USN | | | | | | | | | | |

Internal Assessment Test 2 – Oct. 2018

Sub:	Programming in JAVA					Sub Code:	15CS561	Branch:	ECE/EEE
Date:	17-10-2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	5 (ALL SEC)		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	Discuss the salient features and types of Constructor with programming example.	[10]	CO3	L2
2	Explain the following: a) Use of this keyword b)Garbage collector c) Finalize ()	[10]	CO3	L1
3	Write a java program which creates class Student (Rollno, Name, Number of subjects, Marks of each subject)(Number of subjects varies for each student) Write a parameterized constructor which initializes roll no, name & Number of subjects and create the array of marks dynamically. Display the details of all students with percentage.	[10]	CO2, CO3	L3
4	What is Inheritance? What are the different types of Inheritance? Explain the use of super with a suitable programming example.	[5+5]	CO1, CO3	L2
5	What is Static data members and Static member functions? Write a simple program to count total number of objects in a class.	[5+5]	CO2	L2

USN | | | | | | | | | | |

Internal Assessment Test 2 – Oct. 2018

Sub:	Programming in JAVA					Sub Code:	15CS561	Branch:	ECE/EEE
Date:	17-10-2018	Duration:	90 min's	Max Marks:	50	Sem / Sec:	5 (ALL SEC)		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	Discuss the salient features and types of Constructor with programming example.	[10]	CO3	L2
2	Explain the following: a) Use of this keyword b) Garbage collector c) Finalize ()	[10]	CO3	L1
3	Write a java program which creates class Student (Rollno, Name, Number of subjects, Marks of each subject)(Number of subjects varies for each student) Write a parameterized constructor which initializes roll no, name & Number of subjects and create the array of marks dynamically. Display the details of all students with percentage.	[10]	CO2, CO3	L3
4	What is Inheritance? What are the different types of Inheritance? Explain the use of super with a suitable programming example.	[5+5]	CO1, CO3	L2
5	What is Static data members and Static member functions? Write a simple program to count total number of objects in a class.	[5+5]	CO2	L2

6	Write a program to implement stack using class with the methods to push and pop the elements.	[10]	CO2, CO3	L3
7	Write a difference between Method overloading and Method overriding. Write a program to calculate the area of the square, rectangle using method overloading.	[5+5]	CO1, CO3	L3
8 (a)	Write a note on Access modifiers in Java.	[5]	CO3	L1
(b)	Explain the use of Abstract with a suitable example.	[5]	CO3	L2

6	Write a program to implement stack using class with the methods to push and pop the elements.	[10]	CO2, CO3	L3
7	Write a difference between Method overloading and Method overriding. Write a program to calculate the area of the square, rectangle using method overloading.	[5+5]	CO1, CO3	L3
8 (a)	Write a note on Access modifiers in Java.	[5]	CO3	L1
(b)	Explain the use of Abstract with a suitable example.	[5]	CO3	L2

Solution

1	**Discuss the salient features and types of Constructor with programming example.**	**[10]**

- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor
- A constructor initializes the instance variables of an object.
- It is called automatically immediately after the object is created but before the new operator completes.
  1) it is syntactically similar to a method:
  2) it has the same name as the name of its class
  3) it is written without return type, not even void; the default return type of a class

In Box example the dimensions of a box are automatically initialized when an object is constructed.

```
class Box {
    double width;
    double height;
    double depth;
    Box() {
        System.out.println("Constructing Box");
        width = 10; height = 10; depth = 10;
        }
        double volume() {
            return width * height * depth;
            }
        }
public static void main(String args[]) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
        vol = mybox1.volume();
        System.out.println("Volume is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
        }
    }
```

it generates the following results:

```
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
```

constructor for the class is being called

Parameterized Constructor: The constructor which takes parameter while creating the object of a particular class. Here Box constructor is having width, height and depth parameters which will be initialized at the time of object creation which the supplied values.

```
class Box {
    double width;
    double height;
    double depth;
    Box(double w, double h, double d) {
        width = w; height = h; depth = d;
            }
    double volume(){
        return width * height * depth;
            }
        }
class BoxDemo {
    public static void main(String args[]) {
        Box mybox1 = new Box(10, 20, 15);
```

```
        Box mybox2 = new Box(3, 6, 9);
        double vol;
        vol = mybox1.volume();
        System.out.println("Volume is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
}
}
```
Volume is 3000.0
Volume is 162.0

2	**Explain the following:** **a) Use of this keyword** **b) Garbage collector** **c) Finalize ()**	**[10]**

This Keyword:

- Sometimes a method will need to refer to the object that invoked it
- this is always a reference to the object on which the method was invoked
- Typically used to
  - Avoid variable name collisions
  - Pass the receiver as an argument
  - Chain constructors
  - Keyword this allows a method to refer to the object that invoked it.
- It can be used inside any method to refer to the current object:

```
Box(double w, double h, double d) {
        this.width = w;
        this.height = h;
        this.depth = d;
        }
```

From the main function we can construct the Box object by:

```
public static void main(String args[]) {
        Box mybox1 = new Box(10,20,30);
        Box mybox2 = new Box(myBox1);  //here the myBox1 object has been passed
```

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. section. 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. when a local variable has the same name as an instance variable, the local variable hides the instance variable.

Garbage Collection:

- Garbage collection is a mechanism to remove objects from memory when they are no longer needed.
- Garbage collection is carried out by the garbage collector:

1) The garbage collector keeps track of how many references an object has.
2) 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.
3) It removes an object from memory when it has no longer any references.
4) Thereafter, the memory occupied by the object can be allocated again.
5) The garbage collector invokes the finalize method.

Finalize() :

- .
- Sometimes an object will need to perform some action when it is destroyed. To handle such situations, Java provides a mechanism called finalization
- the finalize method is invoked just before the object is destroyed:
- By using finalization, we 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, we 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.
- implemented inside a class as:
- protected void finalize() { … }
- implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out.
- Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class
- It is important to understand that finalize( ) is only called just prior to garbage collection.

3 **Write a java program which creates class Student (Rollno, Name, Number of subjects, Marks of each subject)(Number of subjects varies for each student) Write a parameterized constructor which initializes roll no, name & Number of subjects and create the array of marks dynamically. Display the details of all students with percentage.** **[10]**

```
import java.io.*;
class Student
  {
    int rollno;
    String name;
    int number_of_subjects;
    int mark[];
   Student(int roll,String stud_name,int noofsub) throws IOException
     {
       rollno=roll;
       name=stud_name;
       number_of_subjects= noofsub;
       getMarks(noofsub);
     }
  public void getMarks(int nosub ) throws IOException
    {
       mark=new int[nosub];
       BufferedReader br= new BufferedReader (new InputStreamReader(System.in));
       for (int i=0; i<nosub;i++)
           {
             System.out.println("Enter "+i+"Subject Marks.:=> ");
             mark[i]=Integer.parseInt(br.readLine());
             System.out.println("");
           }
     }
  public void calculateMarks()
   {
     double percentage=0;
     String grade;
     int tmarks=0;
     for (int i=0;i<mark.length;i++)  {
        tmarks+=mark[i];
     }
   percentage=tmarks/number_of_subjects;
   System.out.println("Roll Number :=> "+rollno);
   System.out.println("Name Of Student is :=> "+name);
   System.out.println("Number Of Subject :=> "+number_of_subjects);
   System.out.println("Percentage Is :=> "+percentage);
   if (percentage>=70)
     System.out.println("Grade Is First Class With Distinction ");
   else if(percentage>=60 && percentage<70)
     System.out.println("Grade Is First Class");
   else if(percentage>=50 && percentage<60)
    System.out.println("Grade Is Second Class");
    else if(percentage>=40 && percentage<50)
     System.out.println("Grade Is Pass Class");
   else
     System.out.println("You Are Fail");
```

```
        }
    }

class StudentDemo  {
    public static void main(String args[])throws IOException {
        int rno,no,nostud;
        String name;
        BufferedReader br= new BufferedReader (new InputStreamReader(System.in));
        System.out.println("Enter How many Students:=> ");
        nostud=Integer.parseInt(br.readLine());
       Student s[]=new Student[nostud];
       for(int i=0;i<nostud;i++)  {
          System.out.println("Enter Roll Number:=> ");
          rno=Integer.parseInt(br.readLine());
          System.out.println("Enter Name:=> ");
          name=br.readLine();
          System.out.println("Enter No of Subject:=> ");
          no=Integer.parseInt(br.readLine());
          s[i]=new Student(rno,name,no);
                }
         for(int i=0;i<nostud;i++)  {
             s[i].calculateMarks();
                          }
                }
           }
```

4 **What is Inheritance? What are the different types of Inheritance? Explain the use of super with a suitable programming example.** **[5+5]**

Inheritance:

- Allows the creation of hierarchical classifications.
- For creating inheritance:

nheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes.

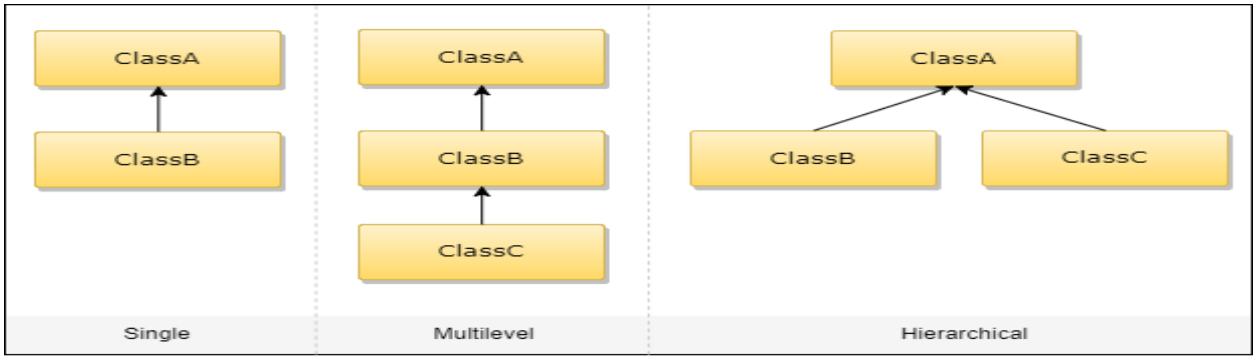
A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

- A class that is inherited is called a superclass. The class that does the inheriting is called a subclass.
- Subclass is a specialized version of a superclass.
- To inherit a class, you simply incorporate the definition of one class into another by using the extends keyword
- The general form of a class declaration that inherits a superclass is shown here:

```
Class subclass-name extends superclass-name {
        // body of class
}
```

A is a superclass for B (In next slide eg.), it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself.

- Further, a subclass can be a superclass for another subclass.



Java Inheritance

Super:

- super is a keyword.
- It is used inside a sub-class method definition to call a method defined in the super class. Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.
- It is also used by class constructors to invoke constructors of its parent class.
- Super keyword are not used in static Method.

Two general form of super:

- The first calls the superclass' constructor.
- The second is used to access a member of the superclass that has been hidden by a member of a subclass
- A subclass can call a constructor defined by its superclass by use of the following form of super:

```
super(arg-list);     //arg-list specifies any arguments needed by the constructor in the superclass.
```

- super( ) must always be the first statement executed inside a subclass' constructor
- Example:

a) First use of super : to call super class constructor:

```
class Box{
  double height;
  double widht;
  double depth;
 Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;

  double vol()
      {
      // body of the function
    }
}
```

```
class BoxWeight entends Box{
          double mass;
          BoxWeight(double w, double h, double d, double m) {
            super(w,h,d);
          mass=m ;
       }
 }
```

Here the value of height,width, depth in Boxweight contructor has been initialized by super class constructor with "super".

b) Second use of super : to call super class constructor:

- This second form of super is most applicable to situations when member names of a subclass hide members by the same name in the superclass
- This usage has the following general form:

```
super.member                    //Here, member can be either a method or an instance variable.
```

```
class A {
     int i;
    }
// Create a subclass by extending class A.
```

```
class B extends A {
        int i; // this i hides the i in A
        B(int a, int b) {
           super.i = a; // i in A
           i = b; // i in B
           }
 void show() {
           System.out.println("i in superclass: " + super.i);

           System.out.println("i in subclass: " + i);
                }
       }
```

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

Output:
i in superclass: 1
i in subclass: 2

5 What is Static data members and Static member functions? Write a simple program to count total number of objects in a class. [5+5]

- ❖ It is possible to create a member that can be used by itself, without reference to a specific instance.
- ❖ To create such a member, precede its declaration with the keyword ***static***
- ❖ The most common example of a static member is main( ). main( ) is declared as static because it must be called before any objects exist.
- ❖ Instance variables declared as static are, essentially, global variables. When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable.
- ❖ Methods declared as static have several restrictions:
  - They can only call other static methods.
  - They must only access static data.
  - They cannot refer to this or super in any way. (The keyword super relates to Inheritance.)
- ❖ If you need to do computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.
- ❖ Outside of the class in which they are defined, static methods and variables can be used independently of any object. To do so, you need only specify the name of their class followed by the dot operator
  classname.method( )

```
package students;

class Students
{
public String name;   //The first 2 variables are nonstatic variables, so they are linked to the objects.
public int age;
public static int numberofobjects=0;  // static variable of int type named numberofobjects. We initially set to 0 at the start of the
                                          program, because no objects have been instantiated yet.

Students (String name, int age)       //Constructor to initialize the pblic and non static variables of the class
{
   this.name= name;
   this.age= age;
   numberofobjects++;         //In the third line, we take the static variable we created called numberofojbects and increment by 1 in
                               the constructor.  Each time an object is created, the constructor is automatically called and the static
                               variable numberofobjects increases by 1. This is how we are able to keep track of all objects created
                               in the class

}    }
```

```
package students;
  class studentsdemo   {
    public static void main(String[]args)   {

           Students student1= new Students("Michelle", 7);     Students student2= new Students("Daniel", 8);
           Students student3= new Students("Vanessa", 9);      Students student4= new Students("Ryan", 8);
           System.out.println ("There are " + Students.numberofobjects + " objects in this class");
} }
```

We then create 4 objects of the Students class, student1, student2, student3, and student4.
With each object that we instantiate, we automatically call the constructor method. This method increments the numberofobjects variable by 1. Being that we have created 4 objects, the numberofobjects variable goes to the value of 4.

Output
There are 4 objects in this class

6 Write a program to implement stack using class with the methods to push and pop the elements. [10]

```
// This class defines an integer stack that can hold 10 values.
        class Stack {
             int stck[] = new int[10];
             int tos;
// Initialize top-of-stack
            Stack() {
              tos = -1;
            }
// Push an item onto the stack
        void push(int item) {
             if(tos==9)
               System.out.println("Stack is full.");
            else
              stck[++tos] = item;
            }
// Pop an item from the stack
        int pop() {
           if(tos < 0) {
              System.out.println("Stack underflow.");
              return 0;
                    }
            else
               return stck[tos--];
              }
   }

class TestStack {
        public static void main(String args[]) {
           Stack mystack1 = new Stack();
           Stack mystack2 = new Stack();
// push some numbers onto the stack
            for(int i=0; i<10; i++)
                      mystack1.push(i);
            for(int i=10; i<20; i++)
                      mystack2.push(i);
// pop those numbers off the stack
           System.out.println("Stack in mystack1:");
            for(int i=0; i<10; i++)
                 System.out.println(mystack1.pop());
           System.out.println("Stack in mystack2:");
           for(int i=0; i<10; i++)
               System.out.println(mystack2.pop());
               }
      }
```

7 Write a difference between Method overloading and Method overriding. Write a program to calculate the area of the square, rectangle using method overloading. [5+5]

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }
    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

	Method Overloading	Method Overriding
Definition	In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.	In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.
Meaning	Method Overloading means more than one method shares the same name in the class but having different signature.	Method Overriding means method of base class is re-defined in the derived class having same signature.
Behavior	Method Overloading is to “add” or “extend” more to method’s behavior.	Method Overriding is to “Change” existing behavior of method.
	Overloading and Overriding is a kind of polymorphism.Polymorphism means “one name, many forms”.	
Polymorphism	It is a **compile time polymorphism**.	It is a **run time polymorphism**.
Inheritance	It may or may not need **inheritance** in Method Overloading.	It always requires inheritance in Method Overriding.
Signature	In Method Overloading, methods must have **different signature**.	In Method Overriding, methods must have **same signature**.
Relationship of Methods	In Method Overloading, relationship is there between methods of same class.	In Method Overriding, relationship is there between methods of super class and sub class.
Criteria	In Method Overloading, methods have same name different signatures but in the same class.	In Method Overriding, methods have same name and same signature but in the different class.
No. of Classes	Method Overloading does not require more than one class for overloading.	Method Overriding requires at least two classes for overriding.

```
class OverloadDemo
{
    void area(float x)
    {
        System.out.println("the area of the square is "+Math.pow(x, 2)+" sq units");
    }
    void area(float x, float y)
    {
```

```
        System.out.println("the area of the rectangle is "+x*y+" sq units");
    }
}
class Overload
{
     public static void main(String args[])
  {
     OverloadDemo ob = new OverloadDemo();
     ob.area(5);
     ob.area(11,12);

        }
}
```

Output:

```
the area of the square is 25.0 sq units
the area of the rectangle is 132.0 sq units
```

8 (a) Write a note on Access modifiers in Java. **[5]**

(b) Explain the use of Abstract with a suitable example. **[5]**

Access modifiers in java:

Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. In the 4 types of access modifiers

- **Public**: keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
- **Default**: (No visibility modifier is specified): it behaves like public in its package and private in other packages. Default Public keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
- **Private** : fields or methods for a class only visible within that class. Private members are *not* visible within subclasses, and are *not* inherited.
- **Protected** : members of a class are visible within the class, subclasses and also within all classes that are in the same package as that class

Abstract :

- Sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.
- Figure used in the following example. The definition of area( ) is simply a placeholder
- Java's solution to this problem is the abstract method.
- You can require that certain methods be overridden by subclasses by specifying the abstract type modifier
- These methods are sometimes referred to as subclasser responsibility because they have no implementation specified in the superclass.
- To declare an abstract method, use this general form:
  abstract type name(parameter-list);
- Any class that contains one or more abstract methods must also be declared abstract.
- There can be no objects of an abstract class. That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods.
- Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract

Here in the Fugure class area() is just a placeholder, doesnot have actual definition. The definition of area will be different and meaningfull I every subclasses of Figure(here I Rectangle class)

```
class Figure {
double dim1;
double dim2;
Figure(double a, double b) {
dim1 = a;
dim2 = b;
}
double area() {
System.out.println("Area for Figure is undefined.");
return 0;
}
}
class Rectangle extends Figure {
Rectangle(double a, double b) {
super(a, b);
}
// override area for rectangle
double area() {
System.out.println("Inside Area for Rectangle.");
return dim1 * dim2;
}
            }
```

So now we can rewrite the above definition with the use of abstract.

```
Abstruct class Figure {                  //As the class definition contains abstract method so class also declared as abstract
       double dim1;
       double dim2;
       Figure(double a, double b) {
         dim1 = a;    dim2 = b;
           }
    abstruct  double area()                 //No definition for area() and declared as abstract method
     }

  class Rectangle extends Figure {
        Rectangle(double a, double b) {
       super(a, b);
     }
// override area for rectangle
    double area() {
           System.out.println("Inside Area for Rectangle.");
           return dim1 * dim2;
          }
     }

class FindAreas {
     public static void main(String args[]) {
          Rectangle r = new Rectangle(9, 5);
         System.out.println("Area is " + r.area());
                       }
            }
```