Scheme Of Evaluation

Internal Assessment Test 2– June 2021

Sub:	Programming in Java							**Code:**	18CS653
Date**:**	24/06/2021	Duration:	90mins	Max Marks:	50	**Sem:**	VI	**Branch:**	ECE A,B ,C &D

Note: Answer Any Five Questions

Question #		**Description**	**Marks Distribution**	**Max marks**	**Total**
1	a	What is super? Explain use of super with example.			10
		Use of super-super,super()	3	6	
		Example	3		
	b	Write a java program to display the words of a sentence by changing the case of first letter of each word to uppercase.			
		Input	1	4	
		Split the words & Change the case	2		
		Display	1		
2	a	What is constructor? Explain its types with example.			10
		Constructor and its use	2	6	
		Types-explanation	2		
		Example	2		
	b	Explain the following methods with example a. charAt() b.regionMatches()			
		charAt()-Use & Example	2	4	
		regionMatches()-Use & Example	2		
3	a	Explain dynamic method dispatch in JAVA.		4	10
		Explanation	4		
	b	A class Shape is defined with two overloading constructors in it and a calculate method to calculate area.Create aclass Test1 which inherits the class Shape. The class Test1 should include two overloading constructors as appropriate for some object instantiation shown in main( ) method. You should define the constructors using the super class constructors. Also, override the method calculate( ) in Test1 to calculate the volume of a Shape.			
		Creation of Shape class with specified methods	2	6	
		Creation of Test1 class with specified methods	2		
		Invoking methods	2		
4	a	Distinguish between Method Overloading and Method Overriding. Write a java program to illustrate the use of method overriding			10
		Differences	3	6	
		program	3		

	b	Write a program in java to search a key String.			
		Taking Input & displaying output	2	4	
		Code to search	2		
5	a	Create a java class Student with following details as variables (USN,Name,Branch,Phone Number). Write a java program to create n student object and print USN,Name,Branch and Phone Number with suitable message.			10
		Creating class Student	2	6	
		Creating n student object	2		
		Displaying	2		

	B	Predict the output for the following
		i)final int x=100;
		System.out.println(x++);
		ii) class Main1 {
		public static void main(String s[]) {
		System.out.println("cmrit");
		}
		static
		{
		System.out.println("Welcome");
		}
		}			

		i) output	2	4	
		ii) output	2		
6	a	Explain the following a. Use of this keyword b.Garbage collection		4	10
		Use of this keyword	2		
		Garbage collection	2		
	b	With a java program to show how final keyword is used with variables and also to prevent inheritance and overriding.			
		a) Illustrating final variable	2	6	
		b) Illustrating final methods	2		
		c) Illustrating final classes	2		

1. **a.What is super? Explain use of super with example.**

super is the keyword using which the subclass refers to its immediate superclass. super() always refers to the superclass immediately above the calling class. This is true even in multilevel hierarchy. super() must always be the first statement executed inside a subclass constructor.

The keyword super has two uses

1. To invoke the superclass' constructor.

2. To access a member of the superclass that has been hidden by member of a subclass.

1. Using super to call superclass constructors:
A subclass can call a constructor defined by its superclass by use of the following form of superclass
super(arg-list);
Here, arg-list specifies any arguments needed by the constructor in the superclass. super() must always be the first statement executed inside a subclass' constructor.

2. Second use of super to access a member of the superclass:
The second form of super always refers to the superclass of the subclass in which it is used.
The general form is
super.member
Here, member can be either a method or an instance variable. The second form of super is most applicable to situations in which member names of a subclass hide members by the same name in the superclass.

Example program to demonstrate both uses of super

```
// create a superclass
class A
{
   int    i, j ;
   A (int a, int b)
          {
      i = a;
      j = b;
   }
   void show()
          {
      System.out.println(" i =  " + i);
      System.out.println("j = " + j);
   }
}

// Define subclass
class B extends A
{
   int k;
   B(int a, int b, int c)
          {
      super(a,b);  // use 1 of superclass
      k = c;
   }
          void show()
          {
      super.show();   // calls show() of class a, use 2 of superclass
      System.out.ptinln("k = " + k);
```

```
   }
}

class Demo
{
   public static void main(String args[])
        {
      B obj = new B(1,2,3);
      obj.show();
   }
}
```

Output:
$javac Demo.java
$java Demo
i = 1
j = 2
k = 3

1.b.Write a java program to display the words of a sentence by changing the case of first letter of each word to uppercase.

```
import java.util.*;
 public class Exercise58 {
     public static void main(String[] args){
       Scanner in = new Scanner(System.in);
     System.out.print("Input a Sentence: ");
       String line = in.nextLine();
       String upper_case_line = "";
        Scanner lineScan = new Scanner(line);
          while(lineScan.hasNext()) {
              String word = lineScan.next();
              upper_case_line                                    +=
Character.toUpperCase(word.charAt(0)) + word.substring(1) + "
";
          }
      System.out.println(upper_case_line.trim());
    }
      }
```

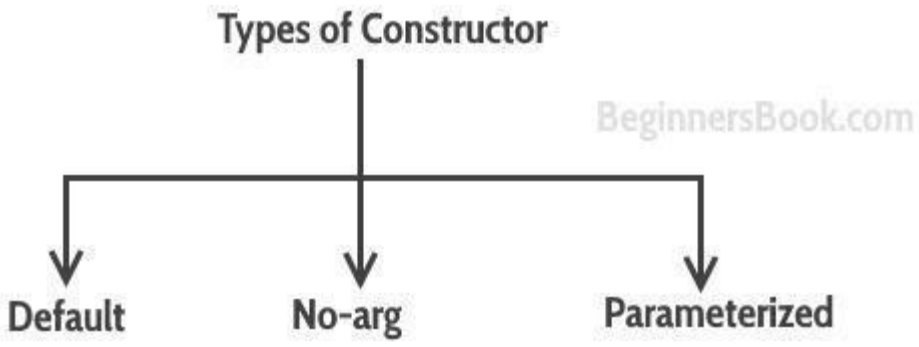
2.a. What is constructor? Explain its types with example.

Constructor is a special type of member method which is invoked automatically when the object gets created. Constructors are used for object initialization. They have the same name as that of the class. Constructors are automatically called immediately after the object is created. Since they are called automatically, there is no return type, not even void. Constructors may or may not take parameters.

- Every class is provided with a ***default constructor*** which initializes all the data members to respective *default values*. (Default for numeric types is zero, for character and strings it is null and default value for Boolean type is false.)
- In the statement *classname ob= new classname();* the term *classname()* is actually a constructor call.
- If the programmer does not provide any constructor of his own, then the above statement will call default constructor.
- If the programmer defines any constructor, then default constructor of Java cannot be used.
- So, if the programmer defines any parameterized constructor and later would like to create an object without explicit initialization, he has to provide the default constructor by his own. For example, in the below program, if we remove ordinary constructor, the statements like Box b1=new Box(); will generate error. To avoid the error, we should write a default constructor like – Box(){ } Now, all the data members will be set to their respective default values.

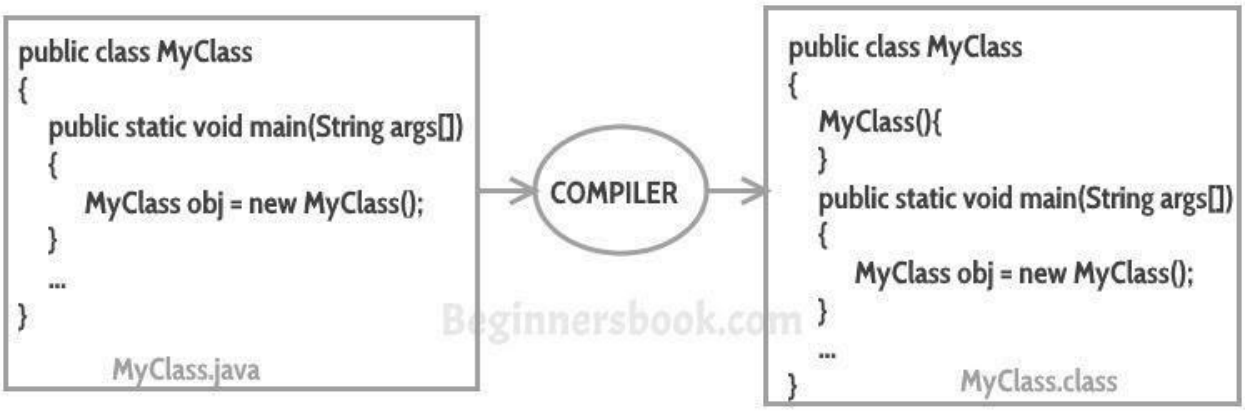
Different types of constructor are

1. Default constructor
2. Non parameterized or No-arg or zero argument constructor
3. Parameterized constructor



1. Default constructor

If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:



If you implement any constructor then you no longer receive a default constructor from Java compiler.

2. No-arg constructor:

Constructor with no arguments is known as **no-arg constructor**. The signature is same as default constructor, however body can have any code unlike default constructor where the body of the constructor is empty.

Although you may see some people claim that that default and no-arg constructor is same but in fact they are not, even if you write **public Demo() { }** in your class Demo it cannot be called default constructor since you have written the code of it.

3. Parameterized constructor

Constructor with arguments(or you can say parameters) is known as Parameterized constructor

```
// program to describe the types of constructor.
class Box
{
   double w, h, d;

   Box() //non parameterized constructor
   {
     w=h=d=5;
   }

   Box(double wd, double ht, double dp) //parameterized constructor
   {
     w=wd;
     h=ht;
     d=dp;
   }
   double volume()
```

```
   {
     return w*h*d;
   }

}

class BoxDemo
{
   public static void main(String args[])
   {
     Box mybox1 = new Box( );
     Box mybox2 = new Box(2, 2, 2);

     double vol;
     // get volume of first box
     vol = mybox1.volume();
     System.out.println("Volume is " + vol);
     // get volume of second box
     vol = mybox2.volume();
     System.out.println("Volume is " + vol);
   }
}
```

Output:

```
$ javac BoxDemo.java
$ java BoxDemo
Volume is 125.0
Volume is 8.0
```

2. **b. Explain the following methods with example**
   a.charAt()
   b.regionMatches()

charAt():

To extract a single character from a String, you can refer directly to an individual character via the charAt( ) method. It has this general form:

char charAt(int where)

Here, where is the index of the character that you want to obtain. The value of where must be nonnegative and specify a location within the string. charAt( ) returns the character at the specified location.

For example,

```
char ch;
ch = "abc".charAt(1);
```

assigns the value “b” to ch.

regionMatches( )

The regionMatches( ) method compares a specific region inside a string with another specific region in another string. There is an overloaded form that allows you to ignore case in such comparisons.

Here are the general forms for these two methods:
boolean regionMatches(int startIndex, String str2,
int str2StartIndex, int numChar

boolean regionMatches(boolean ignoreCase,
int startIndex, String str2,
int str2StartIndex, int numChars)

For both versions, startIndex specifies the index at which the region begins within the invoking String object. The String being compared is specified by str2. The index at which the comparison will start within str2 is specified by str2StartIndex. The length of the substring being compared is passed in numChars. In the second version, if ignoreCase is true, the case of the characters is ignored. Otherwise, case is significant.

3. a. Explain dynamic method dispatch in JAVA.

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism.

A superclass reference variable can refer to a subclass object. Java uses this fact to resolve calls to overridden methods at run time. When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed. Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Here is an example that illustrates dynamic method dispatch:

```
// Dynamic Method Dispatch
class A {
        void callme() {
                System.out.println("Inside A's callme method");
        }
}

class B extends A {
        // override callme()
        void callme() {
```

```
                System.out.println("Inside B's callme method");
        }
}

class C extends A {
        // override callme()
        void callme() {
                System.out.println("Inside C's callme method");
        }
}

class Dispatch {
        public static void main(String args[]) {
                A a = new A(); // object of type A
                B b = new B(); // object of type B
                C c = new C(); // object of type C
                A r; // obtain a reference of type A
                r = a; // r refers to an A object
                r.callme(); // calls A's version of callme
                r = b; // r refers to a B object
                r.callme(); // calls B's version of callme
                r = c; // r refers to a C object
                r.callme(); // calls C's version of callme
        }
}
```

The output from the program is shown here:

```
Inside A’s callme method
Inside B’s callme method
Inside C’s callme method
```

This program creates one superclass called A and two subclasses of it, called B and C. Subclasses B and C override callme( ) declared in A. Inside the main( ) method, objects of type A, B, and C are declared. Also, a reference of type A, called r, is declared. The program then in turn assigns a reference to each type of object to r and uses that reference to invoke callme( ). As the output shows, the version of callme( ) executed is determined by the type of object being referred to at the time of the call. Had it been determined by the type of the reference variable, r, you would see three calls to A’s callme( ) method.

3.b. A class Shape is defined with two overloading constructors in it and a calculate method to calculate area.Create aclass Test1 which inherits the class Shape. The class Test1 should include two overloading constructors as appropriate for some object instantiation shown in main( ) method. You should define the constructors using the super class constructors. Also, override the method calculate( ) in Test1 to calculate the volume of a Shape.

```
import java.util.Scanner;

class Shape{
```

```
	double l,b;
	Shape()
	{
		l=b=2.0;
	}
	Shape(double x, double y )
	{
		l=x;
		b=y;
	}
	double area()
	{
		return l*b;
	}
}
class Test1 extends Shape
{
	double h;
	Test1()
	{
	    super();
		h=2.0;

	}
	Test1(double x,double y,double z)
	{
		super(x,y);
		h=z;
	}
	double area()
	{
		double a=super.area();
		return a*h;
	}
}
public class Program2 {

	public static void main(String[] args) {
		Test1 t= new Test1();
		System.out.println("Area of Obj1"+ t.area());
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter l,b,h values");
		double l=sc.nextDouble();
		double b=sc.nextDouble();
		double h=sc.nextDouble();
		Test1 t1= new Test1(l,b,h);
		System.out.println("Area of Obj2"+t1.area());

	}

}
```

4. **a. Distinguish between Method Overloading and Method Overriding. Write a java program to illustrate the use of method overriding**

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.

```
class AA {
       int i, j;
       AA (int a, int b) {  i = a;
       j = b;
       }
       // display i and j
       void show() {
       System.out.println("i = " + i + "j = " + j);
       }
}
class BB extends AA
{
       int k;
   BB (int a, int b, int c)
   {
          super(a, b);
   k = c;
   }
   // Display k - this overrides show() in A
   void show () {
         super.show();
         System.out.println("k = " + k);
   }
}

public class Overiding {

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

```
        // TODO Auto-generated method stub
        BB subOb = new BB (1, 2, 3);

            subOb.show();    // this calls show in B
    }

}
```

4.b.Write a program in java to search a key String.

```
class SearchKey
{
    public static void main(String args[])
    {
        String
s[]={"Now","is","the","time","for","all","good","men","to","come","to","the","aid","of","their
","country"};
                String key="Country";
                boolean flag=false;
        for(int i=0;i<s.length-1;i++)
        {
            for(int j=0;j<s.length-1-i;j++)
            {
                if(s[j].compareTo(s[j+1])>0) // compare s[j] with s[j+1]
                {
                    String temp=s[j];
                    s[j]=s[j+1];
                    s[j+1]=temp;
                }
            }
        }
        for(int i=0;i<s.length-1;i++)
        {
            if (s[i].compareTo(key) == 0)
            {
                flag=true;
                break;
            }
        }
        if(flag == true)
        {
            System.out.println("Key found ");
        }
        else
        {
            System.out.println("Key not found");
        }
    }
}
```

5. a. Create a java class Student with following details as variables (USN,Name,Branch,Phone Number). Write a java program to create n student object and print USN,Name,Branch and Phone Number with suitable message.

```
import java.util.Scanner;
class Student12
 {
        String USN;
```

```
            String Name;
            String branch;
            int phone;
void insertRecord(String reg,String name, String brnch,int ph) {
USN=reg;
Name=name;
branch=brnch;
phone=ph;
}

void displayRecord()
{
System.out.println(USN+" "+Name+" "+branch+" "+phone);
}

public static void main(String args[])
{
Student12 s[]=new Student12[100];
Scanner sc=new Scanner(System.in);
System.out.println("enter the number of students");
int n=sc.nextInt();
for(int i=0;i<n;i++)
      s[i]=new Student12();
for(int j=0;j<n;j++)
{
       System.out.println("enter the usn,name,branch,phone");
          String USN=sc.next();
          String Name=sc.next();
         String branch=sc.next();
          int phone=sc.nextInt();
    s[j].insertRecord(USN,Name,branch,phone);

}

for( int m=0;m<n;m++)
{
  s[m].displayRecord();
}

}
}
```

5.b. Predict the output for the following

i)final int x=100;
System.out.println(x++);

ii) class Main1 {
public static void main(String s[]) {
System.out.println("cmrit");
}
static
{
System.out.println("Welcome");
}
}

i) Error
ii) Welcome
cmrit

6.a. Explain the following

a. **Use of this keyword**
b. **Garbage collection**

a.Use of this keyword

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword. this can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class' type is permitted.

To better understand what this refers to, consider the following version of Box( ):

// A redundant use of this. Box(double w, double h, double d) { this.width = w;
this.height = h;
this.depth = d;
}

This version of Box( ) operates exactly like the earlier version. The use of this is redundant, but perfectly correct. Inside Box( ), this will always refer to the invoking object. While it is redundant in this case, this is useful in other contexts, one of which is explained in the next section.

b.Garbage collection

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 deallocation 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.

6.b. Write a java program to show how final keyword is used with variables and also to prevent inheritance and overriding.

```
final class FinalClass {
        final int a=10;
  final public void display() {
```

```
    System.out.println("This is a final method.");
    System.out.println(a);
    System.out.println(a++);// can't change value of final variable
  }
}

// try to extend the final class
class Main extends FinalClass //Cant extend final class
{
    public  void display()// can't override final method
    {
    System.out.println("The final method is overridden.");
  }

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