SUMMER– 18 EXAMINATION

Subject Name: Java Programming **Model Answer** Subject Code: 17515

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answers	Marking Scheme
1.	**(A)**	**Attempt any THREE of the following:**	**12Marks**
	(a)	**Define throws & finally statements with its syntax and example.**	**4M**

	Ans:	1) **throws statement :** throws keyword is used to declare that a method may throw one or some exceptions. The caller must catch the exceptions.
		Example :
		import java.io.*;
		class file1
		{
		public static void main(String[] args) throws IOException
		{
		FileWriter file = new FileWriter("Data1.txt");
		file.write("These are contents of my file");
		file.close();
		}
		}
		2) **finally statement :** finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block follows try or catch block.
		Example :
		import java.io.*;
			(Each statement: 2 marks)

```
class file1
{
 public static void main(String[] args)
 {
   try
 {
        FileWriter file = new FileWriter("c:\\Data1.txt");
        file.write("Hello");
 }
catch(IOException)
{}
finally
{
        file.close();
}
}
}
```

	(b)	**Which are the restrictions present for static declared methods?**	**4M**
	Ans:	Restrictions on static variables : 1. Static variables become class variables. 2. They get initialized only once in the entire life time of the program. 3. They cannot be called by the object of the class in which they are defied. 4. A static method can operate only on static variables without objects otherwise non static variables cannot be handled by a static method without using their respective class object.	**(Each point: 1mark)**
	(c)	**Explain any four features of java programming.**	**4M**
	Ans:	**1. Compile & Interpreted**: Java is a two staged system. It combines both approaches. First java compiler translates source code into byte code instruction. Byte codes are not machine instructions. In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compile and interpreted language. **2. Platform independent and portable**: Java programs are portable i.e. it can be easily moved from one computer system to another. Changes in OS, Processor, system resources won't force any change in java programs. Java compiler generates byte code instructions that can be implemented on any machine as well as the size of primitive data type is machine independent. **3. Object Oriented**: Almost everything in java is in the form of object. All program codes and data reside within objects and classes. Similar to other OOP languages java also has basic OOP properties such as encapsulation, polymorphism, data abstraction, inheritance etc. Java comes with an extensive set of classes (default) in packages.	(**Any 4 features : 1mark each)**

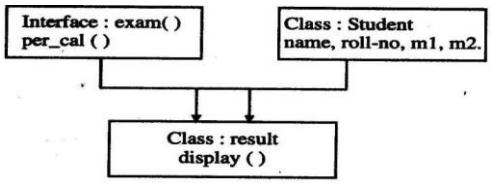
4. Robust & Secure: Java is a robust in the sense that it provides many safeguards to ensure reliable codes. Java incorporates concept of exception handling which captures errors and eliminates any risk of crashing the system. Java system not only verifies all memory access but also ensure that no viruses are communicated with an applet. It does not use pointers by which you can gain access to memory locations without proper authorization.

5. Distributed: It is designed as a distributed language for creating applications on network. It has ability to share both data and program. Java application can open and access remote object on internet as easily as they can do in local system.

6. Multithreaded: It can handle multiple tasks simultaneously. Java makes this possible with the feature of multithreading. This means that we need not wait for the application to finish one task before beginning other.

7. Dynamic and Extensible: Java is capable of dynamically linking new class library's method and object. Java program supports function written in other languages such as C, C++ which are called as native methods. Native methods are linked dynamically at run time.

	(d)	**Explain how interface is used to achieve multiple Inheritance in Java.**	**4M**
	Ans:	**{{**Note:-Any other example can be considered **}}** Multiple inheritances is not possible in java. Multiple inheritance happens when a class is derived from two or more parent classes. Java classes cannot extend more than one parent classes, instead it uses the concept of interface to implement the multiple inheritance. It contains final variables and the methods in an interface are abstract. A sub class implements the interface. When such implementation happens, the class which implements the interface must define all the methods of the interface. A class can implement any number of interfaces. Example of multiple inheritance :	**(Explanation: 2 marks, Example: 2 marks)**




```
import java.io.*;
class Student
{
String name;
int roll_no;
double m1, m2;
Student(String name, int roll_no, double m1, double m2)
{
this.name = name;
this.roll_no = roll_no;
```

```
this.m1 = m1;
this.m2 = m2;
}
}
interface exam {
public void per_cal();
}
class result extends Student implements exam
{
double per;
result(String n, int r, double m1, double m2)
{
super(n,r,m1,m2);
}
public void per_cal()
{
per = ((m1+m2)/200)*100;
System.out.println("Percentage is "+per);
}
void display()
{
System.out.println("The name of the student is"+name);
System.out.println("The roll no of the student is"+roll_no);
per_cal();
}
public static void main(String args[])
{
BufferedReader bin = new BufferedReader(new InputStreamReader(System.in));
try
{
System.out.println("Enter name, roll no mark1 and mark 2 of the student");
String n = bin.readLine();
int rn = Integer.parseInt(bin.readLine());
double m1 = Double.parseDouble(bin.readLine());
double m2 = Double.parseDouble(bin.readLine());
result r = new result(n,rn,m1,m2);
r.display();
} catch(Exception e)
{
System.out.println("Exception caught"+e);
}
}
}
```

	(B)	**Attempt any ONE of the following:**	**6 Marks**
	(a)	**Write a java program to implement visibility controls such as public, private, protected access modes. Assume suitable data, if any.**	**6M**
	Ans:	**{{**Note:- Any common example for all 3 access modes also can be considered **}}**	**(Example : 2 marks for each type)**

Public access specifier :

```
class Hello
{
public int a=20;
public void show()
{
System.out.println("Hello java");
}
}
public class Demo
{
 public static void main(String args[])
 {
  Hello obj=new Hello();
  System.out.println(obj.a);
  obj.show();
 }
}
```

private access specifier :

```
class Hello
{
private int a=20;
private void show()
{
System.out.println("Hello java");
}
}
public class Demo
{
 public static void main(String args[])
 {
  Hello obj=new Hello();
  System.out.println(obj.a); //Compile Time Error, you can't access private data
  obj.show();   //Compile Time Error, you can't access private methods
 }
}
```

protected access specifier :

```
// save A.java
package pack1;
public class A
{
protected void show()
```

```
{
System.out.println("Hello Java");
}
}
//save B.java
package pack2;
import pack1.*;
 class B extends A
{
 public static void main(String args[]){
 B obj = new B();
 obj.show();
  }
}
```

(b) **With proper syntax and example explain following graphics methods:** **6M**

1) **SetColor( )**
2) **SetForeGround( )**
3) **getFont( )**
4) **setSize( )**

Ans:

1) setColor() :

Syntax :
setColor(Color c)
where ‘c’ is the Color class object.
Sets this graphics context's current color to the specified color.
Example : setColor(Color.RED);

2) setForeGround() :

public void setForeground(Color c)
where ‘c’ is Color class object
Sets the foreground color of the component.
Example : setForeground(Color.BLUE);

3) getFont() :

Syntax :
public static Font getFont(String nm)
where
nm - the property name.
Returns a font from the system properties list.
public static Font getFont(String nm, Font font)
where
nm - the property name.
font - a default font to return if property 'nm' is not defined.
Returns the specified font from the system properties list.
Example:
Font f=g.getFont();

(Explanation : 1 mark for each type, Example : ½ mark for each type)

		4) setSize() : Syntax : public void setSize(int width,int height) where width - the new width for this Dimension object height - the new height for this Dimension object Here the dimension object is generally a frame object. Sets the size of this Dimension object to the specified width and height. Example :Frame fm=new Frame(); fm.setSize(200,400);	
2.		**Attempt any TWO of the following:**	**16Marks**
	(a)	**Write a java program to copy the content of the file "file1.txt" into new file "file2.txt".**	**8M**

	Ans:	**{{**Note :-Any other logic can be considered**}}**
		import java.io.*;
		class filecopy
		{
		public static void main(String args[]) throws IOException
		{
		FileReader fr=new FileReader("file1.txt");
		FileWriter fo=new FileWriter("file2.txt");
		int ch;
		try
		{
		while((ch=fr.read())!=-1)
		{
		fo.write(ch);
		}
		fr.close();
		fo.close();
		}
		finally
		{
		if(fr!=null)
		fr.close();
		if(fo!=null)
		fo.close();
		}
		}
		}
			(Correct Logic : 4 marks, Correct Syntax : 4 marks)

	(b)	**Write a java program to implement multilevel inheritance with 4 levels of hierarchy.**	**8M**

	Ans:	**{{**Note:-Any other example can be considered **}}**
		class emp
		{
		int empid;
		String ename;
		emp(int id, String nm)
		{
		empid=id;
		ename=nm;
		}
		}
		class work_profile extends emp
		{
		String dept;
		String job;
		work_profile(int id, String nm, String dpt, String j1)
		{
		super(id,nm);
		dept=dpt;
		job=j1;
		}
		}
		class salary_details extends work_profile
		{
		int basic_salary;
		salary_details(int id, String nm, String dpt, String j1,int bs)
		{
		super(id,nm,dpt,j1);
		basic_salary=bs;
		}
		double calc()
		{
		double gs;
		gs=basic_salary+(basic_salary*0.4)+(basic_salary*0.1);
		return(gs);
		}
		}
		class salary_calc extends salary_details
		{
		salary_calc(int id, String nm, String dpt, String j1,int bs)
		{
		super(id,nm,dpt,j1,bs);
			(Correct logic:4 marks, Correct syntax: 4 marks)

```
}
public static void main(String args[])
{
salary_calc e1=new salary_calc(101,"abc","Sales","clerk",5000);
double gross_salary=e1.calc();
System.out.println("Empid :"+e1.empid);
System.out.println("Emp name :"+e1.ename);
System.out.println("Department :"+e1.dept);
System.out.println("Job :"+e1.job);
System.out.println("BAsic Salary :"+e1.basic_salary);
System.out.println("Gross salary :"+gross_salary);
}
}
```

	(c)	**Define applet. Write a program to create an applet to display message "Welcome to java applet".**	**8M**
	Ans:	Java applet is a small dynamic Java program that can be transferred via the Internet and run by a Java-compatible Web browser. The main difference between Java-based applications and applets is that applets are typically executed in an appletviewer or Java-compatible Web browser. All applets import the java.awt package.	**(Definition : 2 marks, Program : correct logic : 3 marks, Correct syntax : 3 marks)**

```
/*<applet code= WelcomeJava width= 300 height=300></applet>*/
 import java.applet.*;
 import java.awt.*;
 public class WelcomeJava extends Applet
 {
 public void paint(Graphics g)
 {
 g.drawString("Welcome to java",25,50);
 }
}
```

3.		**Attempt any FOUR of the following:**	**16Marks**
	(a)	**Explain any four applet tag.**	**4M**
	Ans:	**{{**Note: - Any 4 attributes shall be considered **}}** **APPLET Tag:** The APPLET tag is used to start an applet from both an HTML document and from an appletviewer will execute each APPLET tag that it finds in a separate window, while web browser will allow many applets on a single page the syntax for the standard APPLET tag is: <APPLET [CODEBASE=codebaseURL] CODE =appletfileName [ALT=alternateText]	**(Any 4 attributes: 1 mark each)**

		[NAME=applet_instance_name] WIDTH=pixels HEIGHT=pixels [ALIGN=aligment] [VSPACE=pixels] [HSPACE=pixels] > [<PARAM NAME=attributeName1 VALUE=attributeValue>] [<PARAM NAME=attributeName2 VALUE=attributeValue>] </APPLET> **CODEBASE:** is an optional attribute that specifies the base URL of the applet code or the directory that will be searched for the applets executable class file. **CODE:** is a required attribute that give the name of the file containing your applets compiled class file which will be run by web browser or applet viewer. **ALT: (**Alternate Text) The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser cannot run java applets. **NAME:** is an optional attribute used to specify a name for the applet instance. **WIDTH AND HEIGHT:** are required attributes that give the size(in pixels) of the applet display area. **ALIGN** is an optional attribute that specifies the alignment of the applet. The possible value is: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM. **VSPACE AND HSPACE:** attributes are optional, VSPACE specifies the space, in pixels, about and below the applet. HSPACE VSPACE specifies the space, in pixels, on each side of the applet **PARAM NAME AND VALUE:** The PARAM tag allows you to specifies applet-specific arguments in an HTML page applets access there attributes with the getParameter() method.	
	(b)	**Which are the ways to access package from another package? Explain with example.**	**4M**
	Ans:	There are two ways to access the package from another package. 1. import package.*; 2. import package.classname; 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 interfaces of another package accessible to the current package.	**(Different ways to access package :1 mark & explanation of each : 1 ½ mark)**

E.g.

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

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

2. Using packagename.classname

If you import packagename.classname then only declared class of this package will be accessible.

Eg.

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

	(c)	**Define a class and object. Write syntax to create class and object with an example.**	**4M**
	Ans:	• Java is complete object oriented programming language. All program code and data reside in object and class. Java classes create objects and objects will communicate between using methods. **Class:** • A 'class' is a user defined data type. Data and methods are encapsulated in class. • It is a template or a pattern which is used to define its properties. • Java is fully object oriented language. All program code and data reside within objects and classes.	**(Definition, syntax and example of each: 2 marks)**

Syntax :

```
class classname
{
type instance-variable1;
.
.
type methodname1(parameter-list)
  { / / body of method }
}
```

Object:

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of state ,behavior and identity.

Syntax:

class_name object=new class_name();

Example:

```
class Student{
 int id;              //field or data member or instance variable
 String name;
  public static void main(String args[]){
  Student s1=new Student();          //creating an object of Student
  System.out.println(s1.id);         //accessing member through reference variable
  System.out.println(s1.name);
 } }
```

	(d)	**With proper syntax and example explain following thread methods:** **(1) wait( )** **(2) sleep( )** **(3) resume( )** **(4) notify( )**	**4M**
	Ans:	**{{**Note :- Separate example for each method can also be considered **}}** **(1) wait():** **syntax : public final void wait()** This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.	**(Each Method: 1 mark)**

(2) sleep():

syntax: public static void sleep(long millis) throws InterruptedException

We can put a thread to sleep for a specified time period using sleep(time) where time is in ms. It reenters the runnable state as soon as period has elapsed /over.

(3) resume():

syntax : public void resume()

This method resumes a thread which was suspended using suspend() method.

(4) notify():

syntax: public final void notify()

notify() method wakes up the first thread that called wait() on the same object.

Eg.

```
class sus extends Thread implements Runnable
    {
        static Thread th;
        float rad,r;
        public sus()
        {
            th= new Thread();
            th.start();
        }
        public void op()
        {
            System.out.println("\nThis is OP");
            if(rad==0)
            {
                System.out.println("Waiting for input radius");
                try
                {
                    wait();
                }
                catch(Exception ex)
                {
                }
            }
        }
        public void ip()
        {
            System.out.println("\nThis is IP");
            r=7;
            rad= r;
            System.out.println(rad);
            System.out.println("Area = "+3.14*rad*rad);
            notify();
        }
```

```
                public static void main(String arp[])
                {
                        try{
                        sus s1 = new sus();
                        System.out.println("\nReady to go");
                        Thread.sleep(2000);
                        System.out.println("\nI am resuming");
                        th.suspend();
                        Thread.sleep(2000);
                        th.resume();
                        System.out.println("\nI am resumed once again");
                        s1.op();
                        s1.ip();
                        s1.op();
                        }
                        catch(Exception e)
                        {}
                }
        }
```

	(e)	**What is type casting? Explain its types with proper syntax and example.**	**4M**
	Ans:	Assigning a value of one type to a variable of another type is known as Type Casting There are 2 types of type casting 1. Widening or Implicit type casting 2. Narrowing or Explicit type casting **1. Widening or Implicit type casting** **byte → short → int → long → float → double** **widening** Implicitly Type casting take place when • The two types are compatible • The target type is larger than the source type Program:	**(Explanation of type casting:1 mark, Explanation of types: 1 mark, Syntax & example:2 marks)**

public class Test {
public static void main(String args[]) {
int i = 100;
long l = i; // no explicit type casting require
float f = l; // no explicit type casting required
System.out.println ("Int value " + i);
System.out.println ("Long value " + l);
System.out.println ("Float value " + f);
}}
	Output:
	
	Int value 100
	Long value 100
	Float value 100.0

2. Narrowing or Explicit type casting

- When you are assigning a larger type value to a variable of smaller type. Then you need to perform explicit type casting.

public class Test
{
public static void main(String args[]) {
double d = 100.04;
long l = (long) d; // explicit type casting required
int I = (int) l; // explicit type casting required
 	
System.out.println ("Double value " + d);
System.out.println ("Logn value " + l);
System.out.println ("Int value " + I);
 	
}
}
	Output:
	
	Double value 100.04
	Long value 100
	Int value 100

4.	(A)	Attempt any THREE of the following:	12Marks
	(a)	State & explain scope of variable with an example.	4M
	Ans:	{{**Note :-Any other example can be considered **}} Scope of Variable : • We can declare variables within any block • One block equal to one new scope in Java thus each time you start a new block, you are creating a new scope. • A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects. Program:	(Explanation: 2 marks & Example:2 marks)

```
// demonstrate block scope.
class Scope {
public static void main (String args[]) {
 int n1;   // Visible in main
 n1   =  10;
if(n1==10) {
                 // start new scope
int n2  = 20; // visible only to this block
                // n1 and n2 both visible here.
System.out.println("n1 and n2 : " + n1 + " " + n2);
          }
          //n2 = 100; // Error! N2 not known here
          //n1 is still visible here.
System.out.println("n1 is " + n1);
               }
}
```

Output:

n1 and n2 ; 10 20

n1 is 10

- n1 is declared in main block thus it is accessible in main block.
- n2 is declared in if block thus it is only accessible inside if block
- Any attempt to access it outside block will cause compilation error.
- Nested Block can have access to its outmost block.
- If block is written inside main block thus all the variables declared inside main block are accessible in if block

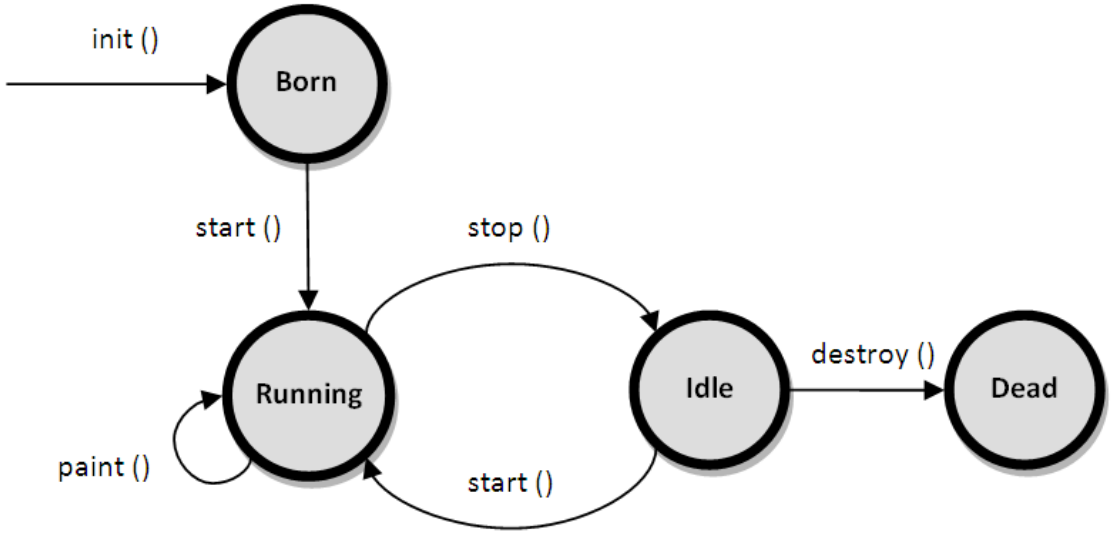
	(b)	With syntax and example explain try & catch statement.	4M
	Ans:	{{**Note :-Any other example can be considered **}} **try**- Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the **try** block, it is thrown.	(Explanation of each with

Syntax: try
{
// block of code to monitor for errors
}

catch- Your code can catch this exception (using **catch**) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java runtime system. A catch block immediately follows the try block. The catch block can have one or more statements that are necessary to process the exception.
Syntax: catch (*ExceptionType1 exOb*)
{
// exception handler for *ExceptionType1*
}
E.g.

```
class DemoException {
public static void main(String args[])
{
try
 {
int b=8;
 int c=b/0;
 System.out.println("Answer="+c);
 }
catch(ArithmeticException e)
 {
System.out.println("Division by Zero");
 } } }
```

syntax:1 mark & Example: 2 marks)

	(c)	**Explain applet life cycle with suitable diagram.**	**4M**
	Ans:		**(Diagram :1 mark, explanation: 3 marks)**

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. The applet states include:

- Born or initialization state
- Running state
- Idle state
- Dead or destroyed state

Initialization state: Applet enters the initialization state when it is first loaded. This is done by calling the init() method of Applet class. At this stage the following can be done:

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

Initialization happens only once in the life time of an applet.

```
public void init()
 {
//implementation
 }
```

Running state: applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. start() can also be called if the applet is already in idle state. start() may be called more than once. start() method may be overridden to create a thread to control the applet.

```
public void start()
 {
 //implementation
}
```

Idle or stopped state: an applet becomes idle when it is stopped from running. Stopping occurs automatically when the user leaves the page containing the currently running applet. stop() method may be overridden to terminate the thread used to run the applet.

```
 public void stop()
{
//implementation
}
```

Dead state: an applet is dead when it is removed from memory. This occurs automatically by invoking the destroy method when we quit the browser. Destroying stage occurs only once in the lifetime of an applet. destroy() method may be overridden to clean up resources like threads.

```
 public void destroy()
 {
 //implementation
 }
```

Display state: applet is in the display state when it has to perform some output operations on the screen. This happens after the applet enters the running state. paint() method is called for this. If anything is to be displayed the paint() method is to be overridden.

```
public void paint(Graphics g)
{
//implementation
}
```

(d) Explain byte stream class in detail. **4M**

Ans:

- It is used for creating & manipulating streams and files for reading and writing bytes.
- Since the steams are unidirectional, they can transmit bytes in only one direction and, therefore, java provides two kinds of byte stream classes –

1. Input Stream classes
2. Output Stream classes

1. Input stream classes

- Input stream classes are used to read 8-bit bytes include a super class known as InputStream and a number of subclasses for supporting various input-related functions.
- The superclass InputStream is an abstract class and therefore, we cannot create instances of this class. Rather we must use the subclasses that inherit from this class. The InputStream includes methods that are designed to perform the following tasks:
- Reading bytes
- Closing streams
- Making position in the streams
- Finding the number of bytes in stream
- Methods of InputStream & FileInputStream class

Methods	Description
int read()	Read byte from Input stream /File Input Stream
int read (byte b [])	Read an arrayof bytes into b
int read(byte b [],int n, int m)	Read m bytes into b starting from n^{th} byte
int available ()	Gives number of bytes available in the input
long skip(long n bytes)	Skips over n bytes from the input stream and return the number of bytes actually ignored.
void close ()	Closes the input stream / file Input Stream

(Explanation of byte stream class : 2 marks, Each type:1 mark)

2. Output Stream classes:

- Output Stream classes are used to write 8-bit bytes
- Output Stream classes are derived from bsae class OutputStream.
- Like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it.
- The several subclasses of the OutputStream can be used for performing the output operations.
- The OutputStream includes methods that are designed to perform the following task:

1. Writing bytes
2. Closing streams
3. Flushing streams

Methods of InputStream & fileInputStream class :

Methods	Description
void write(int b)	Write a bytes to the output stream
void write(byte b[])	Write all bytes in the array b to the output stream
void write(byte []),int n, int m)	Write m bytes from array b starting from n^{th} byte
void close()	Closes the output stream

	(B)	**Attempt any ONE of the following:**	**6 Marks**
	(a)	**Write a java program to implement following functions of string:** **(1) Calculate length of string** **(2) Compare between strings** **(3) Concatenating strings**	**6 M**

	Ans:	**{{**Note:- Any relevant program can be considered**}}**
		
		class StringDemo
		{
		public static void main(String args[])
		{
		String str1="INDIA";
		String str2="India";
		String str3="My India";
		String str4="India";
		System.out.println("The length of string INDIA is "+str1.length()); //Length of string
		System.out.println("Comparing String India and India is "+str2.equals(str4));
		System.out.println("Comparing String INDIA and India is "+str1.equals(str2));
		System.out.println("Comparing String INDIA and India with equalsIgnoreCase is
			(Each function: 2 marks)

		"+str1.equalsIgnoreCase(str2)); String str5="I Love"; System.out.println("Result of concatinating of string is "+str5.concat(str3)); } } **Output (optional)** The length of string INDIA is 5 Comparing String India and India is true Comparing String INDIA and India is false Comparing String INDIA and India with equalsIgnoreCase is true Result of concatenating of string is I LoveMy India	
	(b)	**Write a java program to extend interface assuming suitable data.**	**6 M**

	Ans:	**{{**Note: - Any other relevant program shall be considered**}}**
		
		interface A
		{
		int x=20;
		void display();
		}
		interface B extends A
		{
		int y=30;
		void show();
		}
		class C implements B
		{
		public void display()
		{
		System.out.println("A's X="+x);
		}
		public void show()
		{
		System.out.println("A's X="+x);
		System.out.println("B's Y="+y);
		}
		public static void main(String args[])
		{
		C obj=new C();
		obj.display();
		obj.show();
		}
		}
			(Creation of interface and extend it: 4 marks & implementing into class:2 marks)

5.		Attempt any TWO of the following :	16Marks
	(a)	Write a java program to implement runnable interface with example.	8M
	Ans:	{{**Note:- Any other relevant program shall be considered **}}	(Correct logic : 4 marks, Syntax : 4 marks)

```
class NewThread implements Runnable {
Thread t;
NewThread() {
// Create a new, second thread
t = new Thread(this, "Demo Thread");
System.out.println("Child thread: " + t);
        t.start(); // Start the thread
}
// This is the entry point for the second thread.
public void run() {
        try {
                for(int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
        }
        } catch (InterruptedException e) {
        System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
}
}
class ThreadDemo {
public static void main(String args[]) {
        new NewThread(); // create a new thread
try {
        for(int i = 5; i > 0; i--) {
        System.out.println("Main Thread: " + i);
        Thread.sleep(1000);
}
} catch (InterruptedException e) {
        System.out.println("Main thread interrupted.");
}
        System.out.println("Main thread exiting.");
}
}
```

	(b)	**Write a java program to display all the odd numbers between 1 to 30 using for loop & if statement.**	**8M**

	Ans:	class OddNum
		{
		public static void main(String args[])
		{
		for(int i=1;i<=30;i++)
		{
		if(i%2 ==1)
		{
		System.out.print("Odd number :"+i +"\n");
		
		}
		}
		}
		}
			(Correct logic : 4 marks, Syntax : 4marks)

	(c)	**Explain following methods for applet with an example:** **(1) Passing Parameter to applet** **(2) Embedding <applet> tags in java code.**	**8M**

	Ans:	**{{**Note :-Any other example can be considered **}}**
		1)Passing parameter to applet
		Parameter can be passed to applet through Param attributes of applet tag.
		[<PARAM NAME=attributeName1 VALUE=attributeValue>]
		
		getParameter() Method: The getParameter() method of the Applet class can be used to retrieve the parameters passed from the HTML page.
		Syntax of getParameter() method :
		String getParameter(String param-name)
		2)**APPLET Tag:** Embedding applet tag in java code by two ways.
		• Adding applet tag to html file
		• Adding applet tag in java source file
		1. Example for embedding <applet> tag in html file.
		import java.awt.*;
		import java.applet.*;
		public class hellouser extends Applet
		{
		String str;
		public void init()
		{
		str = getParameter("username");
		str = "Hello "+ str;
			(Method of Passing parameter : 2marks, Applet tag explanation : 2 marks, Example : 4 marks)

```
                    }
                public void paint(Graphics g)
                {
                        g.drawString(str,10,100);
                }
                }
<HTML>
<Applet code = hellouser.class width = 400 height = 400>
 <PARAM NAME = "username" VALUE = abc> </Applet>
</HTML>
```

2. Example for embedding <applet> tag in Java source file.

```
import java.awt.*;
import java.applet.*;
        /*<Applet code = hellouser.class width = 400 height = 400>
        <PARAM NAME = "username" VALUE = abc>
        </Applet>*/
public class hellouser extends Applet
{
        String str;
        public void init()
        {
                str = getParameter("username");
                str = "Hello "+ str;
}
public void paint(Graphics g)
{
        g.drawString(str,10,100);
}
}
```

6.		**Attempt any FOUR of the following :**	**16Marks**
	(a)	**Explain following bitwise operator with an example:** **(1) left shift operator** **(2) write shift operator**	**4M**
	Ans:	**{{**Note: Any other example in program also can be considered **}}** **Ans:** **The Left Shift (<<):** the left shift operator, <<, shifts all of the bits in a 'value' to the left a specified number of times specified by 'num' General form : value <<num **e.g.** x << 2 (x=12) 0000 1100 << 2	**(Each Bitwise operator explanation: 1 mark, Each example:1 mark)**

		= 0011 0000 (decimal 48) **The Right Shift (>>):** the right shift operator, >>, shifts all of the bits in a 'value' to the right a specified number of times specified by 'num' General form: value >>num **e.g.** x>> 2 (x=32) 0010 0000 >> 2 = 0000 1000 (decimal 8)	
	(b)	**State & explain types of errors in Java.**	**4M**
	Ans:	**Types of Error** **1) Compile Time Errors:** All syntax errors given by Java Compiler are called compile time errors. When program is compiled java checks the syntax of each statement. If any syntax error occurs, it will be displayed to user, and .class file will not be created. **Common compile time Errors** 1) Missing semicolon 2) Missing of brackets in classes and methods 3) Misspelling of variables and keywords. 4) Missing double quotes in Strings. 5) Use of undeclared variable. 6) Incompatible type of assignment/initialization. 7) Bad reference to object. **2) Run time error:** After creating .class file, Errors which are generated, while program is running are known as runtime errors. **Results in termination of program.** **Common runtime Errors** 1. Dividing an integer by zero. 2. Accessing an element that is out of the bounds of an array. 3. Trying to store data at negative index value. 4. Opening file which does not exist. 5. Converting invalid string to a number.	**(Two types : 2 marks each)**
	(c)	**Enlist types of constructor. Explain any two with example.**	**4M**
	Ans:	**{{**Note :- Any other example can be considered **}}** **Types of constructors in java:** **1. Default constructor (no-arg constructor)** **2. Parameterized constructor** **3. copy constructor**	**(List : 1 mark , Any 2 explanation with example : 1 ½ mark**

1. Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class_name>()
{ }

Example:

```
class Bike1{
Bike1()
{
        System.out.println("Bike is created");
}
public static void main(String args[]){
        Bike1 b=new Bike1();
}
}
```

2. parameterized constructor

A constructor which has a specific number of parameters is called parameterized constructor.

Parameterized constructor is used to provide different values to the distinct objects.

```
class Student4{
   int id;
   String name;

   Student4(int i,String n){
            id = i;
          name = n;
   }
   void display()
{    System.out.println(id+" "+name);}
     public static void main(String args[])
{
          Student4 s1 = new Student4(111,"Karan");
          Student4 s2 = new Student4(222,"Aryan");
          s1.display();
            s2.display();
  }
}
```

each)

3. Copy Constructor:

A copy constructor is used for copying the values of one object to another object.

Example:

```
class Student6{
        int id;
        String name;
        Student6(int i,String n)
        {
        id = i;
        name = n;
        }

Student6(Student6 s){
        id = s.id;
        name =s.name;
}
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);          //copy constructor called
        s1.display();
        s2.display();
}
}
```

	(d)	**How to add new class to a package? Explain with an example.**	**4M**
	Ans:	**{{ **Note :-Any other example shall be considered **}}** Create a package : simply include a **package** command as the first statement in a Java source file. **Include classes:** Any classes declared within that file will belong to the specified package. The **package** statement defines a name space in which classes are stored. If you omit the **package** statement, the class names are put into the default package, which has no name. **Syntax:** **package *pkg*;** //Here, *pkg* is the name of the package **public class class_name** **{** **//class body goes here** **}** **Example:** package package1; public class Box	**(Explanation :2 marks, Any Example: 2 marks)**

```
{
int l= 5;
int b = 7;
int h = 8;
    public void display()
    {
        System.out.println("Volume is:"+(l*b*h));
    }
}
```

Source file:

```
import package1.Box;
class   VolumeDemo
{
    public static void main(String args[])
    {
Box b=new Box();
b.display();
    }
}
```

	(e)	**Explain Arrary list & Iterator methods of collections with an example.**	**4M**
	Ans:	**Methods of ArrayList class :** **1. void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >size()). **2. boolean add(Object o)** Appends the specified element to the end of this list. **3. boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null. **4. boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null. **5. void clear()** Removes all of the elements from this list. **6. Object clone()** Returns a shallow copy of this ArrayList.	**(Any 2 Methods from each : 2 marks each)**

7. boolean contains(Object o)
Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e))

8. void ensureCapacity(int minCapacity)
Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

9. Object get(int index)
Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

10. int indexOf(Object o)
Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

11. int lastIndexOf(Object o)
Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

12. Object remove(int index)
Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size()).

13. protected void removeRange(int fromIndex, int toIndex)
Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

14. Object set(int index, Object element)
Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

15. int size()
Returns the number of elements in this list.

16. Object[] toArray()
Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.

17. Object[] toArray(Object[] a)
Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

		18. void trimToSize() Trims the capacity of this ArrayList instance to be the list's current size. **Methods of Iterator class :** 1. **boolean hasNext( ) :**Returns true if there are more elements. Otherwise, returns false. 2. **Object next( )**:Returns the next element. Throws NoSuchElementException if there is not a next element. 3. **void remove( ):** Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).	