Department of Computer Science
Faculty of Physical Sciences
Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB06

Objectives:

- Learn how to **overloads** constructors and methods
- Learn how to use the **this** keyword
- Learn how to write and use the **toString()** method

1. Oveloading

Overloading means having more than one constructor in a class or having more than one method with the same name in a class. In the case of constructor, the purpose of overloading is to allow the user to have as many options as possible when creating an object of the class, thus making the class more flexible to use.
In the case of methods, overloading alows the same name to be used for methods that performs similar tasks – Imagine having two plus operators, one for integer addition and another for double addition!
The condition for overloading is that the overloaded methods and/or constructors must have different signatures. The signature of a constructor is determined by the **number**, the **type** and the **order** of its parameters.

Example 1: The following example implements the Employee class. Notice how the constructors are overloaded. Also notice the overloading of the deductions methods.

```
public class Employee1 {
   private int iDNumber;
   private String name;
   private double salary;



   public Employee1(int iD, String employeeName, double employeeSalary) {
      iDNumber = iD;
      name = employeeName;
      salary = employeeSalary;
   }
   public Employee1(String employeeName, int iD, double employeeSalary) {
      iDNumber = iD;
      name = employeeName;
      salary = employeeSalary;
   }
   public Employee1(int iD, String employeeName) {
      iDNumber = iD;
      name = employeeName;
      salary = 0.0;
   }
   public Employee1(String employeeName, int iD) {
      iDNumber = iD;
      name = employeeName;
      salary = 0.0;
   }
   public void setSalary(double employeeSalary) {
      salary = employeeSalary;
   }
   public int getIDNumber() {
      return iDNumber;
   }
```

```
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public void deductions(double telephoneBills) {
        salary -= telephoneBills;
    }
    public void deductions(double telephoneBills, double medicalBills) {
        salary -= (telephoneBills + medicalBills);
    }
    public void raiseSalary(double percentIncrease) {
        salary += salary * percentIncrease/100;
    }
    public void printDetails() {
        System.out.println("\nID Number: "+iDNumber+"\nName: "+name+"\nSalary:
"+salary);
    }
}
```

The following shows how the constructors and the overloaded methods in the above example may be used:

```
import java.util.Scanner;
public class TestEmployee1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number;
        String name;
        double salary;

        System.out.print("Enter Name for Employee 1: ");
        name = input.nextLine();
        System.out.print("Enter ID Number for Employee 1: ");
        number = input.nextInt();
        System.out.print("Enter Salary for Employee 1: ");
        salary = input.nextDouble();

        //any of the following constructors be used to create the object
        Employee1 emp1 = new Employee1(number, name, salary);
// or    Employee1 emp1 = new Employee1(name, number, salary);
        System.out.print("\nEnter Name for Employee 2: ");
        name = input.nextLine();
        System.out.print("Enter ID Number for Employee 2: ");
        number = input.nextInt();

        //if we do not know the salary, we can use one of the following constructors
        Employee1 emp2 = new Employee1(number, name);
//or   Employee1 emp2 = new Employee1(name, number);
        emp2.setSalary(emp1.getSalary());
        emp1.deductions(50);
        emp2.deductions(60, 40);
        emp1.printDetails();
        emp2.printDetails();
    }
}
```

2. The *this* keyword

this is an implicit reference variable (i.e variable that doesn't need to be declared) that refers to the current object. At the point of defining the class, it is used for two purposes as follows:

- To refer to the instance variables of the class, especially when their names happen to be the same with parameters or local variables of a method or constructor.
- To call a constructor from within another constructor of the same class.

The advantages of using ***this*** as can be seen from the following example is that the program becomes shorter and that we do not have to think of different names for the parameters of constructors and methods.

Example 2: The following example modifies the above by using the ***this*** keyword

```
public class Employee2 {
   private int iDNumber;
   private String name;
   private double salary;

   public Employee2(int iDNumber, String name, double salary) {
      this.iDNumber = iDNumber;
      this.name = name;
      this.salary = salary;
   }
   public Employee2(String name, int iDNumber, double salary) {
      this(iDNumber,name,salary);
   }
   public Employee2(int iDNumber, String name) {
      this(iDNumber, name, 0.0);
   }

   public Employee2(String name, int iDNumber) {
      this(iDNumber, name, 0.0);
   }
   public void setSalary(double salary) {
      this.salary = salary;
   }
    public int getIDNumber() {
      return iDNumber;
   }
    public String getName() {
      return name;
   }
    public double getSalary() {
      return salary;
   }
   public void deductions(double telephoneBills) {
      salary -= telephoneBills;
   }
   public void deductions(double telephoneBills, double medicalBills) {
      salary -= (telephoneBills + medicalBills);
    }
   public void raiseSalary(double percentIncrease) {
      salary += salary * percentIncrease/100;
   }
   public void printDetails() {
      System.out.println("\nID Number: "+iDNumber+"\nName: "+name+"\nSalary:
"+salary);
   }
}
```

To test the above class, you need to run the file **TestEmployee2.java** which is the same as **TestEmployee1.java** except of the creation of objects of type **Employee2** class instead of objects of class **Employee1**

3. The *toString()* method

Sometimes we would like to print the values of the instance variable of an object. One way of doing this is to have a method such as the ***printDetails()*** method in the above examples. However, this is not a good idea since in java, there are different output targets such as graphical windows which uses ***drawString()*** method of Graphics/Graphics2D object or output files which require a different object instead of System.out.

Thus, what is normally done is to provide a ***toString()*** method which returns a representation of the object as a String. The application can then use the string returned by this method as it wishes.

the name ***toString()*** is very special in that it does not need to be called explicitly like other methods. The java system automatically calls the ***toString()*** method whenever the object reference variable is used in an operation that requires a string.

Example 3: The file **Employee3.java** is the same as **Employee2.java** except for the replacement of the printDetails() method with toString() method.

```
public String toString() {
   return "\nID Number: "+iDNumber+"\nName: "+name+"\nSalary: "+salary;
}
```

The file **TestEmployee3.java** accordingly modifies **TestEmplyee2.java** to print the two employee objects by implicitly calling the *toString()* method as follows.

```
. . . .
System.out.println(emp1);
System.out.println(emp2);
. . .
```

4. Assignment.

1. Download the folder **lab06.**

   a) Open the files Employee1.java and TestEmployee1.java, study then to understand what each is doing, then compile and execute TestEmployee1.java.

   b) Try the alternative constructor calls in the TestEmployee1.java by removing the comments on them and commenting the lines before each of them, then compile and execute the program again. You should notice no difference.

   c) Open the files Employee2.java and TestEmployee2.java, study them, then compile and execute TestEmployee2.java.

   d) Open the files Employee3.java and TestEmployee3.java, study them, then compile and execute TestEmployee3.java.

2. (a) Design a class called *Author* that contains the following.

(i) Three private instance variables: *name (String)*, *email (String)*, and *gender (char of either 'm' or 'f')*.

(ii) One constructor to initialize name, email and gender with given values.

(iii) Public getters and setters: *getName()*, *getEmail()*, *setEmail()*, and *getGender()*.

There are no setters for name and gender, as these attributes cannot be changed.

(iv) A *toString()* method that returns "*author-name (gender) at email*", eg, "*Aliyu Garba (m) at galiyu@abu.edu.ng*".

(b) Write a test program called *TestAuthor* to test the constructor and the public methods. Try changing the email of an author.

3. Modify the Box.java and BoxDemo.java of Example 1 of lab04 as follows:

a) add methods *getLength()*, *getWidth()* and *getHeight()* that returns the length, width and height of the box object.

b) add another constructor which receives only the length. It then calls the other constructor supplying this length for all of length, width and height (i.e. it forms a cube).

c) add another constructor that receives another Box object as parameter and then uses its length, width and height to initialize the current box object (i.e it creates a box object with same dimension as the one it receives as parameter)

d) add a *toString()* method that returns the langth, width, and height as a string that prints on one line as shown in the figure below.

The class BoxDemo.java should be modified to create three Box objects using each of the three constructors and then prints it (Note: no need to read input). Use the first box object to create the second.