# ID: 16702

# NAME: ABUZAR KHAN

# CLASS: BS (Software Engineering)

# SECTION: B

# SUBJECT: OBJECT OREINTED PROGRAMMING

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

# Answer:

# ACCESS MODIFIERS IN JAVA:

# There are two types of modifiers in Java: access modifiers and non-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:

# 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.

# Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

# 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.

# 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.

# There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

#  OR

# 1. 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. Let’s see an example to understand this:

# DEFAULT ACCESS MODIFIER EXAMPLE IN JAVA:

# To understand this example, you must have the knowledge of [packages in java](https://beginnersbook.com/2013/03/packages-in-java/).

# In this example we have two classes, Test class is trying to access the default method of Addition class, since class Test belongs to a different package, this program would throw compilation error, because the scope of default modifier is limited to the same package in which it is declared.Addition.java

# FOR EXAMPLE:

**package** abcpackage;

**public** **class** Addition {

 /\* Since we didn't mention any access modifier here, it would

 \* be considered as default.

 \*/

 **int** addTwoNumbers(**int** a, **int** b){

 **return** a+b;

 }

}

TEST.JAVA:

**package** xyzpackage;

/\* we are importing the abcpackage

 \* But still we will get error because the

 \* Class we are trying to use has default access

 \* Modifier.

 \*/

**import** abcpackage.\*;

**public** **class** Test {

 **public** **static** **void** main(String args[]){

 Addition obj = **new** Addition();

 /\* It will throw error because we are trying to access

 \* The default method in another package

 \*/

 obj.addTwoNumbers(10, 21);

 }

}

# OUTPUT:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method addTwoNumbers(**int**, **int**) from the type Addition is not visible

at xyzpackage.Test.main(Test.java:12)

# 2. 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.

# Let’s see an example to understand this:

# PRIVATE ACCESS MODIFIER EXAMPLE IN JAVA

# This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example. The private data member and method are only accessible within the class.

# FOR EXAMPLE:

**class** ABC{

 **private** **double** num = 100;

 **private** **int** square(**int** a){

 **return** a\*a;

 }

}

**public** **class** Example{

 **public** **static** **void** main(String args[]){

 ABC obj = **new** ABC();

 System.***out***.println(obj.num);

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

 }

}

# OUTPUT:

***Compile*** - time error

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

# Answer:

# A SPECIFIC PROGRAM TO ILLUSTRATE DEFAULT MODIFIER

//Java program to illustrate default modifier

**package** p1;

//Class Apples is having Default access modifier

**class** Apple

{

 **void** display()

 {

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

 }

}

//Java program to illustrate error while

//using class from different package with

//default modifier

**package** p2;

**import** p1.\*;

//This class is having default access modifier

**class** AppleNew

{

 **public** **static** **void** main(String args[])

 {

 //accessing class Apple from package p1

 Apples obj = **new** Apple();

 obj.display();

 }

}

# OUPUT:

Compile time error

# Explanation:

* 1st step making package p1
* 2nd step making class A, inside the class creating method void display() to print message on screen.
* 3rd step making another package as well making new class and we import 1st package like import pack.
* 4th step creating object in current class which is A obj and call the method of 1st package

# A SPECIFIC PROGRAM TO ILLUSTRATE PRIVATE MODIFIER:

//Java program to illustrate error while

//using class from different package with

//private modifier

**package** p1;

**class** A

{

 **private** **void** display()

 {

 System.***out***.println("GeeksforGeeks");

 }

}

**class** B

{

 **public** **static** **void** main(String args[])

 {

 A obj = **new** A();

 //trying to access private method of another class

 obj.display();

 }

}

# OUTPUT:

error: display() has private access in A

 obj.display();

# Explanation:

* 1st step making package p1
* 2nd step making class A, inside the class creating method void display() to print message on screen.
* 3rd step making another package as well making new class and we import 1st package like import pack.
* 4th step creating object in current class which is A obj and call the method of 1st package

*\*\*\*============///============///===========///============\*\*\**

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

# Answer:

# 1. PROTECTED ACCESS MODIFIER:

Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.
Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

# PROTECTED ACCESS MODIFIER EXAMPLE IN JAVA:

In this example the class Test which is present in another package is able to call the addTwoNumbers() method, which is declared protected. This is because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).
**Addition.java.**

# **FOR EXAMPLE:**

**package** abcpackage;

**public** **class** Addition {

 **protected** **int** addTwoNumbers(**int** a, **int** b){

 **return** a+b;

 }

}

# TEST IN JAVA:

**package** xyzpackage;

**import** abcpackage.\*;

**class** Test **extends** Addition{

 **public** **static** **void** main(String args[]){

 Test obj = **new** Test();

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

 }

}

# Output:

33

# 2. PUBLIC ACCESS MODIFIER:

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn’t put any restriction on the access.

# PUBLIC ACCESS MODIFIER EXAMPLE IN JAVA:

Let’s take the same example that we have seen above but this time the method addTwoNumbers() has public modifier and class Test is able to access this method without even extending the Addition class. This is because public modifier has visibility everywhere.
Addition.java

# FOR EXAMPLE:

**package** abcpackage;

**public** **class** Addition {

 **public** **int** addTwoNumbers(**int** a, **int** b){

 **return** a+b;

 }

}

# TEST.JAVA:

**Package** xyzpackage;

**import** abcpackage.\*;

**class** Test{

 **public** **static** **void** main(String args[]){

 Addition obj = **new** Addition();

 System.***out***.println(obj.addTwoNumbers(100, 1));

 }

}

Output:

101

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

# Answer:

# A SPECIFIC PROGRAM TO ILLUSTRATE PROTECTED MODIFIER:

//Java program to illustrate

//protected modifier

**package** p1;

//Class A

**public** **class** A

{

 **protected** **void** display()

 {

 System.***out***.println("Earth is Round");

 }

}

|  |
| --- |
| //Java program to illustrate //protected modifier **package** p2; **import** p1.\*; //importing all classes in package p1  //Class B is subclass of A **class** B **extends** A {  **public** **static** **void** main(String args[])  {  B obj = **new** B();  obj.display();  }  }  |

# Output:

Earth is Round

# Explanation:

* 1st step making package p1
* 2nd step making class A, inside the class creating method void display() to print message on screen.
* 3rd step making another package as well making new class and we import 1st package like import pack.
* 4th step creating object in current class which is A obj and call the method of 1st package

# A SPECIFIC PROGRAM TO ILLUSTRATE PUBLIC MODIFIER:

|  |
| --- |
| //Java program to illustrate //public modifier **package** p1; **public** **class** A {  **public** **void** display()  {  System.***out***.println("Blood is thicker than water");  } } **package** p2; **import** p1.\*; **class** B {  **public** **static** **void** main(String args[])  {  A obj = **new** A;  obj.display();  } }  |

# OUTPUT:

Blood is thicker than water

# Explanation:

* 1st step making package p1
* 2nd step making class A, inside the class creating method void display() to print message on screen.
* 3rd step making another package as well making new class and we import 1st package like import pack.
* 4th step creating object in current class which is A obj and call the method of 1st package

*\*\*\*============///============///===========///============\*\*\**

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

# Answer:

# INHERITANCE:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

# WHY USE INHERITANCE IN JAVA

# For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).

# For Code Reusability.

# TERMS USED IN INHERITANCE

# Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

# Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

# Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

# Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

# THE SYNTAX OF JAVA INHERITANCE:

1. class Subclass-name extends Superclass-name
2. {
3. //methods and fields
4. }

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

# As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

**class** Employee{

 **float** salary=40000;

}

**class** Programmer **extends** Employee{

 **int** bonus=10000;

 **public** **static** **void** main(String args[]){

 Programmer p=**new** Programmer();

 System.***out***.println("Programmer salary is:"+p.salary);

 System.***out***.println("Bonus of Programmer is:"+p.bonus);

}

}

# OUTPUT:

Programmer salary is:40000.0

 Bonus of programmer is:10000

# TYPES OF INHERITANCE IN JAVA

# On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical



# SINGLE INHERITANCE EXAMPLE

# When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

**class** Animal{

**void** eat(){System.***out***.println("eating...");}

}

**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();

}

}

# Output:

barking...

eating...

# MULTILEVEL INHERITANCE EXAMPLE:

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

|  |
| --- |
| NOTE: Multiple inheritance is not supported in Java through classTo reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error. |

**class** Animal{

**void** eat(){System.***out***.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.***out***.println("barking...");}

}

**class** BabyDog **extends** Dog{

**void** weep(){System.***out***.println("weeping...");}

}

**class** TestInheritance2{

**public** **static** **void** main(String args[]){

BabyDog d=**new** BabyDog();

d.weep();

d.bark();

d.eat();

}

}

# OUTPUT:

weeping...

barking...

eating...

# HIERARCHICAL INHERITANCE EXAMPLE

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

**class** Animal{

**void** eat(){System.***out***.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.***out***.println("barking...");}

}

**class** Cat **extends** Animal{

**void** meow(){System.***out***.println("meowing...");}

}

**class** TestInheritance3{

**public** **static** **void** main(String args[]){

Cat c=**new** Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}

}

# OUTPUT:

meowing...

eating...

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



# **OUTPUT**



# EXPLANATION:

* 1st step making public class Animal
* Declaring protected string type and private string color
* Creating method for eat and sleep and also print message for both
* Creating method for set color
* Creating class using inheritace
* Calling method and declare string in it and print message
* Creating method for bark
* Creating main class
* Creating object and calling method for eat, sleep and bark
* Putting value for color
* Calling method.

*\*\*\*=========///==========///===========///==========\*\*\**

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

# POLYMORPHISM:

Polymorphism is an important concept of object-oriented programming. It simply means more than one form. That is, the same entity (method or operator or object) behaves differently in different scenarios. For example,

The + operator in Java is used to perform two specific functions. When it is used with numbers (integers and floating-point numbers), it performs addition.

int a = 5;

int b = 6;

int sum = a + b; // Output = 11

And when we use + operator with strings, it performs string concatenation. For example,

String firstName = "abc ";

String lastName = "xyz";

name = firstName + lastName; // Output = abc xyz

# TYPES OF POLYMORPHISM

In Java, Polymorphism can be divided into two types:

* Run-time Polymorphism
* Compile-time Polymorphism

# 1.RUN-TIME POLYMORPHISM

In Java, run-time polymorphism can be achieved through method overriding.

Suppose the same method is created in the superclass and its subclasses. In this case, the method that will be called depends upon the object used to call the method. For example,

# Example 1: Method Overriding

**abstract** **class** Animal {

 **public** **abstract** **void** makeSound();

}

**class** Dog **extends** Animal {

 @Override

 **public** **void** makeSound() {

 System.***out***.println("Bark bark..");

 }

}

**class** Cat **extends** Animal {

 @Override

 **public** **void** makeSound() {

 System.***out***.println("Meow meow..");

 }

}

**class** Main {

 **public** **static** **void** main(String[] args) {

 Dog d1 = **new** Dog();

 d1.makeSound();

 Cat c1 = **new** Cat();

 c1.makeSound();

 }

}

**Output**:

Bark bark…

Meow-meow...

# 2. COMPILE-TIME POLYMORPHISM

The compile-time polymorphism can be achieved through method overloading and operator overloading in Java.

# METHOD OVERLOADING

In a Java class, we can create methods with the same name if they differ in parameters. For example,

**void** func() { ... }

**void** func(**int** a) { ... }

**float** func(**double** a) { ... }

**float** func(**int** a, **float** b) { ... }

This is known as method overloading in Java.

Let's take a working example of method overloading.

### Example 3: Method Overloading

**class** Demo {

 **public** **void** displayPattern(){

 **for**(**int** i = 0; i < 10; i++) {

 System.***out***.print("\*");

 }

 }

 **public** **void** displayPattern(**char** symbol) {

 **for**(**int** i = 0; i < 10; i++) {

 System.***out***.print(symbol);

 }

 }

}

**class** Main {

 **public** **static** **void** main(String[] args) {

 Demo d1 = **new** Demo();

 d1.displayPattern();

 System.***out***.println("\n");

 d1.displayPattern('#');

 }

}

**Output**:



In the above program, the displayPattern() method is overloaded.

* If we call the method without passing any arguments, a pattern of \* is created.
* If we call the method by passing a character as an argument, a pattern of that character is created.

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





This will produce the following result −

# OUTPUT



# Explanation:

* 1st step making Employee class as well declaring a static int base and making method for salary.
* 2nd step making class of Manager also inheriate with class Employee to access the salary method.
* 3rd step making class of Clerk also inheriate with class Employee to access the salary method.
* 4th step calling main class
* 5th step static void printsalary (Employee e) can be used to print the salary of any type of employee using base class reference

*\*\*\*=========///==========///===========///==========\*\*\**

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

# Answer:

# ABSTRACTION:

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message.

# USE OF ABSTRACTION:

Abstraction is about hiding unwanted details while showing most essential information. Abstraction is use when we want to hide the implementation from the user and showing only the work or functionality.

For example: Email sending only show the email address and subject and send button.

#  b. Write a program on abstraction in java.

# Answer:

# C:\Users\Abuzar GhaFFari\Desktop\bank.PNGOUTPUT:

Rate of Interest is: 7 %

Rate of Interest is: 8 %

# Explanation:

* 1st step making abstract class and inside the class making abstract method like Abstract void display(); for print message on screen.
* 2nd step making another class B also inheriate with class A and creating method like Void disp() and print message on screen.
* Now making Main class and creating object in the current class B obj and call the methods with object and method name.