# ITI 1521. Introduction à l'informatique II

**Programmation orientée objet** : visibilité, variables et méthodes de classe

by

Marcel Turcotte

Version du 20 janvier 2020

# Préambule

# Préambule

## Aperçu

**Programmation orientée objet : visibilité, variables et méthodes de classe**

Nous découvrons le rôle des modificateurs de visibilité afin de favoriser l'encapsulation. Nous ajoutons à nos connaissances les concepts de variables et méthodes de classe. Finalement, nous verrons le rôle d'une variable référence prédéfinie, **this**.

**Objectif général :**

- Cette semaine, vous serez en mesure de comparer les concepts de variable d'instance et variables de classes, ainsi que de méthodes d'instance et de méthodes de classe.

**Une vidéo d'introduction :**

- `https://www.youtube.com/watch?v=OFFzkropD1A`

# Préambule

## Objectifs d'apprentissage

# Objectifs d'apprentissage

- **Décriver** les mécanismes de Java qui favorisent l'encapsulation.
- **Expliquer** dans vos propres mots les concepts suivants : variable d'instance et variable de classe, méthode d'instance et méthode de classe.
- **Concevoir** un programme Java simple illustrant les concepts de base de la programmation orientée objet.

**Lectures :**

- Pages 573-579 de E. Koffman et P. Wolfgang.

# Lectures (suite) :

**Entrée en matière**

- docs.oracle.com/javase/tutorial/java/concepts/object.html
- docs.oracle.com/javase/tutorial/java/concepts/class.html

**Informations détaillées**

- docs.oracle.com/javase/tutorial/java/javaOO/classes.html
- docs.oracle.com/javase/tutorial/java/javaOO/objects.html
- docs.oracle.com/javase/tutorial/java/javaOO/more.html
- docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

**Exercises**

- docs.oracle.com/javase/tutorial/java/concepts/QandE/questions.html
- docs.oracle.com/javase/tutorial/java/javaOO/QandE/creating-questions.html

# Préambule

## Plan du module

# Plan

1 Préambule

2 Encapsulation

3 Contexte de la classe

4 « What is *this* ? »

5 Final

6 Prologue

# Encapsulation

# Définition : encapsulation

En programmation orientée objet, l'**encapsulation** consiste à regrouper en une même unité (l'objet) les **données** et les **méthodes** qui les manipulent.

# Java : modificateurs de visibilité

En Java, les **modificateurs de visibilité** nous permettent de contrôler l'accès des variables et des méthodes.

```
public class Point {

    private int x, y;

    public int getX() {
        return x;
    }

    public void setX(int value) {
        x = value;
    }

}
```

# Java : modificateurs de visibilité

```
public class Point {

    private int x, y;

    public int getX() { return x; }

    public int getY() { return y; }

    public boolean equals(Point other) {
      if (other == null) {
          return false;
      } else {
          return x == other.getX() && y == other.getY();
      }
    }

}
```

# Java : modificateurs de visibilité

- Est-ce que la déclaration de la méthode **equals** est **valide** ?

```
public class Point {

    private int x, y;

    public int getX() { return x; }

    public int getY() { return y; }

    public boolean equals(Point other) {
      if (other == null) {
          return false;
      } else {
          return x == other.x && y == other.y;
      }
    }
}
```

# Exemple : encapsulation

```
public class Test {
    public static void main(String[] args) {
        Point p;
        p = new Point();
        p.x = 4;
    }
}
```

```
javac Test.java
Test.java:5: error: x has private access in Point
        p.x = 4;
          ^
1 error
```

# Java : modificateurs de visibilité

En Java, les **modificateurs de visibilité** nous permettent de contrôler l'accès des variables et des méthodes.

- Une **variable** ou une **méthode** dont la visibilité est **private** n'est accessible que dans le corps de la classe où elle est définie.
- Une **variable** ou une **méthode** dont la visibilité est **public** est accessible dans le corps de la classe où elle est définie, mais aussi dans toutes les autres classes du programme.

En **ITI1521**, les **variables d'instance** devraient **toujours** être déclarées de visibilité **private** !

# Discussion : encapsulation

- **Analysez** attentivement les deux implémentations de la classe **Pair** qui suivent.
  - Peut-on remplacer une implémentation par l'autre **sans causer d'erreurs de compilation ou d'exécution pour les autres classes d'une application ?**

```java
public class Pair {

    private int first;
    private int second;

    public Pair(int firstInit, int secondInit) {
        first = firstInit;
        second = secondInit;
    }
    public int getFirst() {
        return first;
    }
    public int getSecond() {
        return second;
    }
    public void setFirst(int value) {
        first = value;
    }
    public void setSecond(int value) {
        second = value;
    }
}
```

```java
public class Pair {

    private int[] elems;

    public Pair(int first, int second) {
        elems = new int[2];
        elems[0] = first;
        elems[1] = second;
    }
    public int getFirst() {
        return elems[0];
    }
    public int getSecond() {
        return elems[1];
    }
    public void setFirst(int value) {
        elems[0] = value;
    }
    public void setSecond(int value) {
        elems[1]= value;
    }
}
```

# Discussion : encapsulation

- Peut-on remplacer une implémentation par l'autre **sans causer d'erreurs de compilation ou d'exécution pour les énoncés suivants ?**

```
Pair p;
p = new Pair(2, 4);

System.out.println(p.getFirst());

p.setSecond(12);
```

# Discussion : encapsulation (prise 2)

```
public class Point {

    private int x
    private int y;

    public void setX(int value) {
        if (value < 0 || value > 1024) {
            x = 0;
        } else {
            x = value;
        }
    }
}
```

# Discussion : méthode private

- Pouvez-vous imaginer une situation où l'on souhaiterait déclarer **une méthode «private»** ?

# Discussion : méthode private

```
public class Point {

    private int x
    private int y;

    private boolean isValid(int value) {
        if (value < 0 || value > 1024) {
            return true;
        } else {
            return false;
        }
    }
    public void setX(int value) {
        if (isValid(value)) {
            x = value;
        } else {
            x = 0;
        }
    }
}
```

# Discussion : interface

- L'«**interface**» d'une classe est constituée des méthodes publiques
- Les méthodes qui ne devraient pas faire partie de l'interface seront déclarées «**private**»

# Contexte de la classe

# Définition : variable de classe

Une **variable de classe** est une variable définie dans le corps de la classe et qui est **partagée par les instances** (objets) de cette classe.

```
public class Point {
    public static int MAX_VALUE = 100;
    private int x
    private int y;

    public void moveRight() {
        x = x + 1;
        if (x > MAX_VALUE) {
            x = 0;
        }
    }
}
```

# Définition : variable de classe

- Le mot clé **static** introduit une variable de classe. Comme vous le voyez, il faut **faire un effort** afin de créer une variable de classe, ce n'est pas ce que l'on obtient par défaut.

```
public class Point {
    public static int MAX_VALUE = 100;
    private int x
    private int y;

    public void moveRight() {
        x = x + 1;
        if (x > MAX_VALUE) {
            x = 0;
        }
    }
}
```

# Variable de classe

- La déclaration d'une **constant** est bon exemple d'une situation où les **variables de classes** sont utiles.
- Pouvez-vous trouver d'**autres exemples** où les variables de classes pourraient être utiles ?

```
public class Ticket {

    private static int last = 100;
    private int number;

    public Ticket() {
        number = last;
        last=last+1;
    }

    public int getSerialNumber() {
        return number;
    }
}
```

```java
public class Ticket {

  private static int last = 100;
  private int number;

  public Ticket() {
    number = last;
    last=last+1;
  }

  public int getSerialNumber() {
    return number;
  }
}
```

# Définition : méthode de classe

Les **méthodes de classe n'appartiennent pas à un objet** en particulier. Elles sont donc **partagées** par les instances de la classe. Puisqu'elle ne sont pas associées à un objet, elles **n'ont pas accès aux variables d'instance**.

# Méthode de classe

- Est-ce que la méthode **isValid** utilise une variable d'instance ?

```
public class Point {

    private int x
    private int y;

    private boolean isValid(int value) {
        if (value < 0 || value > 1024) {
            return true;
        } else {
            return false;
        }
    }
}
```

# Méthode de classe

- La méthode **isValid** devrait donc être une méthode de classe (static)

```
public class Point {

    private int x
    private int y;

    private static boolean isValid(int value) {
        if (value < 0 || value > 1024) {
            return true;
        } else {
            return false;
        }
    }
}
```

# Méthode de classe

```
public class Point {

    private int x
    private int y;

    public static int compareTo(int a, int b) {
        int result;
        if (a < b) {
            result = -1;
        } else if (a == b) {
            result = 0;
        } else {
            result = 1;
        }
          return result;
    }
}
```

# Exemples de méthodes de classe

- `http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html`
- `http://docs.oracle.com/javase/8/docs/api/java/lang/System.html`

# Appel à une méthode de classe («static context»)

- On utilise le **nom de la classe** suivi du **nom de la méthode** pour faire un appel à une **méthode de classe**.
- On utilise le **nom de la classe** suivi du **nom de la variable** pour accéder à une **variable de classe**.

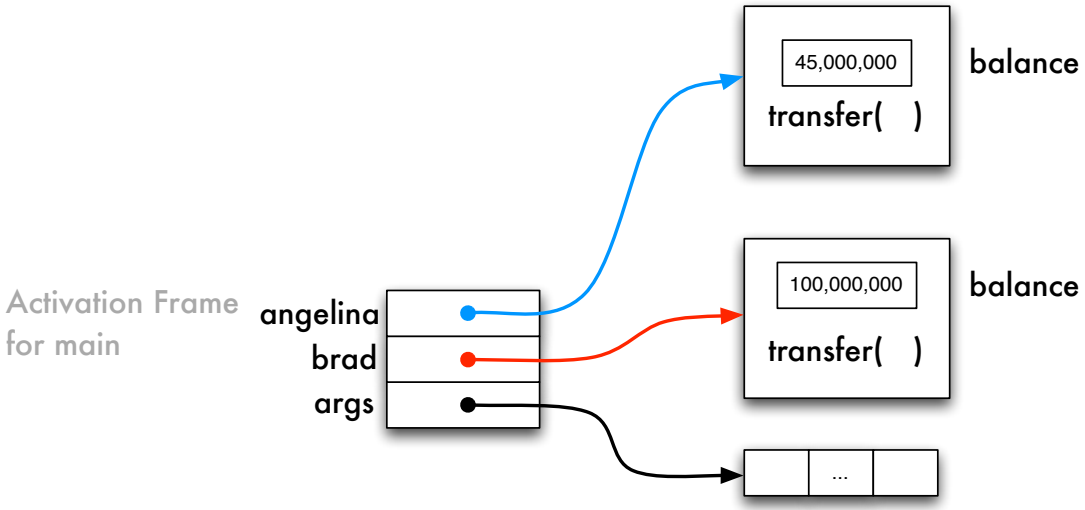
```
double a;

a = Math.abs(-1.6);

a = Math.max(1024.0, Double.MAX_VALUE);

a = Math.random();

a = Math.sqrt(Double.MAX_VALUE);

a = Math.pow(2.0, 32.0);
```

# « What is *this* ? »

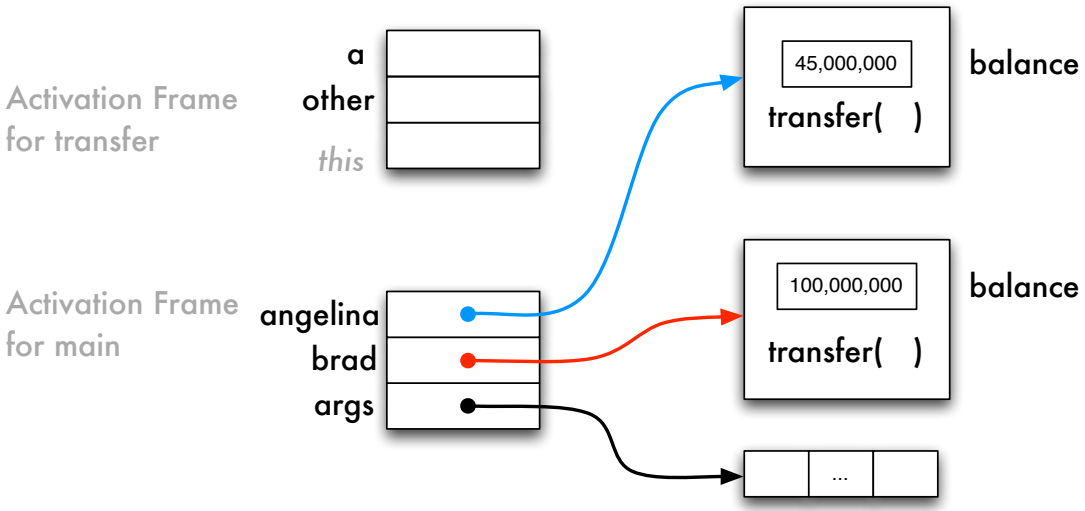
# « What is this ? »

**Chaque objet** possède une référence **this**. Elle désigne l'**objet** lui-même.

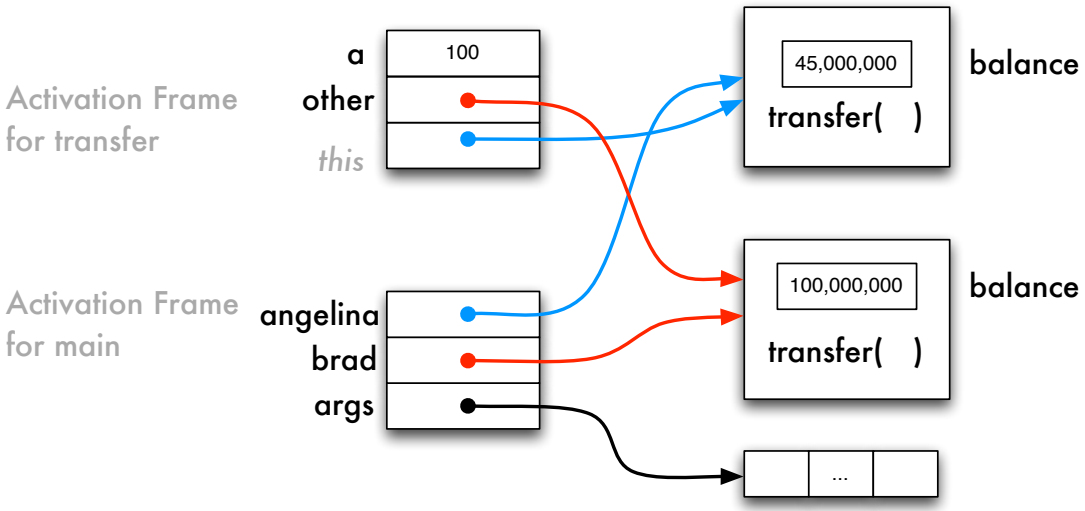
```
public class BankAccount {

  private double balance;

  // ...

  public boolean transfer(BankAccount other, double a) {

    if (this == other) {
        return false;
    }

    ...
  }
}
```



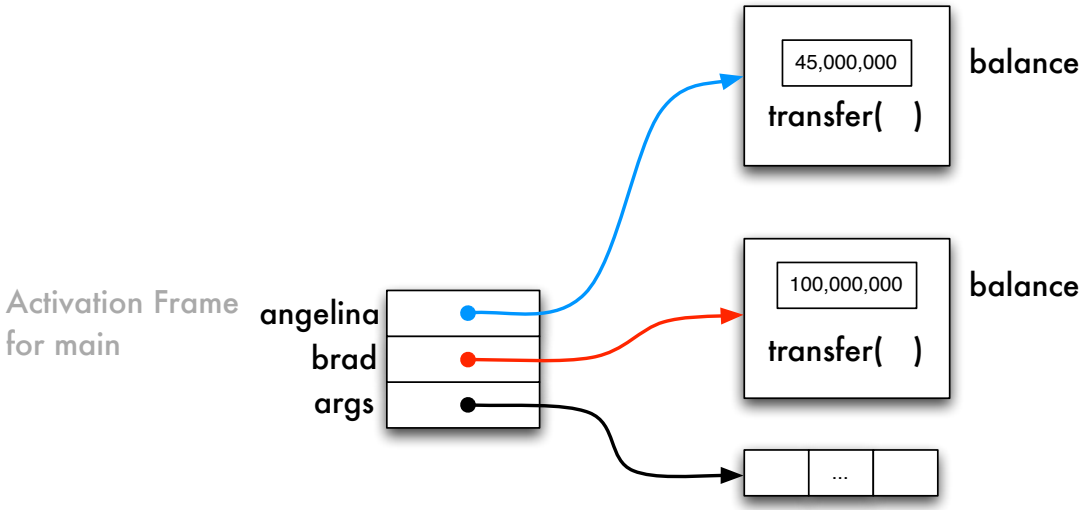

angelina.transfer( brad, 100 )



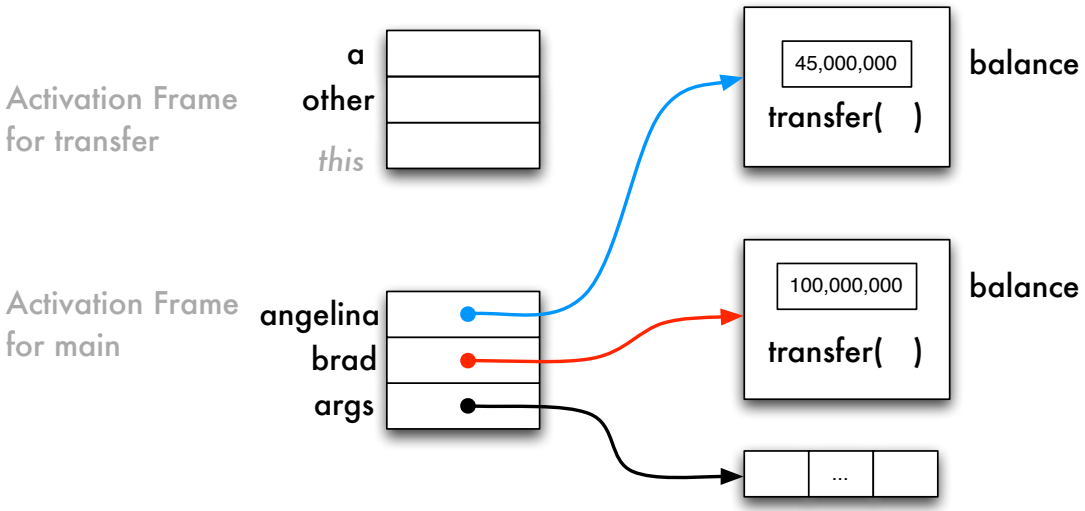

angelina.transfer( brad, 100 )



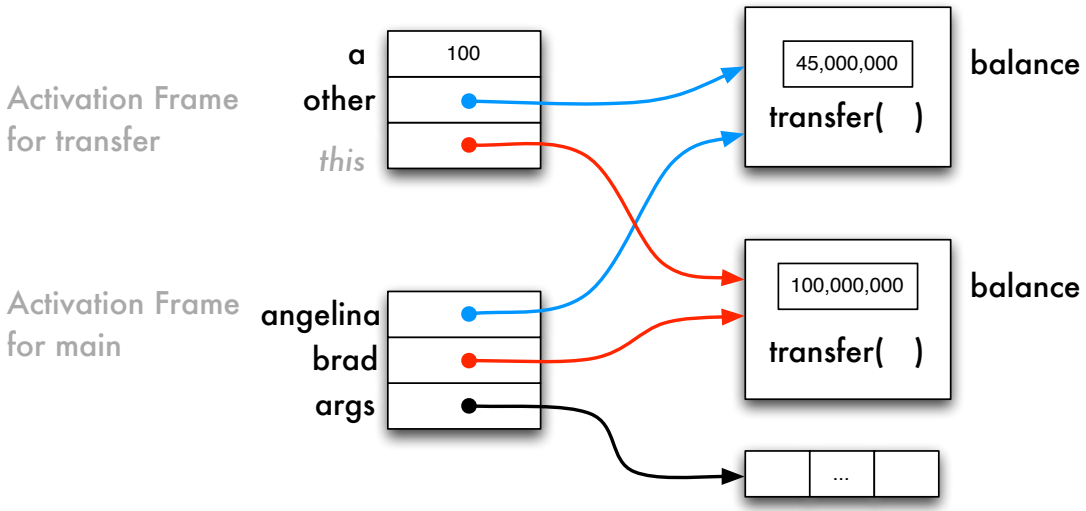

angelina.transfer( brad, 100 )



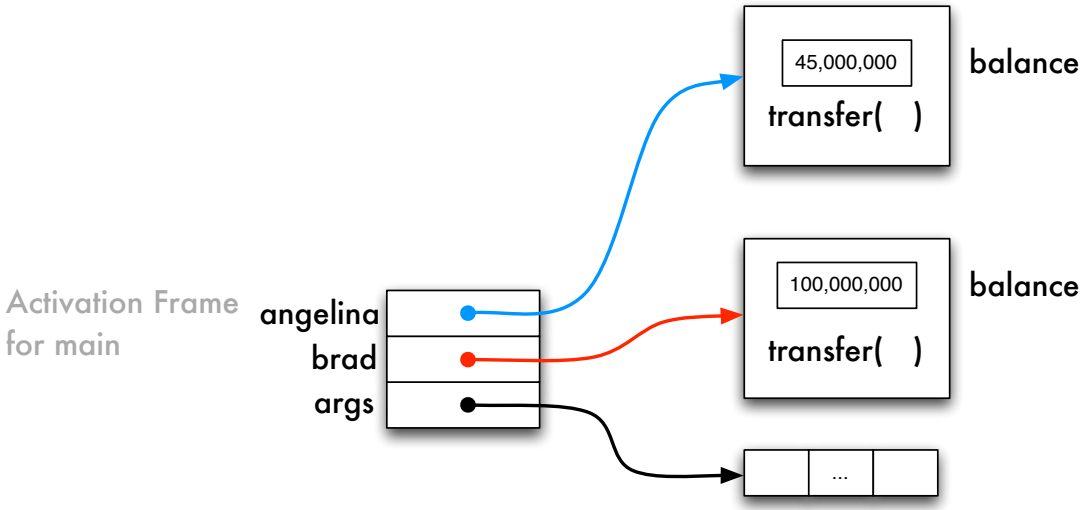

brad.transfer( angelina, 100 )



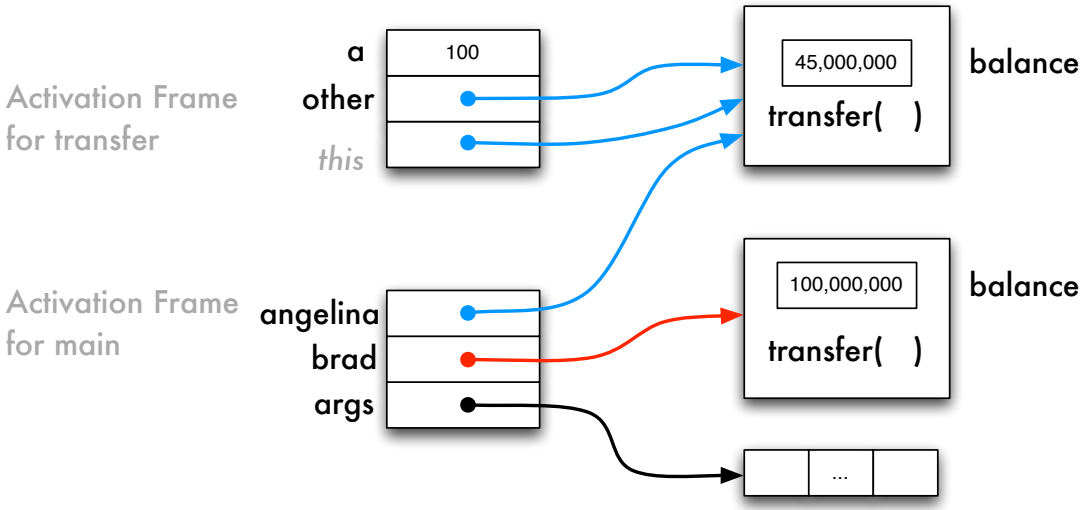

brad.transfer( angelina, 100 )




brad.transfer( angelina, 100 )




angelina.transfer( angelina, 100 )

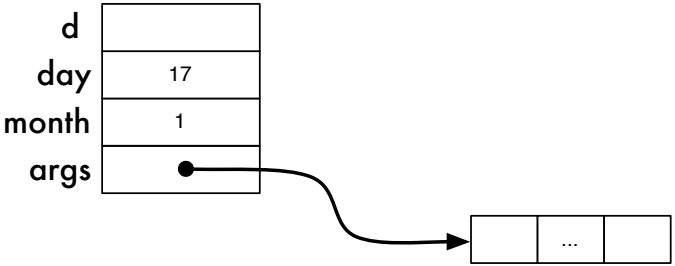



angelina.transfer( angelina, 100 )

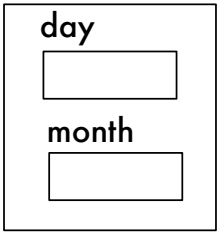
# « What is this »

```
public class Date {
  private int day;
  private int month;
  public Date(int day, int month) {
     this.day = day;
     this.month = month;
  }
  // ...
  public static void main(String[] args) {
     Date d;
     int day, month;
     day = 17; month = 1;
     d = new Date(day, month);
  }
}
```

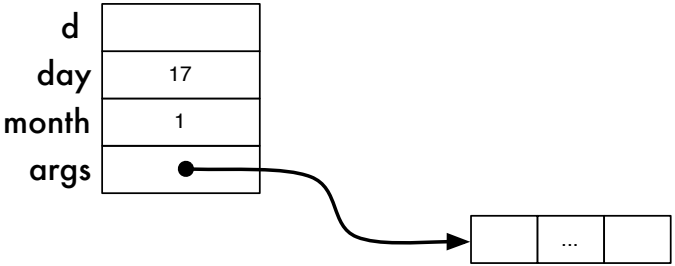
Activation Frame
(working memory)
for main




⇒ Mémoire de travail de la méthode principale (**main**)

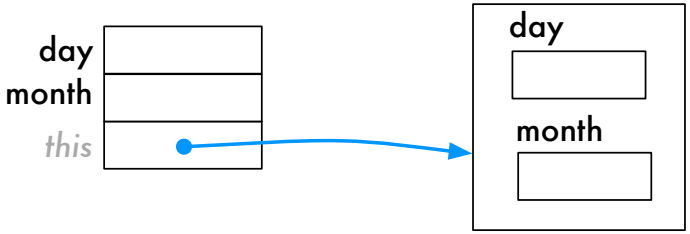



Activation Frame
(working memory)
for main

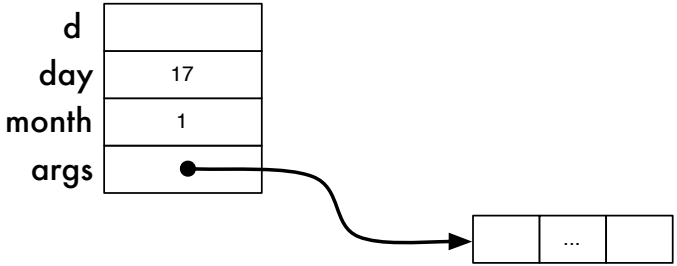



⇒ On exécute “new Date(day,month)”, création d’un objet

Activation Frame
(working memory)
for transfer

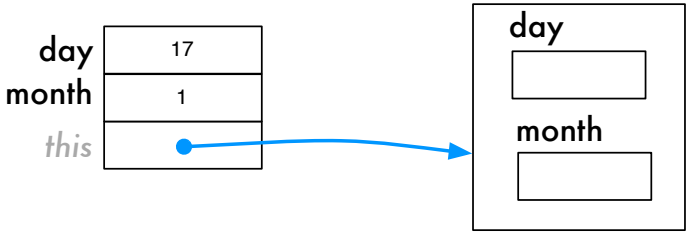



Activation Frame
(working memory)
for main

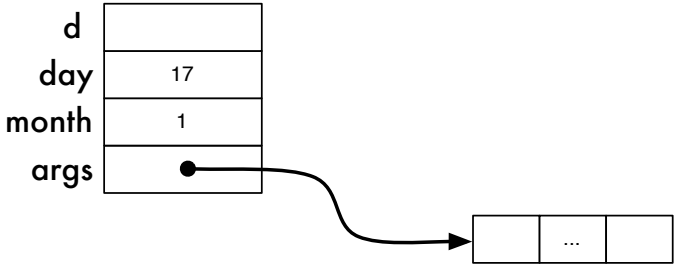



⇒ Appel au constructeur, création de la mémoire de travail associée

Activation Frame
(working memory)
for transfer

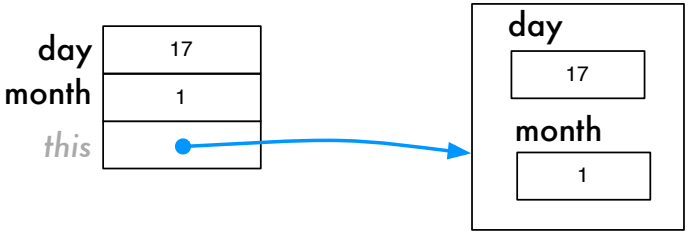



Activation Frame
(working memory)
for main

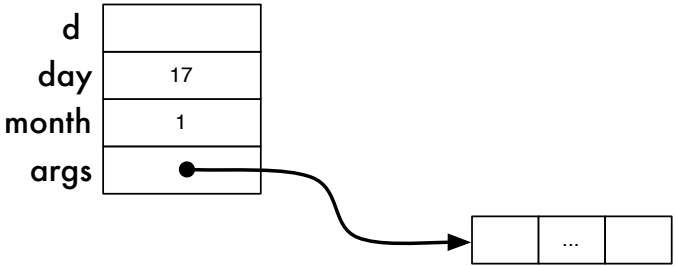



⇒ Copie les valeurs de **paramètres actuels** dans les **paramètres formels**

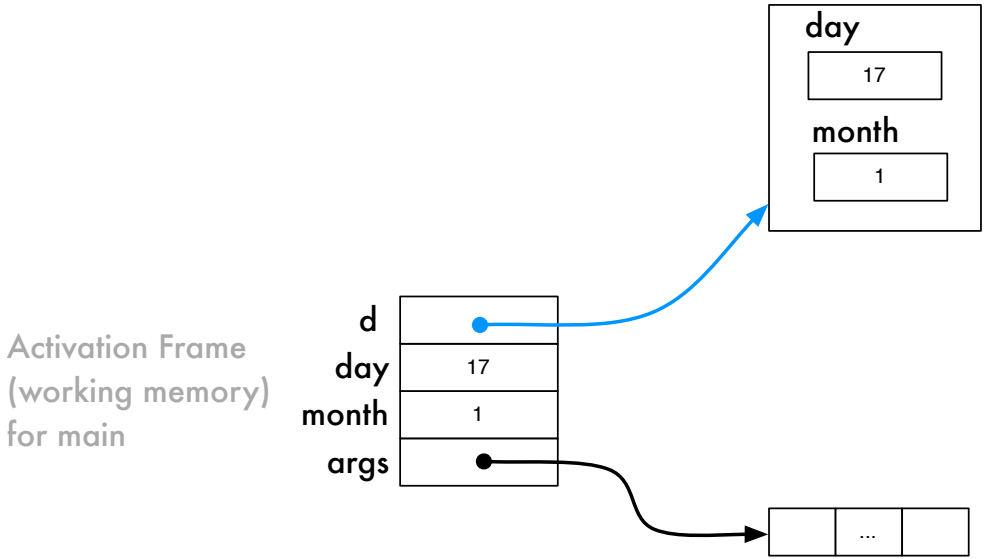
Activation Frame
(working memory)
for transfer




Activation Frame
(working memory)
for main




⇒ On exécute le corps du constructeur, on copie **day** dans **this.day**, **month** dans **this.month**




⇒ Le constructeur termine, on détruit le bloc de mémoire de travail, on assigne la référence à la variable **d**

# « What is this ? »

```
public class Date {
  private int day;
  private int month;
  public Date(int day, int month) {
     this.day = day;
     this.month = month;
  }
  // ...
  public static void main(String[] args) {
     Date d;
     int day, month;
     day = 17; month = 1;
     d = new Date(day, month);
  }
}
```

⇒ **this** lève l'ambiguïté, il permet de distinguer le paramètre **day** de la variable d'instance **day**.

# Final

# Java : final

- Pour une variable, le mot-clé **final** signifie qu'on ne peut changer sa valeur.

```
public class Point {
    public static final int MAX_VALUE = 100;
    private int x
    private int y;

    public void moveRight() {
        x = x + 1;
        if (x > MAX_VALUE) {
            x =0;
        }
    }
}
```

# Prologue

# Résumé

- L'**encapsulation** consiste à mettre les données et les opérations qui les transforment dans une même unité (l'objet)
- Les **modificateurs de visibilité** supportent l'encapsulation
- Les **variables** et les **méthodes de classe** sont partagées par les instances de la classe, et de toutes autres classes si leur visibilité est **public**
- Chaque **objet** possède une référence **this**. Elle désigne **l'objet lui-même**.

# Prochain module

- **Interface**

# References I


E. B. Koffman and Wolfgang P. A. T.
***Data Structures : Abstraction and Design Using Java.***
John Wiley & Sons, 3e edition, 2016.


Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

**Université d'Ottawa**