ITI 1121. Introduction to Computing II

Object oriented programming: visibility, variables and class methods

by

Marcel Turcotte

Version January 20, 2020

Preamble

Preamble

Overview

Overview

Object oriented programming: visibility, variables and class methods

We discover the role of the visibility modifiers in order to promote encapsulation. We add to our knowledge the concepts of class variables and class methods. Finally, we will see the role of a pre-defined reference variable **this**.

General objective :

- This week, you will be able to compare the concepts of instance and class variables, as well as the instance methods and class methods.

Preamble

Learning objectives

Learning objectives

- **Describe** the Java mechanisms to promote encapsulation
- **Explain** in your own words the following concepts: instance and class variables, instance and class methods.
- **Write** simple Java programs based on object-oriented programming.

Lectures:

- Pages 573-579 from E. Koffman and P. Wolfgang.

Lectures (continued):

Basic information

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

Detailed information

- 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

Preamble

Plan

Plan

Encapsulation

Definition: encapsulation

In object-oriented programming, **encapsulation** consists of grouping together in one unit (the object) the **data** and the **methods** that manipulate them.

Java: visibility modifiers

In Java, **visibility modifiers** allow us to control the access of variables and methods.

```
public class Point {

    private int x, y;

    public int getX() {
        return x;
    }

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

}
```

Java: visibility modifiers

```java
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: visibility modifiers

- Is the declaration of the method **equals valid**?

```
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;
      }
    }
}
```

Example : encapsulation

```java
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: visibility modifiers

In Java, **visibility modifiers** allow us to control the access of variables and methods.

- A **variable** or **method** whose visibility is **private** is only accessible in the body of the class where it is defined.
- A **variable** or **method** whose visibility is **public** is accessible in the body of the class where it is defined, but also in all the other classes of the program.

In **ITI1121**, **instance variables** should **always** be declared with visibility **private**!

Discussion: encapsulation

- Carefully **consider** the following two implementations of the class **Pair**.
    - Can one implementation replace the other **without causing any compile or runtime errors for the other classes of an 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

- Can we replace one implementation with the other **without causing compilation or execution errors for the following statements?**

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

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

p.setSecond(12);
```

Discussion: encapsulation (take 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: private methods

- Can you imagine situations for which one would like to declare a method **private**?

```java
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

- The "**interface**" of a class is made up of public methods
- Methods that should not be part of the interface will be declared "**private**"

Static context

Definition: class variable

A **class variable** is a variable defined in the body of the class and that is **shared by the instances** (objects) of this class (one memory location is used by all the instances of the class).

```
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;
        }
    }
}
```

Definition: class variable

- The keyword **static** introduces a class variable. As you can see, you have to make an effort to create a class variable, this is not what you get by default.

```
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;
        }
    }
}
```

Class variable

- The declaration of a **constant** is a good example of a situation where **class variables** are useful.
- Can you find **more examples** where class variables could be useful?

```java
public class Ticket {

    private static int last = 100;
    private int number;

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

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

```
public class Ticket {

  private static int last = 100;
  private int number;

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

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

Definition: class method

Class (static) methods do not belong to an object in particular. They are therefore **shared** by the instances of the class. Since they are not associated with an object, they **do not have access to instance variables**.

Class method

- Does the method **isValid** use any instance variable?

```
public class Point {

    private int x
    private int y;

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

Class method

- The method **isValid** should be a static (class) method.

```
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;
        }
    }
}
```

Class method

```
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;
    }
}
```

Examples of static methods

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

Calling a static method (static context)

- Use the **name of the class** followed by **the name of the method** for calling a **class method**.
- Use the **class name** followed by the **variable name** to access a **class variable**.

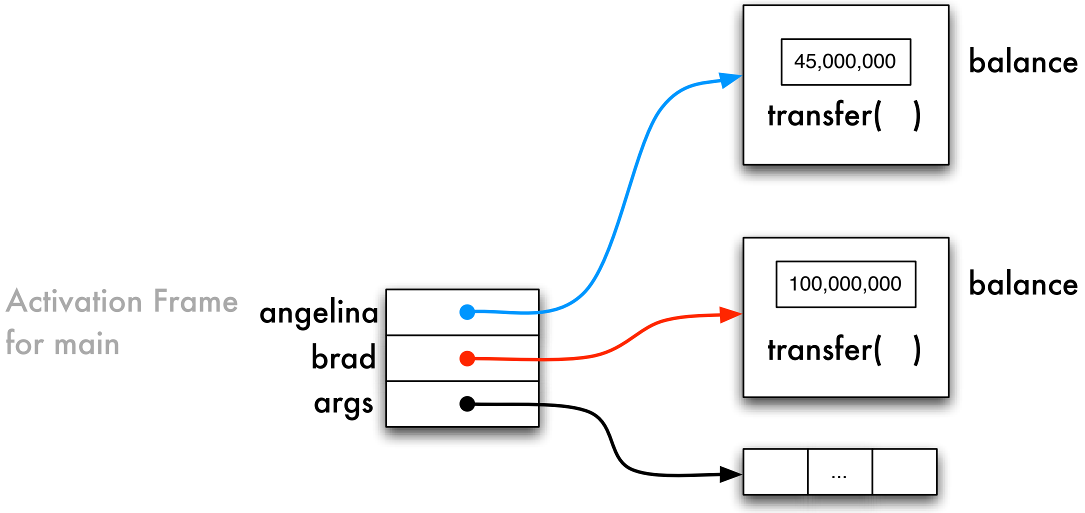
```
double a;

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*? »

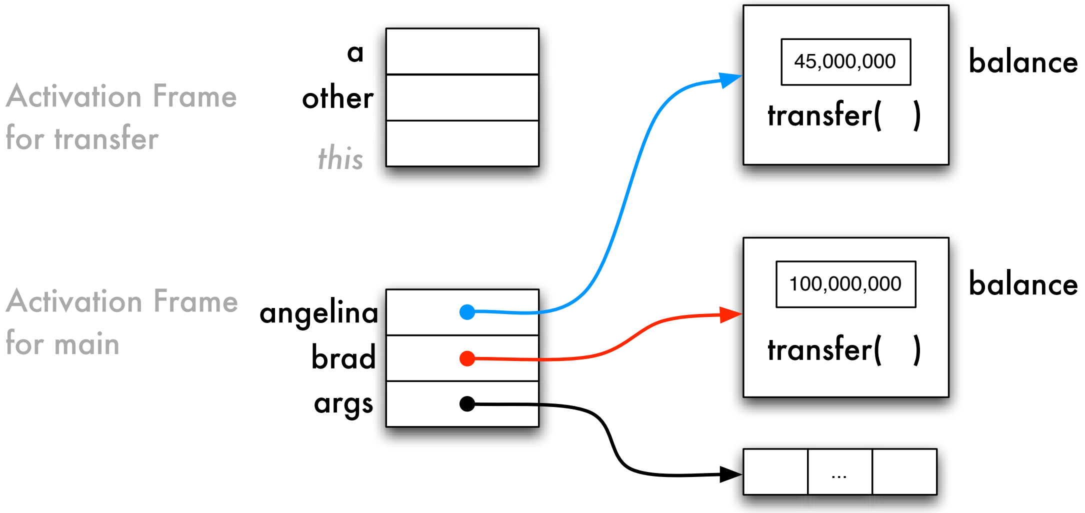
« What is this? »

Each object has a reference called **this**. It designates the object itself.

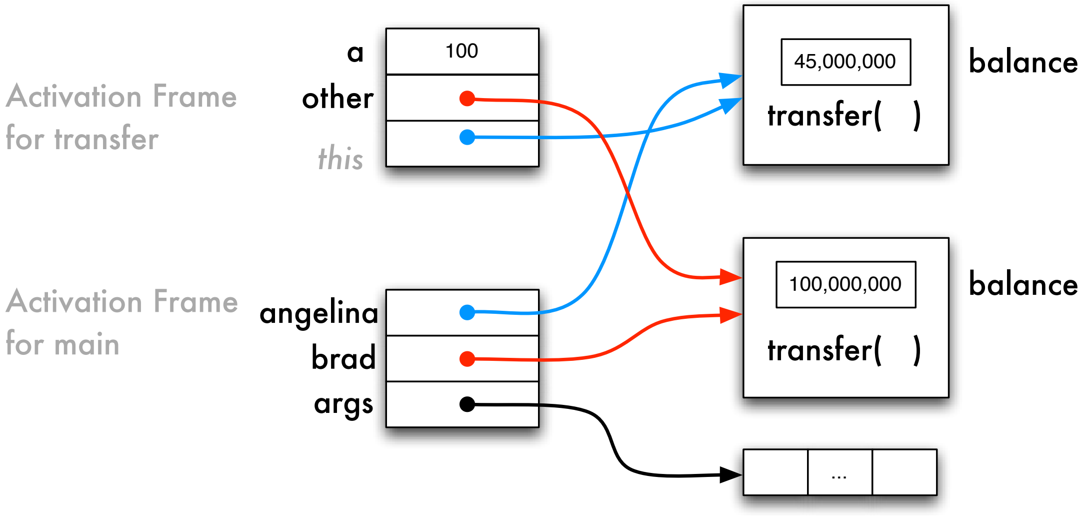
```
public class BankAccount {

  private double balance;

  // ...

  public boolean transfer(BankAccount other, double a) {

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

    ...
  }
}
```



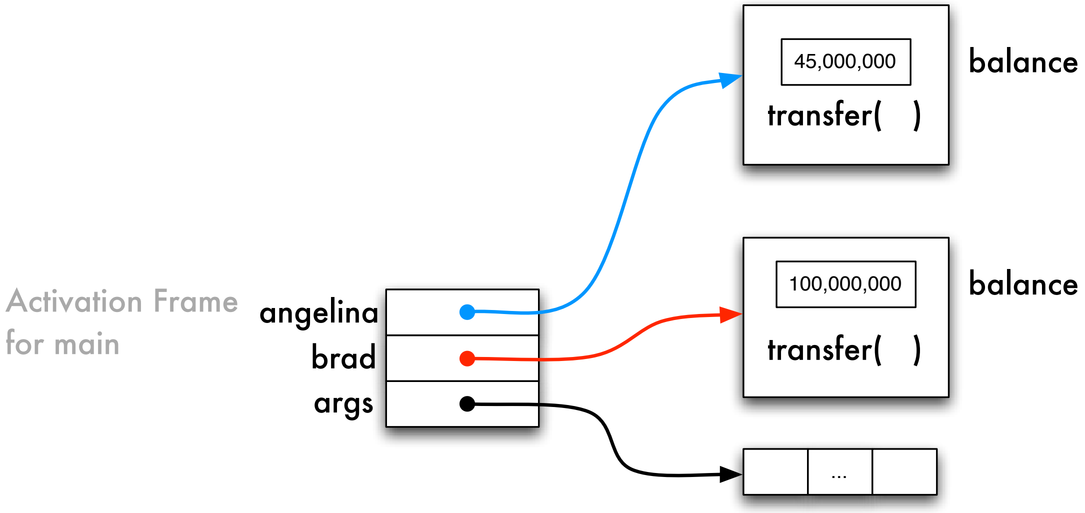
angelina.transfer(brad, 100)



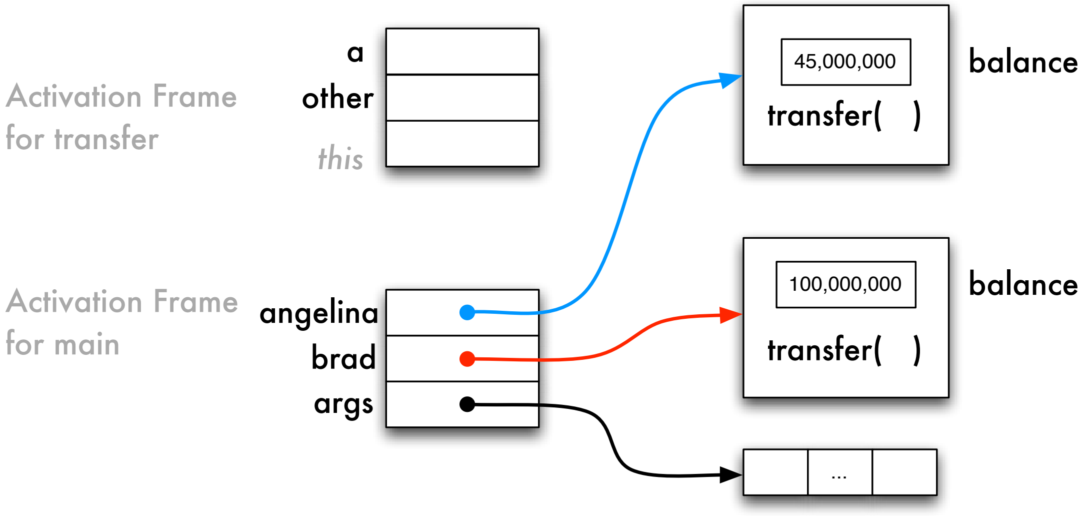
angelina.transfer(brad, 100)



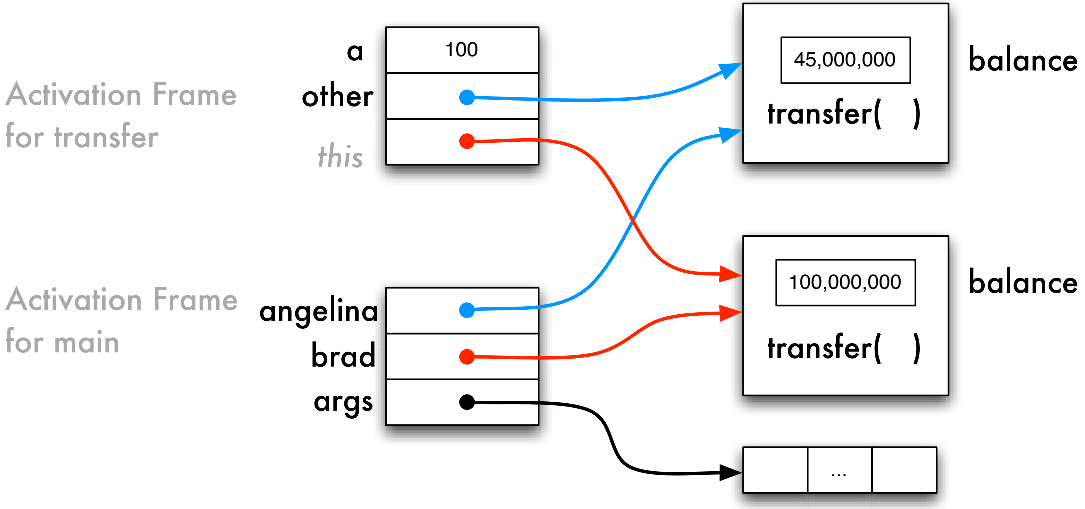
angelina.transfer(brad, 100)



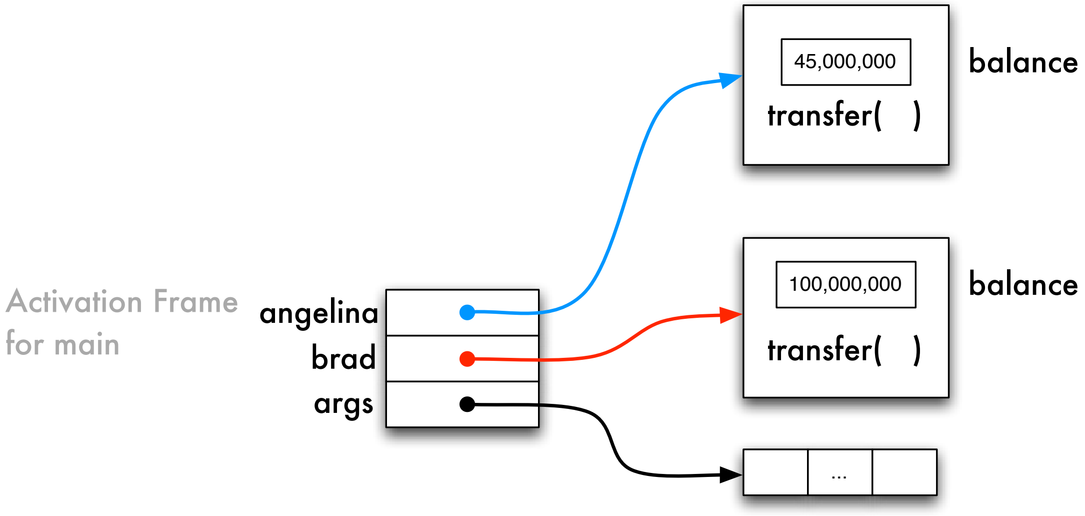
brad.transfer(angelina, 100)



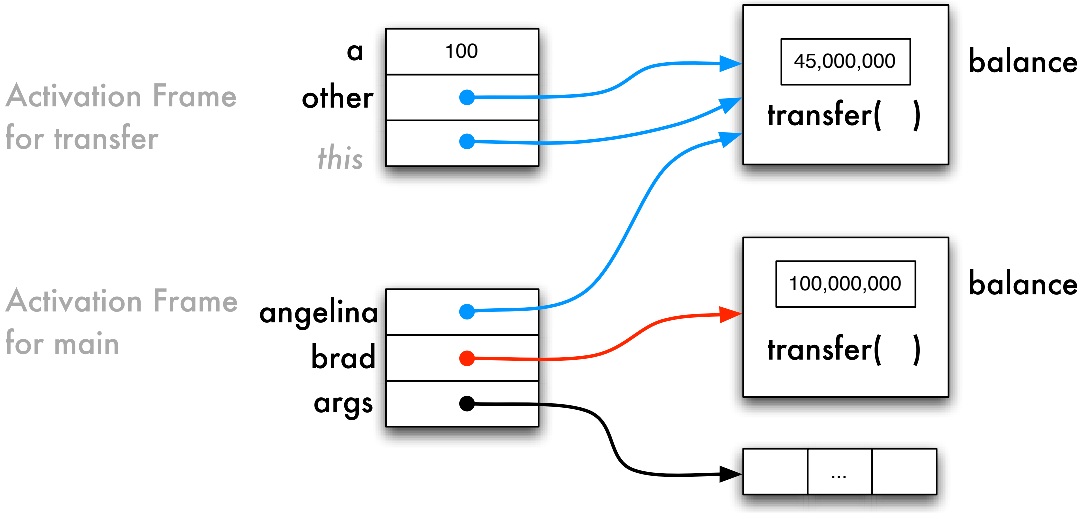
brad.transfer(angelina, 100)



brad.transfer(angelina, 100)



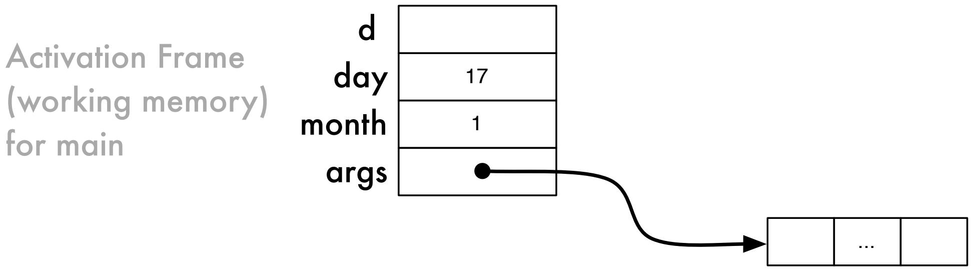
angelina.transfer(angelina, 100)



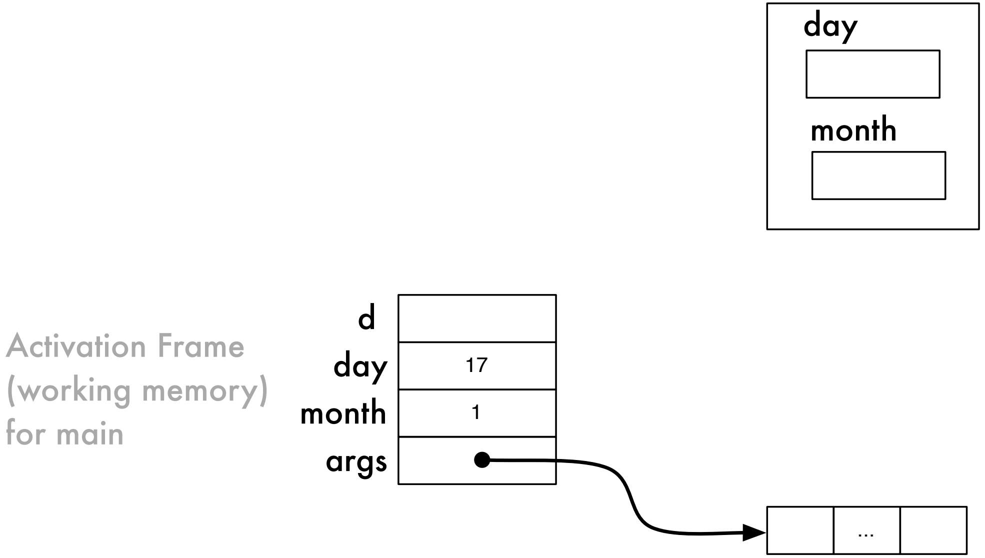
angelina.transfer(angelina, 100)

« What is this »

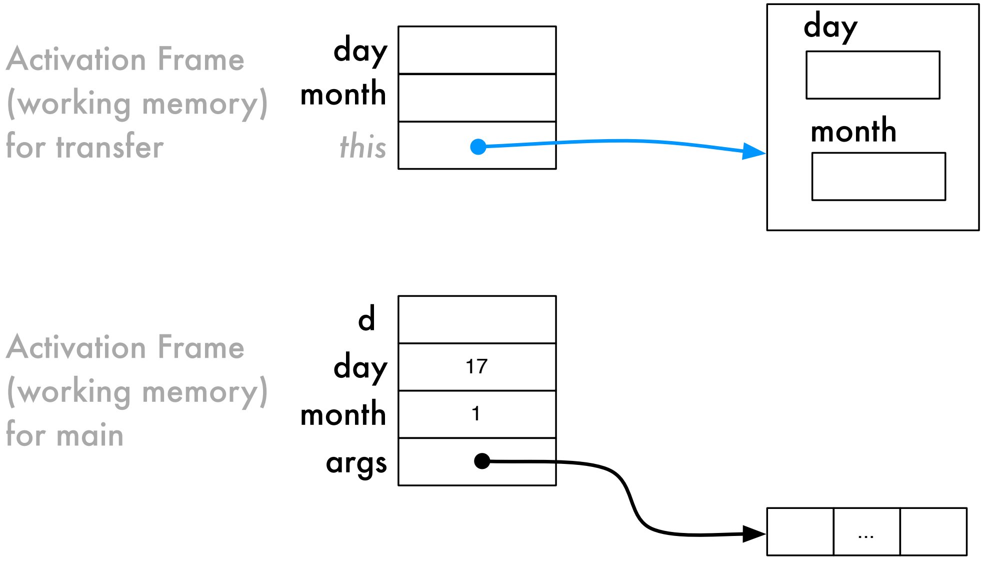
```
public class Date {
  private int day;
  private int month;
  public Date(int day, int month) {
      this.day = day;
      this.month = month;
  }
  // ...
  public static void main(String[] args) {
      Date d;
      int day, month;
      day = 17; month = 1;
      d = new Date(day, month);
 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



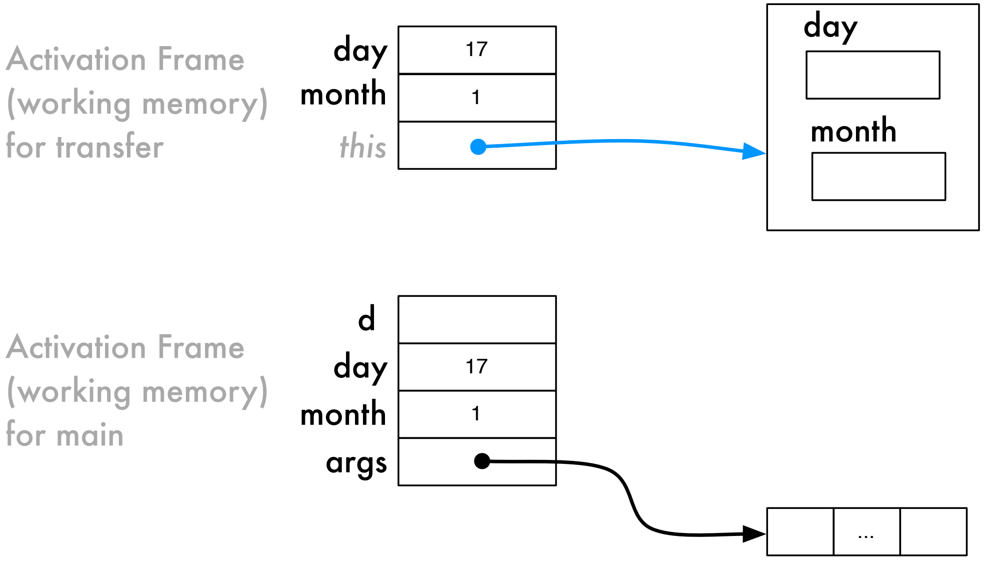
⇒ Working memory for the method **main**



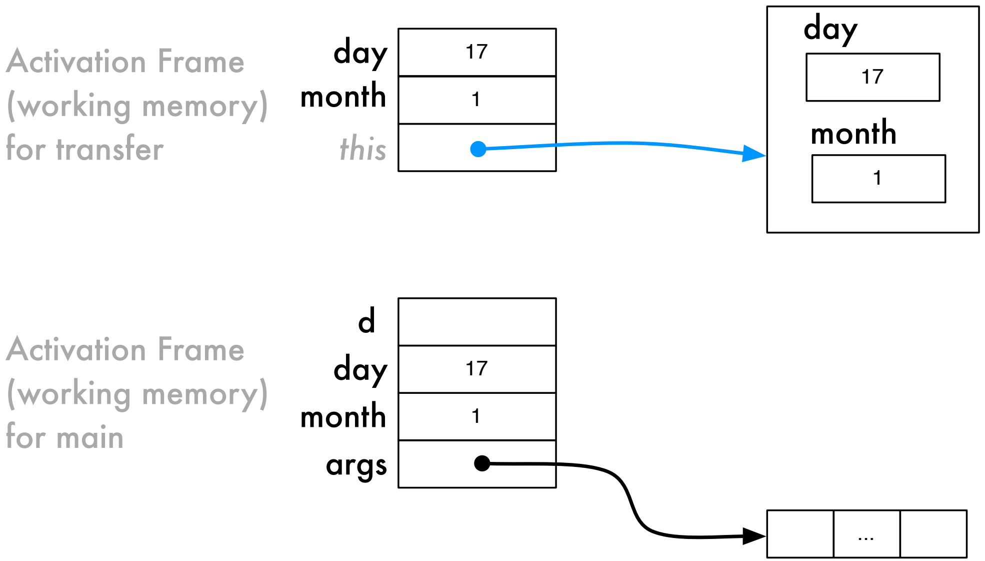
⇒ Executing “new Date(day,month)”, creating an object



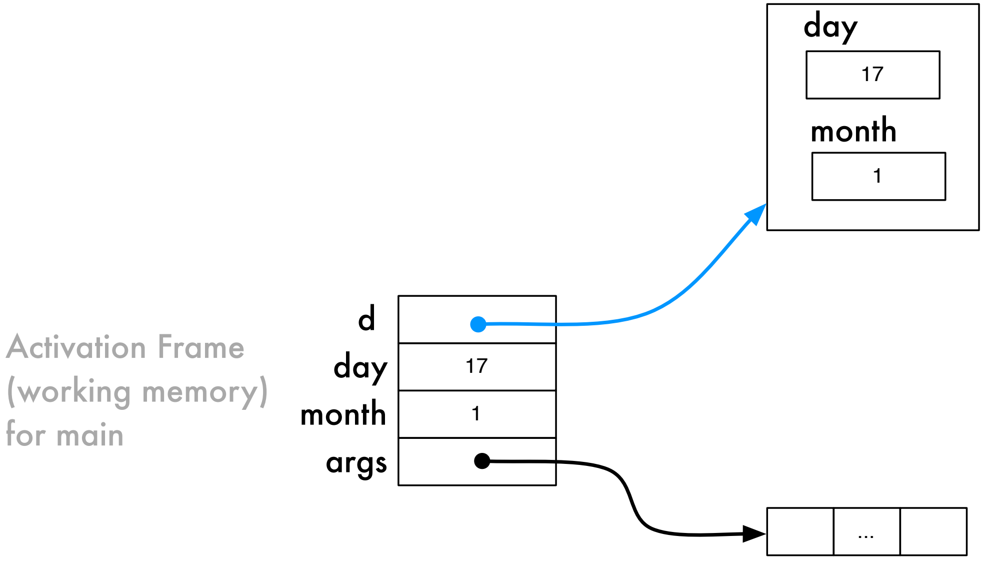
⇒ Calling the constructor, allocating the working memory for the call



⇒ Copying the values of the **actual parameters** to the **formal parameters**



⇒ Executing the body of the constructor, copying the value of **day** to **this.day**, **month** to **this.month**



⇒ The execution of the constructor terminates, its associated working memory is destroyed, assigning the reference of the object to **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** is used to lift (remove) an ambiguity, the parameter **day** and the instance variable **day**.

Final

Java: final

- The keyword **final** implies that the value of the variable cannot be changed.

```
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

Summary

- **Encapsulation** is about putting the data and the operations that transform them into a single unit (the object)
- **visibility modifiers** support encapsulation
- **Static variables** and **methods** are shared by the instances of the class, and all other classes if their visibility is **public**
- Each **object** has a reference **this**. It designates **the object itself**.

Next 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`

School of Electrical Engineering and **Computer Science** (EECS)

University of Ottawa