Basic Concepts

Introducing Nested and Inner Classes

A class defined within another class is termed as Nested class.

The scope of a nested class is bounded by the scope of its enclosing class.

Ex.

class A //Outer class

{

class B //Inner class

{

//B is known to A, but not outside of A

}

}

A nested class has access to the members, including private members, of the class in which it is nested.

However, the enclosing class does not have access to the members of the nested class.

The MouseListener Interface

This interface defines five methods. If the mouse is pressed and released at the same point, mouseClicked( ) is invoked. When the mouse enters a component, the mouseEntered( ) method is called. When it leaves, mouseExited( ) is called. The mousePressed( ) and mouseReleased( ) methods are invoked when the mouse is pressed and released, respectively.

The general forms of these methods are shown here:

void mouseClicked(MouseEvent me)

void mouseEntered(MouseEvent me)

void mouseExited(MouseEvent me)

void mousePressed(MouseEvent me)

void mouseReleased(MouseEvent me)

Dialog Boxes

Dialog boxes are primarily used to obtain user input. They are similar to frame windows, except that dialog boxes are always child windows of a top-level window. Also, dialog boxes don’t have menu bars.

In our application, I have used a dialog box for displaying the top scorers of the players.. It has a “ok” button and clicking on button starts the game.

class ScoreDialog extends JDialog implements ActionListener {

}

To handle the click event on button, this class implements the action as shown above. The code is below:

public void actionPerformed(ActionEvent arg0) {

// This starts the main frame

NumberGameFrame numberGameFrame = new NumberGameFrame();

numberGameFrame.setResizable(false);

numberGameFrame.setSize(400, 550);

numberGameFrame.setVisible(true);

setVisible(false);//to hide dialog box

} }

To create the Graphical interface for the application, one has to define a class that extends the JFrame class. The “NumberGame” provides the frame for application. To make it simple ,I have put all the logic for the Number Game in the MainPanel class as can be seen in the below code.

public class NumberGameFrame extends JFrame implements ActionListener {

JButton brst, bpopulate, bext; //declares button objects

JLabel lpair, lpCnt, lscore, lScore; //declares Label object for score board

String initials = "";// To store Initial

int pairCounter = 0, pScore = 0;

JButton[][] numCell; //2 – D array for number panel

public NumberGameFrame() {

super("Number Game");

brst = new JButton("Restart");//creating and adding buttons to frame

brst.addActionListener(this);//Registration for action event

bpopulate = new JButton("Repopulate");

bpopulate.addActionListener(this);

bext = new JButton("Exit");

bext.addActionListener(this);

………………

}

The GameScorePanel class provides the score panel interface of the game for displaying the score of pair counts and score of the player.

The GameNumberPanel is used to create the Number Panels, I have used a 2 dimensional array of JButtons . These array is placed or added on the form using gridlayout with 12 rows and 9 columns. To display the number, I have setText() method and this number is generated using a random function taken from Internet. Also used setBackground() to set background red or orange.

How colors of number panel changes when mouse is move over it ?

To implement this, I have the MouseAdapter class to handle mouse events. To associate with each button, I defined as class GameNumberMouseHandler that extends the MouseAdapter class.

Java provides a special feature, called an adapter class, that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface. Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface. You can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.

The button when created is associated with the GameNumberMouseAdapter class as shown in below:

for (int row = 0; row < numCell.length; row++) {

for (int col = 0; col < numCell[row].length; col++) {

Random rn = new Random();

int rnd = 1 + rn.nextInt(9 - 1 + 1);

numCell[row][col] = new JButton();

numCell[row][col].setFont(new Font("Times New Roman",

Font.BOLD, 14));

numCell[row][col].setText(Integer.toString(rnd));

numCell[row][col].setBackground(Color.RED);

numCell[row][col]

.addMouseListener(new GameNumberMouseHandler(row,

col));

add(numCell[row][col]);

However if directly use the mouseAdapter class , I was not to get the result of mouse movements on individual buttons.

To handle it, I have defined a class that extends the MouseAdapter class as shown below:

class GameNumberMouseHandler extends MouseAdapter {

private int row, col;

public GameNumberMouseHandler(int row, int col) {

this.row = row;

this.col = col;

}

public void mouseEntered(MouseEvent me) {

prevColor = numCell[row][col].getBackground();

numCell[row][col].setBackground(Color.WHITE);

}

public void mouseExited(MouseEvent me) {

if (numCell[row][col].getBackground() == Color.GREEN)

return;

if (numCell[row][col].getBackground() != Color.ORANGE)

numCell[row][col].setBackground(prevColor);

}

public void mouseClicked(MouseEvent me) {

if (numCell[row][col].getBackground() == Color.ORANGE)

return;

//This occurs when the user clicks button for match or forming pairs

//Here on first click , I change the background color and the x and y position of the button.

//on second click , verify the match, if found updates the score board

}

I have defined this class as an inner class. As it has no other use, it will be used internally by the NumberGameFrame class only.

Similarly To handle the button click events for exit , restart and repopulate actions.

class ActionButtonPanel extends JPanel implements ActionListener {

@Override

public void actionPerformed(ActionEvent ae) {

Object source = ae.getSource();//To know which button clicked

If(source == exit) {

}

Else if(source == restart) {

}

….

}

}

File Handling Basics:

Streams:

Java programs perform I/O through streams.

A stream is an abstraction that either produces or consumes information.

A stream is linked to a physical device by the Java I/O system.

It behaves in the same manner irrespective of physical devices.

java defines:

i) Byte Stream:

For handling i/p & o/p of bytes.

Reading & writing binary data.

1) InputStream: To read sequence of bytes.

2) OutputStream: To write sequence of bytes.

ii) Character Stream:

For handling of i/p & o/p of characters.

It uses UNICODE and hence can be internatiolized.

1) Reader: To read characters.

2) Writer: To write characters.

File class: It describes the attributes of the file or directory.

Constructors:

File(String pathname)

Methods:

1) boolean canRead()

2) boolean canWrite()

3) boolean createNewFile()

4) boolean delete()

5) boolean exists()

6) String getAbsolutePath()

7) String getName()

8) boolean isFile()

9) boolean isdirectory()

10) String getPath()

11) long length()

outputStream class :

An abstract class that defines streaming byte output. All of the methods in this class return a void value and throw IOException

Methods:

1) void write(byte b)

2) void write(byte [] b)

3) void write(byte []b,int offset,int length)

4) void close()

5) void flush()

Note: Methods not exciting. Only can write bytes.

Sub classes:

1) DataOutputStream class: It provides methods for writing ints,shorts or double etc.

Methods:

i) void writeDouble(double d)

ii) void writeFloat(float f)

iii) void writeUTF(String s)

2) FileOutputStream class: Writes bytes to a file.

Constructor:

1) FileOutputStream(String path)

2) FileOutputStream(String path,boolean append)

And similarly the corresponding inputstream class for reading from the file. The FleOutputStream is used to connect to the file on disk “HighScores.hs” . the DataOutputStream class used to write the string and other datatypes on to the file.

The corresponding class FileInputStream and DataInputStream used to read from the file. For this purpose .i have the FileHandler class to read and write the initial,pair counts,score and date from or to the file as shown in the below class.

class DataReaderWriter implements Comparable {

public String initials; //To store information of player in file

public String pdate;

public int pairs;

public int score;

@Override

public int compareTo(Object a) {

int comparescr = ((DataReaderWriter) a).score;

return comparescr - this.score;

}

public void writeDataToFile(String init, String dt, int pair, int score) {

// The name of the file to store Scores.

String fileName = "HighScores.hs";

try {

FileOutputStream fs = new FileOutputStream(fileName, true);

DataOutputStream ds = new DataOutputStream(fs);

ds.writeUTF(init);

ds.writeUTF(dt);

ds.writeInt(pair);

ds.writeInt(score);

// Always close files.

fs.close();

ds.close();

} catch (IOException ex) {

System.out.println("Error!! In writting File '" + fileName + "'");

}

}

public ArrayList<DataReaderWriter> getDataFromFile() {

ArrayList<DataReaderWriter> scores = new ArrayList<DataReaderWriter>();

String fileName = "HighScores.hs";

FileInputStream is = null;

DataInputStream dis = null;

try {

is = new FileInputStream(fileName);

dis = new DataInputStream(is);

while (true) {

DataReaderWriter data = new DataReaderWriter();

data.pdate = dis.readUTF();

data.initials = dis.readUTF();

data.pairs = dis.readInt();

data.score = dis.readInt();

scores.add(data);

}

} catch (EOFException ex) {

try {

dis.close();

is.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

} catch (FileNotFoundException ex) {

System.out.println("Unable to open file '" + fileName + "'");

} catch (IOException ex) {

System.out.println("Error reading file '" + fileName + "'");

// Or we could just do this:

// ex.printStackTrace();

}

Collections.sort(scores);

return scores;

}

}

The class To handle the score is defined as follows

public class Scores implements Comparable{ //It implements comparable to sort in terms of

// scores

//These Fields for score related information

private String initials;

private int pairCount;

private int scoreCount;

private String playDate;

@Override

public int compareTo(Object other) {//this overridden method is used for sorting

int comparescr = ((Scores) other).getScoreCount();

return comparescr - this.getScoreCount();

}

…

}

The ArrayList is used to store the list of scores of the players. The ArrayList is a collection in java used to store list of objects.