**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

**Object Oriented Analysis & Design (CS-309)**

**Muhammad Nadeem || Awais Ahmed || Tooba Ali**

[**Muhammad.nadeem@nu.edu.pk**](mailto:Muhammad.nadeem@nu.edu.pk) **||** [**awais.ahmed@nu.edu.pk**](mailto:awais.ahmed@nu.edu.pk) **||** [**tooba.ali@nu.edu.pk**](mailto:majid.hussain@nu.edu.pk)****

**Revision Object Oriented Concepts & Use Case Diagram**

**Lab Session # 04**

**Objectives:**

* **Hands-on Use Case Diagram**
* **To understand and Implement Object Oriented Concepts**

**Object Oriented Concepts:** OOPS Concepts or Object Oriented Programming Concepts are very important. Without having idea about OOPS concepts, you will not be able to design systems in object oriented programming model.

**Checklist:**

* **Abstraction**

Abstraction is the concept of hiding the internal details and describing things in simple terms. For example, a method that adds two integers. The method internal processing is hidden from outer world. There are many ways to achieve abstraction in object oriented programming, such as encapsulation and inheritance.

A java program is also a great example of abstraction. Here java takes care of converting simple statements to machine language and hides the inner implementation details from outer world.

* **Encapsulation**

Encapsulation is the technique used to implement abstraction in object oriented programming. Encapsulation is used for access restriction to a class members and methods.

Access modifier keywords are used for encapsulation in object oriented programming. For example, encapsulation in java is achieved using private, protected and public keywords.

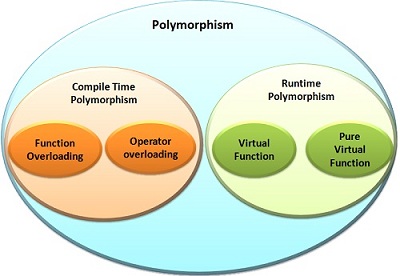
* **Inheritance**

Inheritance is the object oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called superclass and the object that inherits the superclass is called subclass.

* **Polymorphism**

Polymorphism is the concept where an object behaves differently in different situations.

**Types of polymorphism**

****

**Implementation guidelines**

**Compile time polymorphism** is achieved by method overloading. For example, we can have a class as below.

public class Circle {

public void draw(){

System.out.println("Drwaing circle with default color Black and diameter 1 cm.");

}

public void draw(int diameter){

System.out.println("Drwaing circle with default color Black and diameter"+diameter+" cm.");

}

public void draw(int diameter, String color){

System.out.println("Drwaing circle with color"+color+" and diameter"+diameter+" cm.");

}

}

Here we have multiple draw methods but they have different behavior. This is a case of method overloading because all the methods name is same and arguments are different. Here compiler will be able to identify the method to invoke at compile time, hence it’s called compile time polymorphism.

**Runtime polymorphism** is implemented when we have “IS-A” relationship between objects. This is also called as method overriding because subclass has to override the superclass method for runtime polymorphism. If we are working in terms of superclass, the actual implementation class is decided at runtime. Compiler is not able to decide which class method will be invoked. This decision is done at runtime, hence the name as runtime polymorphism or dynamic method dispatch.

public interface Shape {

public void draw();

}

public class Circle implements Shape{

@Override

public void draw(){

System.out.println("Drwaing circle");}}

public class Square implements Shape {

@Override

public void draw() {

System.out.println("Drawing Square");

}

}

**Shape** is the superclass and there are two subclasses **Circle** and **Square**. Below is an example of runtime polymorphism.

Shape sh = new Circle();

sh.draw();

Shape sh1 = getShape(); //some third party logic to determine shape

sh1.draw();

**Inheritance:**

We use extends keyword in java to implement inheritance. Below is a simple example of inheritance in java.

class SuperClassA {

public void foo(){

System.out.println("SuperClassA");

}}

class SubClassB extends SuperClassA{

public void bar(){

System.out.println("SubClassB");

}}

public class Test {

public static void main(String args[]){

SubClassB a = new SubClassB();

a.foo();

a.bar();

}}

**Encapsulation:**

public class Mobile {

private String manufacturer;

private String operating\_system;

public String model;

private int cost;

//Constructor to set properties/characteristics of object

Mobile(String man, String o,String m, int c){

this.manufacturer = man;

this.operating\_system=o;

this.model=m;

this.cost=c;

}

//Method to get access Model property of Object

public String getModel(){ return this.model;}

// We can add other method to get access to other properties

}

**Abstraction:**

public abstract class VehicleAbstract {

public abstract void start();

public void stop(){

System.out.println("Stopping Vehicle in abstract class");

}}

class TwoWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Two Wheeler");

}}

class FourWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Four Wheeler");

}}

public class VehicleTesting {

public static void main(String[] args) {

VehicleAbstract my2Wheeler = new TwoWheeler();

VehicleAbstract my4Wheeler = new FourWheeler();

my2Wheeler.start();

my2Wheeler.stop();

my4Wheeler.start();

my4Wheeler.stop();

}

}