Java Assignment Solution

2014 batch

Q. What are exceptions? Why is it important to handle exceptions? Discuss different keywords that are used to handle exceptions.

Ans: Exception is a runtime error that if generated terminates the running program.

There are two types of exceptions:

1) Checked exception: These are the exceptions that are checked by the compiler during compilation. If these exceptions are not handled compiler shows error. For example,
   - IOException, SQLException, etc

2) Unchecked Exception: These are the exception that are not checked by the compiler during compilation. However, if generated during runtime, terminates the program execution.
   eg Arithmetic Exception, ArrayIndexOutOfBoundException.

It is important to handle exception so that the program continues running in case of different logical errors. Similarly, if exceptions are handled properly it shows error message to the user so that users can read and understand the source of problem which helps to avoid the error in the next run.

Different keywords that are used to handle exceptions are:

i) try: It is used to enclose a block of codes that may generate exceptions.

ii) catch: It is used to handle the exception generated in try block.

iii) finally: It is used to ensure that code written inside it gets executed 100%.

throw: It is used to throw our own exception.
throws: → used when it is used when our code generates checked exception and we don't have try/catch.

Syntax:

```
try {
    // block of code
}
catch(...)
{
    // block of code
}
finally {
    ...
    ...
}
```

Q.2 Write a program using components to add two numbers. Use text fields for input and output.

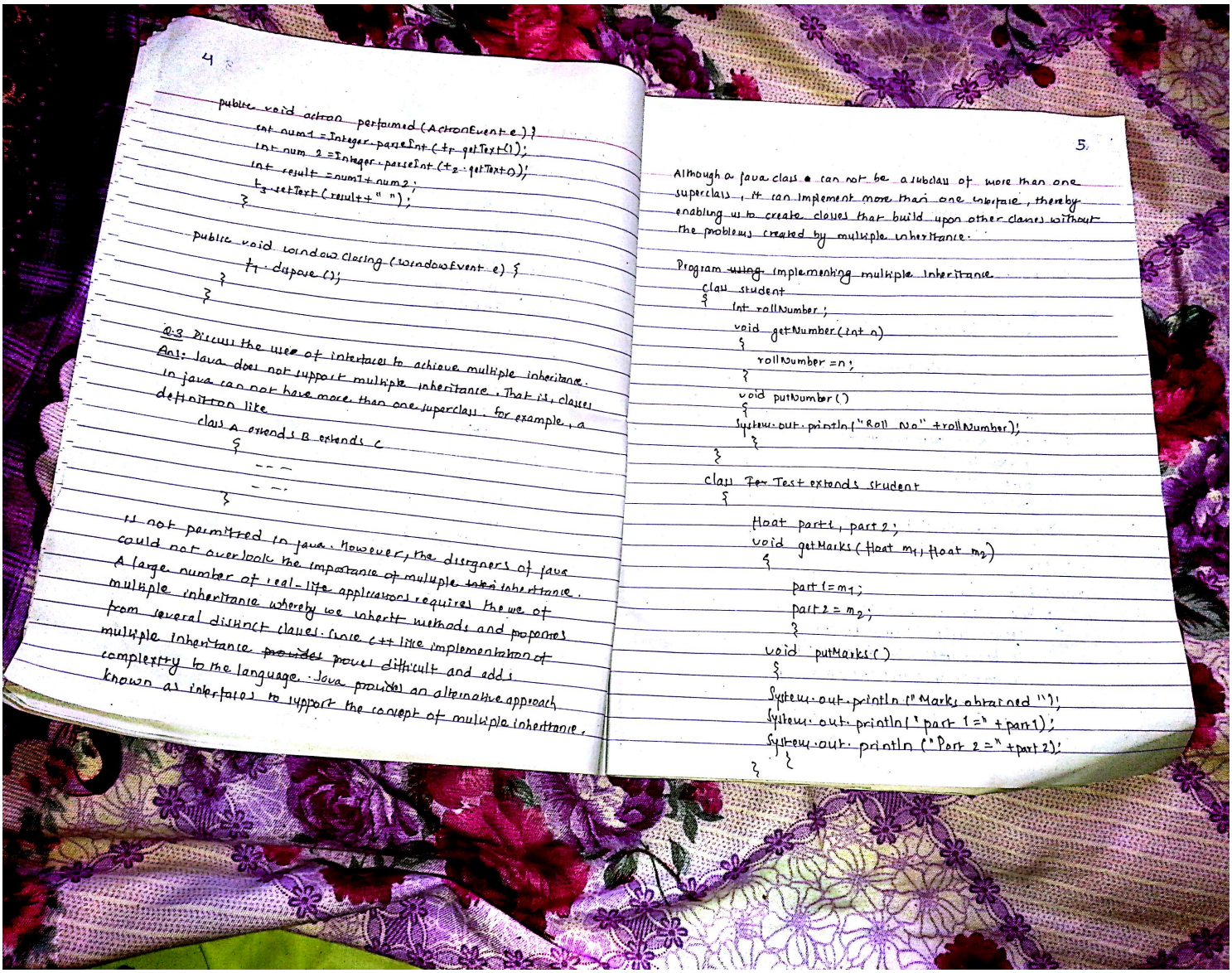
Ans:

```
package eventhandling;
import java.awt.*;
import java.awt.event.*;
public class Addition extends WindowAdapter
        implements ActionListener {
Frame f1;
Label d1, d2, d3;
```



```
Textfield t1, t2, t3;
Button b1;
Addition()
  {
f1 = new Frame();
f1.setLayout(new FlowLayout());
f1.addWindowListener(this);
d1 = new Label("Num 1");
d2 = new Label("Num 2");
t1 = new Textfield(10);
t2 = new Textfield(10);
t3 = new Textfield(10);
b1 = new Button("Add");
b1.addActionListener(this);
f1.add(d1);
f1.add(t1);
f1.add(d2);
f1.add(t2);
f1.add(d3);
f1.add(t3);
f1.add(b1);
f1.setSize(200,200);
f1.setVisible(true);
  }
public static void main(String args[]){
        new Addition();
  }
```

P.T.O.



4

```
public void actionPerformed(ActionEvent e){
    int num1 = Integer.parseInt(t1.getText());
    int num2 = Integer.parseInt(t2.getText());
    int result = num1+num2;
    t3.setText(result+" ");
    }

public void windowClosing(WindowEvent e){
    f1.dispose();
    }
}
```

Q.3 Discuss the use of interfaces to achieve multiple inheritance.

Ans: Java does not support multiple inheritance. That is, classes in java can not have more than one superclass. for example, a definition like

```
class A extends B extends C
    {
    ---
    ---
    }
```

is not permitted in java. However, the designers of java could not overlook the importance of multiple ~~inheritance~~ inheritance. A large number of real-life applications requires the use of multiple inheritance whereby we inherit methods and properties from several distinct classes. Since c++ like implementation of multiple inheritance ~~provides~~ poses difficult and adds complexity to the language. Java provides an alternative approach known as interfaces to support the concept of multiple inheritance.

5.

Although a java class can not be a subclass of more than one superclass, it can implement more than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

Program ~~using~~ implementing multiple inheritance

```
class student
{
    int rollNumber;
    void getNumber(int n)
    {
        rollNumber = n;
    }
    void putNumber()
    {
    System.out.println("Roll No" +rollNumber);
    }
}

class Test extends student
{
    float part1, part2;
    void getMarks(float m1, float m2)
    {
        part1 = m1;
        part2 = m2;
    }
    void putMarks()
    {
    System.out.println("Marks obtained");
    System.out.println("part 1 =" +part1);
    System.out.println("Part 2 =" +part2);
    }
}
```

Q.4 What is JDBC? Explain JDBC drivers.

Ans: JDBC is a Java API used for connecting java with database. It is made of java.sql package. The classes of this package to be used are:

(a) connection – holds the connection to the database.
(b) DriverManager – establish the connection to the database
(c) Statement – used to execute static sql statements.
(d) Prepared statement – used to execute pre-compiled sql statement.
(e) callablestatement – used to execute stored procedure.
(f) ResultSet – used to hold the data returned by sql in "select query"
(g)

steps for JDBC
- import required packages
- register with dbbc driver
- create connection with database
- perform required SQL operation
- close the connection

JDBC technology drivers fit into one of four categories
1. JDBC-ODBC driver
2. Native-API driver
3. Network-Protocol driver (Middleware driver)
4. Database-Protocol driver (Pure Java driver)

Type1 driver: JDBC-ODBC driver
The JDBC type1 driver also known as JDBC-ODBC bridge, is a database driver implementation that employs ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls.

Type 2 - driver - Native API driver
The JDBC type 2 driver, also known as the Native-API driver, is a database driver implementation that uses the client side libraries of the database. The driver converts JDBC method calls into native calls of the database API. For example, Oracle OCI driver is a type 2 driver.

Type 3 driver : Network - Protocol driver.
The JDBC type 3 driver, also known as the pure java driver for database middleware, is a database driver implementation which makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.

Type 4 driver : Database protocol driver
The JDBC type 4 driver, also known as the Direct to Database Pure Java Driver is a database driver implementation that ~~calls~~ converts calls directly into a vendor specific database protocol.

Q.5 What is a layout manager? Explain the types of layout managers available in java. ~~for~~ write programs to show the use of FlowLayout / BorderLayout.
→ A layout manager is an object that determines how components are arranged in a container
① Flow Layout
② Border layout
③ Grid layout
④ GridBagLayout

Borderlayout:
- A layout manager that arranges components in five regions: NORTH, SOUTH, WEST, EAST, CENTER.
- If the components in north, south, east or west are missing then other components expand horizontally or vertically to fill the space.
- If the center component is missing, the space is left blank.
- It has two constructors:
1. Borderlayout() // no gap between the components
2. Borderlayout(int hor, int ver)
- NORTH and SOUTH components use the entire width of the container.

Program using Borderlayout:

```
package layout
import javax.swing.*;
import java.awt.*;
public class BorderlayoutDemo extends JFrame {
JButton b1,b2,b3,b4,b5;
BorderlayoutDemo(){
setLayout(new Borderlayout())
setVisible(true)
setSize(200,200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
b1 =new JButton("north");
b2 = new JButton("south");
b3 = new JButton("west");
b4 = new JButton("center");
b5 =new JButton("east");
```



```
add(b1, Borderlayout.NORTH);
add(b2, Borderlayout.SOUTH);
add(b3, Borderlayout.WEST);
add(b4, Borderlayout.CENTER);
add(b5, Borderlayout.EAST);
}

public static void main(String args[]){
    new BorderlayoutDemo();
}
```

06 Gridlayout:
A layout manager that divides the entire container into a matrix of rows and columns.
- Each component is placed in the cell formed.
- Each component is has equal wide and height
- Size of component varies when we resize the container.

GridBagLayout
- similar to Gridlayout however, the size of each component can be different
- Size of component does not vary when the container is resized.
- A helper class GridBagConstraints is needed

Q.6 Differentiate between

(a) Grid Layout and GridBagLayout

GridLayout	GridBagLayout
- GridLayout is a very simple layout manager that can arrange components in a grid of rows and columns.	- GridBagLayout is much more complex and much flexible
- Each component has equal width and width.	- size of the component can be different.
- size of component varies when we resize the container	- size of component does not vary when the container is resize
- Constructor: GridLayout();	Constructor: GridBagLayout();

(b) TCP and UDP

TCP	UDP
- It is the short form of Transmission control protocol.	- It is the short form of User data gram protocol.
- It acknowledge the transmission of data.	- It does not acknowledge transmission of data.
- It is mostly used for two way communication.	- It is mostly used for unidirectional communication
- It is reliable form of data transmission.	- It is not reliable form of data transmission.
- The speed for TCP is slower than UDP	- UDP is faster because there is no error checking for packets.

TCP	UDP
- TCP does error checking	- UDP does error checking, but no recovery options.

Q.7) Mention the advantage of using adapter class over listener interface. List out some listener interface along with their adapter classes.

Ans: If an event listener is implemented directly by a class, all the methods within that interface need to be implemented making the code unnecessarily large.
This issue can be solved using the adapter class. To use an adapter, you create a subclass of it and override only the methods of interest, rather than directly implementing all methods of listener interface.

Following are some listener interface used in Java:-

i) ActionListener
This interface is used for receiving the action events.

ii) ItemListener
This interface is used for receiving the item event.

iii) TextListener
This interface is used for receiving the text event. The class which ~~provides~~ process the TextEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addTextListener() method.

iv) ActionListener
This interface is used for receiving the action events.

The class which process the ActionEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addActionListener() method. When the action event occurs, that objects actionPerformed method is invoked.

vi) ItemListener

This interface is used for receiving item events. The class which process the ItemEvent should implement this interface. The object of that class must be registered with a component. The object can be ~~regis~~ registered using the addItemListener() method. When the action event occurs, that object's itemStateChanged method is invoked.

vii) WindowListener

This interface is used for receiving window events. The class which processes the ~~window~~ WindowEvent should implement ~~the~~ this interface. The object of that ~~can~~ class must be registered with a component. The object can be registered using the addWindowListener method.

v) MouseListener

This interface is used for receiving Mouse events. The class which process the MouseEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using ~~the~~ the addMouseListener method.

Q8 What are the uses of super keywoid ? Explain any one use with a suitable example.

→ A subclass constructor is used to construct the instance variables of both subclass and the superclass. The subclass

```
class InnerTest
{
    public static void main(String args[])
    {
        BedRoom room1 = new BedRoom(10,12,70);
        int area1 = room1.area();
        int volume1 = room1.volume();
        System.out.println("Area1 = " + area1);
        System.out.println("Volume = " + volume);
    }
}
```

Q.9 What is method overriding? Explain with a suitable example.

→ Using the same method name in the base class and derived class with same method signature.

- It is called runtime polymorphism, dynamic binding or late binding.

There may be occasions when we want an object to respond to the same method but have different behavior when that method is called. That means, we should override the method defined in the superclass. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass. Then, when that method is called, the method defined in the subclass is invoked instead of the one in the superclass. This is known as overriding.


```
class super
{   int x;
    super(int x)
    {
        this.x = x;
    }
    void display()
    {
        System.out.println("super x = " + x);
    }
}
class sub extends super
{
    int y;
    sub(int x, int y)
    {
        super(x);
        this.y = y;
    }
    void display()
    {
        System.out.println("super x = " + x);
        System.out.println("sub y = " + y);
    }
}
class overrideTest
{
    public static void main(String args[])
    {   sub s1 = new sub(700, 200);
        s1.display();
    }
}
```

Q.10] What is constructor? Write a program to show constructor overloading.

Ans A special method that is used to initialize the instance variable. It is special in the sense that its name exactly the same as class name and it does not have return type not even void. It is called automatically when an object is created.

Program to show constructor overloading in java:-

```
class Language{
    String name;
    void setName (String t) {
        name = t;
    }
    void getName (){
        System.out.println ("Language name :"+name);
    }
    Language (){
    System.out.println ("Constructor method called");
    }
    Language (String t) {
        name=t;
    }
public static void main (String [] args)
    {
    Language cpp = new Language ();
    Language java = new Language ("java");
    cpp.setName ("C++");
    java.getName ();
    cpp.getName ();
} }
```


Q.11 What are the three uses of "final" keyword? Explain.

→ The final keyword in java is used to restrict the user. The java final keyword can be used in many context.
final can be
1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Uses of "final" keyword

① To create constant e.g. ~~finaldouble~~ final double PI = 3.14;

② To create final method

A method that cannot be overridden in the derived class

```
eg.  class A
     {
       final void show ()
       {
       }
     class B extends A
       {
         void show ()   // not possible
         {
         }
```

③ To create final class

A class that can not be inherited

eg. final class A
{
}
class B extends A // not possible.

Q.11 Mention the two uses of "abstract" keyword. Write a program to illustrate it.

Ans: The two uses of abstract method are:-

① to create abstract method
- A method which does not have body is called abstract method
- its body must be made in derived derived class i.e. it must be overridden.

② to create abstract class:
- A class which can not be instantiated ie. whose object can not be created is called abstract class.

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


Q.13 Explain serialization and de-serialization. Write a program to illustrate it.

Ans: Process of writing (storing) the state of an object into a persistent store ie file is called serialisation.
A class must implement serializable interface in order to write its object to file.
~~Object output~~ ObjectOutputStream class and its method writeObject() are used

Process of restoring the state of an object from file is called de-serialisation.
ObjectInputStream class and its method readObject() are used

```
import java.io.*;
class Person implements Serializable
{
    private int id;
    private string name;
    private double salary;

Person(int id, string name, double salary){
    this.id = id;
    this.name = name;
    this.salary = salary;
    }
public void showInfo(){
    System.out.println("Id :" + id);
    System.out.println("Name :" + name);
    System.out.println("Salary : " + salary);
}
```

```
public class ReadWriteObject {
    public static void main (String args[]) throws Exception{
        Person ob = new Person (7, "Ram", 10000);
        FileOutputStream fout = new FileOutputStream ("object.txt");
        ObjectOutputStream oout = new ObjectOutputStream(fout);
        oout.writeObject(ob);
        oout.close();
        fout.close();
        FileInputStream fin = new FileInputStream("d:\object.txt");
        ObjectInputStream oin = new ObjectInputStream (fin);
        Person p = (Person) o.(Person)oin.readObject();
        p.showInfo();
        oin.close();
        fin.close();
    }
}
```

Q. Write a program that displays all the records from the table tblemployee that contains the fields id, name and salary. [use data source name : empdatasource]

→

```
package jdbc
import java.awt.*;
import java.sql.*;
import javax.swing.*;
import java.awt.event.*;
public class DisplayRecords implements ActionListener
{
    JFrame f1;
    JTextArea t1;
    JButton b1;
    DisplayRecords()
    {
        f1 = new JFrame ("Display Records");
        f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        t1 = new JTextArea();
        b1 = new JButton ("Load");
        f1.add (t1, BorderLayout.CENTER);
        f1.add (b1, BorderLayout.SOUTH);
        b1.addActionListener (this);
        f1.setSize (300,300)
        f1.setVisible (true);
    }
public static void main (String[] args) {
    new DisplayRecords();
}
public void actionPerformed (ActionEvent e) {
try {
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection
                    ("jdbc:odbc:empdata_source");
    String sql = "select id, name, salary from tblemployee";
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery (sql);
    String str = "";
    while (rs.next()) {
        str = rs.getString ("id") + "\t";
        str += rs.getString ("name") + "\t";
        str += rs.getString ("salary") + "\n";
    }
```

```
String title = "id lt name lt salary \n";
    ta.setText(title+str);
    con.close();
    }
    catch(Exception ex){
        System.out.println(ex);
    }
    }
```

Q.15 Compare AWT and Swing.

Ans: AWT and Swing are the packages in Java that enables the programmer to create GUI based application. Similar to AWT, Swing is also a GUI toolkit that facilitates the creation of highly interactive GUI applications. However, Swing is more flexible and robust when it comes to implementing graphical components. One of the main differences between Swing and AWT is that Swing will always generate similar type of output irrespective of underlying output. AWT on the other hand, is more dependent on the underlying operating system for generating the graphic components; thus the output may vary from one platform to another.

Q.16 Create a TCP client-server application that allows a client to send a message to the server and the server responds the client by sending the same message in uppercase.

Ans: Import java.io.*;
import java.net.*;
class TCPServer
{

```
public static void main(String[] args) throws Exception {
String clientSentence, modifiedSentence;
ServerSocket welcomeSocket = new ServerSocket(6789);
Socket connectionSocket = welcomeSocket.accept();

BufferedReader fromClient = new BufferedReader
    (new InputStreamReader(connectionSocket.getInputStream()));
    clientSentence = fromClient.readLine();
// System.out.println("From client :"+ clientSentence);

modifiedSentence = clientSentence.toUpperCase();

DataOutputStream toClient = new DataOutputStream
    (connectionSocket.getOutputStream());
toClient.writeBytes(modifiedSentence + "\n");
    connectionSocket.close();
    }
    }
```

```
import java.io.*;
import java.net.*;
class TCPClient
{

public static void main(String argv[]) throws Exception
{
```

```
String sentence, modifiedSentence;
Socket clientSocket = new Socket("nd", 6789);
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
sentence = br.readLine();
DataOutputStream toServer = new DataOutputStream
        (clientSocket.getOutputStream());
toServer.writeBytes(sentence + "\n");
BufferedReader fromServer = new BufferedReader(new InputStream
        Reader(clientSocket.getInputStream()));
modifiedSentence = fromServer.readLine();
System.out.println("FROM Server: " + modifiedSentence);
clientSocket.close();
}
}
```

Q.12 Create an applet to display "Nepal College of Information Technology".

Ans:

```
package applet;
import java.applet.*;
import java.awt.*;
public class welcomeApplet extends Applet {
    public void paint(Graphics g) {
```



```
g.drawString("welcome to applet");
g.drawString("Nepal College of Information Technology",70,70)
}
}
```

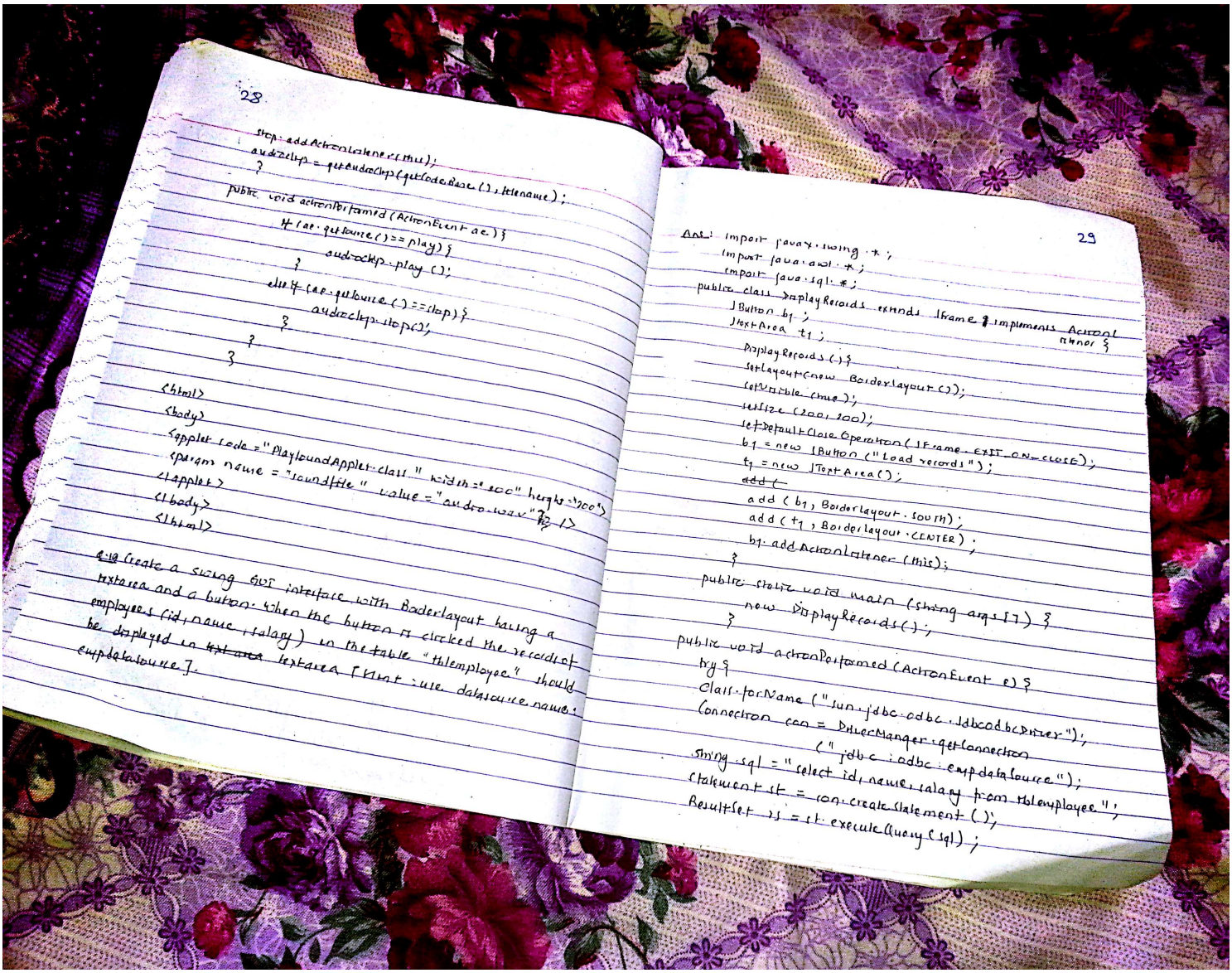
Code to embed applet in html page:

```
<html>
<body>
<applet code = "welcomeApplet.class" width = "300" height = "300">
</applet>
</body>
</html>
```

Q.13 Create an applet to play an audio file that is supplied as parameter from html.

Ans:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class PlaySoundApplet extends Applet implements ActionListener
{
    String fileName = "";
    Button play, stop;
    AudioClip audioClip;
    public void init()
    {
        fileName = getParameter("soundFile");
        play = new Button("play");
        add(play);
        play.addActionListener(this);
        stop = new Button("stop");
        add(stop);
```



28

```
stop.addActionListener(this);
audioClip = getAudioClip(getCodeBase(), fileName);
}
public void actionPerformed(ActionEvent ae){
    if(ae.getSource()==play){
        audioClip.play();
    }
    else if(ae.getSource()==stop){
        audioClip.stop();
    }
}
```

```
<html>
<body>
<applet code="PlaySoundApplet.class" width="300" height="300">
<param name="soundfile" value="audio.wav">
</applet>
</body>
</html>
```

Q.10 Create a swing GUI interface with Borderlayout having a textarea and a button. When the button is clicked the records of employees (id, name, salary) in the table "tblemployee" should be displayed in textarea. [Hint: use datasource name: empdatasource].

29

Ans:

```
import javax.swing.*;
import java.awt.*;
import java.sql.*;
public class DisplayRecords extends JFrame implements ActionListener {
    JButton b1;
    JTextArea t1;
    DisplayRecords(){
    setLayout(new BorderLayout());
    setVisible(true);
    setSize(200,200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    b1 = new JButton("Load records");
    t1 = new JTextArea();
    add(
    add(b1, BorderLayout.SOUTH);
    add(t1, BorderLayout.CENTER);
    b1.addActionListener(this);
    }
    public static void main(String args[]){
        new DisplayRecords();
    }
public void actionPerformed(ActionEvent e){
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManger.getConnection
                ("jdbc:odbc:empdatasource");
String sql = "select id,name,salary from tblemployee";
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(sql);
```

```
InsertRecords1(){
    setLayout(new FlowLayout());
    b1 = new JButton("Save");
    d1 = new JLabel("Id");
    d2 = new JLabel("name");
    d3 = new JLabel("salary");
    t1 = new JTextField(10);
    t2 = new JTextField(10);
    t3 = new JTextField(10);
    add(d1);
    add(t1);
    add(d2);
    add(t2);
    add(d3);
    add(t3);
    add(b1);
    b1.addActionListener(this);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300,300);
    setVisible(true);
    }
public static void main(String[] args){
        new InsertRecords1();
        }

public void actionPerformed(ActionEvent e){
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection
                ("jdbc:odbc:empdatasource");
```

```
String sql = "insert into tblemployee (id,name,salary)
              values (?,?,?)";
PreparedStatement pstmt = con.prepareStatement(sql);
int id = Integer.parseInt(t1.getText());
String name = t2.getText();
float salary = t3.Float.parseFloat(t3.getText());
pstmt.setInt(1,id);
pstmt.setString(2,name);
pstmt.setFloat(3,salary);
pstmt.executeUpdate();
con.close();
System.out.println("inserted in the database");
}
catch(Exception e){
System.out.println(e.getMessage());
}
}
```

Q.21 Lok out any 10 event classes. Also mention when these events are generated.

Ans:

Q.22 How is a component different from container? Create a simple AWT frame and add the close functionality to it.

Ans Component class is the super class to all the other classes from which various GUI elements are ~~st~~ realized. It is primary responsible for affecting the display of a graphic object on the screen. It also handles the various keyboard and mouse events of the GUI application.

Container: As the name suggests, the container object contains the other awt components. It manages the layout and placement of the various awt components within the container. A container object can contain other containers object as well; thus allowing nesting of containers.

```
import java.awt.*;
import java.awt.event.*;
public class FlowLayout GUIAppcn extends WindowAdapter{
    Frame f1;
    GUIAppcn(){
    f1=new Frame("GUI Application");
    f1.setLayout(new FlowLayout());
    f1.setVisible(true);
    f1.setSize(200,200);
    f1.addWindowListener(this);
    }
public static void main(String args[]){
        new GUIAppcn();
    }
public void windowClosing(WindowEvent e){
    f1.dispose();
    }
}
```

Q.23 List out any 10 listener interfaces. Also mention the abstract methods defined in those interface.

① ActionListener
The class which processes the ActionEvent should implement this interface. The object of that class must be registered using the addActionListener() method. When a the action event occurs, the object's actionPerformed method is invoked.

② Component Listener
The class which processes the ComponentEvent should implement this interface. The object of that class must be registered using the addComponentListener() method. When the action event occurs, the object component event are raised for information only.

③ key listener
The class which processes the key Event should implement this interface. The object of that class must be registered with the component. The object can be registered using the addKeyListener() method

④ AdjustmentListener
This interface is used for receiving adjustment events. The class which processes the Adjustment Event should registered with the addAdjustmentListener() method. The class that process events needs to implement this interface.

⑤ ContainerListener
The interface ContainerListener is used for receiving container

events. The class that process container events needs to implements this interface.

⑥ FocusListener
The interface FocusListener is used for receiving keyboard focus events. The class that process focus events needs to implements this interface.

Q.24 Explain event delegation model.
Ans: It is important to consider how users interact with the user interface when designing a graphical user interface (GUI). The GUI may require user to click, retrieve, or drag and drop components in the interface, and input data using the keyboard. These actions will result in an event and you need to write a code to handle these event them.
Event handling code deals with events generated by GUI user interaction. The best practices for coding event handlers are outlined in an event delegation model.

The event delegation model comprises three elements
- Event source
- Event listener
- Adapter

Event source: An event source is a component such as a GUI component that generates an event. The event source can be generated by any type of user interaction. You can also combine a number of different types of events into one event object. For example, if a user clicks and drags an icon, you can use

up the mouse clicked event and mouse moved event into one event object.

Event listener

Event listener are objects that receive notification of an event. Components define the events they fire by registering objects called listeners for those events types. When an event is fired, an event object is passed as an argument to the relevant listener object method. The listener object can handle the event.

Adapter: Adapter are abstract classes that implement listener interfaces using predefined methods. These are provided for convenience.

You can use an adapter to apply one listener's method without having to implement all other methods. Adapter provide empty implementation for all interface methods so you only need to implement override the method you are interested in.

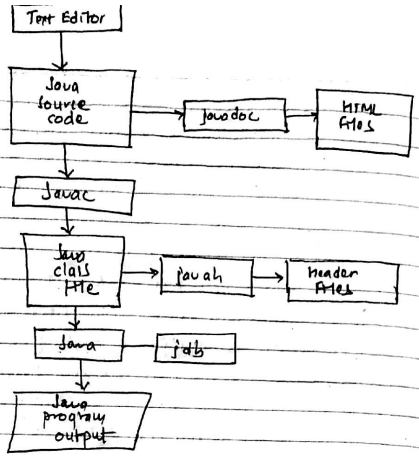
Q.8c Write a program that copies the content of a file into another using byte stream.

Ans:

```
import java.io.*;
public class FileReadWrite {
public static void main (String args[]) throws Exception {
FileInputStream fin = new FileInputStream("d:/abc.txt");
FileOutputStream fout = new FileOutputStream("e:/abc.txt");
```



```
int ch;
while ((ch = fin.read()) != -1)
    {
        fout.write((byte)ch);
    }
fin.close();
fout.close();
System.out.println("operation successful");
}
}
```



JRE (Java Runtime Environment)

The Java Runtime Environment (JRE) facilitates the execution of programs developed in Java. It primarily comprises of the following

→ Java Virtual Machine (JVM): It is a program that interprets the intermediate Java byte code and generates the desired output. It is because of byte code and JVM concepts that programs written in Java are highly portable

→ Runtime class libraries: These are a set of core class libraries that are required for the execution of Java program.

→ User interface toolkits: AWT and Swing are examples of toolkits that support varied input methods for the users to interact with the application program.

→ Deployment technologies → Java plug-in
→ Java web start, etc.

```
import java.lang.*;
class Welcome
{
public static void main
(String [] args)
{
System.out.println("Welcome to
Java");
}
}
```

javac Welcome.java → compiling
java Welcome → interpret and run.

Addition of two numbers using command line arguments:

```
class Add
{
public static void main
(String [] args)
int a = Integer.parseInt(args[0]);
int b = Integer.parseInt(args[1]);
int s = a+b;
System.out.println("sum: "+s);
}
```


if	switch
→ can take any type of variable → can take for equality and other comparisons	→ can take only integer or character variable → checks only for equality.

```
import java.lang.*;
import java.util.*;
class Check
{
public static void main (String
[] args)
{
System.out.println ("Enter a num");
Scanner s = new Scanner (System.in)
int a = s.nextInt();
if (a%2 == 0)
System.out.println("Even
number");
else
System.out.println("Odd number");
}
}
```

Visibility control:

The visibility control modifiers are also known as access modifiers. Java provides three types of visibility control
- public
- private
- protected

1. Use public if the field is to be visible everywhere.
2. Use protected if the field is to be visible everywhere in the current package and also subclasses in other packages.
3. Use default if the field is to be visible everywhere in the current package only.
4. Use private protected if the field is to be visible only in the subclass regardless of package.
5. Use private if the field is to be visible anywhere except in its own class.

JDBC (Java database Connectivity)

→ JDBC is a Java API used for connecting Java with database.

→ It is made of java.sql package. The classes of this package to be used are

Connection - holds the connection to the database.
DriverManager - establishes the connection to the database
Statement - used to execute static sql statements
Prepared Statement → used to execute precompiled sql statements
CallableStatement - used to execute stored procedure
ResultSet - used to hold the data returned by sql "select query".

Steps for JDBC

1. Import required packages
2. Register with JDBC driver
3. Create connection with database
4. Perform required SQL operation
5. close the connection