

**JJIQRA NATIONAL
UNIVERSITY HAYATABAD
PESHAWAR**



NAME: yasir zaman

ID.NO:16729

SEMESTER: 2TH

SECTION: COMPUTER SCIENCE

Paper: programming

Teacher: \$ir Ayub

Question no1;

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

Answer:

Access modifiers:

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
1.class A{
2.private int data=40;

3.private void msg(){System.out.println("Hello java");}
4.}
```

```
5. public class Simple{
6. public static void main(String args[]){
7. A obj=new A();
8. System.out.println(obj.data);//Compile Time Error
9. obj.msg();//Compile Time Error
10. }
11.}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example

```
12.class A{
13.private A(){//private constructor
14.void msg(){System.out.println("Hello java");}
15.}

16.public class Simple{
17. public static void main(String args[]){
18. A obj=new A();//Compile Time Error
19. }
20.}
```

2) Default

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
21. //save by A.java
22. package pack;

23. class A{
24.     void msg(){System.out.println("Hello");}

25. }

26. //save by B.java
27. package mypack;

28. import pack.*;
29. class B{

30.     public static void main(String args[]){
31.         A obj = new A();//Compile Time Error

32.         obj.msg();//Compile Time Error
33.     }

34. }
```

Question no1 : part B;

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

Answer;

Program of private;

```
//Program to show error while using a class from different packages
with private modifier.
package test;
class eduCBA
{
private void display()
{
System.out.println("Hello World!");
}
}
class Check
{
public static void main (String args[])
{
eduCBA obj = new eduCBA();
//make class check to access private method of another class eduCBA.
obj.display();
}
}
```

Output:

error: display() has private access in eduCBA obj.display();

Program of default;

```
//Java program to show the default modifier.

package Test;

//Where Class eduCBA is having Default access modifier as no access modifier
is specified here
```

```
class eduCBA
{
void display ()
{
System.out.println("Hello World!");
}
}
```

Output:

Hello World!



Question no; 2

Explain in detail Public and Protected access modifiers?

Answer;

Public

User can declare a class, method, constructor, and interface with a 'public' access modifier, which can access by any class, method, constructor, and interface within or different packages.

- This access modifier has the Boundless among all modifiers.
- When any class, methods or package marked with 'public' access modifier, where it is accessible to everyone from everywhere from the program.

- There are no limitations on the scope of 'public' access class, methods.

For Example: –

```
//Java program to show to public access modifier
package test;
public class eduCBA
{
public void display ()
{
System.out.println("Hello World!");
}
}
package test2;
import test.*;
class pub
{
public static void main (String args []){
eduCBA obj = new eduCBA ();
obj.display ();
}
}
```

Output:

Hello World

Protected

Syntax 'protected' is used by users when they want to use this access modifier.

- This access modifier is accessible only within the same package or same sub-classes in different classes (but users have to import that package where it was specified).

- User cannot mark class and interfaces with 'protected' access modifier. However, Methods and fields can be declared as protected If methods and fields are in an interface.

For Example:

```
//Java program to show to protected access modifier
```

```
package test;
```

```
//Class eduCBA
```

```
public class eduCBA
```

```
{
```

```
protected void display ()
```

```
{
```

```
System.out.println("Hello World!");
```

```
}
```

```
}
```

```
//Java program to show to protected modifier in same sub-classes of different
```

```
packages
```

```
package test2;
```

```
import test.*;
```

```
//Class pro is subclass of eduCBA
```

```
class pro extends eduCBA
```



```
{  
  
public static void main(String args[])  
  
{  
  
pro obj = new pro();  
  
obj.display();  
  
}  
  
}
```

Output:

Hello World!

Question no:2 paryB

Program of public Access Modifier:

```
// Animal.java file  
// public class  
public class Animal {  
    // public variable  
    public int legCount;  
  
    // public method  
    public void display() {  
        System.out.println("I am an animal.");  
        System.out.println("I have " + legCount + " legs.");  
    }  
}
```

```
// Main.java
public class Main {
    public static void main( String[] args ) {
        // accessing the public class
        Animal animal = new Animal();

        // accessing the public variable
        animal.legCount = 4;
        // accessing the public method
        animal.display();
    }
}
```

Output:

```
I am an animal.
I have 4 legs.
```

Program of Protected Access Modifier:

```
class Animal {
    // protected method
    protected void display() {
        System.out.println("I am an animal");
    }
}

class Dog extends Animal {
    public static void main(String[] args) {


        // create an object of Dog class
        Dog dog = new Dog();

        // access protected method
```

```
        dog.display();
    }
}
```

Output:

```
I am an animal
```



Question no:3:part A.

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

Answer:

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.

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.

Note: The biggest **advantage of Inheritance** is that the code that is already present in base class need not be rewritten in the child class.

This means that the data members(instance variables) and methods of the parent class can be used in the child class as. If you are finding it difficult to understand what is class and object then refer the guide that I have shared on object oriented programming: [OOps Concepts](#)

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
{
}
```

Inheritance Example

In this example, we have a base class `Teacher` and a sub class `PhysicsTeacher`. Since class `PhysicsTeacher` extends the designation and college properties and `work()` method from base class, we need not to declare these properties and method in sub class.

Here we have `collegeName`, `designation` and `work()` method which are common to all the teachers so we have declared them in the base class, this way the child classes like `MathTeacher`, `MusicTeacher` and `PhysicsTeacher` do not need to write this code and can be used directly from base class.

```
class Teacher {
    String designation = "Teacher";
    String collegeName = "Beginnersbook";
    void does(){
        System.out.println("Teaching");
    }
}
```

```

public class PhysicsTeacher extends Teacher{
    String mainSubject = "Physics";
    public static void main(String args[]){
        PhysicsTeacher obj = new PhysicsTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
    }
}

```

Output:

```

Beginnersbook
Teacher
Physics
Teaching

```

Note:

The derived class inherits all the members and methods that are declared as public or protected. If the members or methods of super class are declared as private then the derived class cannot use them directly. The private members can be accessed only in its own class. Such private members can only be accessed using public or protected getter and setter methods of super class as shown in the example below.

```

class Teacher {
    private String designation = "Teacher";
    private String collegeName = "Beginnersbook";
    public String getDesignation() {
        return designation;
    }
    protected void setDesignation(String designation) {
        this.designation = designation;
    }
    protected String getCollegeName() {
        return collegeName;
    }
    protected void setCollegeName(String collegeName) {
        this.collegeName = collegeName;
    }
    void does(){
        System.out.println("Teaching");
    }
}

public class JavaExample extends Teacher{
    String mainSubject = "Physics";
    public static void main(String args[]){
        JavaExample obj = new JavaExample();
        /* Note: we are not accessing the data members
        * directly we are using public getter method
        * to access the private members of parent class

```

```
        */
        System.out.println(obj.getCollegeName());
        System.out.println(obj.getDesignation());
        System.out.println(obj.mainSubject);
        obj.does();
    }
}
```

The output is:

```
Beginnersbook
Teacher
Physics
Teaching
```

Question no: 3

Write a program using Inheritance class on Animal in java.

Program;

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** BabyDog **extends** Dog{
8. **void** weep(){System.out.println("weeping...");}
9. }
10. **class** TestInheritance2{
11. **public static void** main(String args[]){
12. BabyDog d=**new** BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
- 16.}}

Output:

```
weeping...
```

barking...
eating...



Question no;4 PartA: What is polymorphism and why it is used, discuss in detail ?

Answer:

Polymorphism; Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Example

Let us look at an example.

```
public interface Vegetarian{}  
public class Animal{}  
public class Deer extends Animal implements Vegetarian{}
```

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above examples –

- A Deer IS-A Animal
- A Deer IS-A Vegetarian
- A Deer IS-A Deer
- A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal –

Example

```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

All the reference variables d, a, v, o refer to the same Deer object in the heap.

Virtual Methods

In this section, I will show you how the behavior of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

Example

```
/* File name : Employee.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;  
}  
}
```

Question no;4 part no B;

Program:

```
/* File name : Employee.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;
}
}
```

Now suppose we extend Employee class as follows –

```
/* File name : Salary.java */
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;
    }
}
```

Now, the program carefully and try to determine its output –

```
/* File name : VirtualDemo.java */
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();
}
}
```

This will produce the following result –

Output

Constructing an Employee
Constructing an Employee

Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with



Question no;5 PART A

ANSWER: Abstraction is one of the [key concepts](#) of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

That's a very generic concept that's not limited to object-oriented programming. You can find it everywhere in the real world.

Abstraction in the real world

I'm a coffee addict. So, when I wake up in the morning, I go into my kitchen, switch on the coffee machine and make coffee. Sounds familiar?

Making coffee with a coffee machine is a good example of abstraction.

You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get.

The thing you don't need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee. You don't need to know the ideal temperature of the water or the amount of ground coffee you need to use.

Someone else worried about that and created a coffee machine that now acts as an abstraction and hides all these details. You just interact with a simple interface that doesn't require any knowledge about the internal implementation.

You can use the same concept in object-oriented programming languages like Java.

Abstraction in OOP

Objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation. But you don't need to understand how this method is implemented and which kinds of actions it has to perform to create the expected result.

Let's implement the coffee machine example in Java. You do the same in any other object-oriented programming language. The syntax might be a little bit different, but the general concept is the same.

Use abstraction to implement a coffee machine

Modern coffee machines have become pretty complex. Depending on your choice of coffee, they decide which of the available coffee beans to use and how to grind them. They also use the right amount of water and heat it to the required temperature to brew a huge cup of filter coffee or a small and strong espresso.

Implementing the *CoffeeMachine* abstraction

Using the concept of abstraction, you can hide all these decisions and processing steps within your *CoffeeMachine* class. If you want to keep it as simple as possible, you just need a constructor method that takes a *Map* of *CoffeeBean* objects to create a new *CoffeeMachine* object and a *brewCoffee* method that expects your *CoffeeSelection* and returns a *Coffee* object.

```
import org.thoughts.on.java.coffee.CoffeeException;
import java.util.Map;

public class CoffeeMachine {
    private Map<CoffeeSelection, CoffeeBean> beans;

    public CoffeeMachine(Map<CoffeeSelection, CoffeeBean> beans)
    {
        this.beans = beans
    }

    public Coffee brewCoffee(CoffeeSelection selection) throws
    CoffeeException {
        Coffee coffee = new Coffee();
        System.out.println("Making coffee ...");
        return coffee;
    }
}
```

CoffeeSelection is a simple enum providing a set of predefined values for the different kinds of coffees.

```
public enum CoffeeSelection {
    FILTER_COFFEE, ESPRESSO, CAPPUCCINO;
}
```

And the classes *CoffeeBean* and *Coffee* are simple POJOs (plain old Java objects) that only store a set of attributes without providing any logic.


```
public class CoffeeBean {
    private String name;
    private double quantity;

    public CoffeeBean(String name, double quantity) {
        this.name = name;
        this.quantity;
    }
}

public class Coffee {
```

```
private CoffeeSelection selection;
private double quantity;

public Coffee(CoffeeSelection, double quantity) {
    this.selection = selection;
    this.quantity = quantity;
}
}
```



Question no5:Part B:

Program:

example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class

```
35. abstract class Bike{
36.  abstract void run();
37.}
38. class Honda4 extends Bike{
39. void run(){System.out.println("running safely");}
40. public static void main(String args[]){
41. Bike obj = new Honda4();
42. obj.run();
43.}
44.}
```

Output;

running safely



Finish paper 😊😊😊