

---

## **Lab 07**

### **Method Overriding and Abstract Classes**

#### **Objective:**

The objective of this lab is to familiarize the students with various concepts and terminologies of method overriding and concept of abstract classes.

#### **Activity Outcomes:**

This lab teaches you the following topics:

- Method overriding where a base class method version is redefined in the child class with exact method signatures.
- Abstract classes along with the access of base class members.

#### **Instructor Note:**

As pre-lab activity, read Chapter 11 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

---

## 1) Useful Concepts

### a. Method Overriding

The definition of an inherited method can be changed in the definition of a derived class so that it has a meaning in the derived class that is different from what it is in the base class. This is called overriding the definition of the inherited method.

For example, the methods `toString` and `equals` are overridden (redefined) in the definition of the derived class `HourlyEmployee`. They are also overridden in the class `SalariedEmployee`. To override a method definition, simply give the new definition of the method in the class definition, just as you would with a method that is added in the derived class.

In a derived class, you can override (change) the definition of a method from the base class. As a general rule, when overriding a method definition, you may not change the type returned by the method, and you may not change a void method to a method that returns a value, nor a method that returns a value to a void method. The one exception to this rule is if the returned type is a class type, then you may change the returned type to that of any descendent class of the returned type. For example, if a function returns the type `Employee`, when you override the function definition in a derived class, you may change the returned type to `HourlyEmployee`, `SalariedEmployee`, or any other descendent class of the class `Employee`. This sort of changed return type is known as a covariant return type and is new in Java version 5.0; it was not allowed in earlier versions of Java.

### b. Abstract Class

A class that has at least one abstract method is called an abstract class and, in Java, must have the modifier `abstract` added to the class heading. An abstract class can have any number of abstract methods. In addition, it can have, and typically does have, other regular (fully defined) methods. If a derived class of an abstract class does not give full definitions to all the abstract methods, or if the derived class adds an abstract method, then the derived class is also an abstract class and must include the modifier `abstract` in its heading.

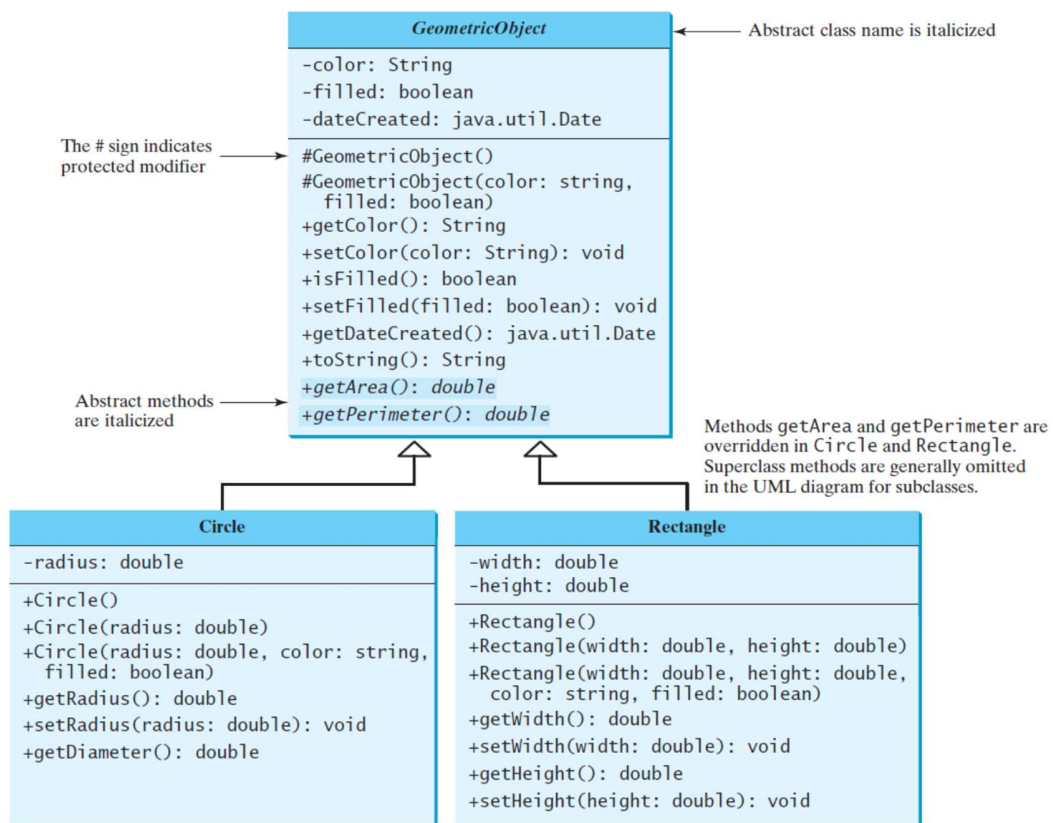
In contrast with the term abstract class, a class with no abstract methods is called a concrete class.

```
public abstract class GeometricObject
{
    private instanceVariables;
    . . .
    public abstract double getArea();
}
```

```

public abstract double getPerimeter();
...
}

```



## 2) Solved Lab Activities (Allocated Time 1 Hr.)

Sr.No	Allocated Time	Level of Complexity	CLO Mapping
Activity 1	10 mins	Medium	CLO-4
Activity 2	20 mins	Medium	CLO-4
Activity 2	20 mins	Medium	CLO-4

---

## Activity 1:

*The following activity demonstrates the creation of overridden methods.*

### Solution:

```
class A
{
    int i, j;

    A(int a, int b) { i = a; j = b; }

    // display i and j
    void show() {
        System.out.println("i and j: " + i + " " + j);
    }
}

-----

class B extends A
{
    int k;
    B(int a, int b, int c) { super(a, b); k = c; }

    // display k - this overrides show() in A
    void show() {
        System.out.println("k: " + k);
    }
}

-----

Public class OverrideRunner
{
    public static void main(String args[])
    {
        B subOb = new B(1, 2, 3); subOb.show(); // this calls show() in B
    }
}
```

---

## Output

k: 3

## Activity 2:

*The following activity explains the use of overriding for customizing the method of super class. The classes below include a CommisionEmployee class that has attributes of firstname, lastName, SSN, grossSales, CommisionRate. It has a constructor to initialize, set and get functions, and a function to display data members.*

*The other class BasePlusCommisionEmployee is inherited from CommisionEmployee. It has additional attributes of Salary. It also has set and get functions and display function.*

*The Earning method is overridden in this example.*

```
public class commissionEmployee
{
protected String FirstName; protected String LastName; protected
String SSN; protected double grossSales; protected double commonRate;

public commissionEmployee()
{
FirstName="Nagina"; LastName="Nazar"; SSN="S003";
grossSales=1234.1; commonRate=12.5;
}

public commissionEmployee (String a,String e,String b, double c,
double d){ FirstName=a;
LastName=e; SSN=b;
grossSales=c; commonRate=d;
}

public void setFN(String a){ FirstName=a;}
public void setLN(String e){ LastName=e;}
public void setSSN(String b){ SSN=b;}
public void setGS(double c){ grossSales=c;}
public void setCR(double d){ commonRate=d;}
public String getFN(){return FirstName;}
```

---

```
public String getSSN(){return SSN;}
public double getGS(){return grossSales;}
public double getCR(){return commonRate;}

public double earnings(){
return grossSales*commonRate;
}

public void display(){
System.out.println("first name:"+FirstName+"last name:"
+LastName+"SSN:"+SSN+" Gross Sale:"+grossSales+" and
commonRate:"+commonRate);
}
}

-----
public class BasePlusCommEmployee extends commissionEmployee
{
private double salary;

BasePlusCommEmployee(){ salary=48000; }

BasePlusCommEmployee(String A,String E,String B, double C, double D,
double S)
{
super(A,E,B,C,D);
salary=S;
}
//overridden method
public double earnings()
{
return super.earnings()+salary;
}

public void display(){
super.display();
System.out.println("Salary : "+salary);
}
}
```

---

```
}

Public class OverrideRunner
{
public static void main(String args[])
{
BasePlusCommEmployee b = new BasePlusCommEmployee("ali", "ahmed",
"25-kkn", 100, 5.2, 25000);
double earn = b.earnings();

System.out.println("Earning of employee is " + earn);
}
}
```

## Output

Earning of employee is 25520.0

## Activity 3:

*A Simple demonstration of abstract.*

```
public abstract class A {
abstract void callme();
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
} }

-----

public class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
} }

-----

public class AbstractDemo {
public static void main(String[] args) {
B b = new B();
b.callme(); b.callmetoo(); } }
```

---

## Output

B's implementation of Call me. This is a concrete method

## Graded Lab Tasks( Allocated Time 2 Hr.)

**Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.**

### Lab Task 1

Create a class named *Movie* that can be used with your video rental business. The *Movie* class should track the Motion Picture Association of America (MPAA) rating (e.g., Rated G, PG-13, R), ID Number, and movie title with appropriate accessor and mutator methods. Also create an *equals()* method that overrides *Object*'s *equals()* method, where two movies are equal if their ID number is identical. Next, create three additional classes named *Action*, *Comedy*, and *Drama* that are derived from *Movie*. Finally, create an overridden method named *calcLateFees* that takes as input the number of days a movie is late and returns the late fee for that movie. The default late fee is \$2/day. Action movies have a late fee of \$3/day, comedies are \$2.50/day, and dramas are \$2/day. Test your classes from a main method.

### Lab Task 2

Write a program that declares two classes. The parent class is called *Simple* that has two data members *num1* and *num2* to store two numbers. It also has four member functions.

The *add()* function adds two numbers and displays the result. The *sub()* function subtracts two numbers and displays the result.

The *mul()* function multiplies two numbers and displays the result. The *div()* function divides two numbers and displays the result.

The child class is called *VerifiedSimple* that overrides all four functions. Each function in the child class checks the value of data members. It calls the corresponding member function in the parent class if the values are greater than 0. Otherwise it displays error message.

### Lab Task 3

Create an abstract class that stores data about the shapes e.g. Number of Lines in a Shape, Pen Color, Fill Color and an abstract method *draw*. Implement the method *draw* for *Circle*, *Square* and *Triangle* subclasses, the better approach is to draw these figures on screen, if you can't then just use a display message in the *draw* function.