

**Iqra National University**

**program BS Software Engineering Section B**

**Name: IJAZ KHAN**

**ID# 16764**

**Subject object oriented programming**

**Semester (2)**

**Submitted to M.Ayub sir .**

Date29/June/2020 Final term paper

Q1. a. Why access modifiers are used in java, explain in detail Private and Default

access modifiers?

Ans:

A Java access modifier specifies which classes can access a given class and its fields, constructors and methods. Access modifiers can be specified separately for a class, its constructors, fields and methods. Java access modifiers are also sometimes referred to in daily speech as Java access specifiers, but the correct name is Java access modifiers. Classes, fields, constructors and methods can have different Java access modifiers:

Examples: private double num = 100;

public double num = 100;

**Default access modifier:**

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class. Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package.

## Private access modifier:

The scope of private modifier is limited to the class only.

1. Private Data members and methods are only accessible within the class
2. Class and [Interface](https://beginnersbook.com/2013/05/java-interface/) cannot be declared as private
3. If a class has [private constructor](https://beginnersbook.com/2013/12/java-private-constructor-example/) then you cannot create the object of that class from outside of the class.

Examples: private double num = 100;

b. Write a specific program of the above mentioned access modifiers in java.

Ans: for private access modifiers code is written below but I have tried to access it from outside class which will show error because that is not possible. So methods and variables cant be accessed from outside function

public class JavaApplication28 {

public static void main(String args[]){

Calculator obj = new Calculator();

System.out.println(obj.num);

System.out.println(obj.square(10));

} }

class Calculator{

private double num = 100;

private int square(int a){

return a\*a;

}

}

Now for default access modifier

public class Student {

long marks = 0;

}

public class StudentMain {

public static void main(String args){

Student s = new Student();

System.out.println(s.marks);

}}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q2. a. Explain in detail Public and Protected access modifiers?

**ANS: Protected access modifiers:**

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces,

The protected access modifier is accessible within the package. However, it can also accessible outside the package but through inheritance only We can't assign protected to outer class and interface If you make any constructor protected, you cannot create the instance of that class from outside the package If you are overriding any method, overridden method (i.e., declared in the subclass) must not be more restrictive According to the previous point, if you assign protected to any method or variable, that method or variable can be overridden to sub-class using public or protected access modifier only.

**Public access modifiers:**

A Java public keyword is an access modifier. It can be assigned to variables, methods, constructors, and classes. It is the most non-restricted type of access modifier.

## Points to remember

* The public access modifier is accessible everywhere. So, we can easily access the public inside and outside the class and package.
* If you are overriding any method, overridden method (i.e., declared in the subclass) must not be more restrictive. So, if you assign public to any method or variable, that method or variable can be overridden to sub-class using public access modifier only.
* If a program contains multiple classes, at most one class can be assigned as public.
* If a class contain a public class, the name of the program must be similar to the public class name.

B. Write a specific program of the above mentioned access modifiers in java.

Example for public:

Ans: Public:

class A {

    public String msg="Try to access a public variable outside the class";

     String info;

     public void display()

     {

         System.out.println("Try to access a public method outside the class");

         System.out.println(info);

     }

     public A(String info)

     {

         this.info=info;

     }

}

  public class PublicExample1 {

    public static void main(String[] args) {

        A a=new A("Try to create the instance of public constructor outside the class");

       System.out.println(a.msg);

       a.display();

            }

}

Protected example:

public class Addition {

protected int addTwoNumbers(int a, int b){

return a+b;

}

}

class Test extends Addition{

public static void main(String args[]){

Test obj = new Test();

System.out.println(obj.addTwoNumbers(11, 22));

}

}

  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q3. a. What is inheritance and why it is used, discuss in detail ?

**Ans : Inheritance:**

The process by which one class acquires the properties (data members) and functionalities(methods) of another class is called **inheritance**. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.  
**Child Class:**  
The class that extends the features of another class is known as child class, sub class or derived class.

**Parent Class:**  
The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

**Importance of inheritance:**

Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.  
Inheritance allows us to reuse of code, it improves reusability in your java application.  
The biggest advantage ofInheritance is that the code that is already present in base class need not be rewritten in the child class.

## Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class

class XYZ extends ABC

{

}

b. Write a program using Inheritance class on Animal in java.

Ans: class Animal{

void eat()

{

System.out.println("eating...")

;}

void Sleep(){System.out.println("sleeping...");

}

}

  class Dog extends Animal{

void bark(){

System.out.println("barking...");

}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q4. a. What is polymorphism and why it is used, discuss in detail ?

**Ans: Polymorphism:** Polymorphism in Java is a concept by which we can perform a single action in different ways. We can perform polymorphism in java by method overloading and method overriding. If you overload a static method in Java, it is the example of compile time polymorphism.

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like specified in [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound

**Why is it used:**

The good reason for why Polymorphism is need in java is because the concept is extensively used in implementing inheritance.

It plays an important role in allowing objects having different internal structures to share the same external interface.

Polymorphism gives you benefits only if you need Polymorphism. It's used when an entity of your conceptual project can be seen as the specialization of another entity. The main idea is "specialization". A great example stands in the so called Taxonomy for example applied to living beings. Dogs and Humans are both Mammals. This means that, the class Mammals group all the entities that have some properties and behaviors in common.

b. Write a program using polymorphism in a class on Employee in java.

public class Employee {

private String name;

private String address;

private int number;

public Employee(String name, String address, int number) {

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public void mailCheck() {

System.out.println("Mailing a check to " + this.name + " " + this.address);

}

public String toString() {

return name + " " + address + " " + number;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public void setAddress(String newAddress) {

address = newAddress;

}

public int getNumber() {

return number;

}

}

public class Salary extends Employee {

private double salary; // Annual salary

public Salary(String name, String address, int number, double salary) {

super(name, address, number);

setSalary(salary);

}

public void mailCheck() {

System.out.println("Within mailCheck of Salary class ");

System.out.println("Mailing check to " + getName()

+ " with salary " + salary);

}

public double getSalary() {

return salary;

}

public void setSalary(double newSalary) {

if(newSalary >= 0.0) {

salary = newSalary;

}

}

public double computePay() {

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

public class VirtualDemo {

public static void main(String [] args) {

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q5. a. Why abstraction is used in OOP, discuss in detail?

Ans: Data Abstraction is the property by virtue of which only the essential details are displayed to the user The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

**Abstraction in JAVA**: “shows” only the essential attributes and “hides” unnecessary details of the object from the user. In Java, abstraction is accomplished using Abstract classes, Abstract methods, and interfaces. Abstraction helps in reducing programming complexity and effort.

### **Abstract Class:**

A class which is declared “abstract” is called as an abstract class. It can have abstract methods as well as concrete methods. A normal class cannot have abstract methods.

### **Abstract Method:**

A method without a body is known as an Abstract Method. It must be declared in an abstract class. The abstract method will never be final because the abstract class must implement all the abstract methods.

**Rules of Abstract Method:**

* Abstract methods do not have an implementation; it only has method signature
* If a class is using an abstract method they must be declared abstract. The opposite cannot be true. This means that an abstract class does not necessarily have an abstract method.
* If a regular class extends an abstract class, then that class must implement all the abstract methods of the abstract parent

b. Write a program on abstraction in java.

**Code:**

abstract class Shape

{

    String color;

    // these are abstract methods

    abstract double area();

    public abstract String toString();

    // abstract class can have constructor

    public Shape(String color) {

        System.out.println("Shape constructor called");

        this.color = color;

    }

    // this is a concrete method

    public String getColor() {

        return color;

    }

}

class Circle extends Shape

{

    double radius;

    public Circle(String color,double radius) {

        // calling Shape constructor

        super(color);

        System.out.println("Circle constructor called");

        this.radius = radius;

    }

    @Override

    double area() {

        return Math.PI \* Math.pow(radius, 2);

    }

    @Override

    public String toString() {

        return "Circle color is " + super.color +

                       "and area is : " + area();

    }

      }

class Rectangle extends Shape{

    double length;

    double width;

    public Rectangle(String color,double length,double width) {

        // calling Shape constructor

        super(color);

        System.out.println("Rectangle constructor called");

        this.length = length;

        this.width = width;

    }

    @Override

    double area() {

        return length\*width;

    }

    @Override

    public String toString() {

        return "Rectangle color is " + super.color +

                           "and area is : " + area();

    }

}

public class Test

{

    public static void main(String[] args)

    {

        Shape s1 = new Circle("Red", 2.2);

        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());

        System.out.println(s2.toString());

    }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_