# National University of Computer & Emerging Sciences, Karachi



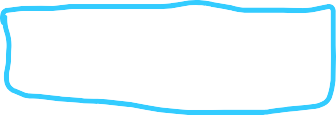
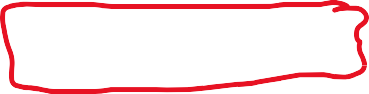
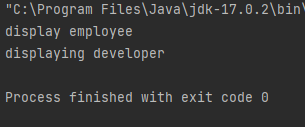
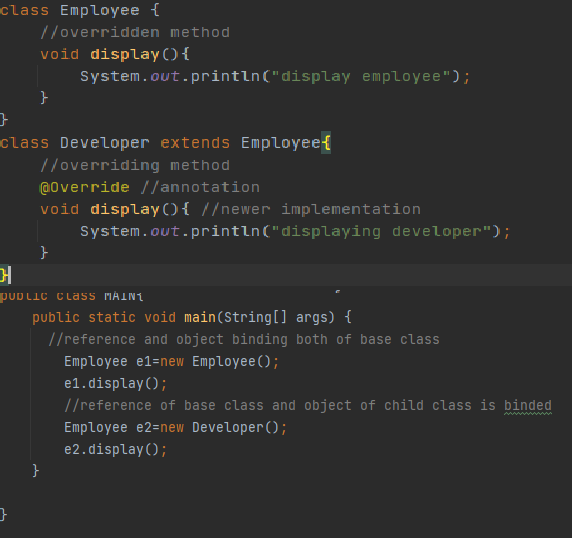
**Computer Science Department Spring 2024, Lab Manual – 07**

|  |  |
| --- | --- |
| **Course Code: CL-217** | **Course : Object Oriented Programming Lab** |
| **Instructor(s) :** | **Ali Fatmi** |

LAB - 7

Polymorphism

# Polymorphism:



*Figure 2: static vs dynamic binding e.g.*

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

# Method Overloading:

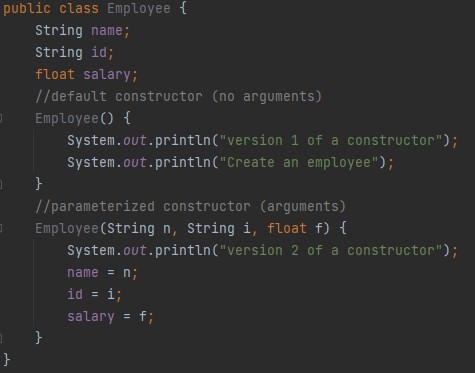
A feature that allows a class to have more than one method having the same name but with different argument list. It is similar to [constructor overloading](https://beginnersbook.com/2013/05/constructor-overloading/) in Java, that allows a class to have more than one constructor having different argument 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.

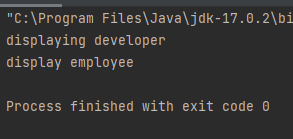
Methods with different return types doesn’t matter in case of overloading.

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.



*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.

*Figure 4:method overriding*

## TASK 1:

Write a Java program that has a class named as Employee.

* The class Employee has a default constructor that displays “I am an employee”.
* It has the attributes name, position, number of years, and salary. Initialize the salary to 30000.
* Input all the details from the user and display them using input and display function respectively.

Derive a class FulltimeEmployee.

* The class has a default constructor that displays “I am a full time employee in the company”.
* The class has an attribute increment initialized to 20%.
* The class has a function that displays the incremented salary of a full time employee.

Derive a class PartTimeEmployee.

* The class has a default constructor that displays “I am a part time employee in the company”.
* The class has an attribute increment initialized to 5%.
* The class has a function that displays the incremented salary of a part time employee.

In the main, display all the details as required.

## Task 2:

Write a Java program that has a class named as Vehicle.

* The class Vehicle has a default constructor that displays “It’s a vehicle”.
* It has the attributes VehicleName, VehicleNo and wheels.
* Create an accelerate method that will take speed as argument and displays the speed of the vehicle as km/hr.

Derive a class FourWheeler.

* Create a parameterized constructor that will initialize VehicleName, VehicleNo, wheels and PowerSteering.
* The class has a Boolean attribute power steering. If its trye then then vehicle has a power steering otherwise not.
* Create a display method to show the details of the vehicle and if PowerSteering is true, then print your car has a power steering otherwise print an appropriate message. Moreover, it also displays the spped at which the vehicle is operating.

In the main, create a fourwheeler object and call the display functions to display the details.

## 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 class named Adder

* Create a static add function that will add the two integers
* Create another static function that will add an array of as many integers as a person wants.
* Create a function named display that will display the sum of both the functions separately.