Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No.04

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Constructor overloading and method overriding

AIM: Create an abstract **class SalaryAccount** with an **abstract method bank_name()** and normal method status_employee(). Create a class PermEmployee with 3 constructors default constructor, parametrized constructor with **2 arguments name, id** and another parametrized constructor with **3 arguments name, id and salary**. **Class PermEmployee** should extend SalaryAccount and define bank_name() that displays name of bank of Permanent employee. Override status_employee() and display employee is permanant. Create another **class TempEmployee** with 1 constructor which initializes name, id and salary also override bank_name() where bank name is different from permanent employee. Implement status_employee() where status should be displayed temporary. Display the details of 2 classes of employee using **constructor overloading** and **dynamic method dispatch** from another **class Test**. Draw class Diagram for the same. Clearly show attributes multiplicities and aggregations/compositions/Association between classes in the class diagram.

Expected OUTCOME of Experiment:

CO1: Understand the basic object oriented concept.

CO3: Implement scenarios using object oriented concepts(Drawing class diagram, relationship between classes)

Books/ Journals/ Websites referred:

1.Ralph Bravaco , Shai Simoson , “Java Programing From the Group Up” Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design .

Pre Lab/ Prior Concepts:

Constructors:
Constructors are used to initialize the object's state. Like methods, a constructor also contains **collection of statements (i.e. instructions)** that are executed at time of Object creation. A constructor has no return type.
Overloading:
Overloading allows different methods to have same name, but different signatures where signature can differ by number of input parameters or type of input parameters or both. Overloading is related to compile time (or static) polymorphism.
Constructors Overloading:
In addition to overloading methods, we can also overload constructors in java. Overloaded constructor is called based upon the parameters specified when new is executed. Sometimes, there is a need of initializing an object in different ways. This can be done using constructor overloading example:

```
class Box
{
    double width, height, depth;

    // constructor used when all dimensions
    // specified
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }

    // constructor used when no dimensions
    // specified
    Box()
    {
        width = height = depth = 0;
    }

    // constructor used when cube is created
    Box(double len)
    {
        width = height = depth = len;
    }
}
```

Method Overriding:
In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type (or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.

Method overriding is one of the way by which java achieve Run Time Polymorphism.The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed.

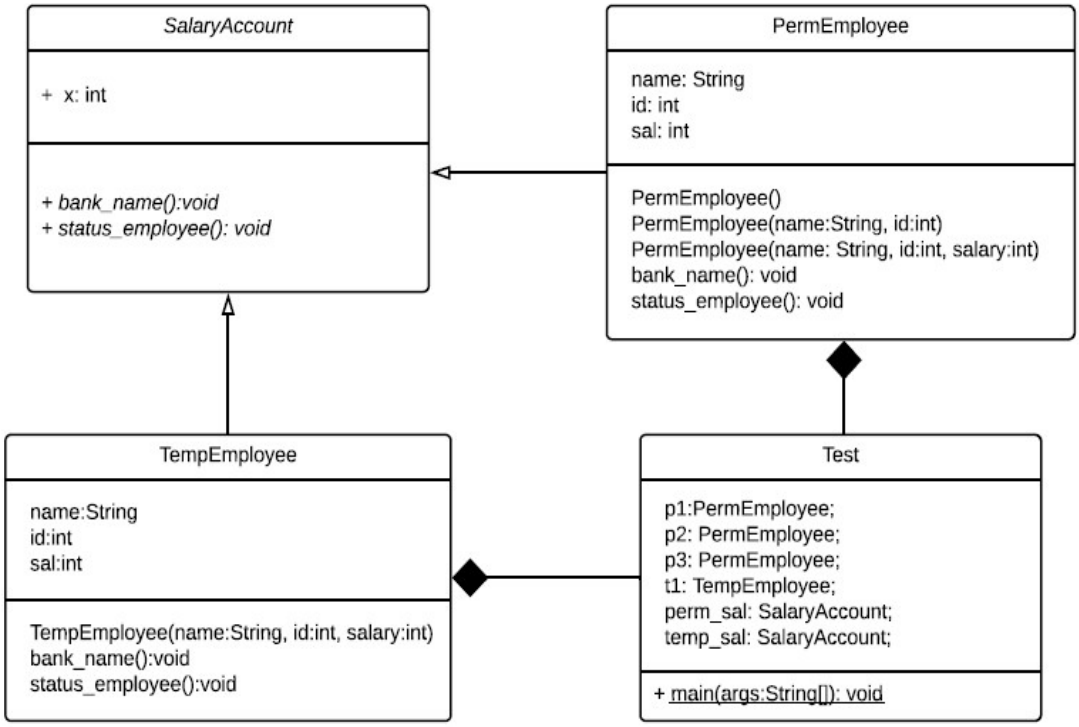
```
// Base Class
class Parent
{
    void show() { System.out.println("Parent's show()"); }
}

// Inherited class
class Child extends Parent
{
    // This method overrides show() of Parent
    @Override
    void show() { System.out.println("Child's show()"); }
}

// Driver class
class Main
{
    public static void main(String[] args)
    {
        // If a Parent type reference refers
        // to a Parent object, then Parent's
        // show is called
        Parent obj1 = new Parent();
        obj1.show();

        // If a Parent type reference refers
        // to a Child object Child's show()
        // is called. This is called RUN TIME
        // POLYMORPHISM.
        Parent obj2 = new Child();
        obj2.show();
    }
}
```

Class Diagram:



Algorithm:

STEP 1: START.
STEP 2: Declare an abstract class SalaryAccount and declare abstract method bank_name() and normal method status_employee().
STEP 3: Create a class PermEmployee and inherit SalaryAccount.
STEP 4: Define and declare constructors (parameterized and non-parameterized) and then override method status_employee() and define bank_name(), display().
STEP 5: Create a class TempEmployee and inherit SalaryAccount.
STEP 6: Define and declare constructors (parameterized) and then override method status_employee() and define bank_name(), display().
STEP 7: In class Main, create objects of class PermEmployee and TempEmployee and call the methods to see method overriding.
STEP 8: For upcasting, create reference variables of SalaryAccount type and call the methods of PermEmployee and TempEmployee classes.
STEP 9: END.

Implementation details:

```
import java.util.*;
import java.lang.*;
abstract class SalaryAccount{
  abstract void bank_name();
  int x=100;
  public void status_employee(){
    System.out.println("This is status_employee of SalaryAccount
class");
  }
}
class PermEmployee extends SalaryAccount{
String name;
int id;
int sal;
int x=50;
PermEmployee(){
//System.out.println("This is default constructor.");
name="Deep Dama";
id=70;
sal=100;
}
PermEmployee(String name, int id){
this.name=name;
this.id=id;
sal=50000;
}
PermEmployee(String name, int id, int sal){
this.name=name;
this.id=id;
this.sal=sal;
}
void display(){
  System.out.println("For PermEmployee class: ");
  System.out.println("Name: "+name+"\nID: "+id+"\nSalary:
"+sal);
}

  public void status_employee(){
      System.out.println("This is status_employee of
PermEmployee class.");
  }
  public void bank_name(){
```

```
    System.out.println("This is bank_name of PermEmployee
class.");
  }
}
class TempEmployee extends SalaryAccount{
  String name;
  int id, sal;
  int x=10;
  TempEmployee(String name, int id, int sal){
    this.name=name;
    this.id=id;
    this.sal=sal;
  }
  public void status_employee(){
      System.out.println("This is status_employee of
TempEmployee class.");
  }
  public void bank_name(){
    System.out.println("This is bank_name of TempEmployee
class.");
  }
  void display(){
    System.out.println("For TempEmployee class: ");
    System.out.println("Name: "+name+"\nID: "+id+"\nSalary:
"+sal);
  }
}
class Main{
public static void main(String[] args){
PermEmployee p1=new PermEmployee();
p1.display();
p1.bank_name();
p1.status_employee();
System.out.println("Printing value of x from PermEmployee class:
"+p1.x);
PermEmployee p2=new PermEmployee("Arghyadeep", 72);
p2.display();
p2.bank_name();
p2.status_employee();
System.out.println("Printing value of x from PermEmployee class:
"+p2.x);
PermEmployee p3=new PermEmployee("Gaurang", 65, 100000);
p3.display();
p3.bank_name();
```

```
p3.status_employee();
System.out.println("Printing value of x from PermEmployee class:
"+p3.x);
TempEmployee t1=new TempEmployee("Gaurav", 66, 10000);
t1.display();
t1.bank_name();
t1.status_employee();
System.out.println("Printing value of x from TempEmployee class:
"+t1.x);
SalaryAccount perm_sal=new PermEmployee(); //upcasting
SalaryAccount temp_sal=new TempEmployee("Kaustubh Damania", 71,
200000); //upcasting
System.out.println("Now we will see dynamic method dispatch for
PermEmployee: ");
//perm_sal.display();  //this will throw error as we cannot call
child class method with parent class reference.
perm_sal.status_employee();
perm_sal.bank_name();
System.out.println("Printing value of x from Abstract class:
"+perm_sal.x);
System.out.println("Now we will see dynamic method dispatch for
TempEmployee: ");
//temp_sal.display();  //this will throw error as we cannot call
child class method with parent class reference.
temp_sal.status_employee();
temp_sal.bank_name();
System.out.println("Printing value of x from Abstract class:
"+temp_sal.x);
  }
}
```

For verification, the code can be found at:
https://repl.it/@ARGHYADEEPDAS/EmployeeJava

Output Screen:

```
For PermEmployee class:
Name: Deep Dama
ID: 70
Salary: 100
This is bank_name of PermEmployee class.
This is status_employee of PermEmployee class.
Printing value of x from PermEmployee class: 50
For PermEmployee class:
Name: Arghyadeep
ID: 72
Salary: 50000
This is bank_name of PermEmployee class.
This is status_employee of PermEmployee class.
Printing value of x from PermEmployee class: 50
For PermEmployee class:
Name: Gaurang
ID: 65
Salary: 100000
This is bank_name of PermEmployee class.
This is status_employee of PermEmployee class.
Printing value of x from PermEmployee class: 50
For TempEmployee class:
Name: Gaurav
ID: 66
Salary: 10000
This is bank_name of TempEmployee class.
This is status_employee of TempEmployee class.
Printing value of x from TempEmployee class: 10
```

```
Now we will see dynamic method dispatch for PermEmployee:
This is status_employee of PermEmployee class.
This is bank_name of PermEmployee class.
Printing value of x from Abstract class: 100
Now we will see dynamic method dispatch for TempEmployee:
This is status_employee of TempEmployee class.
This is bank_name of TempEmployee class.
Printing value of x from Abstract class: 100
```

Conclusion: Constructor overloading, method overriding and dynamic method dispatch were implemented successfully.

Date: 24/08/2018 **Signature of faculty in-charge**