CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | |

Internal Assessment Test - 2

Sub:	**Programming in JAVA(open elective)**							Code:	**15CS561**
Date:	09 / 11 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	V	Branchs	**EC,TC,EE,ME,CV**

Answer Any **FIVE FULL** Questions

		Marks	OBE	
			CO	RBT
1.	What is Method overloading? Explain with help of program.	[10]	CO3	L2
2.	What is Dynamic method dispatch? Explain with suitable example.	[10]	CO3	L2
3.	a) Explain how constructors are called in hierarchy of inheritance with an example.	[05]	CO3	L2
	b) Write a program to illustrate use of 'this' keyword and explain the same.	[05]	CO3	L3
4.	Write java program to implement stack of integer elements and make use of 'static' and 'final' keyword for suitable instances. .	[10]	CO3	L3

CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | |

Internal Assessment Test - 2

Sub:	**Programming in JAVA(open elective)**							Code:	**15CS561**
Date:	09 / 11 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	V	Branchs	**EC,TC,EE,ME,CV**

Answer Any **FIVE FULL** Questions

		Marks	OBE	
			CO	RBT
1.	What is Method overloading? Explain with help of program.	[10]	CO3	L2
2.	What is Dynamic method dispatch? Explain with suitable example.	[10]	CO3	L2
3.	a) Explain how constructors are called in hierarchy of inheritance with an example.	[05]	CO3	L2
	b) Write a program to illustrate use of 'this' keyword and explain the same.	[05]	CO3	L3
4.	Write java program to implement stack of integer elements and make use of 'static' and 'final' keyword for suitable instances. .	[10]	CO3	L3

5.	(a) Explain concept of abstract class with help of program.	[05]	CO3	L3
	(b) What is need of Access protection? Explain Access Modifiers available in java.	[05]	CO4	L2
6.	Explain need of package and write a program to create your own package and use it.	[10]	CO4	L3
7.	a) Explain difference between abstract class and interface.	[04]	CO4	L2
	b) What is Multiple Inheritance? How it is achieved in java. Explain with an example.	[06]		
8.	What is Exception handling? Write a java program to create your own exception and handle it	[10]	CO4	L3

5.	(a) Explain concept of abstract class with help of program.	[05]	CO3	L3
	(b) What is need of Access protection? Explain Access Modifiers available in java	[05]	CO4	L2
6.	Explain need of package and write a program to create your own package and use it.	[10]	CO4	L3
7.	a) Explain difference between abstract class and interface.	[04]	CO4	L2
	b) What is Multiple Inheritance? How it is achieved in java. Explain with an example	[06]		
8.	What is Exception handling? Write a java program to create your own exception and handle it	[10]	CO4	L3

1. What is method overloading in Java? Explain with Example?

   (explanation-5+program 5)

   If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.If we have to perform only one operation, having same name of the methods increases the readability of the program.

   Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. Method overloading increases the readability of the program. Different ways to overload the method

   There are two ways to overload the method in java

   By changing number of arguments
   By changing the data type

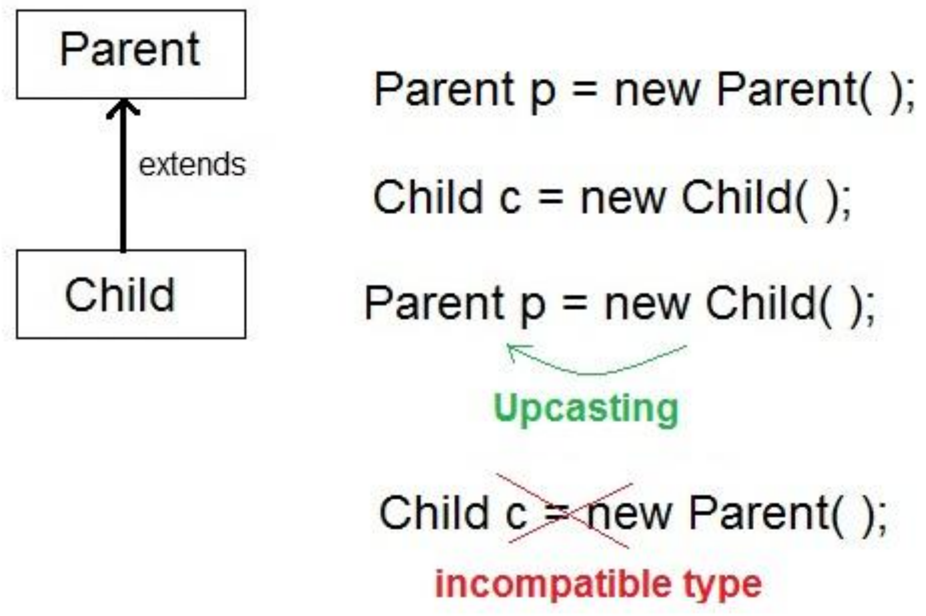
```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
static float add(int a,int b,float c){return a+b+c;}

}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
System.out.println(Adder.add(11,11,11.102));

13.     }}
```

2. What is dynamic method dispatch explain with example?
(explanation-5+program 5)

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refer to. In simple words the type of object which it referred determines which version of overridden method will be called.



Upcasting

When **Parent** class reference variable refers to **Child** class object, it is known as **Upcasting**

Example

```
class Game

{

 public void type()

 {  System.out.println("Indoor & outdoor"); }

}
```

```
Class Cricket extends Game

{

 public void type()

 {  System.out.println("outdoor game"); }


 public static void main(String[] args)

 {

   Game gm = new Game();

   Cricket ck = new Cricket();

   gm.type();

   ck.type();

   gm=ck;      //gm refers to Cricket object

   gm.type();  //calls Cricket's version of type

 }

}
```

Output :

```
Indoor & outdoor

Outdoor game

Outdoor game
```

Notice the last output. This is because of **gm = ck**; Now `gm.type()` will call Cricket version of type method. Because here gm refers to cricket object.

Q. Difference between Static binding and Dynamic binding in java ?

Static binding in Java occurs during compile time while dynamic binding occurs during runtime. Static binding uses type(Class) information for binding while dynamic binding uses instance of class(Object) to resolve calling of method at run-time. Overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding at runtime.

In simpler terms, Static binding means when the type of object which is invoking the method is determined at compile time by the compiler. While Dynamic binding means when the type of object which is invoking the method is determined at run time by the compiler.

3.Explain how constructors are called in inheritance hierarchy with a program?
(explanation-2.5+program 2.5)

In Java, constructor of base class with no argument gets automatically called in derived class constructor. For example, output of following program is:
Base Class Constructor Called
Derived Class Constructor Called

```
class Base {
  Base() {
    System.out.println("Base Class Constructor Called ");
  }
}

class Derived extends Base {
  Derived() {
    System.out.println("Derived Class Constructor Called ");
  }
}

public class Main {
  public static void main(String[] args) {
    Derived d = new Derived();
  }
}
```

But, if we want to call parameterized contructor of base class, then we can call it using super(). The point to note is **base class comstructor call must be the first line in derived class constructor**.

3.b write a program to illustrate use of this keyword in java?

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Local parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

4. Write java program to to implement stack of integer elements and make use of 'static' and 'final' keyword for suitable instances.

```
/*
 * Java Program to Implement Stack
 */

import java.util.*;

/*  Class arrayStack  */
class arrayStack
{
    protected int arr[];
    protected int top, size, len;
    /*  Constructor for arrayStack */
    public arrayStack(int n)
    {
        size = n;
        len = 0;
        arr = new int[size];
        top = -1;
    }
    /*  Function to check if stack is empty */
    Public static boolean isEmpty()
    {
        return top == -1;
    }
    /*  Function to check if stack is full */
    public boolean isFull()
    {
        return top == size -1 ;
    }
    /*  Function to get the size of the stack */
    public int getSize()
    {
        return len ;
    }
    /*  Function to check the top element of the stack */
    public int peek()
    {
        if( isEmpty() )
            throw new NoSuchElementException("Underflow Exception");
        return arr[top];
    }
    /*  Function to add an element to the stack */
    public void push(int i)
    {
        if(top + 1 >= size)
            throw new IndexOutOfBoundsException("Overflow Exception");
        if(top + 1 < size )
            arr[++top] = i;
        len++ ;
    }
    /*  Function to delete an element from the stack */
    public int pop()
    {
        if( isEmpty() )
            throw new NoSuchElementException("Underflow Exception");
        len-- ;
        return arr[top--];
    }
    /*  Function to display the status of the stack */
    public void display()
    {
        System.out.print("\nStack = ");
        if (len == 0)
        {
            System.out.print("Empty\n");
            return ;
        }
        for (int i = top; i >= 0; i--)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
}

/*  Class StackImplement  */
public class StackImplement
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Stack Test\n");
        System.out.println("Enter Size of Integer Stack ");
         final int n = scan.nextInt();
        /* Creating object of class arrayStack */
        arrayStack stk = new arrayStack(n);
        /* Perform Stack Operations */
        char ch;
        do{
            System.out.println("\nStack Operations");
            System.out.println("1. push");
            System.out.println("2. pop");
            System.out.println("3. peek");
            System.out.println("4. check empty");
            System.out.println("5. check full");
            System.out.println("6. size");
            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to push");
                           try
                           {
                               stk.push( scan.nextInt() );
                           }
                           catch (Exception e)
                           {
                            System.out.println("Error : " + e.getMessage());
                        }
                        break;
                    case 2 :
                        try
                        {
                            System.out.println("Popped Element = " + stk.pop());
                        }
                        catch (Exception e)
                        {
                            System.out.println("Error : " + e.getMessage());
                        }
                        break;
                    case 3 :
                        try
                        {
                            System.out.println("Peek Element = " +      stk.peek());
                        }
                        catch (Exception e)
                        {
                            System.out.println("Error : " + e.getMessage());
                        }
                        break;
                    case 4 :
                        System.out.println("Empty status = " + stk.isEmpty());
                        break;
                    case 5 :
                        System.out.println("Full status = " + stk.isFull());
                        break;
                    case 6 :
                        System.out.println("Size = " + stk.getSize());
                        break;
                    default :
                        System.out.println("Wrong Entry \n ");
                        break;
                    }
                    /* display stack */
                    stk.display();
                    System.out.println("\nDo you want to continue (Type y or n) \n");
                    ch = scan.next().charAt(0);

```

```
            } while (ch == 'Y'|| ch == 'y');
        }
    }
```

5(a) Explain concept of abstract class with help of program. [05]

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
       }
```

(b) What is need of access protection? Explain all access specifiers in java.?

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

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

6.Explain need of package and write a program to create your own package and use it.

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

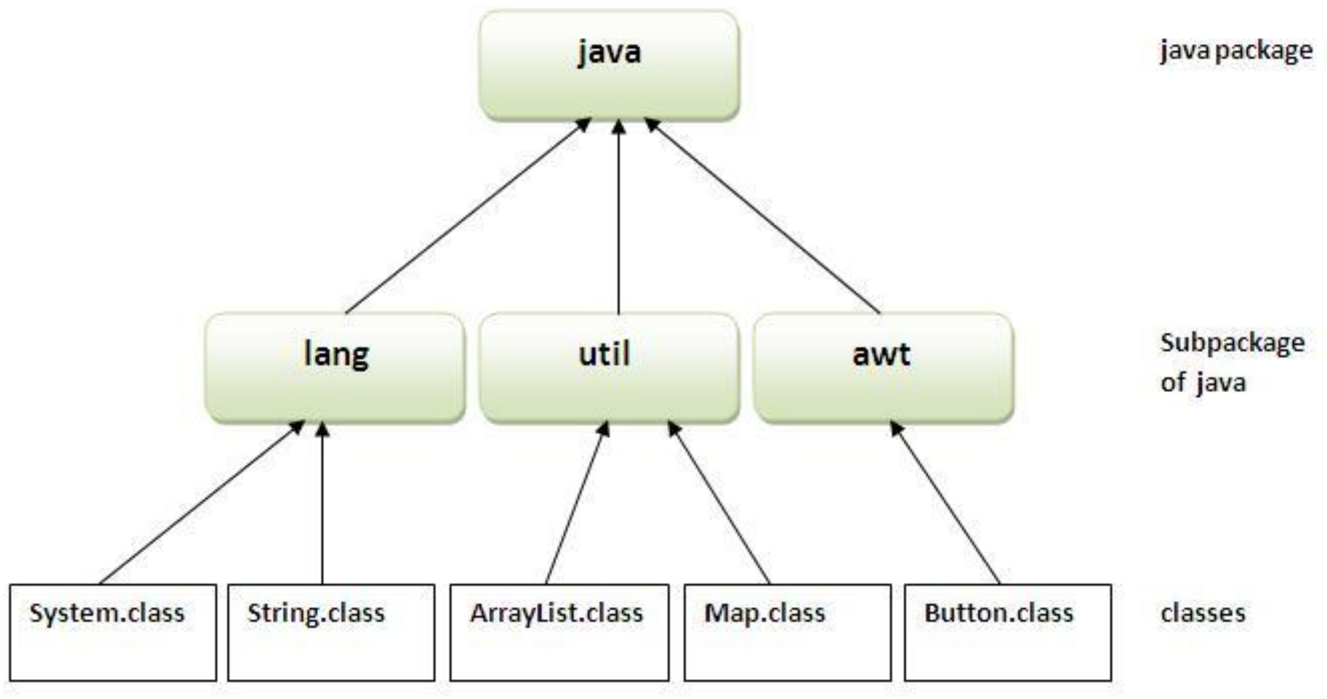
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

```
Output:Welcome to package
```

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

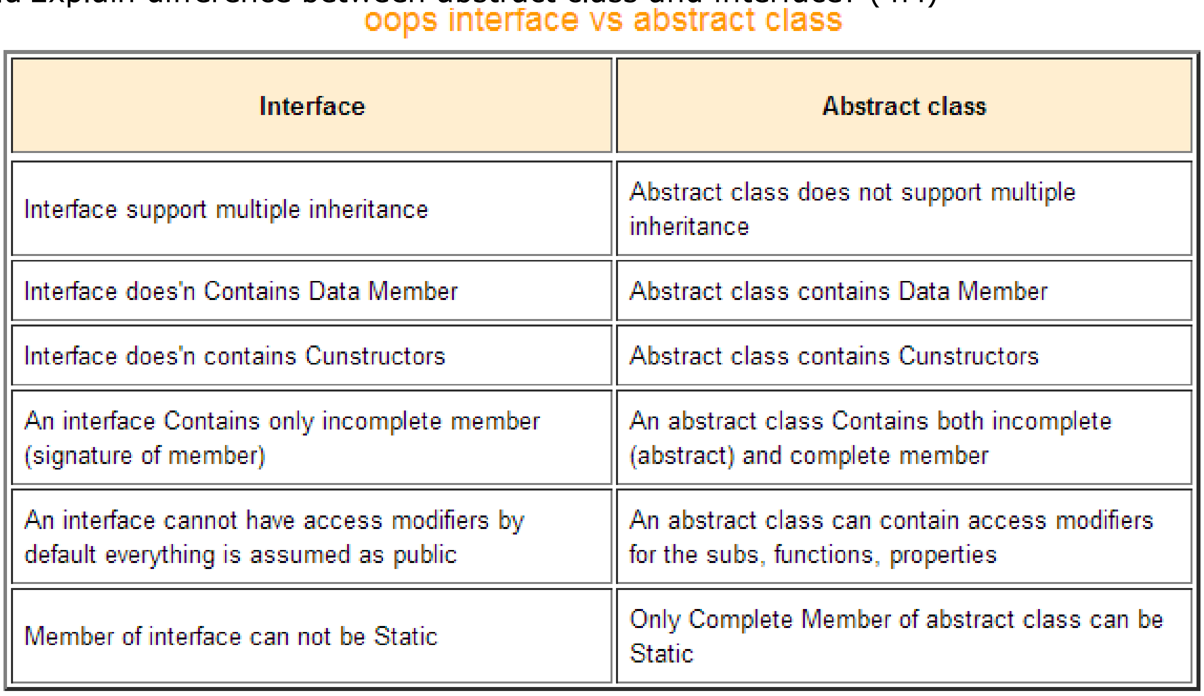
```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
```

```java

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
      }
```

6.a Explain difference between abstract class and interface? (4M)



oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

b. Write a program to implement multiple inheritance in java.(6M)

Inheritance is when an object or class is based on another object or class, using the same implementation specifying implementation to maintain the same behavior. It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a hierarchy. Multiple Inheritance allows a class to have more than one super class and to inherit features from all parent class. it is achieved using interface.

```java
interface vehicleone{
        int  speed=90;
        public void distance();
}
```

```
interface vehicletwo{
	int distance=100;
	public void speed();
}

class Vehicle  implements vehicleone,vehicletwo{
	public void distance(){
		int  distance=speed*100;
		System.out.println("distance travelled is "+distance);
	}
	public void speed(){
		int speed=distance/100;
	}
}

class MultipleInheritanceUsingInterface{
	public static void main(String args[]){
		System.out.println("Vehicle");
		obj.distance();
		obj.speed();
	}
}
```

8. What is Exception handling? Write a java program to create your own exception and handle it.
   Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an **exception object** and then the normal flow of the program halts and JRE tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **"throwing the exception"**.

   Once runtime receives the exception object, it tries to find the handler for the exception. Exception Handler is the block of code that can process the exception object. The logic to find the exception handler is simple – starting the search in the method where error occurred, if no appropriate handler found, then move to the caller method and so on. So if methods call stack is A->B->C and exception is raised in method C, then the search for appropriate handler will move from C->B->A. If appropriate exception handler is found, exception object is passed to the handler to process it. The handler is said to be **"catching the exception"**. If there are no appropriate exception handler found then program terminates printing information about the exception.

Java provides a lot of exception classes for us to use but sometimes we may need to create our own custom exception classes to notify the caller about specific type of exception with appropriate message and any custom fields we want to introduce for tracking, such as error codes. For example, let's say we write a method to process only text files, so we can provide caller with appropriate error code when some other type of file is sent as input.

Here is an example of custom exception class and showing it's usage.

```
package com.journaldev.exceptions;


public class MyException extends Exception {


	private static final long serialVersionUID =
4664456874499611218L;


	private String errorCode="Unknown_Exception";


	public MyException(String message, String errorCode){

		super(message);

		this.errorCode=errorCode;

	}


	public String getErrorCode(){

		return this.errorCode;
```

```
          }




}
```