Scheme of Evaluation
Internal Assessment Test 2 – June 2022

Sub:	Programming in Java							**Code:**	18CS653
Date**:**	10/6/2022	Duration:	90mins	Max Marks:	50	**Sem:**	VI	**Branch:**	EEE/MECH

Note: Answer Any five full questions.

Question #		**Description**	**Marks Distribution**		**Max Marks**
1	a)	With suitable example, explain the following jump statements i) continue ii) break continue: syntax and example break: syntax and example	3M 3M	6M	10M
1	b)	Explain the different access specifiers in Java. Public Private Default Protected	1M*4	4M	
2	a)	Differentiate between method overloading and overriding with suitable example. Any four differences with example	1M*4	4M	10M
2	b)	Create a java class called Student with following details as variable (USN, Name, Branch, Phone number). Write a Java program to create a student object and print USN, Name, Branch, Phone number with suitable message. Class definition Instance variable initialization Creating objects Display information	1M 2M 1M 2M	6M	
3	a)	What is super? Explain with suitable example Super usage		6M	10M

		Example	1M 5M		
3	b)	What is a class? How an object is created for the class? Class definition Object creation	 1M 3M	4M	
4	a)	Explain the Garbage Collection in Java. Garbge collection usage Its working	2M 3M	5M	10M
4	b)	Explain the following i)final ii) finalize() Final – syntax and example Finalize()- syntax and example	3M 3M	5M	
5	a)	What are the salient features of constructors? Explain different types of constructors in Java with suitable example. Features of constructors Default constructor Parameterized constructor	4M 3M 3M	10M	10M
6	a)	List the various types of inheritance in Java and explain with suitable example Single inheritance Multilevel inheritance Hierarchical inheritance Multiple inheritance Hybrid inheritance	2M*5	10M	10M

Q.1 With suitable example, explain the following jump statements
i) continue ii) break

break : breaks the loop

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

Output:

```
0
1
2
3
```

Continue: skips the particular iteration

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

Output:

```
0
1
2
3
5
6
7
8
9
```

Q. 1b) Explain the different access specifiers in Java.

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.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Q. 2 a) Differentiate between method overloading and overriding with suitable example.

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Java Method Overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

Q. 2b) Create a java class called Student with following details as variable (USN, Name, Branch, Phone number). Write a Java program to create a student object and print USN, Name, Branch, Phone number with suitable message.

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 String USN;//field or data member or instance variable
 String name;
String branch;
int phone;
 void displayInformation(){System.out.println("My usn is"+ USN+" "+name+ " " +branch + " " +phone); }
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();

S1.USN="1cr19is001";
S1.name= "Rahul";
S1.branch="ise";
S1.phone=1234;
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Output:

Q. 3 a) What is super? Explain with suitable example

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.

2. super can be used to invoke immediate parent class method.

   super() can be used to invoke immediate parent class constructor

super is used to invoke parent class constructor.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

Q. 3 b) What is a class? How an object is created for the class?

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

> **An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Create an object called "myObj" and print the value of x:

```
public class Myclass {
  int x = 5;

  public static void main(String[] args) {
    Myclass myObj = new Myclass();
    System.out.println(myObj.x);
  }
}
```

Q.4a) Explain the Garbage Collection in Java.

Since objects are dynamically allocated by using the new operator, you might be wondering how such objects are destroyed and their memory released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator. Java takes a different approach; it handles ideal location for you automatically.

The technique that accomplishes this is called garbage collection. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++. Garbage collection only occurs sporadically (if at all) during the execution of your program. It will not occur simply because one or more objects exist that are no longer used. Furthermore, different Java run-time implementations will take varying approaches to garbage collection, but for the most part, you should not have to think about it while writing your programs.

Q 4b) Explain the following
i)final ii) finalize()

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

1) Java final variable

```
1.  class Bike9{
2.   final int speedlimit=90;//final variable
3.   void run(){
4.    speedlimit=400;
5.   }
6.   public static void main(String args[]){
7.   Bike9 obj=new  Bike9();
8.   obj.run();
9.   }
10. }//end of class
```

Output:compile time error

The finalize( ) Method

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called finalization. By using finalization, you can define

specific actions that will occur when an object is just about to be reclaimed by the garbage collector.
To add a finalizer to a class, you simply define the finalize( ) method. The Java run time calls that method whenever it is about to recycle an object of that class. Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the Java run time calls the finalize( ) method on the object.
The finalize( ) method has this general form:

```
protected void finalize( )
{
// finalization code here
}
```

Q. 5a) What are the salient features of constructors? Explain different types of constructors in Java with suitable example.

Constructors cannot be private.

A constructor cannot be abstract, static, final, native, strictfp, or synchronized

A constructor can be overloaded.

Constructors cannot return a value.

Constructors do not have a return type; not even void.

An abstract class can have the constructor.

Constructors name must be similar to that of the class name inside which it resides.

Constructors are automatically called when an object is created.

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)

2. Parameterized constructor

Default Constructor – A constructor that accepts no parameter is called Default Constructor. It is not necessary to have a constructor block in your class definition. If you don't explicitly write a constructor, the compiler automatically inserts one for you.

Example illustrating Default Constructor in Java:

Example illustrating Default Constructor in Java:

```
public class Edureka{
Edureka()
{ System.out.println("I am a constructor");}
public static void main(String args[]){
Edureka obj = new Edureka();
}
}
```

Output: I am a constructor

Parameterized Constructor – A constructor is called Parameterized Constructor when it accepts a specific number of parameters. To initialize data members of a class with distinct values.

Example illustrating Parameterized Constructor:

Example illustrating Parameterized Constructor:

```
public class Edureka{
String studentName;
int studentAge;
//constructor
Edureka(String name, int age){
studentName = name;
studentAge = age;
}
void display(){
System.out.println(studentName+ " "+studentAge);
}
public static void main(String args[])
{
Edureka myObj = new Edureka("Manan" , 19);
myObj.display();
}
}
```

Output: Manan-19

Q.6a) List the various types of inheritance in Java and explain with suitable example
Below are the different types of inheritance which is supported by Java.
1. **Single Inheritance :** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.

```
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 TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10. d.bark();
11. d.eat();
12. }}
```

Output:

```
barking...
eating...
```

- **Multilevel Inheritance :** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.

```
•
• 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...

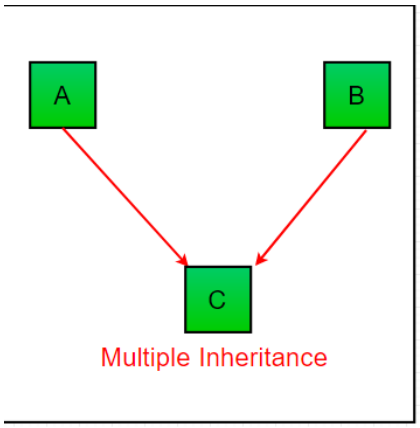
3. • **Hierarchical Inheritance :** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.

```
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 Cat extends Animal{
8. void meow(){System.out.println("meowing...");}
9. }
10. class TestInheritance3{
11. public static void main(String args[]){
12. Cat c=new Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}
```

Output

meowing...
eating...

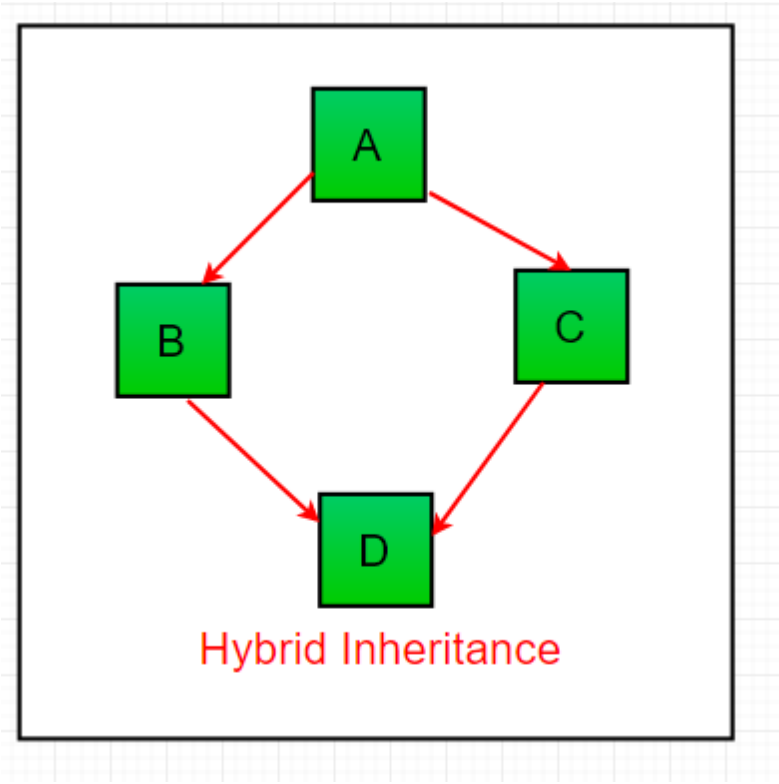
4. **Multiple Inheritance (Through Interfaces) :** In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B




```
class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
8.
9. public static void main(String args[]){
10. C obj=new C();
11. obj.msg();//Now which msg() method would be invoked?
12. }
13. }
```

Output

`Compile Time Error` **Hybrid Inheritance(Through Interfaces) :** It is a mix of two or more of the above types of
inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.


A
B
C
D
Hybrid Inheritance