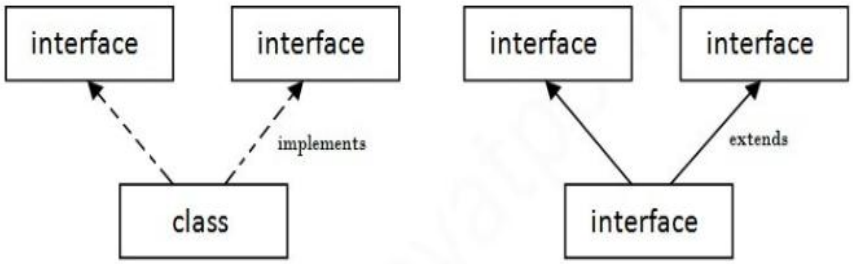
USN | | | | | | | | | | |

Internal Assessment Test_solution 3 – July. 2022

Sub:	**Programming in JAVA**					Sub Code:	18CS653	Branch:	ECE
Date:	11-07-2022	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	**6 (A, B,C,D)**		**OBE**

Answer any FIVE FULL Questions		MARKS	CO	RBT

1	Explain with an example how an interface can be extended and also explain interface variables.
	
	Syntax of interface and variables—2+2
	
	Explanation with an example—6M
	
	• One interface can inherit another by use of the keyword **extends**
	• The syntax is the same as for inheriting classes
	• When a class implements an interface that inherits another interface, it **must provide implementations for all methods defined** within the interface inheritance chain
	• If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as **multiple inheritance**
	
	 
	
	Multiple Inheritance in Java
	
	// One interface can extend another.
	
	interface A {
	
	void meth1();
	
	void meth2();
	
	}
		(10)M	CO3.CO4	L3

```
// B now includes meth1() and meth2() -- it adds meth3().
interface B extends A {
void meth3();
}
// This class must implement all of A and B
class MyClass implements B {
public void meth1() {
System.out.println("Implement meth1().");
}
public void meth2() {
System.out.println("Implement meth2().");
}
public void meth3() {
}
}
class IFExtend {
public static void main(String arg[]) {
MyClass ob = new MyClass();
ob.meth1();
ob.meth2();
ob.meth3();
}
}
```

Interface vairales

- Interfaces can import shared constants into multiple classes

	by simply declaring an interface that contains variables that are initialized to the desired values Example:			

2	Design and explain with help of Java program fixed integer stack using an interface.
	
	interface IntStack {
	
	void push(int item); // store an item
	
	int pop(); // retrieve an item
	
	}
	
	// An implementation of IntStack that uses fixed storage.
	
	class FixedStack implements IntStack {
	
	private int stck[];
	
	private int tos;
	
	// allocate and initialize stack
	
	FixedStack(int size) {
	
	stck = new int[size];
	
	tos = -1;
	
	}
	
	// Push an item onto the stack
	
	public void push(int item) {
	
	if(tos==stck.length-1) // use length member
	
	System.out.println("Stack is full.");
	
	else
	
	stck[++tos] = item;
	
	}
		(10)M	CO3,CO4	L3

```
// Pop an item from the stack
public int pop() {
if(tos < 0) {
System.out.println("Stack underflow.");
return 0;
}
else
return stck[tos--];}}
class IFTest {
public static void main(String args[]) {
FixedStack mystack1 = new FixedStack(5);
FixedStack mystack2 = new FixedStack(8);
// push some numbers onto the stack
for(int i=0; i<5; i++) mystack1.push(i);
for(int i=0; i<8; i++) mystack2.push(i);
// pop those numbers off the stack
System.out.println("Stack in mystack1:");
for(int i=0; i<5; i++)
System.out.println(mystack1.pop());
System.out.println("Stack in mystack2:");
for(int i=0; i<8; i++)
System.out.println(mystack2.pop());}}
class IFTest3 {
public static void main(String args[]) {
```

	IntStack mystack; // create an interface reference variable
	DynStack ds = new DynStack(5);
	FixedStack fs = new FixedStack(8);
	mystack = ds; // load dynamic stack
	// push some numbers onto the stack
	for(int i=0; i<12; i++) mystack.push(i);
	mystack = fs; // load fixed stack
	for(int i=0; i<8; i++) mystack.push(i);
	mystack = ds;
	System.out.println("Values in dynamic stack:");
	for(int i=0; i<12; i++)
	System.out.println(mystack.pop());
	mystack = fs;
	System.out.println("Values in fixed stack:");
	for(int i=0; i<8; i++)
	System.out.println(mystack.pop());
	}
	}			

3	Explain exception with syntax and example and also Explain nested try with the help of example • Java permits using a try block inside another try block, called as nested try block • Every statement that we enter in try block, context of that exception is pushed onto the stack • For example, the **inner try block** is handeling **ArrayIndexOutOfBoundsException** while the **outer try block** handling	[5+5]M	CO3,CO4	L2

	the **ArithemeticException** (division by zero)
	• Syntax:
	//main try block
	try
	{
	statement 1;
	statement 2;
	//try catch block within another try block
	try
	{
	statement 3;
	statement 4;
	//try catch block within nested try block
	try
	{
	statement 5;
	statement 6;
	}
	catch(Exception e2)
	{
	//exception message
	}
	
	}
	catch(Exception e1)
	{
	//exception message
	}
	}
	//catch block of parent (outer) try block
	catch(Exception e3)
	{
	//exception message
	}			

4	Design an Interface called Polygon with a method called area. Implement this Interface to create different classes like Square, Rectangle and print the area of Square and Rectangle.
	
	interface Polygon {
	void getArea(int length, int breadth);
	}
		[7+3]M	CO3,CO4	L2
				
				L3

```
// implement the Polygon interface
class Square implements Polygon {

  // implementation of abstract method
  public void getArea(int length, int breadth) {
    System.out.println("The area of the square is " + (length
* breadth));
  }
}
class Rectangle implements Polygon {

  // implementation of abstract method
  public void getArea(int length, int breadth) {
    System.out.println("The area of the rectangle is " +
(length * breadth));
  }
}

class Main {
  public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    r1.getArea(5, 6);
Square s1 = new Square();
    s1.getArea(50,50);

  }
}
```

5	Explain package with syntax. Write a simple program to create a package and import the package to make use of a class declared in the package. • The **package keyword** is used to create a package in java. • **How to compile java package:** • If not using any IDE, follow the **syntax**: • javac -d directory javafilename **-d switch** specifies the destination where to put the generated class file	[3+7]M	CO3,CO4	L2

//save as Simple.java

package mypack;

This is file **Protection.java**:

```
package p1;

public class Protection {

int n = 1;

private int n_pri = 2;

protected int n_pro = 3;

public int n_pub = 4;

public Protection() {

System.out.println("base constructor");

System.out.println("n = " + n);

System.out.println("n_pri = " + n_pri);

System.out.println("n_pro = " + n_pro);

System.out.println("n_pub = " + n_pub);

}

}
```

This is file **Derived.java**:

```
package p1;

class Derived extends Protection {

Derived() {

System.out.println("derived constructor");

System.out.println("n = " + n);
```

```
// class only

// System.out.println("n_pri = "4 + n_pri);

System.out.println("n_pro = " + n_pro);

System.out.println("n_pub = " + n_pub);}}
```

This is file **Protection2.java**:

```
package p2;

class Protection2 extends p1.Protection {

Protection2() {

System.out.println("derived other package constructor");

// class or package only

// System.out.println("n = " + n);

// class only

// System.out.println("n_pri = " + n_pri);

System.out.println("n_pro = " + n_pro);

System.out.println("n_pub = " + n_pub);

}

}

package p2;

// Instantiate the various classes in p2.

public class Demo {

public static void main(String args[]) {

Protection2 ob1 = new Protection2();

OtherPackage ob2 = new OtherPackage();

}
```

	}
	
	public class Simple{
	public static void main(String args[]){
	System.out.println("Welcome to package");
	}
	}			

6	Explain with the help of program how throw, throws, and finally keywords are used with respect to exception. • Java throw keyword is used to throw an exception explicitly • ***throw*** specifies the **exception** object which is to be thrown • The Exception has some message with it that provides the error description like related to user inputs, server, etc. • ***throw*** sends checked or unchecked, custom exceptions in Java • We can also define our own set of conditions and throw an exception explicitly using throw keyword. • **Example:** • we can throw ***ArithmeticException*** if we divide a number by another number. Here, we just need to set the condition and throw exception using ***throw*** keyword throw *ThrowableInstance;* *ThrowableInstance :object of type* ***Throwable/***subclass of ***Throwable*** There are two ways you can obtain a **Throwable object:** using a parameter in a **catch clause, or creating one with the new operator** The flow of execution stops immediately after the **throw statement; any subsequent** statements are not executed. The **syntax** of the Java throw keyword is given below.	[10]]M	CO3,CO4	L2

throw Instance i.e:

throw new exception_class("error message");

Example of throw IOException:

throw new IOException("sorry device error");

A **throws clause lists the types** of exceptions that a method might throw

This is necessary for all exceptions, except those of

return_type method_name() **throws** exception_class_name{

//method code

}

- **finally creates a block of code that will be executed after a try/catch block has** completed and before the code following the **try/catch block**
- The **finally block will** execute whether or not an exception is thrown
- If an exception is thrown, the **finally** block will execute even if no **catch statement matches the exception**
- Any time a method is about to return to the caller from inside a **try/catch block, via an uncaught exception or** an explicit return statement, the **finally clause is also executed just before the method** returns
- The **finally clause is optional, but each try statement requires at** least one **catch or a finally clause**
- *If a **finally block is associated with a try, the finally block will be executed upon** conclusion of the **try***

// Demonstrate finally.

class FinallyDemo {

// Through an exception out of the method.

static void procA() {

try {

System.out.println("inside procA");

```
throw new RuntimeException("demo");
} finally {
System.out.println("procA's finally");
}
}
// Return from within a try block.
static void procB() {
try {
System.out.println("inside procB");
return;
} finally {
System.out.println("procB's finally");
}
}
// Execute a try block normally.
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();
	
	}
	
	}			

7	What is chained Exception? Write a program to generate chained exception using throw.
	• Chained Exceptions allows to relate one exception with another exception, i.e one exception describes cause of another exception
	• **Constructors** Of Throwable class Which support chained exceptions in java :
	• Throwable(Throwable cause) :- Where cause is the exception that causes the current exception.
	• Throwable(String msg, Throwable cause) :- Where msg is the exception message and cause is the exception that causes the current exception.
	• **Methods** Of Throwable class Which support chained exceptions in java :
	• getCause() method :- This method returns actual cause of an exception.
	• initCause(Throwable cause) method :- This method sets the cause for the calling exception.
	import java.io.IOException;
	public class ChainedException
	{
	public static void divide(int a, int b)
	{
	if(b == 0)
	{
	ArithmeticException ae = new ArithmeticException("top layer");
		[7+3]	CO3,CO4	L1

```
      ae.initCause(new IOException("cause"));
      throw ae;
    }
    else
    {
      System.out.println(a/b);
    }
  }
public static void main(String[] args)
  {
    try
    {
      divide(5, 0);
    }
    catch(ArithmeticException ae) {
      System.out.println( "caught : " +ae);
      System.out.println("actual cause: "+ae.getCause());
    }
  }
}
// Demonstrate exception chaining.
class ChainExcDemo {
static void demoproc() {
// create an exception
NullPointerException e =
new NullPointerException("top layer");
// add a cause
e.initCause(new ArithmeticException("cause"));
throw e;
}
public static void main(String args[]) {
try {
demoproc();
} catch(NullPointerException e) {
// display top level exception
System.out.println("Caught: " + e);
// display cause exception
System.out.println("Original cause: " +
e.getCause());
}
}
}
```

HOD CI CCI