du constructeur spécifié

# L'instanciation

```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```




1. création de la structure en mémoire (tas), allocation
2. initialisation des attributs
3. exécution du constructeur spécifié

# L'instanciation

```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```


```
public class BankAccount {
  private String firstName;
  private String lastName;
  private float balance;

  public BankAccount(String f,String l,float bal) { ... }
  public BankAccount(String f,String l) { ... }

  ...
}
```

L'objet conserve un lien sur sa classe

1. création de la structure en mémoire (tas), allocation
2. initialisation des attributs
3. exécution du constructeur spécifié
4. renvoi d'une référence sur l'objet

# L'instanciation

```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```


```
public class BankAccount {
  private String firstName;
  private String lastName;
  private float balance;

  public BankAccount(String f,String l,float bal) { ... }
  public BankAccount(String f,String l) { ... }

  ...
}
```

L'objet conserve un lien sur sa classe

1. création de la structure en mémoire (tas), allocation
2. initialisation des attributs
3. exécution du constructeur spécifié
4. renvoi d'une référence sur l'objet

# Avancement


1 Java

2 Compiler et exécuter en Java

3 Instanciation d'objets

4 Manipulations des références

5 Un exemple complet

# Un objet particulier

- La référence sur rien **`null`** qui est de tous les types références
- Utiliser une référence nulle (valant **`null`**) lève l'exception `NullPointerException` :
- ⇒ Penser à tester une référence reçue en paramètre avant de s'en servir.
  Précondition `arg != null` :

```
public void transfer(float amount, BankAccount target) {
  if (target != null && amount < balance) {
    balance -= amount;
    target.deposit(amount);
  }
}
```

# Comparaison de références

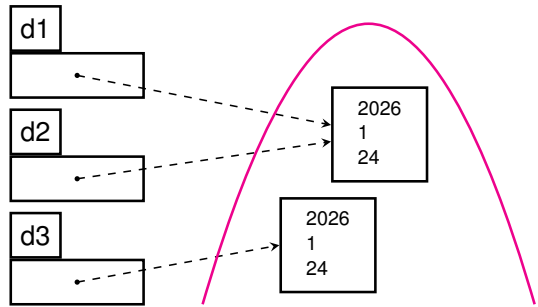
- La comparaison == compare les valeurs
  - sur les types primitifs compare bien la valeur
  - sur les types références compare les adresses pas les contenus
- Méthode `equals` (voir plus tard)

```
Date d1 = new Date(2026,1,24);
Date d2 = d1;
Date d3 = new Date(2026,1,24);

d1 == d2;
d1 == d3;
d1.equals(d2);
d1.equals(d3);
```

# Comparaison de références

- La comparaison == compare les valeurs
  - sur les types primitifs compare bien la valeur
  - sur les types références compare les adresses pas les contenus
- Méthode `equals` (voir plus tard)

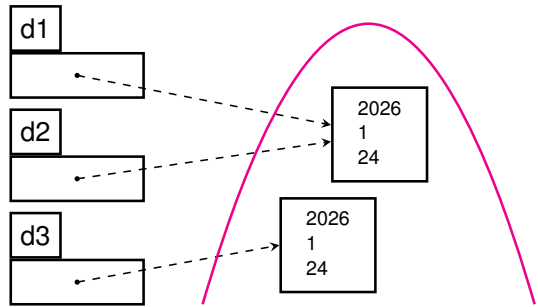
```
Date d1 = new Date(2026,1,24);
Date d2 = d1;
Date d3 = new Date(2026,1,24);

d1 == d2;
d1 == d3;
d1.equals(d2);
d1.equals(d3);
```



# Comparaison de références

- La comparaison == compare les valeurs
  - sur les types primitifs compare bien la valeur
  - sur les types références compare les adresses pas les contenus
- Méthode `equals` (voir plus tard)

```
Date d1 = new Date(2026,1,24);
Date d2 = d1;
Date d3 = new Date(2026,1,24);

d1 == d2;           true
d1 == d3;           false
d1.equals(d2);      true
d1.equals(d3);      true
```



# Avancement


1 Java

2 Compiler et exécuter en Java

3 Instanciation d'objets

4 Manipulations des références

5 Un exemple complet

# Classe BankAccount

```java
public class BankAccount{
    private String firstName;
    private String lastName;
    private float balance;
    private static float minBalance = 0;
    public BankAccount(String first,String last,float balance){
        this.firstName = first;
        this.lastName = last;
        this.balance = balance;
    }
    public float getBalance(){
        return balance;
    }
    public String getOwner(){
        return firstName + " " + lastName;
    }
    public void setOwner(String first,String last){
        this.firstName = first;
        this.lastName = last;
    }
    public static void setMinBalance(float min) {
        BankAccount.minBalance = min;
    }
    public void deposit(float amount){
        if (amount >= 0) this.balance += amount;
    }
    public void withdraw(float amount){
        float newBalance = this.balance - amount;
        if (amount >= 0 && newBalance > BankAccount.minBalance){
            this.balance = newBalance;
        }
    }
    public static void main(String[] a){
        BankAccount b = new BankAccount("thomas","ledoux",100);
        System.out.println(b.getOwner()+" with "+b.getBalance());
        b.deposit(100);
        b.withdraw(150);
    }
}
```