Chapter 5: Classes and Objects in Depth

Introduction to methods

What are Methods

- Objects are entities of the real-world that interact with their environments by performing services on demand.

- Objects of the same class have:
  - the same characteristics: store the same type of data.
  - And the same behavior: provide the same services to their environment.

- Services that objects provide are called methods.

Why Methods

- Information hiding prevent the data an object stores from being directly accessed by outsiders.

- Encapsulation allows objects containing the appropriate operations that could be applied on the data they store.

- So, the data that an object stores would be accessed only through appropriate operations.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Method Declaration

- Method declaration is composed of:
  - Method header.
  - Method body

```
<method header> {

        <method body>

}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Method Declaration (cont.)

```
<modifiers>   <return type>   <method name>   ( <parameters>   ){

      <method body>

}
```

Modifier	Return Type	Method Name	Parameters

```
public      void      setOwnerName  (   String  name   ) {

      ownerName = name;                          ⟵  Method body

}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Method Header

```
<modifiers>  <return type>  <method name>  ( <parameters>  ){

        <method body>

}
```

- The *modifiers* represent the way the method is accessed.
- The *return type* indicates the type of value (if any) the method returns.
  - If the method returns a value, the type of the value must be declared.
  - Returned values can be used by the calling method.
  - Any method can return at most one value.
  - If the method returns nothing, the keyword *void* must be used as the *return type*.
- The *parameters* represent a list of variables whose values will be passed to the method for use by the method.
  - They are optional.
  - A method that does not accept parameters is declared with an empty set of parameters inside the parentheses.

Types of methods

- There 3 different criteria defining types of methods:
  - Modifiers: this criteria is also composed of 3 sub-criteria:
    – Visibility: public or private (or protected in csc 113)
    – Shared between all instances or not: class member (static) or instance method.
    – Override able or not (final): to be discussed in CSC 113.
  - Return type: method with or without (void) return value.
  - Parameters: with or without parameters.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Example of Methods with No-Parameters and No-Return value

```java
import java.util.Scanner;

public class Course {
    // Attributes
    private String studentName;
    private String courseCode ;
    private static Scanner input = new Scanner(System.in); //Class att.
    // Methods
    public void enterDataFromKeyBoard() {
        System.out.print ("Enter the student name: ");
        studentName = input.next();

        System.out.print ("Enter the course code: ");
        courseCode = input.next();
    }

    public void displayData() {
    System.out.println ("The student name is: " + studentName);
    System.out.println ("The the course code is: "+ courseCode);
    }

}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Message Passing Principle or Method Invocation

- Message passing is the principle that allows objects to communicate by exchanging messages.

- Passing a message to an object means ordering this latter to execute a specific method.

- Passing messages to objects is also known as method invocation.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Method Invocation

- Invoking a method of a given object requires using:
  - the instance variable that refers to this object.
  - the dot (.) operator as following:

    instanceVariable.methodName(arguments)

```java
public class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
//Create and assign values to course1
        course1 = new Course( );
        course1.enterDataFromKeyBoard();   course1.display();
//Create and assign values to course2
        course2 = new Course( );
        course2.enterDataFromKeyBoard();   course2.display();

    }
}
```

Method Invocation Execution Schema

```java
class Client {

    public static void
        main(String[] arg) {

      X obj = new X();
      // Block statement 1

      obj.method();
      // Block statement 2
    }

      . . .

}
```
The client

```java
class X {

    . . .

    public void method() {

        // Method body
    }

    . . .

}
```

Block statement 1 executes

The method Invocation

Passing Parameters
if exist

The method body starts

Return result if any

Block statement 2 starts

The method body finishes

The client

Returning a Value from a Method

- A method returns to the code that invoked it when it:
  - completes all the statements in the method,
  - reaches a return statement, or
  - throws an exception (covered in CSC 113),

- If the method returns a value:
  - The caller must declare a variable of the same type of the return value.
  - The caller assigns the return value to the variable:

    variableName = instanceVariable.methodName(args);

The *return* keyword

- The *method's return type* is declared in its method declaration.
- The *return* statement is used within the body of the method to return the value.
- Any method declared *void* doesn't return a value.
  - It does not need to contain a return statement.
  - It may use a return statement to branch out of a control flow block and exit the method. The return statement is simply used like this:
    - return;
  - Return a value from a such method, will cause a compiler error.
- Any method that is not declared void:
  - must contain a return statement with a corresponding return value, like this:
    - return returnValue;
  - The data type of the return value must match the method's declared return type.
  - you can't return an integer value from a method declared to return a boolean.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Example of a Method with Return value

```java
public class Student {
    // Attributes
    private String studentName;
    private int midTerm1, midTerm2, lab, final ;
    // Methods

    public int computeTotalMarks() {
        int value = mid1 + mid2 + lab + final;

        return value;
    }
}
```

```java
public class TestStudent {
    public static void main (String [] args) {
        Student st = new Student();
        int total;

        …

        total = st.computeTotalMarks();
        System.out.println(total);
    }
}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Template for Methods with Return value

```java
public class ClassName {
    // Attributes
    ...
    // Methods
    ...
    public returnType methodName(…) {
        returnType variableName;
        // 1 - calculate the value to return
        // 2 - assign the value to variableName
        return variableName;
    }
}
```

```java
public class ClientClass {
    public static void main (String [] args) {
        ClassName instanceVariable = new ClassName();
        returnType receivingVaraiable;

        ...

        receivingVaraiable = instanceVariable.methodName(…);

        ...
    }
}
```

Passing Information to a Method

- The declaration for a method declares the number and the type of the data-items to be passed for that method.

- *Parameters* refers to the list of variables in a method declaration.

- *Arguments* are the actual values that are passed in when the method is invoked.

- When you invoke a method, the arguments used must match the declaration's parameters in type and order

Arguments and Parameters

- An argument is a value we pass to a method.
- A parameter is a placeholder in the called method to hold the value of the passed argument.

```
class Sample {

    public static void
            main(String[] arg) {

    Account acct = new Account();
    . . .

    acct.add(400);
    . . .
    }

    . . .
}
```
argument

```
class Account {

    . . .

    public void add(double amt) {

        balance = balance + amt;
    }

    . . .
}
```
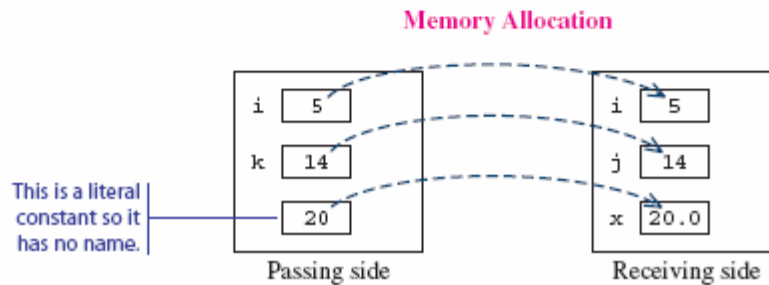parameter

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Matching Arguments and Parameters


```
Demo demo = new Demo( );

int i = 5;
int k = 14;          Passing side

demo.compute( i, k, 20 );
```

```
class Demo {

  public void compute(int i, int j, double x) {
    ...
  }
}
                                    Receiving side
```

Memory Allocation

This is a literal constant so it has no name.

i	5
k	14
	20

Passing side

i	5
j	14
x	20.0

Receiving side

- The number or arguments and the parameters must be the same
- Arguments and parameters are paired left to right
- The matched pair must be assignment-compatible (e.g. you cannot pass a double argument to a int parameter)

Parameter Passing

- When a method is called:
  - The parameters are created.
  - The values of arguments are copied into the parameters' variables.
  - The variables declared in the method body (called local variables) are created.
  - The method body is executed using the parameters and local variables.

- When the method finishes:
  - Parameters and local variables are destroyed.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

Passing Objects to a Method

- As we can pass primitive data type values, we can also pass object references to a method using instance variables.

- Pass an instance variable to a method means passing a reference of an object.
    - It means that the corresponding parameter will be a copy of the reference of this objects.
        - Because the passing parameter mechanism copies the value of the argument (which is an object reference) into the parameter.
    - The argument and its corresponding parameter refer to the same object.
        - The object is not duplicated.
        - There are two instance variables (the argument and the parameter) referring to the same object.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

How Private Attributes could be Accessed

- Private attributes are not accessible from outside.
  - Except from objects of the same class.
- They are accessible:
  - From inside: from the object containing the data itself.
  - From objects of the same class.
- They are accessible from outside using accessor operations.
  - Getters
  - Setters

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

```java
class Course {

    // Data Member
    private String studentName;
    private String courseCode ;

}
```

```java
public class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
//Create and assign values to course1
        course1 = new Course( );
        course1.courseCode= "CSC112";
        course1.studentName= "Majed AlKebir";

//Create and assign values to course2
        course2 = new Course( );
        course2.courseCode= "CSC107";
        course2.studentName= "Fahd AlAmri";


        System.out.println(course1.studentName + " has the course "+
                            course1.courseCode);
        System.out.println(course2.studentName + " has the course "+
                            course2.courseCode);

    }
}
```
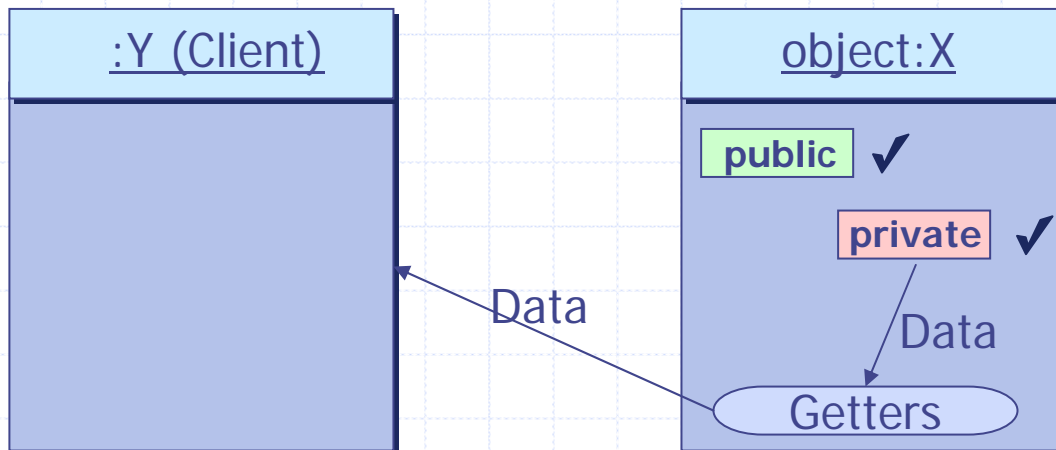
Getters

The object point of view

- Are operations performed by the object returning to outsiders data retrieved from the object state.

The user point of view

- Are services called from outside allowing to retrieve data from the object state.

:Y (Client)

object:X

public ✔

private ✔

Data

Getters

Data

Getters are:
- Public
- With no parameters
- With return value

Template for Getters

```java
public class ClassName {

    private dataType1 attribute1;
    . . .
    private dataTypen attributen;
    . . .


    public dataType1 getAttribute1() {

        return attribute1;
    }


    . . .


    public dataTypen getAttributen() {

        return attributen;
    }
    . . .
}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Setters

The object point of view

- Are operations performed by the object allowing to receive and store in the object state the data provided by outsiders.

The user point of view

- Are services used by outsiders allowing to provide to the object the data that should be stored in the object state.

```
:Y (Client)
```

```
object:X

public ✓

private ✓

Data

Setters
```
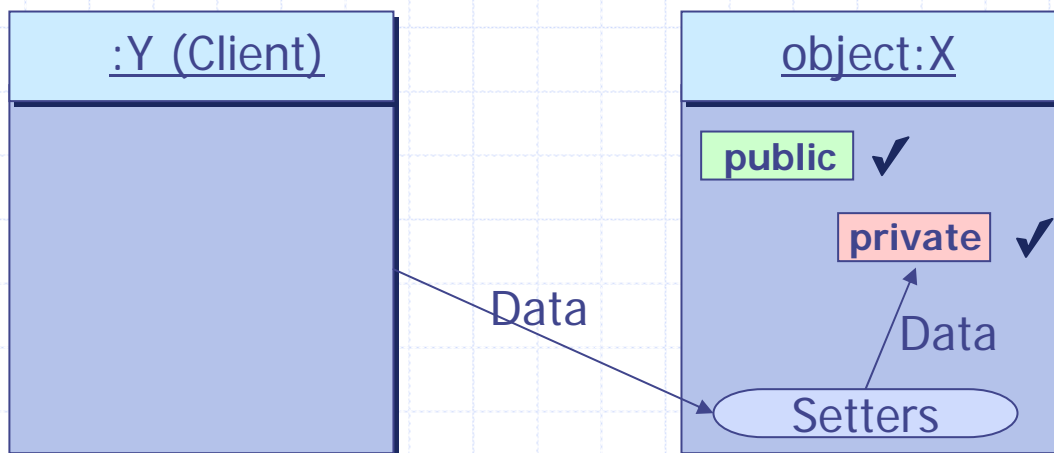
Data

Setters are:
- •Public
- •With 1 parameter
- •With no return value

Template for Setters

```java
public class ClassName {

    private dataType1 attribute1;
    . . .
    private dataTypen attributen;
    . . .


    public void setAttribute1(dataType1 param){

        attribute1 = param;
    }


    . . .


    public void setAttributen(dataTypen param) {

        attributen = param;
    }
    . . .
}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

```java
public class Course {

    // Attributes
    private String studentName;
    private String courseCode ;

    ...
    public String getStudentName() {
        return studentName;
    }
    public String getCourseCode() {
        return courseCode;
    }
    ...
    public void setStudentName(String val) {
        studentName = val;
    }
    public void setCourseCode(String val) {
        courseCode = val;
    }

}
```

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

```java
public class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
//Create and assign values to course1
        course1 = new Course( );
        course1.setCourseCode("CSC112");
        course1.setStudentName("Majed AlKebir");

//Create and assign values to course2
        course2 = new Course( );
        course2.setCourseCode("CSC107");
        course2.setStudentName("Fahd AlAmri");

        System.out.println(course1.getStudentName() +
                " has the course " +  course1.getCourseCode());
        System.out.println(course2.getStudentName() +
                " has the course " + course2.getCourseCode());

    }
}
```
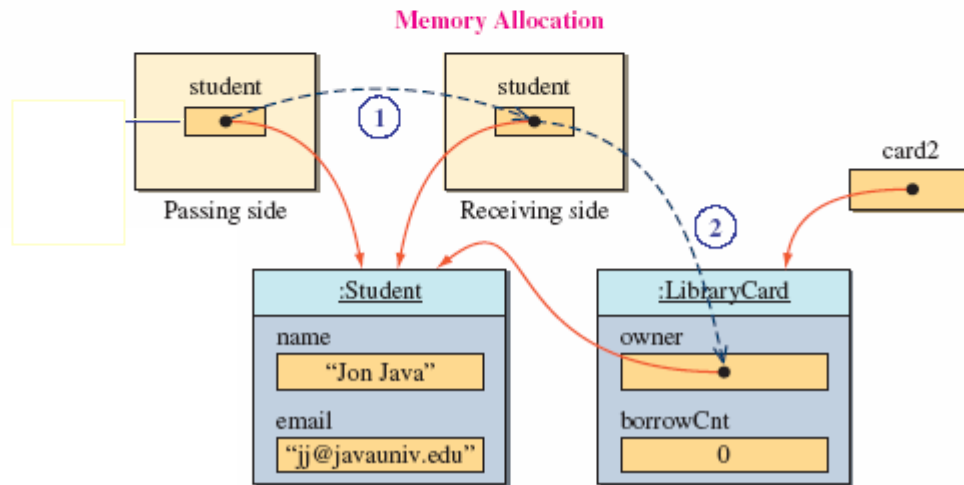
Passing an Object to a Setter

```
LibraryCard card2;     Passing side

card2 = new LibraryCard( );

card2.setOwner(student);
```

①

```
class LibraryCard {

 public void setOwner(Student student) {
    owner = student; ②
 }
}                              Receiving side
```

Memory Allocation

Using Setters and sharing the same Object

- The same Student object reference is passed to card1 and card2 using setters

```
Student      student;
LibraryCard card1, card2;

student = new Student( );
student.setName('Jon Java');
student.setEmail("jj@javauniv.edu");

card1 = new LibraryCard( );
card1.setOwner(student);
card1.checkOut(3);

card2 = new LibraryCard( );
card2.setOwner(student); //the same student is the owner
                         //of the second card, too
```
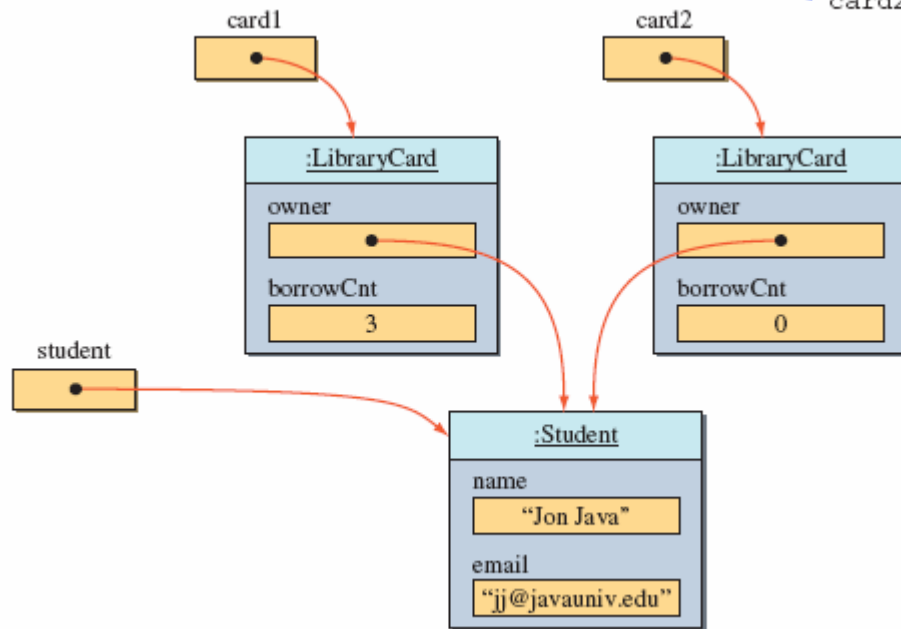
- Since we are actually passing the same object reference, it results in the owner of two LibraryCard objects referring to the same Student object

Class Constructors

- A class is a blueprint or prototype from which objects of the same type are created.

- Constructors define the initial states of objects when they are created.
  - *ClassName x = new ClassName();*

- A class contains at least one constructor.

- A class may contain more than one constructor.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP

The Default Class Constructor

- If no constructors are defined in the class, the default constructor is added by the compiler at compile time.

- The default constructor does not accept parameters and creates objects with empty states.
  - *ClassName x = new ClassName();*

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Class Constructors Declaration

```
public   <constructor name>   ( <parameters>   ){

       <constructor body>

}
```

- The ***constructor name***: a constructor has the name of the class .

- The ***parameters*** represent values that will be passed to the constructor for initialize the object state.

- Constructor declarations look like method declarations—except that they use the name of the class and have no return type.
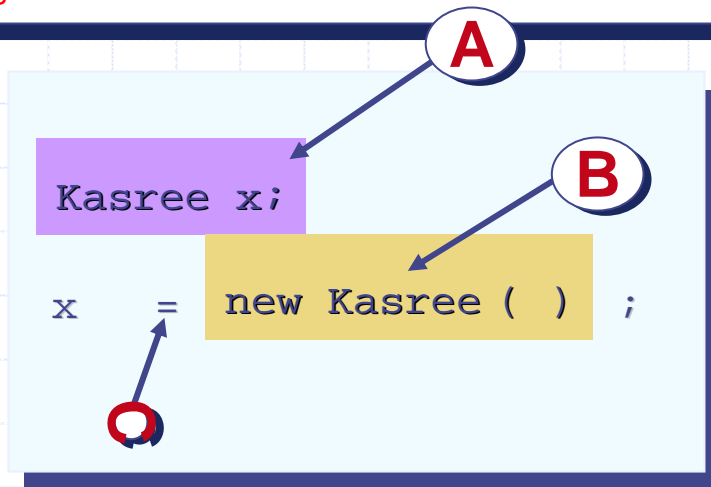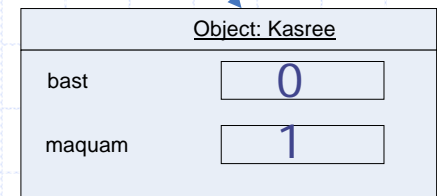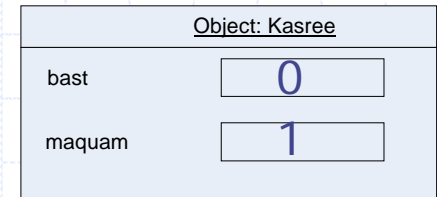
Example of
a Constructor with No-Parameter

Code

```
public class Kasree {

    private int bast;

    private int maquam;

    public Kasree() {

        bast = 0; maquam =1;
    }
    . . .
}
```

A

B

```
Kasree x;

x    =    new Kasree ( )   ;
```

C

State of Memory

A. The instance variable is allocated in memory.

X

B. The object is created with initial state

Object: Kasree	
bast	0
maquam	1

C. The reference of the object created in B is assigned to the variable.

X

Object: Kasree	
bast	0
maquam	1

Dr. S. GANNOUNI & Dr. A. TOUIR

Introduction to OOP

Class with Multiple Constructors

```java
public class Kasree {
    private int bast;
    private int maquam;

    public Kasree() {
        bast = 0; maquam =1;
    }
    public Kasree(int a, int b) {
        bast = a;
        if (b != 0) maquam = b;
        else maquam = 1;
    }
    . . .
}
```
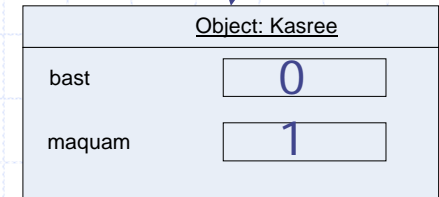
Code

```
Kasree x , y;

x   =  new Kasree()

y   = new Kasree(4, 3);
```
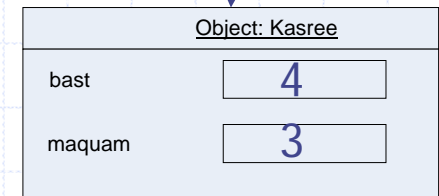
A

B

A. The constructor declared with no-parameter is used to create the object

x

Object: Kasree

bast | 0

maquam | 1

B. The constructor declared with parameters is used to create the object

y

Object: Kasree

bast | 4

maquam | 3

State of Memory

Overloading

- Two of the components of a method declaration comprise the *method signature:*
    - the method's name
    - and the parameter types.
    - The signature of the constructors declared above are:
        - Kasree()
        - Kasree(int, int)

- *overloading* methods allows implementing different versions of the same method with different *method signatures.*
    - This means that methods within a class can have the same name if they have different parameter lists.

Overloading (cont.)

- Overloaded methods are differentiated by:
    - the number,
    - and the type of the arguments passed into the method.
- You cannot declare more than one method with:
    - the same name,
    - and the same number and type of parameters.
- The compiler does not consider return type when differentiating methods.
    - No declaration of two methods having the same signature even if they have a different return type.

Dr. S. GANNOUNI & Dr. A. TOUIR   Introduction to OOP