12

```
        catch(IOException e)
        {
          System.out.println(e);
          System.exit(-1);
        }
        finally
        {
          try
          {
            fr.close();
            fw.close();
        System.out.println("\n..Its Done....\n");
        System.out.println("..See input.txt and  output.txt files.....\n\n");
          }
          catch(IOException e)
          {}
        }}//main}//class
```

(c) **Explain the semantic and functionality of the given statements:**
1m each

i) b.addActionListener(this);
ii) repaint()
iii) public void paint(Graphics g){..}
iv) <applet>tag in java program
v) appletviewer

(d) **What is thread? Discuss the various ways of creating threads in java.**
Thread concept: 1m
Explanation about two ways for creation:4m

(e) **What do you mean by Member Inner class? Explain it with suitable example.**

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{
 //code
 class Inner{
  //code
 }
}
```

Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
 TestMemberOuter1 obj=new TestMemberOuter1();
 TestMemberOuter1.Inner in=obj.new Inner();
 in.msg();
 }
}
```

****

```
        print.setText("Simple Interest is "+Simple_Interest);
      }
      if(b2==ae.getSource())
      {
Final_Amount=Principal_Amount+simple_interest(Principal_Amount,No_of_Years,Rate_of_Interest)
;
        print.setText("Final Amount is "+Final_Amount);
      }
    }
    public float simple_interest(float P,float R,float T)
    {
      SI=(P*R*T)/100;
      return SI;
    }
  }
```

(f) **Briefly Explain: Event Delegation Model**

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source – the class which broadcasts the events
- Event Listeners – the classes which receive notifications of events
- Event Object – the class object which describes the event.

An event occurs (like mouse click, key press, etc) which is followed by the event is broadcasted by the event source by invoking an agreed method on all event listeners. The event object is passed as argument to the agreed-upon method. Later the event listeners respond as they fit, like submit a form, displaying a message / alert etc.

Q. 5 **Attempt the following (Any THREE)**

(a) **Write a program to accept 3 numbers from command line and perform their sum.**

Program:5m

(b) **Write a java program to copy the content of abc.txt file to pqr.txt file.**

```
import java.io.*;
class CopyFile{
public static void main(String args[]){
File FI =new File("input.txt");
File FO=new File("output.txt");
FileReader fr=null;
FileWriter fw=null;
try{
fr=new FileReader(FI);
fw=new FileWriter(FO);
int ch;
  while((ch=fr.read())!=-1) {
     fw.write(ch);
System.out.print(" "+ch);
  }
 }
```

```
        Integer dec = Integer.decode(decimal);
        System.out.println("decode(45) = " + dec);
        dec = Integer.decode(octal);
        System.out.println("decode(005) = " + dec);
        dec = Integer.decode(hex);
        System.out.println("decode(0x0f) = " + dec);

        int valrot = 2;
        System.out.println("rotateLeft(0000 0000 0000 0010 , 2) =" +
                        Integer.rotateLeft(valrot, 2));
        System.out.println("rotateRight(0000 0000 0000 0010,3) =" +
                        Integer.rotateRight(valrot, 3));
    }
}
```

(d) **Differentiate between Choice and List AWT controls.**

1m for each difference

(e) **Write an applet program that accepts Principle Amount, No. of Years & Rate of Interest from 3 text fields, when you click "Calculate Interest" button, simple interest should be calculated. When you click on "Final Amount" button, the final amount by adding principle amount and interest should be displayed.**

Program 5m

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class SimpleInterest extends Applet implements ActionListener
{
  Label l1,l2,l3,print;
  TextField t1,t2,t3;
  Button b1,b2;
  float Simple_Interest,Principal_Amount,No_of_Years,Final_Amount,Rate_of_Interest,SI;
  public void init()
  {
    l1=new Label("Enter the Principal Amount : ");
    l2=new Label("Enter the No. of Years : ");
    l3=new Label("Enter the Rate of Interest : ");
    t1=new TextField(20);
    t2=new TextField(20);
    t3=new TextField(20);
    b1=new Button(" Calculate Interest ");
    b2=new Button(" Final Amount ");
    print=new Label("                                    ");
    add(l1);    add(t1);    add(l2);    add(t2);    add(l3);    add(t3);
    add(b1);    add(b2);    add(print);
    b1.addActionListener(this);
    b2.addActionListener(this);
  }
  public void actionPerformed(ActionEvent ae)
  {
    Principal_Amount=Float.parseFloat(t1.getText());
    No_of_Years=Float.parseFloat(t2.getText());
    Rate_of_Interest=Float.parseFloat(t3.getText());
    if(b1==ae.getSource())
    {
      Simple_Interest=simple_interest(Principal_Amount,No_of_Years,Rate_of_Interest);
```

4 isEmpty( )Returns true if the collection has no elements.

5 iterator( )Returns an Iterator object for the collection, which may be used to retrieve an object.

6 remove( )Removes a specified object from the collection.

7 size( )Returns the number of elements in the collection.

(c) **Demonstrate the working of Integer wrapper class with suitable program.**

```
//Java program to illustrate
//various Integer methods
public class Integer_test
{
   public static void main(String args[])
   {
      int b = 55;
      String bb = "45";

      Integer x = new Integer(b);
      Integer y = new Integer(bb);

      System.out.println("toString(b) = " + Integer.toString(b));


      System.out.println("toHexString(b) =" + Integer.toHexString(b));
      System.out.println("toOctalString(b) =" + Integer.toOctalString(b));
      System.out.println("toBinaryString(b) =" + Integer.toBinaryString(b));

      Integer z = Integer.valueOf(b);
      System.out.println("valueOf(b) = " + z);
      z = Integer.valueOf(bb);
      System.out.println("ValueOf(bb) = " + z);
      z = Integer.valueOf(bb, 6);
      System.out.println("ValueOf(bb,6) = " + z);


      int zz = Integer.parseInt(bb);
      System.out.println("parseInt(bb) = " + zz);
      zz = Integer.parseInt(bb, 6);
      System.out.println("parseInt(bb,6) = " + zz);

      int prop = Integer.getInteger("sun.arch.data.model");
      System.out.println("getInteger(sun.arch.data.model) = " + prop);
      System.out.println("getInteger(abcd) =" + Integer.getInteger("abcd"));

      System.out.println("getInteger(abcd,10) =" + Integer.getInteger("abcd", 10));

      String decimal = "45";
      String octal = "005";
      String hex = "0x0f";
```

```
            System.err.println("Could not listen on port: 44.");
            System.exit(1);
        }
        Socket clientSocket = null;
        try
        {
            clientSocket = serverSocket.accept();
        }
        catch (IOException e)
        {
            System.err.println("Accept failed.");
            System.exit(1);
        }
        BufferedReader br = new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream( )));
        String namesfromclient[ ] = new String[5];
        for(i=0;i<=4;i++)
        {
            namesfromclient[i]=br.readLine( );
        }
        System.out.println("Names received from client");
        pw.write("Good bye");
        pw.flush( );
        pw.close( );
        br.close( );
        clientSocket.close( );
        serverSocket.close( );
    }
}
```

Q. 4 Attempt the following (Any THREE)

(a) **What is layout manager? Explain any two.**

2.5m for each Layout manager

(b) **Discus various methods of set interface in detail.**

1m for each method

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

The methods declared by Set are summarized in the following table –

Sr.No.	Method & Description
1	add( ) Adds an object to the collection.
2	clear( )Removes all objects from the collection.
3	contains( )Returns true if a specified object is an element within the collection.

7

void	**setSoTimeout**(int timeout) Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
String	**toString**() Returns the implementation address and implementation port of this socket as a String.

(f) Write a client socket that will accept n names from user and send them to the server. After receiving the names, the server socket should send the message "Good Bye" and close the connection.

Client program

```
import java.net.*;
import java.io.*;
public class ClientConnection
{
        String names[ ]=new String[5];
        public static void main(String ar[ ]) throws Exception
        {
                DataInputStream dis = new DataInputStream(System.in);
                System.out.println("Enter 5 names");
                for(int i=0;i<=4;i++)
                {
                        names[i]=dis.readLine( );
                }
                new ClientConnection().connectandsend( );
        }

        public void connectandsend( )
        {
                InetAddress ser = InetAddress.getByName("NetworkServer");
                Socket clientSocket = new Socket(ser,80);
                PrintWriter pw = new PrintWriter(clientSocket.getOutputStream(),true);
                BufferedReader br = new BufferedReader
                (new InputStreamReader(clientSocket.getInputStream( )));
                pw.write(message);
                pw.flush( );
                pw.close( );
                br.close( );
        }
}
```

Server program

```
import java.net.*;
public class ServerConnection
{
        public static void main(String ar[ ]) throws Exception
        {
                int i;
                ServerSocket serverSocket = null;
                try
                {
                        serverSocket = new ServerSocket(44);
                }
                catch (IOException e)
                {
```

6

1m each difference.

(e) **Discuss any five methods of java.net.ServerSocket class.**

1m each methods

Socket	**accept**() Listens for a connection to be made to this socket and accepts it.
void	**bind**(SocketAddress endpoint) Binds the ServerSocket to a specific address (IP address and port number).
void	**bind**(SocketAddress endpoint, int backlog) Binds the ServerSocket to a specific address (IP address and port number).
void	**close**() Closes this socket.
ServerSocketChannel	**getChannel**() Returns the unique **ServerSocketChannel** object associated with this socket, if any.
InetAddress	**getInetAddress**() Returns the local address of this server socket.
int	**getLocalPort**() Returns the port number on which this socket is listening.
SocketAddress	**getLocalSocketAddress**() Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
int	**getReceiveBufferSize**() Gets the value of the SO_RCVBUF option for this ServerSocket, that is the proposed buffer size that will be used for Sockets accepted from this ServerSocket.
boolean	**getReuseAddress**() Tests if SO_REUSEADDR is enabled.
int	**getSoTimeout**() Retrieve setting for SO_TIMEOUT.
protected void	**implAccept**(Socket s) Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
boolean	**isBound**() Returns the binding state of the ServerSocket.
boolean	**isClosed**() Returns the closed state of the ServerSocket.
void	**setPerformancePreferences**(int connectionTime, int latency, int bandwidth) Sets performance preferences for this ServerSocket.
void	**setReceiveBufferSize**(int size) Sets a default proposed value for the SO_RCVBUF option for sockets accepted from this ServerSocket.
void	**setReuseAddress**(boolean on) Enable/disable the SO_REUSEADDR socket option.
static void	**setSocketFactory**(SocketImplFactory fac) Sets the server socket implementation factory for the application.

5

{//statement to be synchronized
}Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.
If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.
Consider an example, Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.
To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods:

Using Synchronized methods is a way to accomplish synchronization

Synchronized Keyword:

To synchronize above program, we must *synchronize* access to the shared **display()** method, making it available to only one thread at a time. This is done by using keyword **synchronized** with display() method.

synchronized void display (String msg)

Using Synchronized block

If you have to synchronize access to an object of a class or you only want a part of a method to be synchronized to an object then you can use synchronized block for it.

(c) Write an application that generates custom exception if any value from its command line arguments is negative.

```
class CustomExceptionDemo
    {
        public static void main(String ar[ ])
        {
            int n = Integer.parseInt(ar[0]);
            System.out.println("main starts");
            try
            {
            System.out.println("In try block");
            if(n<0)
            {
            throw new CustomException("Number is negative");
            }
            }catch(CustomException ce)
            {
            System.out.println("Custom Exception occurs");
            }
            System.out.println("main ends");
        } //end of main
    } //end of class
    public class CustomException extends Exception
    {
        public CustomException(String m)
        {
            super(m);
        }
    }
```

(d) **Differentiate between byte stream classes and character stream classes.**

- Default – No keyword required
- Private
- Protected
- Public

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

(f) **Differentiate between classes and interfaces.**

BASIS FOR COMPARISON	CLASS	INTERFACE
Basic	A class is instantiated to create objects.	An interface can never be instantiated as the methods are unable to perform any action on invoking.
Keyword	class	interface
Access specifier	The members of a class can be private, public or protected.	The members of an interface are always public.
Methods	The methods of a class are defined to perform a specific action.	The methods in an interface are purely abstract.
Implement/Extend	A class can implement any number of interface and can extend only one class.	An interface can extend multiple interfaces but can not implement any interface.
Constructor	A class can have constructors to initialize the variables.	An interface can never have a constructor as there is hardly any variable to initialize.

Q. 3 Attempt the following (Any THREE)

(a) **Explain five keywords of exception handling in java.**

1m for each keyword

Try throw thows catch finaaly.

(b) **What do you mean by synchronising thread? Explain it in detail.**

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called **synchronization**. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax :

synchronized (object)

3

```
{
public static void main(String args[])
{
int phno,calls;
double ans;

phno=Integer.parseInt(args[0]);
calls=Integer.parseInt(args[1]);

BillRate br=new BillRate();
ans=br.check(calls);

System.out.println("The bill for Telephone no" +phno+"for "+calls+"calls is "+ans);
}}
```

(c) **What do you mean by extending an interface? Explain it with suitable example. Concept: 1m**

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.

Program: 4m

```
interface A
{
void meth1();
void meth2();
}
interface B extends A
{
void meth3();
}
class MyClass implements B
{
public void meth1()
{ System.out.println("Implement meth1()."); }
public void meth2()
{ System.out.println("Implement meth2()."); }
public void meth3()
{ System.out.println("Implement meth3()."); }
}
class IFExtend
{
public static void main(String arg[])
{
MyClass ob = new MyClass();
ob.meth1();
ob.meth2();
ob.meth3();
}}
```

(d) **Explain the following string library functions: toCharArray(), getBytes(), getChars(), charAt(), lastIndexOf().**

1m each method

(e) **List and explain access specifiers in java.**

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

of the execution.

- It is a variable which belongs to the class and not to object(instance)
- Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
- A single copy to be shared by all instances of the class
- A static variable can be accessed directly by the class name and doesn't need any object

Static method in Java is a method which belongs to the class and not to the object. A static method can access only static data.

- It is a method which belongs to the class and not to the object(instance)
- A static method can access only static data. It can not access non-static data (instance variables)
- A static method can call only other static methods and can not call a non-static method from it.
- A static method can be accessed directly by the class name and doesn't need any object
- A static method cannot refer to "this" or "super" keywords in anyway

(c) **Can we declare main() of java class as private? Why?**

Ans: Yes, we can declare main method as private. It compiles without any errors, but in runtime, it says main method is not public.

(d) **Which method is used add an element and corresponding key to a map?**

Ans: public put(K key, V value)

(e) When the constructor of class is invoked?

Ans: the first time (and only the first time) execution reaches point where object is declare.

Q. 2 **Attempt the following (Any THREE)**

(a) **Explain the concept final methods and final variables with a program.**

Final variable: When a variable is declared with *final* keyword, its value can't be modified, essentially, a constant. This also means that you must initialize a final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but internal state of the object pointed by that reference variable can be changed

Final Method: When we use *final* specifier with a method, the method cannot be overridden in any of the inheriting classes. Methods are made final due to design reasons.

Since private methods are inaccessible, they are implicitly final in Java. So adding *final* specifier to a private method doesn't add any value

Program :3m

(b) **Write a java program to input a telephone no. and number of calls calculate and display bill amount which includes a fix rent of rupees 400, the first 150 cost are free with excess calls charge 80 paise each.**

```
class BillRate
{
double rate;
        double check(int x)
        {
                rate=400;
                if(x>150)
                        rate=400+(x-150)*0.80;
                return rate;
        }
}

class BillDemo
```

53232

(1)

Q. 1 Attempt All (Each of 5Marks)

(a) Multiple Choice Questions:

1. Which of these events will be generated if we close an applet's window?
   d) **WindowEvent**
2. Method overriding is combination of inheritance and polymorphism?
   a) **True**
3. Which of these method is used to tell the calling thread to give up monitor and go to sleep until some other thread enters the same monitor?
   a) wait()
4. Which of these methods of Character wrapper can be used to obtain the character value contained in Character object.
   c) charValue()
5. What will be the output of following code?

```
class exception_handling
  {
    public static void main(String args[])
    {
      try
      {
        int a[] = {1, 2,3 , 4, 5};
        for (int i = 0; i < 7; ++i)
          System.out.print(a[i]);
      }
      catch(ArrayIndexOutOfBoundsException e)
      {
       System.out.print("0");
      }
    }
  }
```

b) **123450**

(b) Fill in the blanks: (class, raise, java.net, 32, JVM, java.lang, JIT, 64, object, throw java.util, new, throws, JDK)

1. IPV4 address consist of **32** bits in it.
2. To compile, debug and execute java program **JVM** component is used.
3. All java built in exceptions are located in **java.lang** package.
4. To make a class **class** keyword is used.
5. **throw** keyword is used to generate an exception explicitly.

(c) Answer in 1 – 2 sentences:

a) What is JVM? Is it platform independent?

Ans: A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages and compiled to Java bytecode. The JVM is detailed by a specification that formally describes what is required of a JVM implementation. YES it is platform independent.

(b) What is the purpose of static method and static variable?

Ans: Static variable in Java is variable which belongs to the class and initialized only once at the start