National University of Computer & Emerging Sciences,
Karachi
Computer Science Department
Spring 2023, Lab Manual – 07

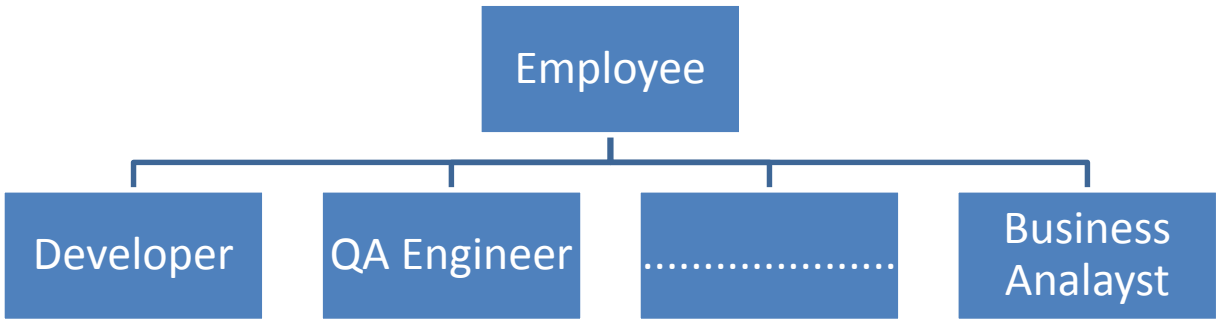
Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Abeer Gauher, Hajra Ahmed, Shafique Rehman

LAB - 7

Hierarchical inheritance & Polymorphism

Hierarchical Inheritance:

When two or more classes inherits a single class, it is known as hierarchical inheritance. The base class has attributes that are protected but are accessible from the subclass.



Example:

```
public class Employee {
    String name;
    String id;
    float salary;
    //default constructor
    Employee(){
        System.out.println("Create an employee");
    }
    void display(){
        System.out.println("display employee");
    }
    public static void main(String[] args) {
        QAEngineer qaEngineer = new QAEngineer(); //instantiate QAEngineer
        qaEngineer.display(); //call the method of employee class
        qaEngineer.display_QAEngineer();
        Developer developer = new Developer(); //instantiate developer
        developer.display(); // call the method of employee class
        developer.display_Developer();
    }
}
class Developer extends Employee{
    String department;
    Developer(){
        super();
        department = "development";
    }
    void display_Developer(){
        System.out.println("display developer");
    }
}
class QAEngineer extends Employee{
    String department;
    QAEngineer(){
        super();
        department = "Quality Assurance & Testing";
    }
    void display_QAEngineer(){
        System.out.println("display QA");
    }
}
```

Figure 1: hierarchial inheritance example

Polymorphism:

Polymorphism is derived from poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms. It is a concept by which we can perform a *single action in different ways*. There are two types of polymorphism in Java:

- compile-time polymorphism / static polymorphism / static binding
- runtime polymorphism/ dynamic polymorphism/ dynamic binding.

Static binding	**Dynamic binding**
Done at compile time	Done at run time
Actual object not used	Actual objects are used
Also early binding	Also, late binding
e.g., method overloading	e.g., method overriding

Static binding:

Early binding done at compile time is static binding. Here, in successive example I have shown that (in red box) when an employee object is called, all its methods are bind with the functions call. So, when I call display function using e1 that will show the output of the Employee class as the reference object is bind with the member function of employee class.

Dynamic binding:

Late binding or runtime binding is the concept in which the JVM binds the function at runtime with their definition so any overridden method will have newer definition at runtime. As in the example (in blue box), we can see that the reference object e2 binds the display method of Developer class with the e2's display function call

```
class Employee {
    //overridden method
    void display(){
        System.out.println("display employee");
    }
}
class Developer extends Employee{
    //overriding method
    @Override //annotation
    void display(){ //newer implementation
        System.out.println("displaying developer");
    }
}
public class MAIN{
    public static void main(String[] args) {
     //reference and object binding both of base class
        Employee e1=new Employee();
        e1.display();
     //reference of base class and object of child class is binded
        Employee e2=new Developer();
        e2.display();
    }
}
```

```
"C:\Program Files\Java\jdk-17.0.2\bin
display employee
displaying developer
```

Figure 2: static vs dynamic binding e.g.

Method Overloading:

A feature that allows a class to have more than one method having the same name but with different argument list. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

In order to overload a method, the argument lists of the methods must differ using either of the given options:

- Different number of parameters.
- Different data type of parameters
- Sequence of data type of parameters.

Methods with different return types doesn't matter in case of overloading.

Figure 1 shows the overloading of constructor. We can see that; the function names are same but only the argument list varies in both the cases. Same goes for any other function in which return type and function remains same. Only the argument lists will change. It is like making multiple versions of a functions in a class.

```
public class Employee {
    String name;
    String id;
    float salary;
    //default constructor (no arguments)
    Employee() {
        System.out.println("version 1 of a constructor");
        System.out.println("Create an employee");
    }
    //parameterized constructor (arguments)
    Employee(String n, String i, float f) {
        System.out.println("version 2 of a constructor");
        name = n;
        id = i;
        salary = f;
    }
}
```

Figure 3: constructor overloading

Method Overriding:

Two methods having exactly same signature can have different definitions when they are in different classes. This is the concept of overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.

In this case the method in parent class is called **overridden method** and the method in child class is called **overriding method.**

Some important points to note are:

- Annotation (i.e., @override) for overriding may or may not be used as IntelliJ allowed it without using annotation. Annotations provide additional information about a program. Annotations have no direct effect on the functioning of the code they annotate
- We **cannot override the static methods** as overriding is the concept of dynamic binding and static methods are bind using static binding at compile time.

```
class Employee {
    String name;
    String id;
    float salary;
    //overridden method
    void display(){
        System.out.println("display employee");
    }
}
class Developer extends Employee{
    String department;
    Developer(){
        super();
        department = "development";
    }
    //overriding method
    @Override //annotation
    void display(){ //newer implementation
        System.out.println("displaying developer");
        // calling overidden method of base class using super
        super.display();
    }
}
public class MAIN{
    public static void main(String[] args) {
        Developer developer = new Developer(); //instantiate developer
        developer.display();
       // developer.display();
    }

}
```

```
"C:\Program Files\Java\jdk-17.0.2\bi
displaying developer
display employee

Process finished with exit code 0
```

Figure 4:method overriding

TASK 1:

Create a Java program that implements a class hierarchy for a library system. The hierarchy should include the following classes:

- Item: a base class that represents an item in the library and has attributes such as title, author, and year. It also defines a method toString that returns a string representation of the item's attributes.
- Book: a subclass of Item that represents a book and has additional attributes such as publisher and ISBN. It also overrides the toString method to include the book's publisher and ISBN in the string representation.
- Magazine: a subclass of Item that represents a magazine and has additional attributes such as publisher and issueNumber. It also overrides the toString method to include the magazine's publisher and issue number in the string representation.
- DVD: a subclass of Item that represents a DVD and has additional attributes such as director and length. It also overrides the toString method to include the DVD's director and length in the string representation.

Your program should include appropriate constructors for each class.
It should create instances of each subclass and demonstrate how to call their methods to retrieve their attributes and behavior, including the overridden toString method.

Task 2:

Create a Java program that implements a class hierarchy for a university system. The hierarchy should include the following classes:

- Person: a base class that represents a person and has attributes such as name, email, and phone. It also defines a method toString that returns a string representation of the persons attributes.
- Student: a subclass of Person that represents a student and has attributes such as studentId and major.
- It also defines a method getGPA that returns the students grade point average.
- Faculty: a subclass of Person that represents a faculty member and has attributes such as facultyId and department. It also defines a method getRank that returns the faculty members rank (e.g., professor, assistant professor, etc.).
- Staff: a subclass of Person that represents a staff member and has attributes such as staffId and jobTitle.
- It also defines a method getSalary that returns the staff members salary.

Your program should include appropriate constructors for each class.

Task 3:

Create a class named Robot that will input:

- the x and y coordinates of a Robot and the direction in which he wants to move using parameterized constructor. (direction could be E,W,N,S)
- Create a method to display the intial position of the robot.

Create another class named Moving Robot inherited from robot

- This class have a function named moveRobot; function will take steps to move as argument and move robot in that direction.
  For example, if initially the direction = N and Y = 2, and user entered the steps=3, so after movement the updated coordinates are; Y = 5. (Since moving in North will update the +Y in Quadrant System)
- Create a display method to show the updated X and Y coordinates.

Task 4:

Crete a polymorphic banking application.
The application contains a class named **Accounts**. Two classes named saving account and Checking account inherit it. The account class have:

- an attribute named balance
- a member function named debit to withdraw amount from the account.
- a member function named credit function to deposit a particular amount.
- A getBalance function to show current balance.

Create a class named SavingAccount that contains:

- An attribute named timeSpan (to show the time passed over money saved in the account)
- A calculateIntereset method to calculate interest over the current balance amount
  - Formula for that is Interest = (current Balance) * InterestRate * timespan
  - Consider whatever interestRate you want
- Override the credit method to add the interest to account balance.

After transaction processing print the updated account balance using getBalance function.

Task 5:
Create a Java program that implements a class named "Sorting" with the following methods:

- sort (int[] array) sorts the given integer array in ascending order.
- sort(String[] array) sorts the given string array in ascending order
- sort (int[] array, boolean descending) sorts the given integer array in either ascending or descending order, depending on the value of the boolean parameter.
- sort(String[] array, boolean descending) sorts the given string array in either ascending or descending order, depending on the value of the boolean parameter.

Create an instance of the "Sorting" class, populate some arrays with random values, and execute all four "sort" methods with appropriate arguments.