

UEF4.3. Object-Oriented Programming

Mrs. Sadeg and Mrs. Bousbia

s_sadeg@esi.dz, n_bousbia@esi.dz

National School of Computer Science (ESI)

Previous course

- Introduction to the Object-Oriented Programming Paradigm
- Some basic concepts
 - Abstraction
 - Encapsulation
 - Inheritance
 - Polymorphism

Chapter II

Classes and Objects

A series of horizontal lines in shades of green and yellow, with varying lengths and offsets, creating a modern, layered effect across the width of the slide.

The concept of class

Reminder

- A *class* can be thought of as a mold or a template from which objects can be created.
- A set of objects with the same characteristics and behaviors belong to the same class
 - All cars belong to the Car class
 - All students belong to the Student class
- A class itself cannot be used to run programs. To do so, it must be instantiated (creating objects)
- The class simply describes the structure of objects (their attributes and methods)

The concept of class

Example: Point Class

- Suppose a point is represented by two integer coordinates: abscissa and ordinate.
- So we need to define two attributes in the Point class
 - x (abscissa)
 - y (ordinate)
- Suppose we want to be able to:
 - Initialize the coordinates of a point
 - Move this point in the plane
 - Display the coordinates of this point
- So we need to have three methods in the Point class
 - initialize
 - move
 - display

Point Class

X
Y

initialize()
move()
display()

The concept of class

Defining a Point class (Java code)

The general template of a class in Java is:

```
class Point {
```

```
// instructions for defining the attributes and methods of the class
```

```
}
```

The concept of class

Defining the attributes of the Point class (code in Java)

```
class Point {  
private int x ; // abscissa  
private int y ; // ordinate  
// definition of methods  
}
```

- The attributes (properties) of a class can be of primitive type (integer, float, character or boolean) or of object type.
- Attribute declarations can be made anywhere within the class, but typically they are declared at the beginning or at the end of the class.

The private keyword specifies that the x and y fields are not accessible from outside the class (i.e. outside its own methods)

Point of class

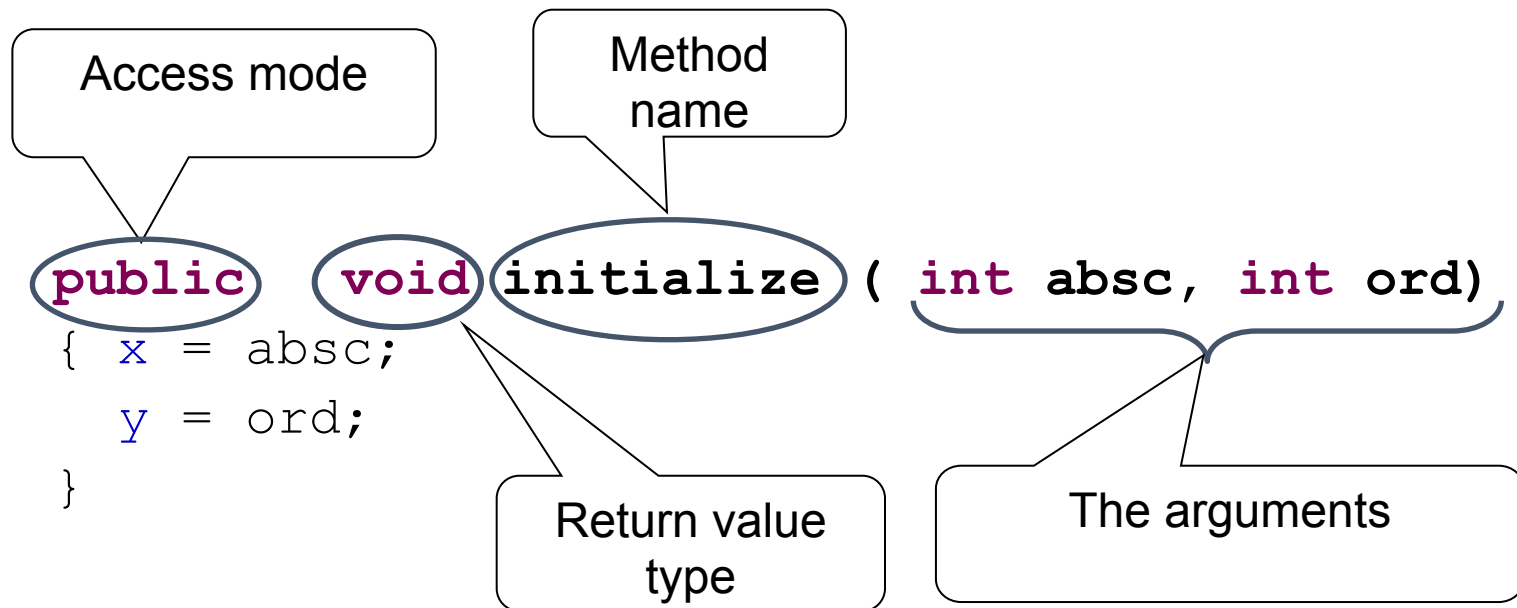
Attributes of the Point class
(Java)

```
class Point {  
    private int x ; // abscissa  
    private int y ; // ordinate  
    // definition of methods  
}
```

- The attributes (properties) of a class can be of primitive type (integer, float, character or boolean) or of object type.
- Attribute declarations can be made anywhere within the class, but typically they are declared at the beginning or at the end of the class.

The concept of class

Defining Point Class Methods (code in Java)



The concept of class

Example Point class: the complete code in Java

```
class Point {  
    private int x ;  
    private int y ;  
  
    public void initialize ( int absc, int ord){  
        x = absc;  
  
        y = ord;  
    }  
    public void move ( int dx, int dy){  
        x = x + dx;  
  
        y = y + dy;  
    }  
  
    public void display(){  
        System.out.println( "i am a point with coordinates: " + x + " " + y );  
    }  
}
```

The concept of object

Reminder

- Objects are shaped from the class, like moldings from a mold.
- An object of a class is also called ***an instance*** of that class.
- Once we have the class, we can create as many objects (instances) as we want.
- These objects will have:
 - Different identities.
 - States defined by the same attributes but with values that are generally different.
 - Identical behavior (Same methods)

Concept of object

- **Class = Mold = Model**
 - Describes the structure of the state (attributes and their types)
 - Defines the behavior of the object (its methods) and its interface (the message sendings accepted by the object)
- **Instance = Molding = Object**
 - Has a state (attribute values) that corresponds to the structure described by the class
 - Only responds to message sends allowed by the class (interface)

Classes and Objects

- **class : abstract**
 - There is no student called "student"
- **instance : concrete**
 - The student “Halimi Amina born on 03/17/2006”

Creating an object

How are objects created?

- To create a new object, we need to allocate memory space for it.
- Each object will have its own space where the values of its attributes will be stored.
- If we create 1000 objects of type *Point*, we will have 1000 locations for the variable x and 1000 locations for the variable y.
- The methods are not duplicated (useless)

Creating an object

Example: the Point class

- Objects of the Point class can be created and its public methods - *initialize*, *move* and *display* - can be freely applied these instances.
- Before creating an object, it must first be declared.

Point a;

- This declaration does not allocate memory for an object of type Point, but only for a reference to an object of type Point.

Creating an object

Example: Point class

To create a location for the object itself, we use the **new** keyword

```
a = new Point();
```

In this statement, the **new** operator creates an object of type Point and provides its reference.

This reference is assigned to the previously declared variable *a*.

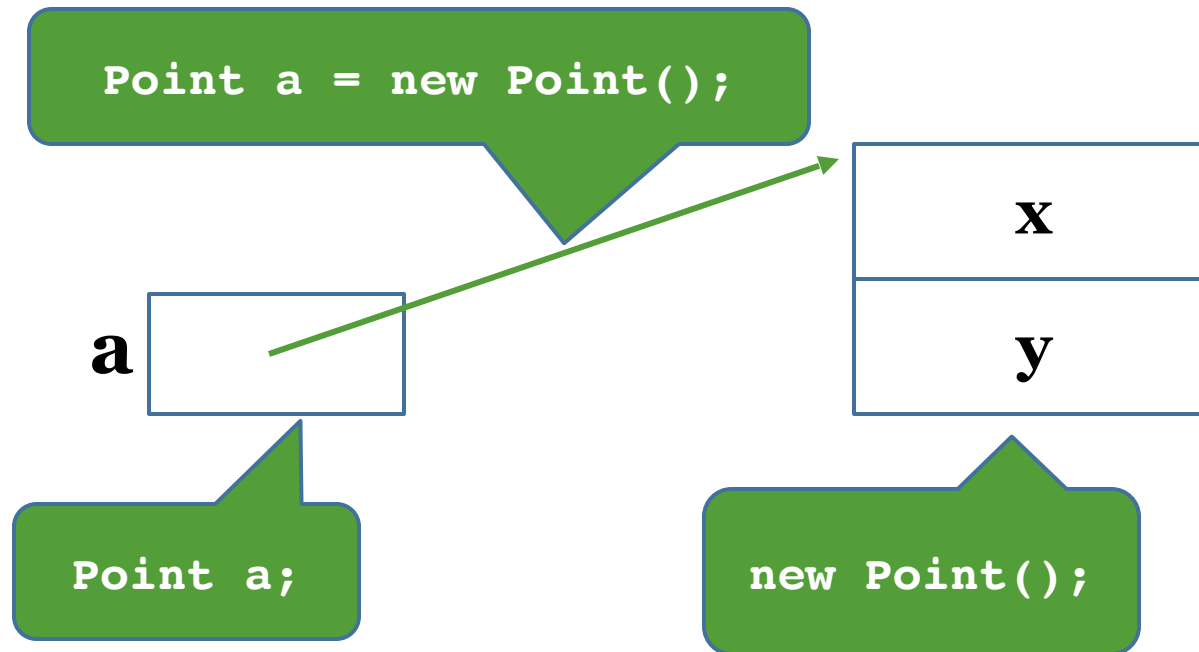
```
Point a;  
a = new Point();
```



```
Point a = new Point();
```


Creating an object

Example: the Point class



The concept of constructor

Point Class with a constructor

```
class Point {  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    {  
        x = absc;  
        y = ord;  
    }  
  
    public void move ( int dx, int dy)  
    {  
        x = x + dx;  
        y = y + dy;  
    }  
  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

The concept of constructor

Point Class with a constructor

```
class Point {  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
    }  
  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

This is the constructor for the Point class. Now the initialize method becomes useless. We can delete it

The concept of constructor

Definition

- A constructor is a special method used to create a new object.
- The object will obey the structure defined in the class, that is, it will have the attributes and behavior defined in the class. This requires allocating memory for storing state.
- The constructor is often used to initialize the various attributes of the object.
- It has the same name as the class
- Does not return a value
- Has no return type, not even `void`

The concept of constructor

1. Should a class have a constructor?
2. Can we create an object of a class that does not have a constructor?

The concept of constructor

Point Class without constructor

```
class Point {  
    private int x ;  
    private int y ;  
  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

No constructor is defined. How can objects be created ?

Creating an object

Example without constructor

In the absence of a constructor, we can create an object of type `Point` as follows:

```
a = new Point();
```

In this statement, the `new` operator creates an object of type `Point` and provides its reference.

This reference is assigned to the previously declared variable `a` and the **attributes `x` and `y` are initialized to the default values (0 for `int`)**

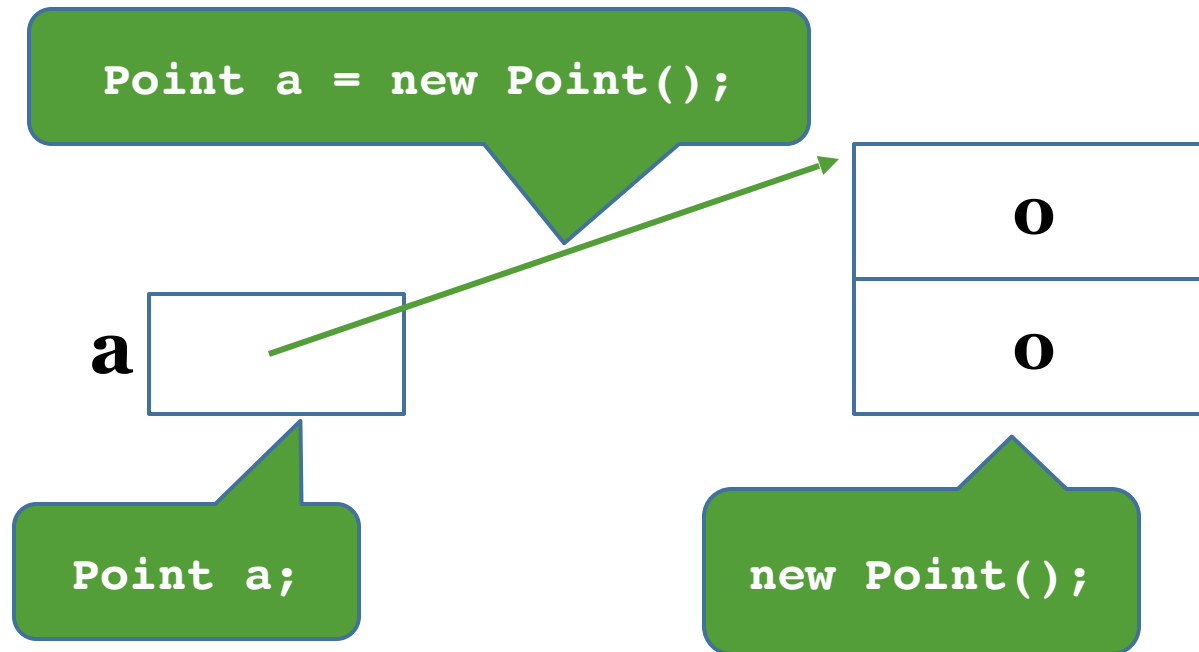
```
Point a;  
a = new Point();
```



```
Point a = new Point();
```

Creating an object

Example: the Point class



The concept of constructor

Attention !

```
class Point {  
  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
    }  
  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

Is it allowed to write

```
Point a = new Point()
```

?

The concept of constructor

Attention !

```
class Point {  
  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
    }  
  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

Is it allowed to write

```
Point a = new Point()
```

?

NO !

The concept of constructor

Attention !

```
class Point {  
  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
    }  
  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

Is it allowed to write

```
Point a = new Point()
```

?

NO !

What to do?

The concept of constructor

Back to example: Point Class

```
class Point {  
  
    private int x ;  
    private int y ;  
  
    public Point()  
    {  
    }  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
    }  
    public void moves ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        // display the values of x and y on the screen  
    }  
}
```

Point a = new
Point();

Correct !

The concept of constructor

Some rules

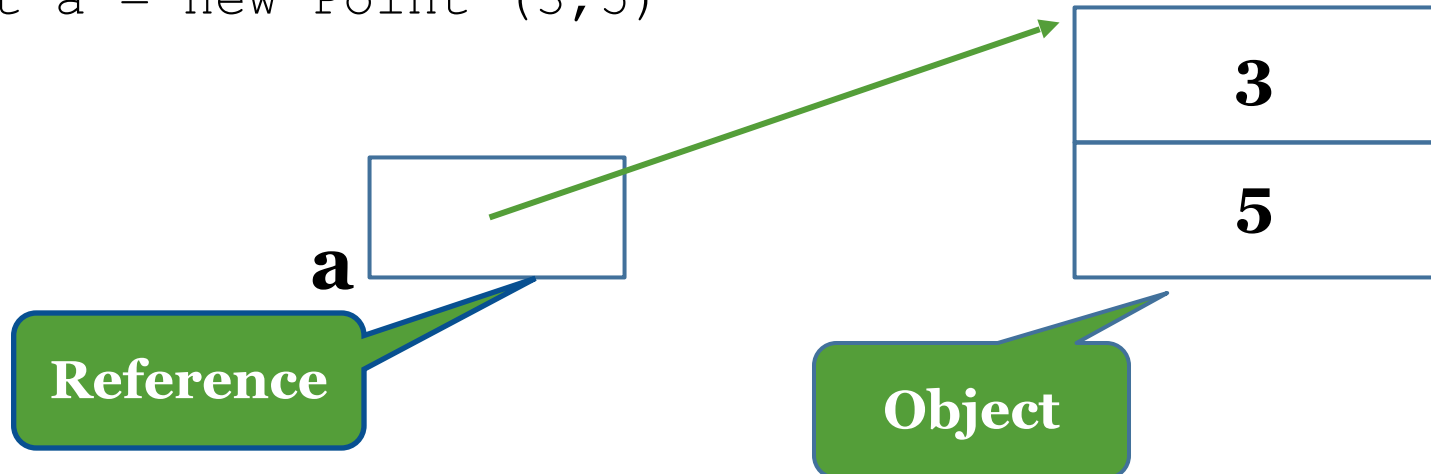
1. A class may have no constructor (as in our first example of a `Point` class). In this case objects are created using a default constructor with no arguments as in the statement `a = new Point () ;`
2. A class can have multiple constructors (what does this remind you of?).
3. Once a class has at least one constructor, the default constructor can no longer be used unless a no-argument constructor has been defined.

The concept of reference

- To use objects, you must reference them.
- The reference provides access to an object, but is not the object itself. It contains the memory address where the object is stored.

Example

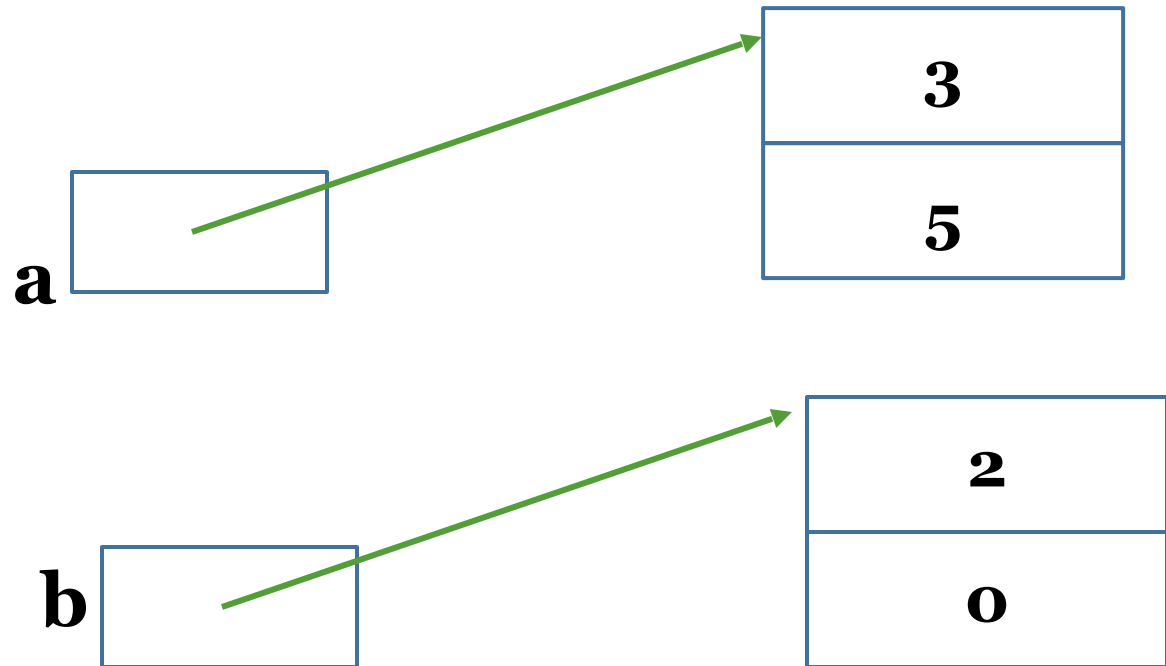
```
Point a = new Point (3,5)
```



The concept of reference

First example

```
Point a,b;  
a = new Point(3,5);  
b = new Point(2,0);
```



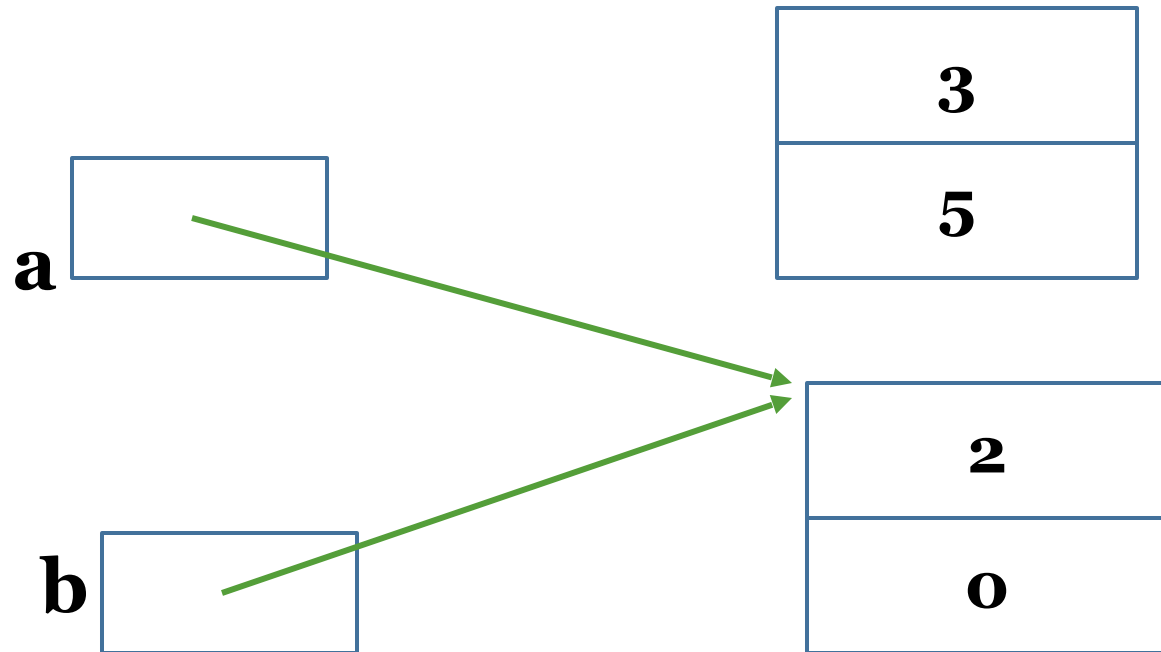
The concept of reference

First example: Assigning objects

Let's run the assignment

`a = b;`

The assignment only copies into 'a' the reference stored in 'b'



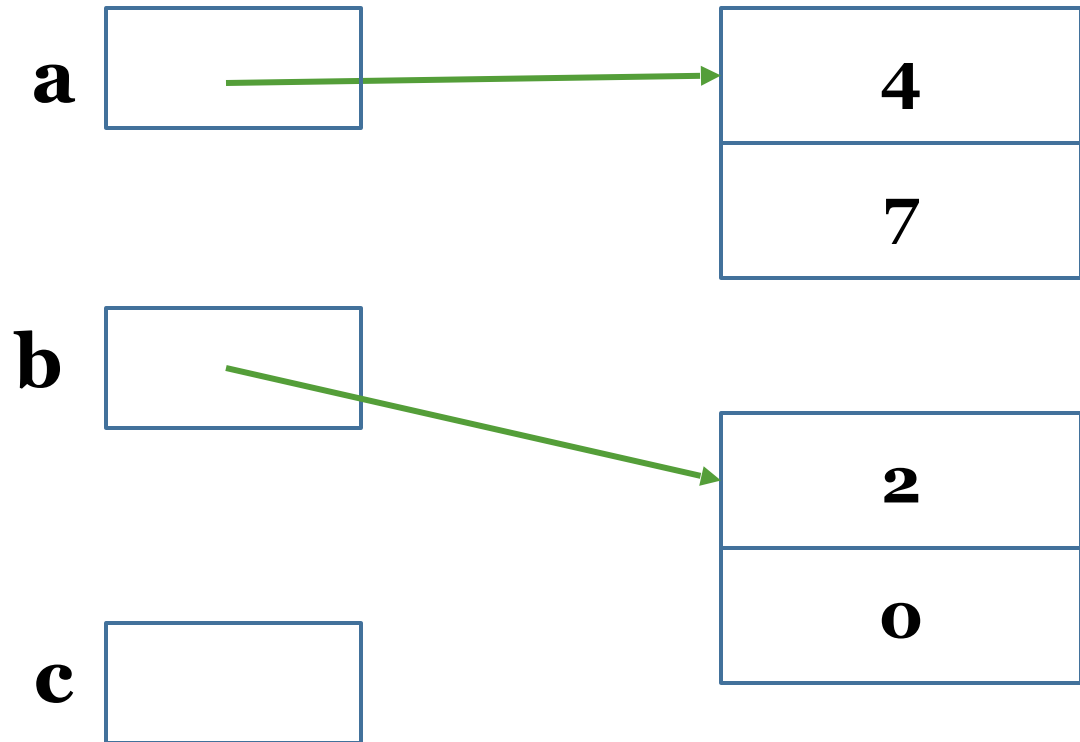
The concept of reference

Second example: object assignment

```
Point a, b, c;
```

```
a = new Point(4,7);
```

```
b = new Point(2,0);
```



The concept of reference

Second example: comparing objects

```
Point a, b, c;
```

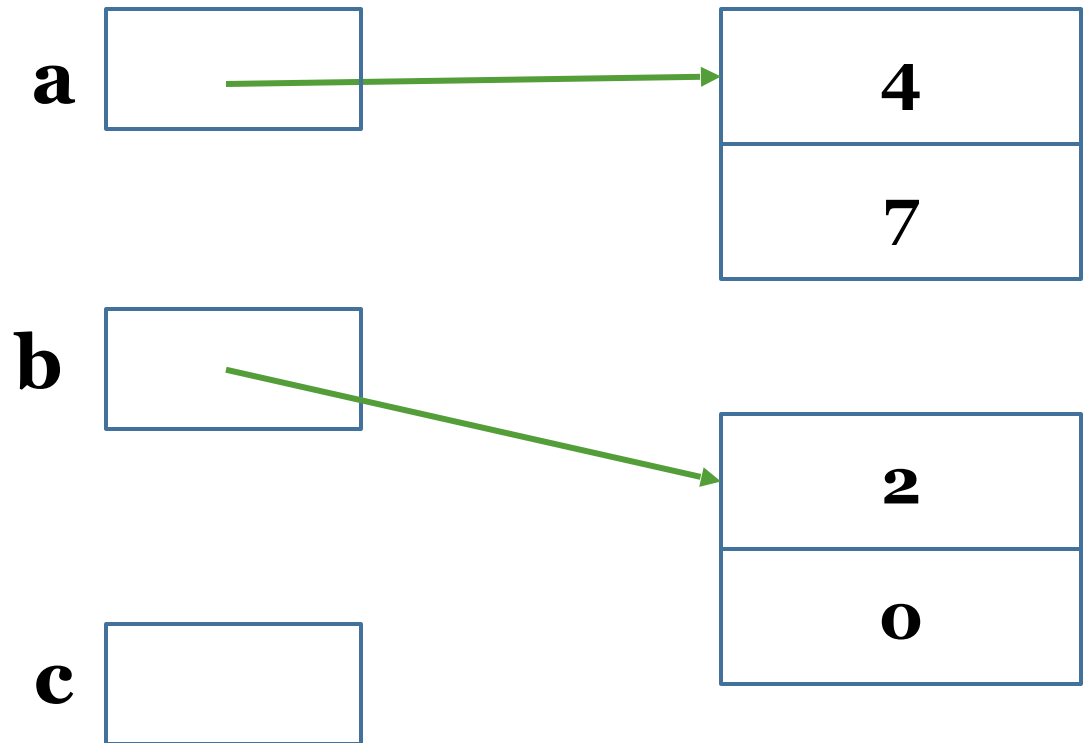
```
a = new Point(4,7);
```

```
b = new Point(2,0);
```

```
c = a;
```

```
a = b;
```

```
b = c;
```



The concept of reference

Second example: Comparing objects

```
Point a, b, c;
```

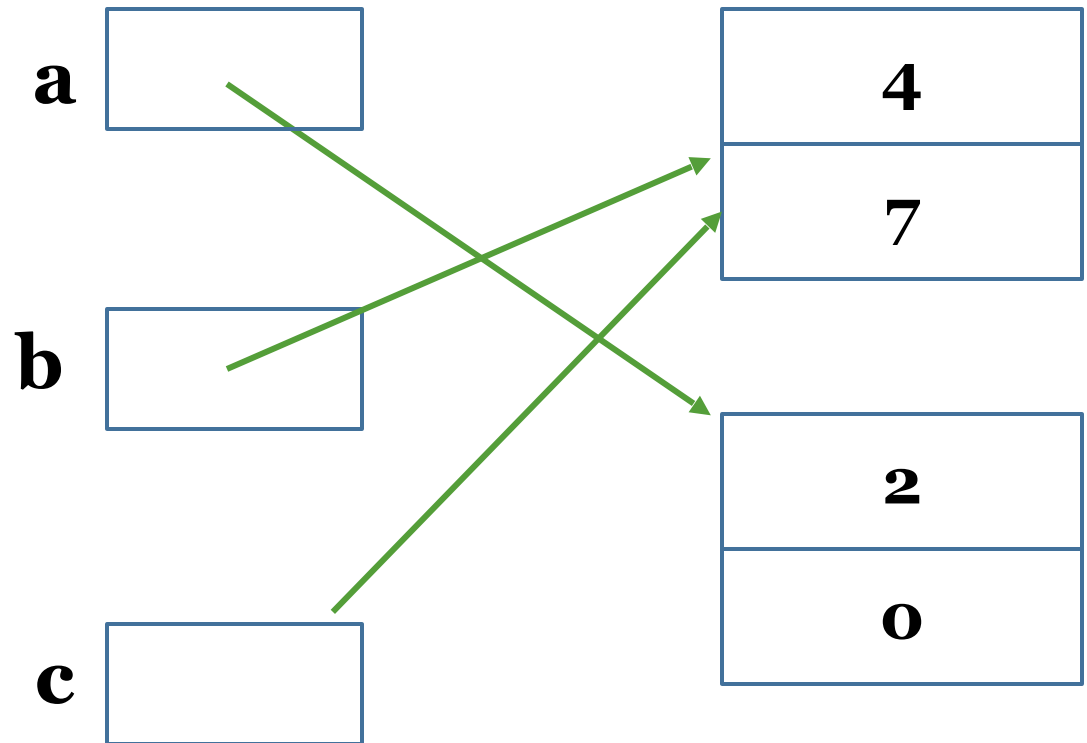
```
a = new Point(4,7);
```

```
b = new Point(2,0);
```

```
c = a;
```

```
a = b;
```

```
b = c;
```

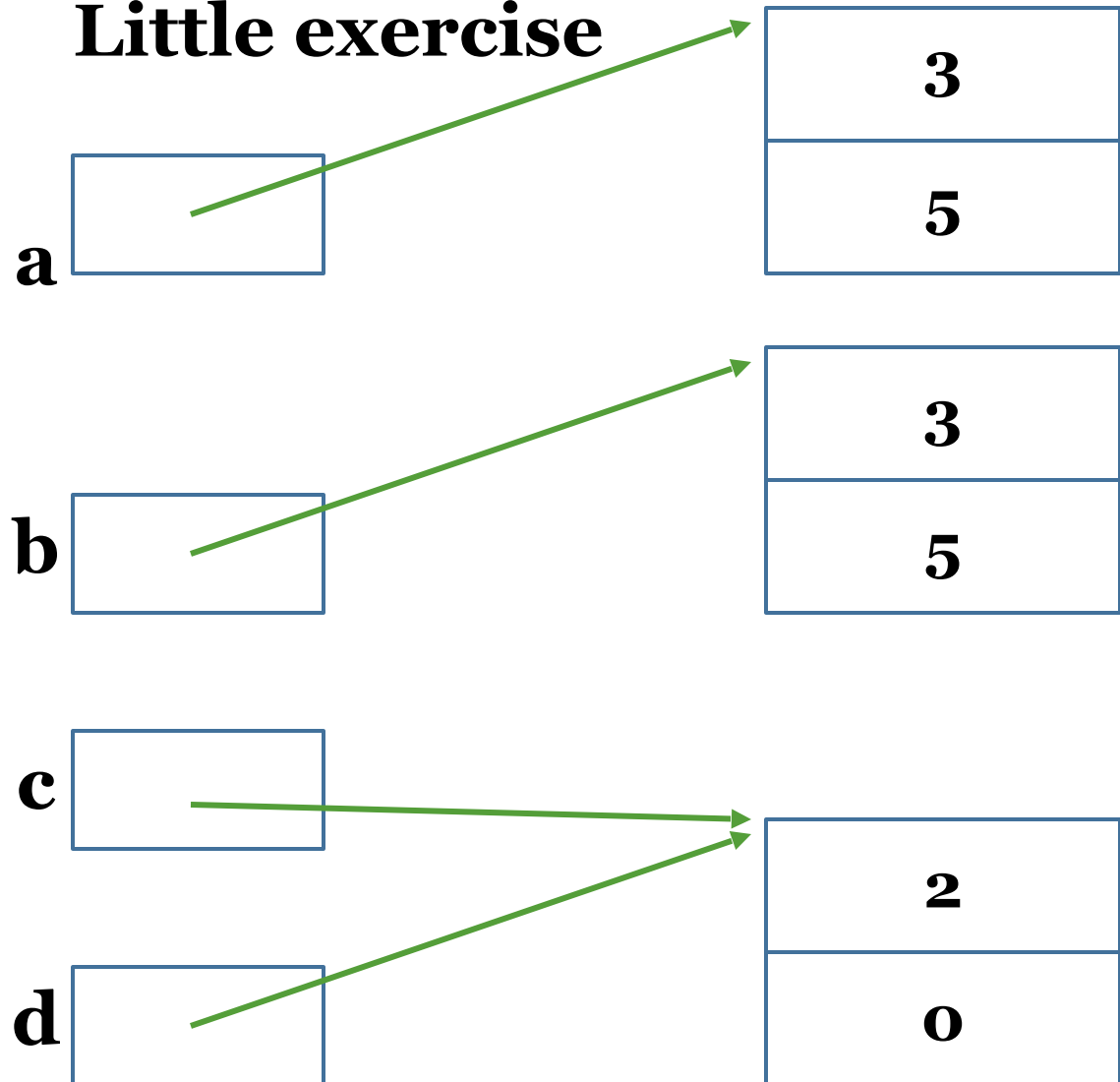


The concept of reference

Little exercise

`a == b ?`

`c == d ?`



Self-reference

- When processing one of its methods, an object may have to send itself a message (to access one of its attributes or invoke one of its methods). To do this, it uses the **this keyword**.
- Using the keyword **this** also helps avoid ambiguity.

Example: the Point constructor

```
public Point(int x, int y) {  
    this.x = x;  
    this.y = y;  
}
```

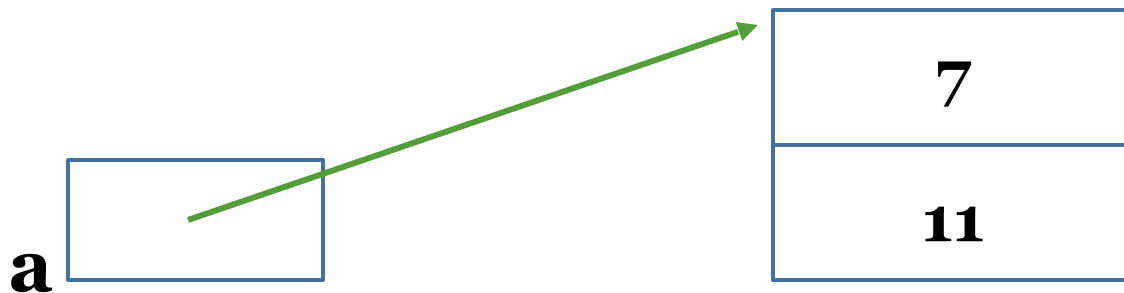
Exercise 1 What does the following program provide?

```
class Entier{
private int n ;
public Entier (int n) {this. n = n ; }
public void incr (int dn) { n += dn ; }
public void imprime () { System.out.println (n) ; }
}

public class TestEnt{
    public static void main (String args[]){
Entier n1 = new Entier (2) ; System.out.print ("n1 = ") ; n1.imprime();
Entier n2 = new Entier (5) ; System.out.print ("n2 = ") ; n2.imprime();
n1.incr(3) ; System.out.print ("n1 = ") ; n1.imprime() ;
System.out.println ("n1 == n2 est " + (n1 == n2)) ;
n1 = n2 ; n2.incr(12) ; System.out.print ("n2 = ") ; n2.imprime() ;
System.out.print ("n1 = ") ; n1.imprime() ;
System.out.println ("n1 == n2 est " + (n1 == n2)) ;
}
}
```

Using an object

- Once the object is created, any method can be applied to it.
- Example: `a.move(4, 6);`
- This is a call to the move method which will add the value 4 to the **x** field and the value 6 to the **y** field



Creating and using an object

Example : Circle class

Let's imagine that we want to create a *Circle class* to represent circles defined by a center (an object of type Point) and a radius of type float. The functionality of this class is limited to:

- Displaying the characteristics of a circle (coordinates of the center point and the radius)
- The movement of the center

Circle Class

center
radius

move()
display()

Point Class

X
Y

move()
display()

Creating and using an object

Circle class example: code in java

```
public class Circle
{
    private Point center ;
    private float radius ;

    public Circle ( int x, int y, float r) {
        center = new Point(x, y);
        radius = r;
    }

    public void display() {
        // display the center and radius of the circle
    }
}
```

Some particular methods

Among the methods that a class includes, we distinguish:

- The constructors
- Accessor methods (*Getter*): provide the values of some private attributes without modifying them.
 - Often, we use names in the form `getXXX`
- Mutator methods (Setters): Modify the values of certain private attributes.
 - Often, we use names in the form `setXXX`

Some design rules

For a good class design, there are some rules that are not mandatory but are strongly recommended to follow

1. Respect the principle of encapsulation by declaring all fields as private.
2. Rely on the concept of **contract** which considers that a class is characterized by:
 1. The headers of its public methods (interface)
 2. The behaviors of these methods.
3. Everything else(attributes, private methods and body) is called **implementation** and should remain private
4. The contract defines what the class does, while the implementation describes how it does it.

The Garbage Collector

- To create an object, a Java program uses the new operator.
- To free the memory space occupied by an object, a destructor is invoked (finalizers in Java).
- Java ensures automatic memory management through the Garbage Collector, which operates based on the following principle:
 - At any given time, we know the number of references to an object.
 - When an object is no longer referenced (there is no reference to it), it is certain that the program will no longer be able to access it. Therefore, its memory can be safely released.

Class Attributes

Exercise : We want to use the Point class to create points at will, but we want to know the number of points created.

What should be done?

Point Class

X
Y
nbPoints
move() display()

Class Attributes and Methods

Class Fields: Solution

```
class Point {  
    private int x ;  
    private int y ;  
    int nbPoints ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
      nbPoints ++;  
    }  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        System.out.println( "i am a point with coordinates: " + x + " " + y );  
    }  
}
```

Class Attributes and Methods

Class Fields: Solution

```
class Point {
    private int x ;
    private int y ;
    int nbPoints ;
```

This attribute will be duplicated in each object and will have the value 1

Incorrect solution

```
public Point( int absc, int ord)
```

```
{ x = absc;
  y = ord;
  nbPoints ++;
```

```
}
```

```
public void move ( int dx, int dy)
```

```
{ x = x + dx;
  y = y + dy;
}
```

```
public void display()
```

```
{
System.out.println( "i am a point with coordinates: " + x + " " + y );
}
```

```
}
```

Class Attributes

- When an object is created, a copy of each attribute is created for it.
- We can define attributes that exist only in a single copy for all instances of the same class (global data shared by all objects of the class).
- These attributes are called ***class attributes*** or ***static attributes***

Class Attributes and Methods

Class attributes: Solution

```
class Point {  
    private int x ;  
    private int y ;  
    static int nbPoints ;  
  
    public Point( int absc, int ord)  
    { x = absc;  
      y = ord;  
      nbPoints ++;  
    }  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
    public void display()  
    {  
        System.out.println( "I am a point with coordinates: " + x + " " + y );  
    }  
}
```

Class Methods

- Similarly, we can define *class* (or *static*) methods that can be called independently of any object
- A class method has a role that is independent of any specific object. This is the case, for example, for a method that operates only on class fields.
- In Java, the `static` keyword is used for this purpose
- A class method cannot operate on regular (non-static) attributes since it is not tied to any particular object.

Class Attributes and Methods

Class Methods: Back to Example Point

```
class Point {  
    private int x;  
    private int y;  
    static int nbPoints;  
  
    public Point(int absc, int ord)  
    { x = absc;  
      y = ord;  
      nbPoints++;  
    }  
  
    public void move(int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
  
    public static int nbPoints(){  
        return nbPoints;  
    }  
}
```

How can we call the
nbPoints() method?

Class Attributes and Methods

Class Methods: Back to Example Point

```
class Point {  
    private int x;  
    private int y;  
    static int nbPoints;  
  
    public Point(int absc, int ord)  
    { x = absc;  
      y = ord;  
      nbPoints++;  
    }  
  
    public void move(int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
  
    public static int nbPoints(){  
    return nbPoints;  
    }  
}
```

How can we call the
nbPoints() method?

```
int nbp = Point.nbPoints();
```

We don't need to create an
object to use this method
because it is a class method

Exercise 2.1 What mistakes were made in this code ?

```
class A {  
    static private int p = 20 ;  
    private int q ;  
  
    static int f (int n){ q = n ;}  
    void g(int n){ q = n ; p = n ;}  
}  
  
public class TestA{  
    public static void main(String args[]){  
        A a = new A() ;  
        int n = 5 ;  
        a.g(n) ;  
        a.f(n) ;  
        f(n) ;  
    }  
}
```

Overloading of methods

- We talk about overloading when a symbol has multiple meanings, and the appropriate one is chosen based on the context.
- In Java, it is possible to overload the methods of a class, including static ones. Multiple methods can have the same name as long as the number and type of their arguments allow the compiler to differentiate between them and make its choice.

Overloading of methods

```
class Point {  
    private int x ;  
    private int y ;  
  
    public Point( int absc, int ord)  
    { x = absc; y = ord;  
    }  
    public void move ( int dx, int dy)  
    { x = x + dx;  
      y = y + dy;  
    }  
  
    public void move ( int dx)  
    { x + = dx;  
    }  
  
    public void move ( short dx)  
    {x+ = dx;  
    }  
}
```

Overloading of methods

```
class Point {
    private int x ;
    private int y ;

    public Point( int absc, int ord)
    { x = absc; y = ord;
    }

    public void move ( int dx, int dy)
    { x = x + dx;
      y = y + dy;
    }

    public void move ( int dx)
    { x + = dx;
    }

    public void move ( short dx)
    { x+ = dx;
    }
}
```

```
Point a = new Point(1,2);
int n = 2; int b = 5; short p = 3;

a.move(n,b); //call move(int,int)

a.move(n); //call move(int)

a.move(p); // call move(short)
```


Overloading of methods





General rules

When encountering a method call, the compiler searches for all acceptable methods and selects the best one if it exists.

For a method to be acceptable, it must meet the following criteria:

- It must have the required number of arguments.
- The types of the actual arguments must be compatible with the types of the formal parameters.
- It must be accessible (a private method will not be acceptable outside its class).

The method selection process follows these rules:

- If no method is acceptable  Compilation error
- If only one method is acceptable  it is used.
- If multiple methods are acceptable  The most specific one is chosen
- If there is ambiguity  Compilation error

Overloading of methods

Example

```
public void move ( float dx, int dy) // move (float, int)
{...
}
public void move ( int dx, float dy) // move (int, float)
{...
}
```

What are the correct calls?

```
Point a = new Point(0,3);
float f ; int n ;
a.move(f,n) //
a.move(n,f) //
a.move(f,f) //
a.move(n,n) //
```

Overloading of methods

Example

```
public void move ( float dx, int dy) // move (float, int)
{...
}
public void move ( int dx, float dy) // move (int, float)
{...
}
```

What are the correct calls?

```
Point a = new Point(0,3);
float f ; int n ;
a.move(f,n) //
a.move(n,f) //
a.move(f,f) //
a.move(n,n) //
```

OK: Call move(float,int)
Ok: Call move(int, float)
ERROR: No method move(float,float)
ERROR: Ambiguity

Exercise 2.2 What errors are in this code?

```
class Surdef {  
    public void f(int n) { ..... }  
    public int f (int p) { ..... }  
    public void g (float x) { ..... }  
    public void g(double y) { ..... }  
    public void h(long n) { ..... }  
}
```

Exercise 3

Consider the following class definition:

```
class A{  
  public void f(int n) { ..... }  
  public void f(int n, int q) { ..... }  
  public void f(int n, double y) { ..... }  
}
```

With these statements:

```
A a ; byte b ; shorts p; int n ; long q ; float x; double y ;
```

Which instructions are correct, and in that case, which methods are called and what conversions, if any, are involved?

```
a.f(n);  
a.f(n, q) ;  
a.f(q) ;  
a.f(p, n) ;  
a.f(b, x) ;  
a.f(q, x) ;
```

Exchange of information with methods

- In programming languages there are two ways to transfer information.
 - By value: the method receives a copy of the actual argument's value, works on this copy, and modifies it without affecting the original argument
 - By address (or by reference): the method receives the address of the actual argument, works on it directly, and can modify its value
- Java always passes information by value.
 - What happens if a variable is of an object type?
 - What is actually passed, and what can be modified?

Exchange of information with methods

Variable of type Object

In the case where the manipulated variable is of an object type, its name represents its reference, so the method receives a copy of this reference. This means the method can modify the referenced object

Exercise 4 What results does this program provide?

```
class A {  
    private int n ;  
    public A(int nn){n = nn;}  
    public int getn() { return n; }  
    public void setn (int nn) { n = nn; }  
}  
  
class Util {  
    public static void incre (A a, int p) { a.setn(a.getn()+p);}  
    public static void incre (int n, int p) { n += p ;}  
}
```

```
public class Trans {  
    public static void main(String args[]) {  
        A a = new A(2) ;  
        int n = 2 ;  
        System.out.println("value of a before: " + a.getn());  
        Util.incre(a,5);  
        System.out.println("value of a after: " + a.getn());  
        System.out.println("value of n before: " + n);  
        Util.incre(n, 5);  
        System.out.println("value of n after: " + n);  
    }  
}
```


Exercise 5

The Java program below was written by an OOP beginner.

1. What do you think about this code? Does it follow the OO approach? Explain.
2. Suggest an improved version.

```
public class intsum{
public static void main(String args []){
    int suma = Integer.parseInt( args [0]);
    int sumb = Integer.parseInt( args [1]);
    int sum = suma + sumb ;
    int diff = suma - sumb ;
    int prod = suma * sumb ;
    double avg = ( suma + sumb )/2;
    System.out.println( "The sum of the two numbers is " + sum );
    System.out.println( "The difference of the two numbers is " + diff );
    System.out.println( "The product of the two numbers is " + prod );
    System.out.println( "The average of the two numbers is " + avg );
    if ( suma > sumb )
        System.out.println( "The largest number is " + suma );
    else
        System.out.println( "The largest number is " + sumb );
}
}
```