Internal Assessment Test 3 – November 2019								
Sub:	Programming in Java				Sub Code:	**17CS561**	Branch:	**EC/TCE**
Date:	19/11/19	Duration:	90 mins	Max Marks: 50	Sem / Sec:	IV (All Sections)		OBE

Answer any FIVE FULL Questions		MARKS	CO	RBT
1 (a)	Define package. Describe the various levels of access protection for packages and their implications.	[06]	CO4	L1, L4

Package:
Packages are containers for classes that are used to keep the class name space compartmentalized.
For example, a package allows you to create a class named List, which you can store in your own package without concern that it will collide with some other class named List stored elsewhere.
Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.
Packages act as containers for classes and other subordinate packages.
Classes act as containers for data and code. The class is Java's smallest unit of abstraction.
Java addresses four categories of visibility for class members, they are
1. Subclasses in the same package
2. Non-subclasses in the same package
3. Subclasses in different packages
4. Classes that are neither in the same package nor subclasses
The three access specifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories.
Anything declared public can be accessed from anywhere.
Anything declared private cannot be seen outside of its class.
When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package. This is the default access.
If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then declare that element protected.
A non-nested class has only two possible access levels: default and public. When a class is declared as public, it is accessible by any other code.
If a class has default access, then it can only be accessed by other code within its same package.
When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class.

The table shows class member access

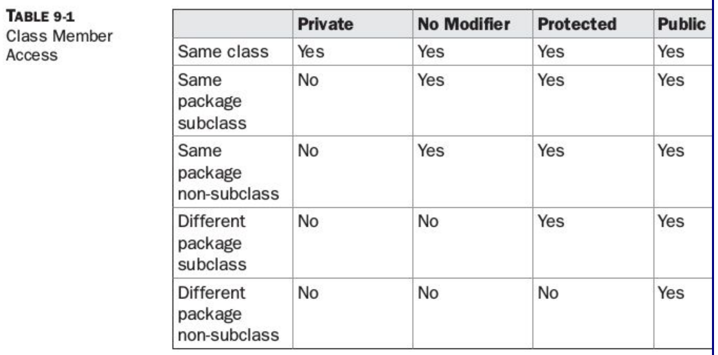
TABLE 9-1 Class Member Access

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

(b)	Explain how to create user defined exceptions with an example program.	[04]	CO4	L2

Custom Exception:
A user defined exception is known as custom exception.
Java custom exceptions are used to customize the exception according to user need.
With the help of custom exception, we can have our own exception and message.
We can create our own exception types to handle situations specific to our applications.
To create our own Exception, define a subclass of Exception, which is inturn a subclass of Throwable.
Our subclass need not implement anything. Its existence in the type system allows you to use them as exceptions.
The Exception class does not define any methods of its own. It inherits those methods provided by Throwable.
Thus, all exceptions, including those that you create, have the methods defined by Throwable available to them.
To name a few, the methods are
1. Throwable fillInStackTrace()
2. Throwable getCause()
3. String getLocalizedMessage()
4. String getMessage()
5. StackTraceElement [] getStackTrace()
6. Throwable initCause (Throwble causeExc)
7. void printStackTrace()
8. String toString ()

You can override one or more of these methods in exception classes that you create.
Exception defines four constructors
1. Exception()
2. Exception(String msg)
3. Throwable(Throwable causeExc)
4. Throwable(String msg, Throwable causeExc)

Example Program

Program Explanation:
The following example declares a new subclass of Exception and then uses that subclass to signal an error condition in a method.
It overrides the toString() method, allowing a description of the exception to be displayed.
Program:

```
//This program creates a custom exception type
class MyException extends Exception {
private int detail;
MyException(int a) {
detail = a;
}
public String toString() {
return "MyException[" + detail + "]";
}
}
class ExceptionDemo {
static void compute (int a) throws MyException {
System.out.println("Called compute (" + a + ")");
if (a > 10)
throw new MyException(a);
System.out.println("Normal exit");
}
public static void main (String args[]) {
```

```
try {
compute(1);
compute(20);
}
catch (MyException e) {
System.out.println("Caught " + e);
}
}
}
```
Output:
```
$ javac ExceptionDemo.java
$ java ExceptionDemo
Called compute (1)
Normal exit
Called compute (20)
```

2.	What is interface? Explain how to define, implement and assign variables in interface.	[10]	CO4	L3

Defining Interface:
An interface is defined much like a class. This is the general form of an interface:
```
access interface name {
return-type method-name1(parameter-list);
return-type method-name2(parameter-list);
type final-varname1 = value;
type final-varname2 = value;
// ...
return-type method-nameN(parameter-list);
type final-varnameN = value;
}
```
When no access specifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as public, the interface can be used by any other code.

//Program to define, implement and assign variable in interface
Defining an Interface:
An example of an interface definition. It declares a simple interface that contains one method called callback( ) that takes a single integer parameter.
```
interface Callback {
void callback(int param);
}
```
Implementing an interface:
Example class that implements the Callback interface.
```
class Client implements Callback {
// Implement Callback's interface
public void callback(int p) {
System.out.println("callback called with " + p);
}
}
```
Accessing Implementations Through Interface References
The following example calls the callback( ) method via an interface reference variable:
```
class TestIface {
public static void main(String args[]) {
Callback c = new Client();
c.callback(42);
}
}
```
Output:

	The output of this program is callback called with 42			

3.	Define exception. Explain how Java exception handling is managed by the five keywords with example programs.
	
	A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method.
	
	An exception is an abnormal condition which arises in the code during runtime. It is a runtime error.
	In Java, an exception is an object that describes an exceptional condition that has occurred in
	a piece of code.
	When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error.
	Java exception handling is managed via five keywords
	1. try
	2. catch
	3. throw
	4. finally
	5. throws
	try:
	◦ Program statements that you want to monitor for exceptions are placed within the try block.
	◦ If an exception occurs within the try block, it is thrown
	catch:
	The exception thrown in the try block is caught in the catch block. Thus the catch clause catches the exception and handles it in some rational manner. A catch clause must be placed immediately following a try block. We need to specify the exception type that we wish to catch in the catch clause.
	throw:
	◦ The keyword throw is used to manually throw an exception.
	◦ Syntax: throw ThrowableInstance;
	◦ where ThrowableInstance is a object of type Throwable or a subclass of Throwable.
	◦ There are two ways to obtain a Throwable object
	1. using a parameter in a catch clause
	2. creating one with the new operator
	throws:
	◦ If a method is causing an exception that it does not handle then it must specify this behavior so that callers of the method can guard themselves against that exception. We do this by including a throws clause in the method's declaration.
	◦ A throws clause lists the type of exceptions that a method might throw.
	◦ The general form of a method declaration that includes a throws clause
	type method-name (parameter-list) throws exception-list
	{
	// body of method
	}
	finally:
	◦ Any code that must be executed after a try block completes is placed in a finally block.
	The general form of an exception handling block is as follows
		[10]	CO4	L1, L3

```
try {
// block of code to monitor for errors
}
catch (ExceptionType1 exob) {
// exception handler for ExceptionType1
}
catch(ExceptionType1 exob) {
// exception handler for ExceptionType2
}
// ...
finally {
// block of code to be executed after try block ends
}
```
Example program to illustrate try and catch block.

```
// Program with try and catch clause that processes the ArithmeticException generated by
division by zero error.
class Exception1
{
public static void main(String args[])
{
int d,a;
try
{
//monitor a block of code
d=0;
a = 42 / d;
}
catch (ArithmeticException e) // catch divide by zero error
{
System.out.println("Divide by zero error");
}
}
}
```
Output:

```
$javac Exception1.java
$java Exception1
Divide by zero error
```

```
//Program to demonstrate throw
class ThrowDemo
{
static void demoproc ()
{
try
{
throw new NullPointerException("demo");
}
catch (NullPointerException e)
{
System.out.println("caught inside demoproc");
throw e;
}
}
public static void main(String args[])
{
try
{
demoproc();
}
catch ( NullPointerException e)
{
System.out.println("Recaught : " + e);
}
}
}
```
Output:
```
$javac ThrowDemo.java
$java ThrowDemo
caught inside demoproc
Recaught : java.lang.NullPointerException: demo
```
```
// Program to demonstrate throws
class ThrowsDemo
{
static void throwOne() throws IllegalAccessException
{
System.out.println("Inside throwOne");
throw new IllegalAccessException("Demo");
}
public static void main (String args[])
{
try
{
throwOne();
}
catch (IllegalAccessException e)
{
System.out.println("Caught " + e);
}
}
}
```
Output:
```
Inside throwOne
Caught java.lang.IllegalAccessException: Demo
```
```
// Program to demonstrate finally
class FinallyDemo {
static void procA() {
try {
System.out.println("Inside procA");
throw new RuntimeException("demo");
}
finally {
System.out.println("procA's finally");
}
}
static void procB () {
try {
System.out.println("Inside procB");
return;
}
finally {
System.out.println("procC's finally");
}
}
static void procC () {
try {
System.out.println("inside procC");
}
finally {
System.out.println("procC's finally");
}
}
public static void main (String args[]) {
try {
procA();
}
catch (Exception e) {
System.out.println("Exception Caught");
}
procB();
procC();
}
}
```

Output:

```
Inside procA
procA's finally
Exception Caught
Inside procB
procC's finally
inside procC
procC's finally
```

4 (a)	What is enum? Write a Java program to demonstrate valuses() and valueOf() methods.	[06]	CO4	L3

In its simplest form, an enumeration is a list of named constants.
In their simplest form, Java enumerations appear similar to enumerations in other languages. However, this similarity is only skin deep. In languages such as C++, enumerations are simply lists of named integer constants.

In Java, an enumeration defines a class type. By making enumerations into classes, the concept of the enumeration is greatly expanded. For example, in Java, an enumeration can have constructors, methods, and instance variables.

Enumeration Fundamentals
An enumeration is created using the enum keyword. For example, here is a simple enumeration that lists various apple varieties:

```
// An enumeration of apple varieties.
enum Apple {
Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

Once you have defined an enumeration, you can create a variable of that type.
For example, this declares ap as a variable of enumeration type Apple:
Apple ap;
Because ap is of type Apple, the only values that it can be assigned (or can contain) are those defined by the enumeration. For example, this assigns ap the value RedDel:
ap = Apple.RedDel;
Two enumeration constants can be compared for equality by using the = = relational operator. For example,
this statement compares the value in ap with the GoldenDel constant:
if(ap == Apple.GoldenDel) // …

An enumeration value can also be used to control a switch statement.
The values( ) and valueOf( ) Methods : explain in detail
All enumerations automatically contain two predefined methods: values( ) and valueOf( ).
Their general forms are shown here:
public static enum-type[ ] values( )
public static enum-type valueOf(String str)
Apple.valueOf("Winesap") is
Winesap.

```
enum Apple {
Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
public class EnumDemo {
public static void main(String args[])
{
Apple ap;
System.out.println("Here are all Apple constants:");
// use values()
for(Apple a : Apple.values())
System.out.println(a);System.out.println();
// use valueOf()
ap = Apple.valueOf("Winesap");
System.out.println("ap contains " + ap);
}
}
```

The output from the program is shown here:

```
Here are all Apple constants:
Jonathan
GoldenDel
RedDel
Winesap
Cortland
ap contains Winesap
```

(b)	Write a Java program to sort string elements using bubble sort.	[04]	CO4	L3

```
class StringSort
{
public static void main(String args[])
{
String
s[]={"Now","is","the","time","for","all","good","men","to","come","to","the","aid","of","th
eir","co
untry"};
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;
}
}
}
System.out.println("the sorted array is");
for(String temp:s)
{
System.out.println(temp);
}
}
}
```

Output:

```
the sorted array is
Now
aid
all
```

	come country for good is men of the the their time to to			
5 (a)	Explain the following with syntax (i) charAt() (ii) toCharArray() (iii) concat **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. **toCharArray( )** If you want to convert all the characters in a String object into a character array, the easiest way is to call toCharArray( ). It returns an array of characters for the entire string. It has this general form: char[ ] toCharArray( ) **concat( )** You can concatenate two strings using concat( ), shown here: String concat(String str) This method creates a new object that contains the invoking string with the contents of str appended to the end. concat( ) performs the same function as +. For example, String s1 = "one"; String s2 = s1.concat("two"); puts the string "onetwo" into s2. It generates the same result as the following sequence: String s1 = "one"; String s2 = s1 + "two";	[06]	CO5	L2
(b)	Explain the role of interface in implementing multiple inheritance in Java. Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority. Therefore, in order to avoid such complications Java does not support multiple inheritance of classes. But, a class can implement two or more interfaces. A class can implement more than one interface, which can contain default methods that have the same name. The Java compiler provides some rules to determine which default method a particular class uses.	[04]	CO4	L1

Example program to demonstrate multiple inheritance using interface:

```
// Program to demonstrate multiple inheritance using interface
// Define the interface I1
interface I1 {
void showI1() ;
}
// Define the interface I2
interface I2 {
void showI2();
}
// Define MInheritance that implements both I1 and I2
class MInheritance implements I1, I2 {
// Implement I1's interface
public void showI1() {
System.out.println("Inside showI1");
}
// Implement I2's interface
public void showI2() {
System.out.println("Inside showI2");
}
}
class TestMI {
public static void main(String args[]) {
MInheritance MI = new Minheritance();
MI.showI1();
MI.showI2();
}
}
```

Output:

```
$ javac TestMI.java
$ java TestMI
Inside showI1
Inisde showI2
```

Program Explanation:
The program defines two interfaces I1 and I2.
The class Minheritance implements from both interfaces I1 and I2. The method of both the interfaces showI1() and showI2 are implemented in this class.
The class TestMI creates an instance of Minheritance and calls both the methods showI1() and showI2().

6.	What is an Applet? Write a program to create a simple Applet. Explain the two ways in which you can run an Applet.	[10]	CO5	L4

Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. After an applet arrives on the client, it has limited access to resources so that it can produce a graphical user interface and run complex computations without introducing the risk of viruses or breaching data integrity.

```
//Program to create a simple Applet
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleApplet" width=200 height=60>
</applet>
*/
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("A Simple Applet", 20, 20);
}
}
```

There are two ways in which you can run an applet:
1. Executing the applet within a Java-compatible web browser.
2. Using an applet viewer, such as the standard tool, appletviewer. An appletviewer executes your
applet in a window. This is generally the fastest and easiest way to test your applet.
1. To execute an applet in a web browser:
To execute an applet in a web browser, you need to write a short HTML text file that contains a tag that loads the applet.
Here is the HTML file that executes SimpleApplet:

```
<applet code="SimpleApplet" width=200 height=60>
</applet>
```

The width and height statements specify the dimensions of the display area used by the applet. After you create this file, you can execute your browser and then load this file, which
causes SimpleApplet to be executed.
2. To execute SimpleApplet with an applet viewer:
To execute SimpleApplet with an applet viewer you Simply include a comment at the head of your
Java source code file that contains the APPLET tag. By doing so, your code is documented with a
prototype of the necessary HTML statements, and you can test your compiled applet merely by
starting the applet viewer with your Java source code file. If you use this method, the SimpleApplet
source file looks like this:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleApplet" width=200 height=60>
</applet>
*/
public class SimpleApplet extends Applet {
public void paint(Graphics g) {
g.drawString("A Simple Applet", 20, 20);
}
}
```

Execute the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the APPLET tag within the comment and execute your applet.
The window produced by SimpleApplet, as displayed by the applet viewer, is shown in the following illustration:

```
$javac SimpleApplet.java
$appletviewer SimpleApplet.java
```

7. Write a program to copy a file called FIRST.TXT to a file called SECOND.TXT. Use the following command line: [10] CO5 L2

```
java  CopyFile    FIRST.TXT    SECOND.TXT
/* Copy a text file.
To use this program, specify the name
of the source file and the destination file.
For example, to copy a file called FIRST.TXT
to a file called SECOND.TXT, use the following
command line.
java CopyFile FIRST.TXT SECOND.TXT
*/
import java.io.*;
class CopyFile
{
public static void main(String args[]) throws IOException
{
int i;
FileInputStream fin;

FileOutputStream fout;

try { // open input file

try {
fin = new FileInputStream(args[0]);
}
catch(FileNotFoundException e) {
System.out.println("Input File Not Found");
return;
}
// open output file
try {
fout = new FileOutputStream(args[1]);
}
catch(FileNotFoundException e) {
System.out.println("Error Opening Output File");
return;
}
}
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Usage: CopyFile From To");
return;
}
// Copy File
try {
do {
i = fin.read();
if(i != -1) fout.write(i);
} while(i != -1);
}
catch(IOException e) {
System.out.println("File Error");
}
fin.close();
fout.close();
}
}
```