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

Les objets en Java

F. Dagnat, J. Mallet & T. Ledoux

Diaporama – UE-IR-S6 – Session 1

1er semestre 2025

Plan

- 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
  - Appendre un langage objet, c'est apprendre sa syntaxe. . .
  - et connaître ses bibliothèques de code

```
1  class BankAccount {
2    // Des attributs
3    String firstName;
4    String lastName;
5    float balance;
6    // Un constructeur
7    BankAccount(String firstName,String lastName,float balance) {
8      ...
9    }
10   // Des méthodes
11   float getBalance() {
12     ...
13   }
14   void deposit(float amount) {
15     ...
16   }
17 }
```

- déclaration de variable et assignation : `type variable = value;`
- déclarations de fonctions

<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/`

- **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 :

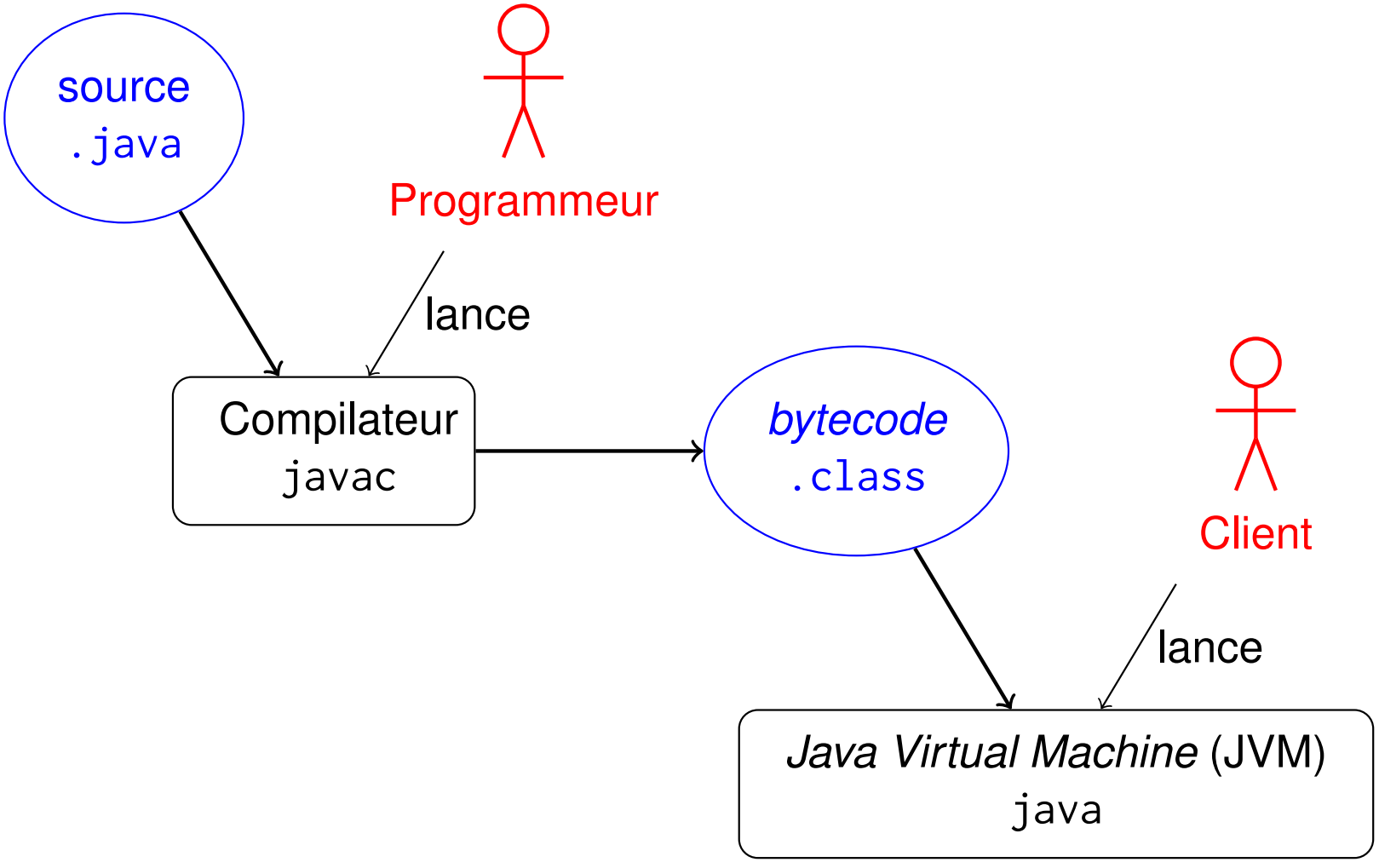
Java	accessible par
`private`	les objets de même classe
`public`	tous (pas d'encapsulation)

- 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

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 :

```
1 public class BankAccount {
2   // Des attributs
3   ...
4   private static float minBalance;
5   // Des méthodes
6   public static void setMinBalance(float min) {
7     BankAccount.minBalance = min;
8   }
9 }
```


source
.java
Programmeur
lance
Compilateur
javac
bytecode
.class
Client
lance
Java Virtual Machine (JVM)
java

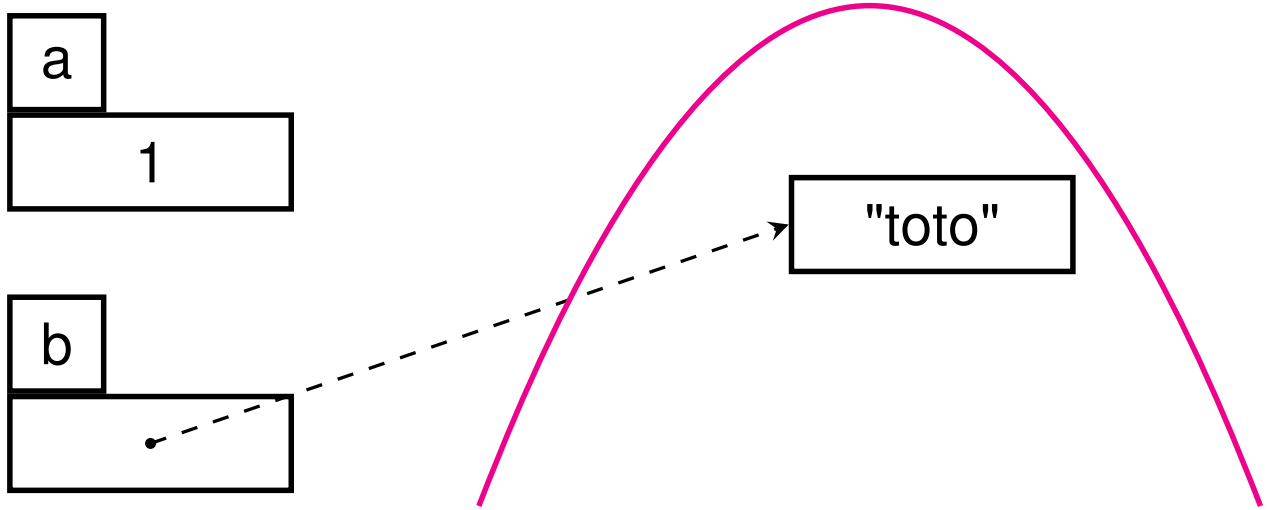
- Une classe publique peut être exécutée
- La JVM lance alors l'exécution de sa méthode `main`
- Cette méthode doit suivre la signature suivante

```
1       public static void main(String[] args){
2         ...
3       }
```

Seul le nom de l'argument peut être modifié

- 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

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



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

```
1       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

```
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) { ... }

  ...
}
```

```
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) { ... }

  ...
}
```

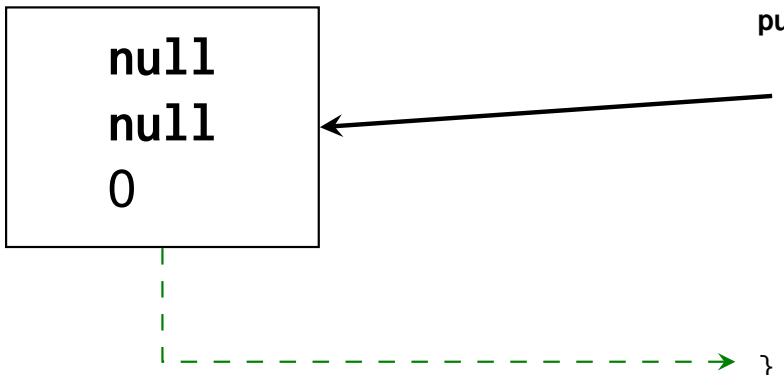
```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```

```
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

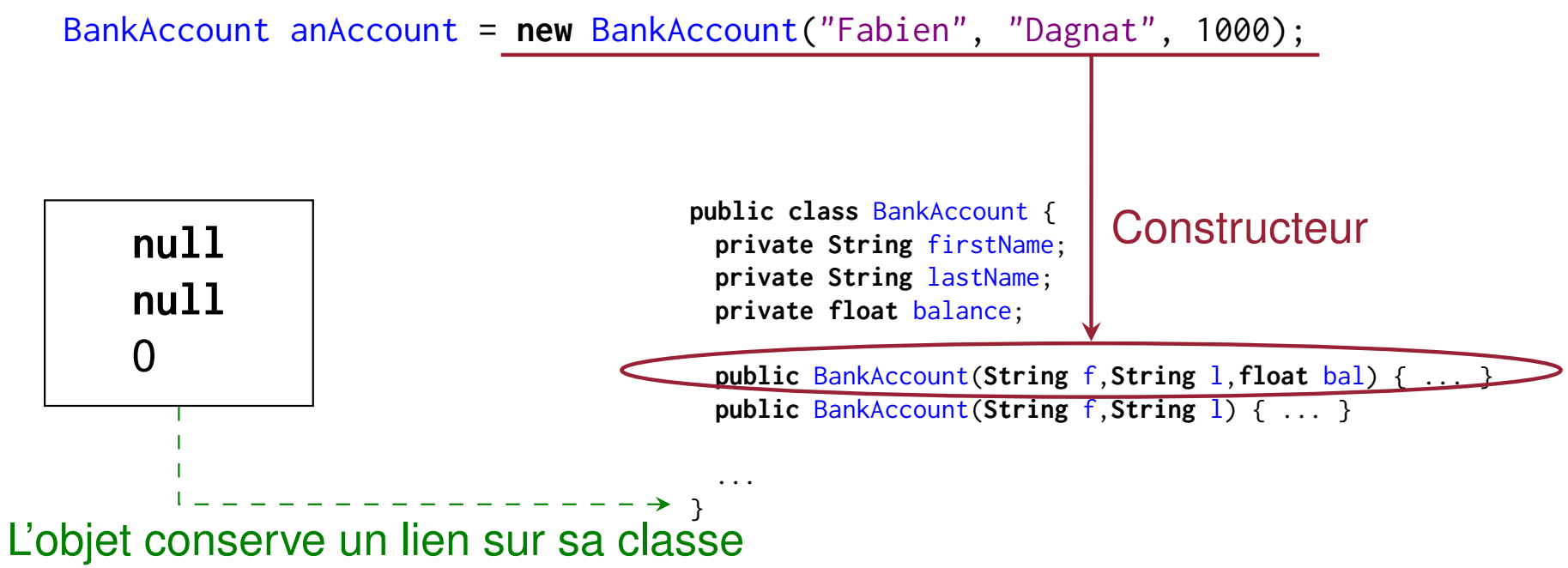
```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```




```
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) { ... }

  ...
}
```

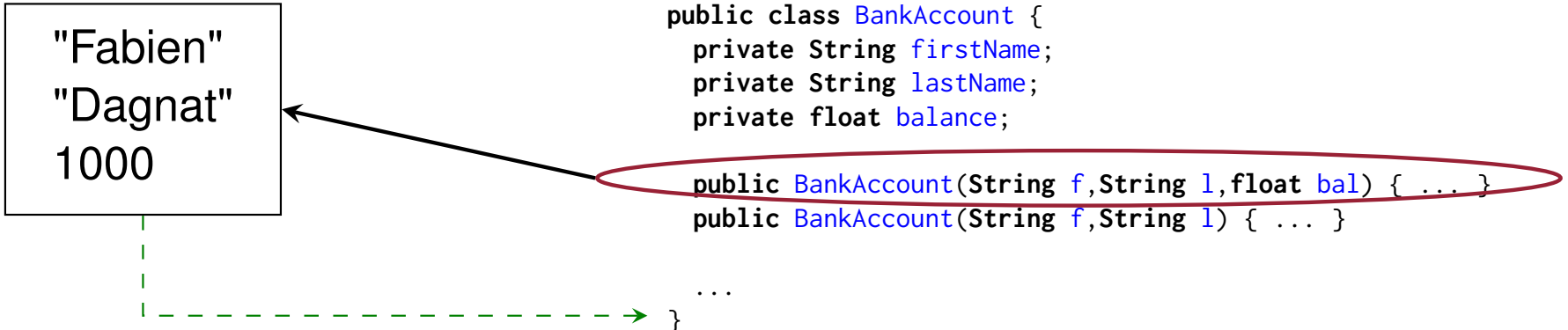
L'objet conserve un lien sur sa classe

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



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

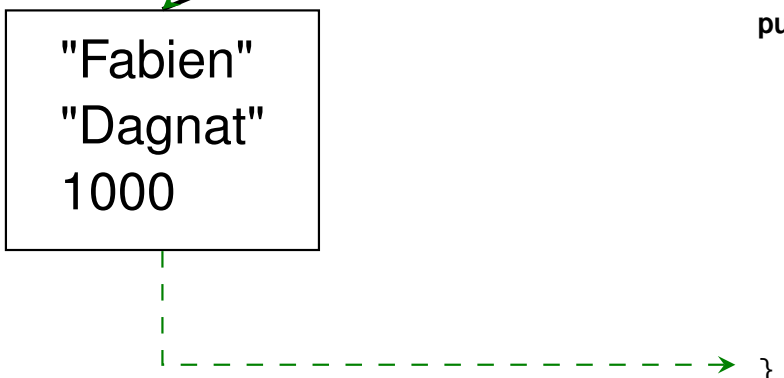
```
BankAccount anAccount = new BankAccount("Fabien", "Dagnat", 1000);
```



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

```
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

```
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

- 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` :

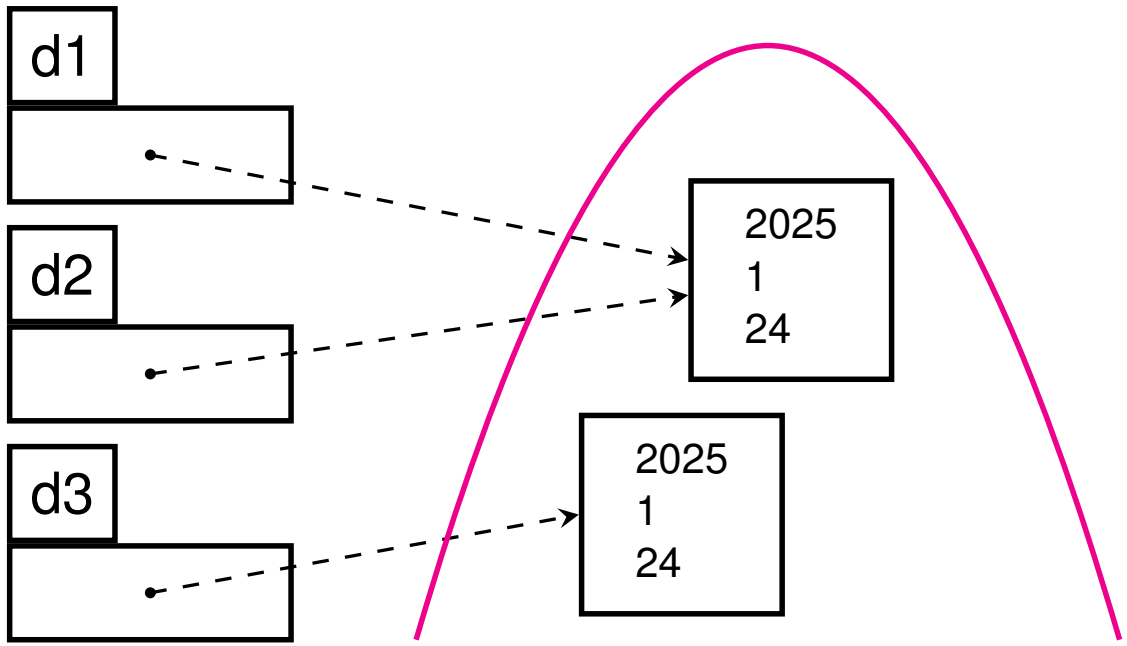
```
1 public void transfer(float amount, BankAccount target) {
2   if (target != null && amount < balance) {
3     balance -= amount;
4     target.deposit(amount);
5   }
6 }
```

- 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)

```
1 Date d1 = new Date(2025,1,24);
2 Date d2 = d1;
3 Date d3 = new Date(2025,1,24);
4
5 d1 == d2;
6 d1 == d3;
7 d1.equals(d2);
8 d1.equals(d3);
```

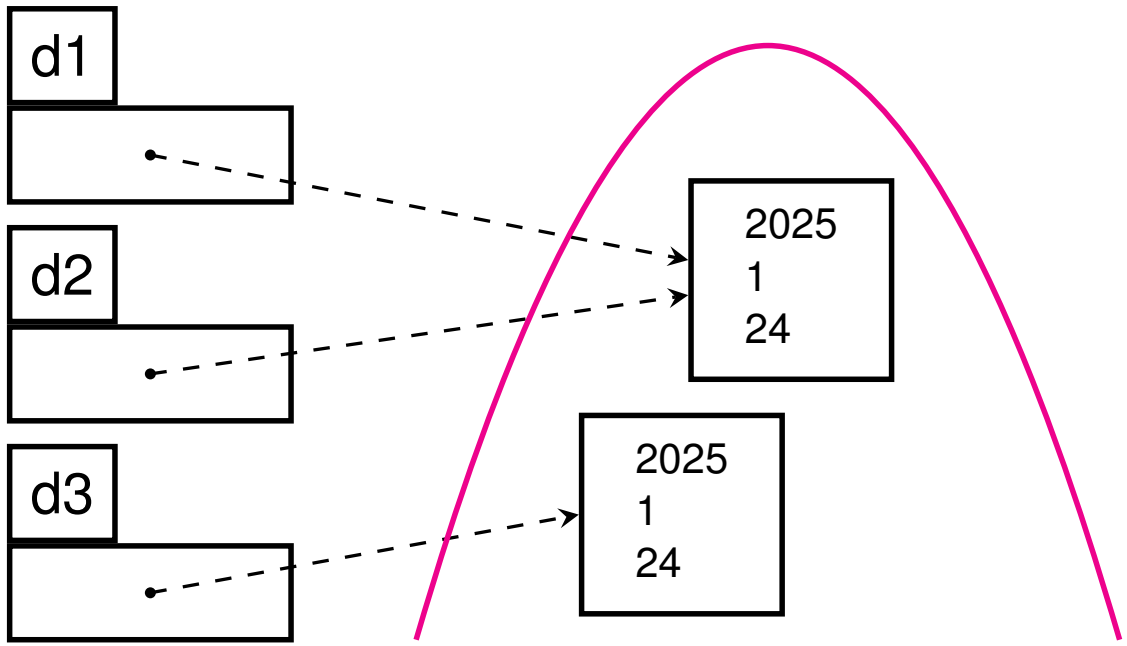
- La comparaison == compare les valeurs
  - sur les types primitifs compare bien la valeur
  - sur les types références compare les adresses pas les contenus
- Méthode `equals` (voir plus tard)

```
1 Date d1 = new Date(2025,1,24);
2 Date d2 = d1;
3 Date d3 = new Date(2025,1,24);
4
5 d1 == d2;
6 d1 == d3;
7 d1.equals(d2);
8 d1.equals(d3);
```



- 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)

```
1 Date d1 = new Date(2025,1,24);
2 Date d2 = d1;
3 Date d3 = new Date(2025,1,24);
4
5 d1 == d2;           true
6 d1 == d3;           false
7 d1.equals(d2);      true
8 d1.equals(d3);      true
```



```java
public class BankAccount{
    private String firstName;
    private String lastName;
    private float balance;
    private static float minBalance = 0;
    public BankAccount(String firstName,String lastName,float balance){
        this.firstName = firstName;
        this.lastName = lastName;
        this.balance = balance;
    }
    public float getBalance(){
        return balance;
    }
    public String getOwner(){
        return firstName + " " + lastName;
    }
    public void setOwner(String firstName,String lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }
    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){
        if (amount >= 0 && this.balance - amount > BankAccount.minBalance) this.balance -= amount;
    }
    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);
    }
}
```