**Q1.Constructors & Inheritance**

Which of the following pieces of code is incorrect?

Ans: **public** **class** **A** {

**public** **A**(**int** i) {

}

}

**class** **B** **extends** A {

}

**✓ Correct**

**Feedback:**

The given code is incorrect because class B inherits class A; so, when an object of class B is created, the default constructor of class B will be invoked, which will also invoke the super() method that will reach the constructor of class A. Since the constructor is not defined in class A, because there is no integer argument passed in the default constructor invocation, an error will be thrown.

#### Q2. Access Modifier in Java

Select the correct sentence from the options given below:

More than one option can be correct in this case.

Ans: i) A method of the super class with a default access modifier can be overridden as protected or public.

**✓ Correct**You missed this!

**Feedback:**

The order of access modifiers in the increasing order of their restrictive nature is:  
                   public > protected > default > private  
A more restrictive access modifier can be converted into a less restrictive access modifier but not vice versa, which means that a protected method can be made public but a public method cannot be made protected. Similarly, in this option, you can override a default method to either public or protected because it is less restrictive.

ii) A method of the super class with a default access modifier can be overridden as public.

**✓ Correct**

**Feedback:**

The order of access modifiers in the increasing order of their restrictive nature is:  
                   public > protected > default > private  
A more restrictive access modifier can be converted into a less restrictive access modifier but not vice versa, which means that a protected method can be made public but a public method cannot be made protected. In this option, you can easily override a default method as public since default is more restrictive than public.

#### Q3: Interface

Which of the following is/are the **incorrect** implementation(s) of interfaces in **Java 8**?

More than one option can be correct in this case.

Ans: i) **interface** **bookTickets** {

**int** number = **0**;

**public** **void** **addSeat**() {

        number++;

    }

**public** **void** **display**();

}

**✓ Correct**

**Feedback:**

You cannot define a method in an interface unless it is default or static. In this option, the method **addSeat** **()** is defined and is not default; therefore, there will be a compilation error.

ii) **interface** **bookTickets** {

**int** number = **0**;

**default** **void** **addSeat**() {

number = number + **1**;

}

**void** **display**();

}

**✓ Correct**

**Feedback:**

Variables in an interface are **static** and **final**. This is because you cannot use an interface directly and for implementation; it needs a class or another interface. Therefore, to make the instance variable available to all the classes and the interface that implements it, you need to make the variables static. As you know, a static variable carries a unique value throughout the code, and hence, the value cannot be changed even within the interface. In this code, the default method **addSeat()** will add 1 to the value of the final variable**number**, which will not be allowed, and the program will throw a compilation error.

Q4: **final** **public** **abstract** **class** **Calculation** {

**int** num1;

**int** num2;

Calculation(**int** n1, **int** n2) {

**this**.num1 = n1;

**this**.num2 = n2;

}

**public** **abstract** **void** **add**();

}

Identify the mistake in the piece of code given above.

Ans: The abstract class cannot be declared as final.

**✓ Correct**

**Feedback:**

Abstract and final are the opposite of each other. While abstract allows you to independently change, define, and inherit its functioning in the subclasses according to your convenience and need, the final keyword makes sure that you are not able to change or extend variables or methods of the class. A final class cannot participate in inheritance at all, and once initialized, it cannot be changed. You cannot create an object for an abstract class and you need to extend it using other subclasses. These two are contradictory functions; hence, final and abstract cannot be used together.

#### Q5: Find Output - I

What will the output of the following code be?

**class** **A** {

**public** **void** **display**() {

**double** i = **99.99**;

System.out.println(i);

}

}

**class** **B** **extends** A {

**public** **void** **display**() {

**double** i = **100**;

**super**.display();

}

}

**public** **class** **Main** {

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

B object = **new** B();

object.display();

}

}

Ans: 99.99

**✓ Correct**

**Feedback:**

Class B is derived from class A, and both have a display() method that can print the value of ‘i’. An object of class B is created in the main method and calls its display() method. The display() method of class B contains the super.display() method. The ‘super’ keyword is used to access the function of the parent class, which has the same name as one of the functions in the child class. Therefore, super.display() will call the display method of class A that will print the value of ‘i’ as 99.99.

#### Q6: Find Output - II

What will be the output of the code given below?

**abstract** **class** **Parent** {

**abstract** **int** **display**(**int** i);

**int** **returnValue**(**int** i) {

**return** display(++i);

}

}

**class** **Child** **extends** Parent {

**int** **display**(**int** i) {

**return** ++i;

}

}

**public** **class** **Source** {

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

Child myObj = **new** Child();

System.out.println(myObj.returnValue(**1011**));

}

}

Ans: 1013

**✓ Correct**

**Feedback:**

returnValue() takes 1011 as the argument and increments it by 1; then, it passes the value 1012 to display() to return the result of display(). display() of class Child again increments the value by, 1 and hence, 1013 is the result. Note that it was possible because the abstract class Parent containing the abstract returnValue() is called by the object of the subclass Child that inherits the class Parent.

4/6 Ans correct- 40

# Graded Questions - II

#### Q1: Referring to Parent Class

Which of the following keywords is used to refer to the parent class members in a child class?

Ans: super

**✓ Correct**

**Feedback:**

Whenever a child class needs to reference a member of a parent class, it can do so by prefixing the variable name or the method name, with the keyword super and a ( . ) operator.

#### Q2: Superclass & Subclass

Which of the following statements is/are true about superclasses and subclasses?

1. A superclass is one that contains the common functionality for other classes to use.
2. A subclass can consist of more functionalities than its superclass.
3. A superclass is a specialised version of a subclass.
4. Both the superclass and subclass must have the same functionality.

Ans: 1 and 2

**✓ Correct**

**Feedback:**

The concept of inheritance is based on the fact that the common functionality code is defined in the superclass from which other classes (subclasses) can inherit the same. Subclasses thus become specialised versions of their superclasses.

#### Q2: Access

Based on the code snippet given below, which member variable cannot be referred to inside the Policeman class or by a Policeman object?

**class** **Person** {

**private** String eyeColor;

**protected** String name;

**public** **Person**(String name) {

**this**.name = name;

}

}

**class** **Adult** **extends** Person {

**protected** **int** age;

**public** **Adult**(String name, **int** age) {

**super**(name);

**this**.age = age;

}

}

**class** **Policeman** **extends** Adult {

**protected** **final** String job;

**public** **Policeman**(String name, **int** age) {

**super**(name, age);

**this**.job = "policeman";

}

}

Ans: String eyeColor

**✓ Correct**

**Feedback:**

The Policeman class or object(s) cannot refer to the instance variable String eyeColor because eyeColor is a private variable of the Person class, and private variables cannot be referred to outside of the class that it is declared in.

#### Q3: Subclasses

Find out all the subclasses that exist in the code given below.

(Note that more than one option may be true.)

**class** **Person** {

**private** String eyeColor;

**protected** String name;

**public** **Person**(String name) {

**this**.name = name;

}

}

**class** **Adult** **extends** Person {

**protected** **int** age;

**public** **Adult**(String name, **int** age) {

**super**(name);

**this**.age = age;

}

}

**class** **Policeman** **extends** Adult {

**protected** **final** String job;

**public** **Policeman**(String name, **int** age) {

**super**(name, age);

**this**.job = "policeman";

}

}

Assume that the default class Object is not taken into consideration meaning that a class inheriting the default Object class in Java will not be counted as a subclass.

Ans: i) Adult

**✓ Correct**

**Feedback:**

The class *Adult* inherits from the class *Person*. So, the class *Adult*can be termed as a subclass. This option is one of the correct choices.

ii) Policeman

**✓ Correct**

**Feedback:**

The class *Policeman* inherits from the class *Adult*. So, the class *Policeman* can be termed as a subclass. This option is one of the correct choices.

#### Q4: OOP

Go through the code given below:

**class** **Person** {

**private** String eyeColor;

**protected** String name;

Car myCar;

**public** **Person**(String name) {

**this**.name = name;

}

**public** **void** **setMyCar**(Car car){

**this**.myCar = car;

}

}

**class** **Adult** **extends** Person {

**protected** **int** age;

**public** **Adult**(String name, **int** age) {

**super**(name);

**this**.age = age;

}

}

**class** **Car** {

String brand;

String model;

**int** year;

**public** **Car**(String brand, String model, **int** year) {

**this**.brand = brand;

**this**.model = model;

**this**.year = year;

}

}

Which Object Oriented Principle is exhibited when the Person class is referring to the Car class using the following line of code?

Car myCar;

Ans: Composition

**✓ Correct**

**Feedback:**

Composition means using instance variables that are references to class A inside Class B. In this case, the Person class is referring to an instance variable that is of the Car class.

¾ ans correct 30marks

# Graded Questions - III

#### Q1: Access Specifier

Which of the following superclass members can be accessed both in the superclass and subclass declarations?

More than one option can be correct in this case.

Ans: public

**✓ Correct**

**Feedback:**

The public members of a class can be accessed by any class. Thus, the public members of a superclass can be easily accessed by the subclass. Thus, this option is one of the correct choices.



protected

**✓ Correct**

**Feedback:**

The protected members of a superclass can be accessed within the class as well as in its subclass. Thus, this option is one of the correct choices.

#### Q2: Abstract Classes & Interfaces

Identify the correct statement from the following.

Ans: A class can implement more than one interface.

**✓ Correct**

**Feedback:**

A class can implement more than one interface. The syntax for implementing more than one interface is — public class className implements interface1,interface2,interface3…..

#### Q3: Invoking Method of Superclass in Subclass

Which of the following is the correct way to invoke a method named myMethod taking in two integer arguments? Note that the method is defined in the superclass but needs to be invoked in its subclass.

Ans: **super**.myMethod(**2**, **3**);

**✓ Correct**

**Feedback:**

super.myMethod(2,3) will call the superclass version of myMethod(), with two integer-type values.

#### Q4: OOP Principle

Go through the code snippet given below:

**public** **class** **GradedQuestionDemo** {

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

Square mySquare = **new** Square(**5.0**);

mySquare.myMethod(**10**);

}

}

**class** **Rectangle** {

**private** String type;

**public** **Rectangle**(**double** width, **double** height) {

**this**.width = width;

**this**.height = height;

}

**public** **void** **myMethod**(**int** someVariable){

...

}

}

**class** **Square** **extends** Rectangle {

**public** **Square**(**double** side) {

**super**(side, side);

}

**public** **void** **myMethod**(**int** someVariable){

...

}

}

When encountering the following line of code:

mySquare.myMethod(**10**);

in the code above, Java is able to leverage the power of one of the principles of OOP and invoke

myMethod(...)

in the Square class instead of the

myMethod(...)

in the Rectangle class. Which of the following OOP principles is demonstrated by this example?

Ans: Polymorphism

**✓ Correct**

**Feedback:**

Polymorphism is the concept that a method can behave differently based on the type of object that is used to call it. This is the correct answer because mySqaure.myMethod(10); will refer to the myMethod(...) of the Square class instead of the myMethod(...) of the Rectangle class.

#### Q5: Abstract Class

Which of the following options is/are correct for the piece of code given below:

(Multiple options can be correct)

**abstract** **class** **Vehicle** {

**private** **char** vehicleType;

**public** **abstract** **void** **move** ();

}

**public** **class** **Truck** **extends** Vehicle {

**private** **int** truckNo;

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

Truck mahindra= **new** Truck();

mahindra.move();

}

}

Ans: Truck is not abstract, and it extends from the abstract class Vehicle. However, Truck does not provide an implementation for the abstract method move().

**✓ Correct**

**Feedback:**

In order to be a concrete class, the Truck class should provide a definition for the abstract method move() of the abstract class Vehicle. If not, then Truck should also be declared abstract.



You need to write a method definition in the truck class, for the abstract method defined in the Vehicle class.

**✓ Correct**

**Feedback:**

An abstract class can have concrete methods as well as abstract methods (methods without any method definitions).

4/5 ans correct = 40 marks

# Graded Questions - Arrays and ArrayLists

#### Q1: Data Structure Length

Select the correct statement.

Ans: Array is a fixed-length data structure and ArrayList is a variable-length data structure.

**✓ Correct**

**Feedback:**

Correct. Array is a fixed-length data structure and ArrayList is a variable-length data structure.

#### Q2: Primitives in ArrayList

State whether the following statement is true or false.

You can store primitives in ArrayList.

Ans: False

**✓ Correct**

**Feedback:**

Correct. ArrayList cannot store primitives.

2/2 = 20 marks

2nd attempt = 5marks

#### Q1@Override Annotation’s Target

To which of the following elements can the @Override annotation be applied?

Ans: Method

**✓ Correct**

**Feedback:**

Correct. The @Override annotation can only be applied to method declaration while overriding a method from the superclass.

5Marks

#### Q2:@Deprecated Annotation’s Target

To which of the following elements can the @Deprecated annotation be applied?

All of the above

**✓ Correct**

**Feedback:**

Correct. The @Deprecated annotation can be applied to fields, methods and classes.

1 proper correct and 2 -2nd attempt = 5 == 15marks

# Graded Questions - Git

#### Q1: Version Control

Version control can help us in which of the following situations?

Ans: All

**✓ Correct**

**Feedback:**

Version control helps in managing and tracking the changes to your files, collaborating and communicating changes to projects, and reverting to previous versions of your files.

#### Q2: Pushing a File

Which command is used to push a file to your remote repository on GitHub?

Ans: You can use the command ‘git push -u origin master’ to push a file to your remote repository on GitHub

**✓ Correct**

**Feedback:**

The command ‘git push -u origin master’ will push the most recent commit in your local repository to your online repository on GitHub.

#### Q3: Steps to push a git repository

What is the sequence of steps to be followed in order to push a Git repository from your local machine to a remote repository?

1. Adding the file to the staging area, so you can review the changes before making any commits
2. Initialising and creating an empty Git repository
3. Pushing all your local changes to your online repository, that is, your repository on GitHub.
4. Making a commit and writing a commit message

Ans: 2,1,4,3

**✓ Correct**

**Feedback:**

Here’s the correct sequence of the steps to push changes from a Git repo on your local computer to a remote repo:

* First step: Initialising and creating an empty Git repository
* Second step: Adding the file to the staging area, so you can review the changes before making any commits
* Third step: Making a commit and writing a commit message
* Fourth step: Pushing all your local changes to your online repository, that is, your repository on GitHub.

3/3 = 30 marks

# Graded Questions- I

Before you start with the module on **Inheritance and Polymorphism**, please attempt these graded questions based on the previous module on**Abstraction and Encapsulation**.

#### Q1-Find Output - I

What would be the output of the code given below:

**class** **BalanceAmount** {

**int** balance;

**public** **void** **setBalance**(**int** amount) {

balance=amount;

}

**public** **void** **incrementBalance**() {

balance+=**10**;

}

**public** **int** **getCurrentBalance**() {

**return** balance;

}

}

**public** **class** **Source** {

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

BalanceAmount obj= **new** BalanceAmount();

obj.setBalance(**20**);

obj.incrementBalance();

System.out.println(obj.getCurrentBalance());

obj.balance=**1000**;

System.out.println(obj.getCurrentBalance());

}

}

Ans: 30  
1000

**✓ Correct**

**Feedback:**

The setBalance() method will set the value of the variable ‘balance’ as 20, but after the incrementBalance() method is called, the value of balance will be changed to 30. Therefore the first output is 30. Now, when the variable ‘balance’ is called through obj.balance and is assigned a value of 1000, the value of balance will change to 1000 because the variable balance is declared public, i.e., it can be easily accessed outside the class. Hence, 1000 will be printed next.

#### Q2: Find Output - II

Select the correct output for the code given below:

**public** **class** **Employee** {

Integer salary;

**int** EmpId;

String EmpName;

**void** **display**() {

System.out.println(EmpId);

System.out.println(EmpName);

System.out.println(salary);

}

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

Employee object = **new** Employee();

object.display();

}

}

Ans: 0  
null  
null

**✓ Correct**

**Feedback:**

There is no constructor in the given class, and thus, the class variables EmpId, EmpName, and salary are not initialised. Thus, both the integer and the string variables are set to the default values, which are 0 and null respectively for int and string variables whereas salary is an object of ‘Integer’ class whose default value is null. Integer is a class which contains methods like ‘parseInt()’ and store references to integer objects on the other hand, ‘int’ is a primitive data type that defines the type of instance variables. Thus, this option is the correct choice.

#### Q3: Find Output - III

Predict the output of the code given below:

**class** **Display** {

**public** **Display**(**int** i) {

System.out.println("ONE");

}

**public** **Display**() {

**this**(**1**);

System.out.println("TWO");

}

**void** **Display**() {

Display(**1**);

System.out.println("THREE");

}

**void** **Display**(**int** i) {

System.out.println("FOUR");

}

}

**public** **class** **Source** {

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

Display obj = **new** Display();

obj.Display();

}

}

Ans: ONE  
TWO  
FOUR  
THREE

**✓ Correct**

**Feedback:**

In the given code, you create an object of the class Display in the ‘main’ method and when you create it, the constructor of the class Display is automatically invoked. Since no argument is passed, the constructor **public Display()** will be invoked with no arguments. The constructor will pass an integer argument to another overloaded constructor which will display ‘ONE’ and then ‘TWO’. Another statement in the main method calls the Display method with an object of the class Display. The first statement of the method ‘void Method’ passes an integer 1 in the method and calls another overloaded method which displays ‘FOUR’ and then ‘THREE’.

3/3 = 30 Marks

# Graded Questions - II

#### Q1: Object-oriented programming

As per the concept of classes and objects, what changes you will have to make in the following code to make it more object-oriented ?

public class ClassesAndObjects {

public static void main(String[] args) {

Ball.spinAndBounce();

}

}

class Ball {

public String state;

public static void bounce() {

state = "bouncing";

System.out.println("Ball is bouncing");

}

public static void spin() {

state = "spinning";

System.out.println("Ball is spinning");

}

public static void spinAndBounce() {

spin();

bounce();

}

}

Ans: All static methods and variables in the Ball have to be converted into non-static ones and should be called using Ball b = new Ball(); & b.spinAndBounce(); statements.

**✓ Correct**

**Feedback:**

To make this code more object oriented, we should create the objects of Ball class, because if one object of the Ball class calls the bounce() method and sets its state variable to “bouncing”, it would not change the state variable of the other objects of the Ball class. Since static methods cannot access non-static variables or methods, we need to make bounce(), spin(), and spinAndBounce() methods non-static.

#### Q2: Calling methods on objects

What would this code print?

public class Main {

public static void main(String[] args) {

TestClass c1 = new TestClass(2, 5.2);

TestClass c2 = new TestClass(3, 7.5);

System.out.println(c1.a + ", " + c1.b);

}

}

class TestClass {

private int a;

public double b;

public TestClass(int first, double second) {

this.a = first;

this.b = second;

}

}

Ans: Will show an error

**✓ Correct**

**Feedback:**

‘a’ is a private variable declared inside the TestClass, so it cannot be accessed from the main class. So this will show an error.

#### Q3: Read-only objects

**Mutable objects** can have their fields changed after construction. **Immutable objects**(**also called read-only objects**) are objects whose state cannot change after construction.

How can you create the read-only objects?

Ans: By providing parameterised constructor, getters and no setters

**✓ Correct**

**Feedback:**

Parameterised constructors will help you initialise variables while creation of objects. Getters would help you read the values stored in instance variables of your objects. Not providing setters would ensure that there will be no changes to the state of your objects at any point of time after construction, hence resulting in immutable (or read-only) objects.

4/4 = 40 marks

# Graded Questions - III

#### Q1: Calling a method

public class Robot {

// Other code here

public Robot() {

// default constructor code here

}

// draw a line from current location to the location specified

public void passLocation(int x, int y) {

// more code

}

}

Which is the right way to call the passLocation method above?

Ans: Robot r = new Robot( );r.passLocation(5, 5);

**✓ Correct**

**Feedback:**

This is the correct notation to create a new object r of Robot class. Also, the passLocation method can be called by the statement r.passLocation(5, 5), which has a void return type.

#### Q2: Calling methods on objects

public class Main {

public static void main(String[] args) {

TestClass c1 = new TestClass(2, 5.2);

TestClass c2 = new TestClass(3, 7.5);

System.out.println(c1.a + ", " + c1.b);

}

}

class TestClass {

public int a;

public double b;

public TestClass(int first, double second) {

this.a = first;

this.b = second;

}

}

What would this code print?

Ans: 2, 5.2

**✓ Correct**

**Feedback:**

Here you are creating two objects c1 and c2, and then reading the variables of c1. The c1.a variable would return 2 and c1.b would return 5.2.

#### Q3: Methods of a class

Consider the following method play, and assume that appropriate import statements are there for classes used in this function.

public void play () {

Dog tom = new Dog();

Cat shian = new Cat();

tom.bite(4);

shian.jump(5, fred);

String greet = “I am hungry";

tom.bark(greet, 3);

Which methods belong to the Dog class ?

**More than one option can be correct in this case.**

Ans: bark

**✓ Correct**

**Feedback:**

Here, tom is an object of the Dog class. So, whichever methods are called using the object tom are methods of the Dog class. So, bark would be a method of the Dog class.

bite

**✓ Correct**You missed this!

**Feedback:**

Here, tom is an object of the Dog class. So, whichever methods are being called using the object tom are the methods of the Dog class. So, bite would be a method of the Dog class.

2/3 = 20Marks