14-07-2022

**JAVA NOTES**

1. Precedence operator :

| Operators | Precedence |
|---|---|
| postfix increment and decrement | ++-- |
| prefix increment and decrement, and unary | ++--+-~! |
| multiplicative | */% |
| additive | +- |
| shift | <<>>>>> |
| relational | <><=>=instanceof |
| equality | ==!= |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | \| |
| logical AND | && |
| logical OR | \|\| |
| ternary | ? : |
| assignment | =+=-=*=/=%= &=^=\|=<<=>>=>>>= |

**Example: Operator Precedence**

```
class Precedence

{

public static void main(String[] args)

{

int a = 10, b = 5, c = 1, result;

result = a-++c-++b;

System.out.println(result);

}

}
```

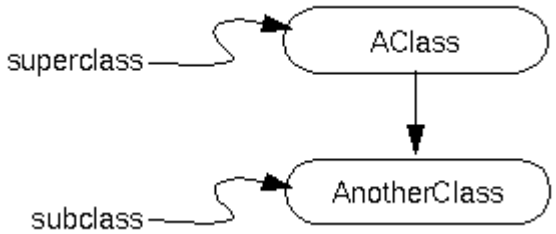
**Output**:

2

The table below shows the associativity of Java operators along with their associativity.

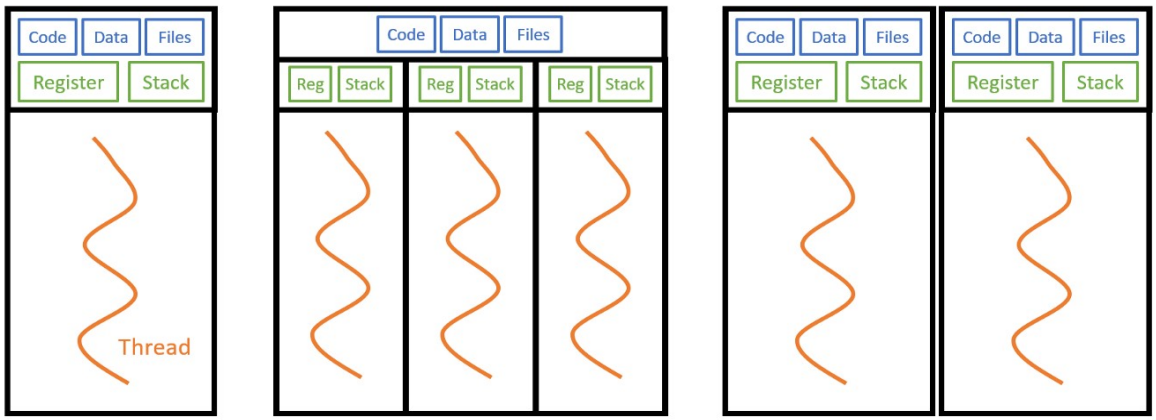
| Operators | Precedence | Associativity |
|---|---|---|
| postfix increment and decrement | ++-- | left to right |
| prefix increment and decrement, and unary | ++--+-~! | right to left |
| multiplicative | */% | left to right |
| additive | +- | left to right |
| shift | <<>>>>> | left to right |
| relational | <><=>=instanceof | left to right |
| equality | ==!= | left to right |
| bitwise AND | & | left to right |
| bitwise exclusive OR | ^ | left to right |
| bitwise inclusive OR | \| | left to right |
| logical AND | && | left to right |
| logical OR | \|\| | left to right |
| ternary | ? : | right to left |
| assignment | =+=-=*=/=%= &=^=\|=<<=>>=>>>= | right to left |

2. Sub class :




3. Difference between Multi Process and Multi Thread :

By formal definition, ***multithreading*** refers to the ability of a processor to execute multiple threads concurrently, where each thread runs a process. Whereas ***multiprocessing*** refers to the ability of a system to run multiple processors concurrently, where each processor can run one or more threads.




Single Processor Single Thread  Single Processor Multithread  Multiprocessing

4. Types of Mouse handling events in Java :

java.lang.Object

└ java.util.EventObject

└ java.awt.AWTEvent

└ java.awt.event.ComponentEvent

└ java.awt.event.InputEvent

└ **java.awt.event.MouseEvent**

public class MouseEvent

extends InputEvent

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. Component bounds can be obscurred by the visible component's children or by a menu or by a top-level window. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).

This low-level event is generated by a component object for:

- Mouse Events
  - a mouse button is pressed
  - a mouse button is released
  - a mouse button is clicked (pressed and released)
  - the mouse cursor enters the unobscured part of component's geometry
  - the mouse cursor exits the unobscured part of component's geometry
- Mouse Motion Events
  - the mouse is moved
  - the mouse is dragged

A MouseEvent object is passed to every MouseListener or MouseAdapter object which is registered to receive the "interesting" mouse events using the component's addMouseListener method. (MouseAdapter objects implement the MouseListener

interface.) Each such listener object gets a MouseEvent containing the mouse event.

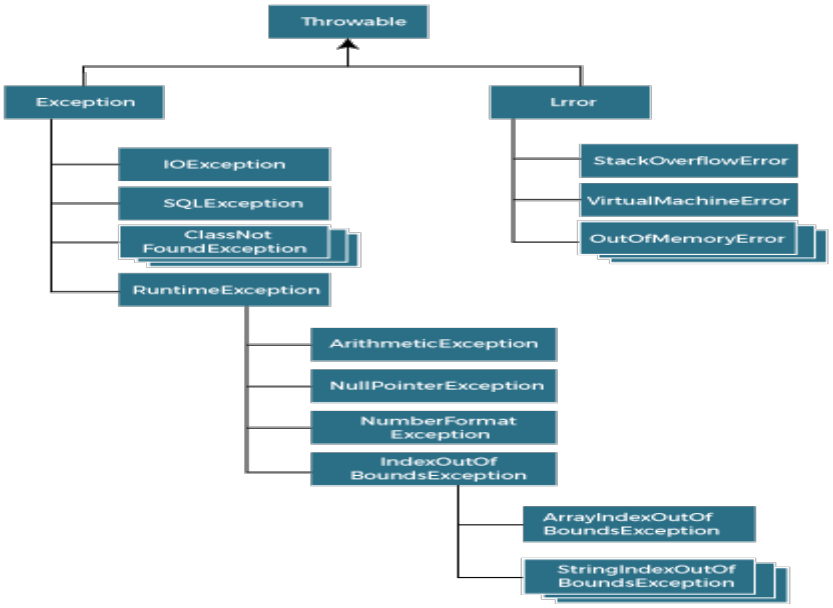
5. Garbage Collector :

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

```
// Garbage collection in Java

public class TestGarbage1
{
public void finalize ( )
{
System.out.println("Garbage collection completed successfully");
}// End of finalize ( ) method
public static void main(String args[])
{
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
s2=null;
System.gc();
} // End of main method
}// End of class
```

6. Exception Hierarchy in Java :




7. Thread Priorities in Java :

```
public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY
```

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

8. Random Access File in Java :

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructor

| Constructor | Description |
|---|---|
| RandomAccessFile(File file, String mode) | Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument. |
| RandomAccessFile(String name, String mode) | Creates a random access file stream to read from, and optionally to write to, a file with the specified name. |

Method

| Modifier and Type | Method | Method |
|---|---|---|
| void | close() | It closes this random access file stream and releases any system resources associated with the stream. |
| FileChannel | getChannel() | It returns the unique FileChannel object associated with this file. |
| int | readInt() | It reads a signed 32-bit integer from this file. |
| String | readUTF() | It reads in a string from this file. |
| void | seek(long pos) | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |
| void | writeDouble(double v) | It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, |

| | | |
|---|---|---|
| | | high byte first. |
| void | writeFloat(float v) | It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first. |
| void | write(int b) | It writes the specified byte to this file. |
| int | read() | It reads a byte of data from this file. |
| long | length() | It returns the length of this file. |
| void | seek(long pos) | It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs. |

---

Example

```
import java.io.IOException;
import java.io.RandomAccessFile;

        public class RandomAccessFileExample{
        static final String FILEPATH="myFile.TXT";
        public static void main(String[] args) {
        try{
                System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
                writeToFile(FILEPATH,"I love my country and my people",31);
        }catch(IOException e){ e.printStackTrace(); }
}
private static byte[] readFromFile(String filePath, int position, int size)
throws IOException{
RandomAccessFile file = new RandomAccessFile(filePath, "r");
file.seek(position);
byte[] bytes=new byte[size];
file.read(bytes);
file.close();
return bytes;
}
```

```
private static void writeToFile(String filePath, String data, int position)
throws IOException {
RandomAccessFile file = new RandomAccessFile(filePath, "rw");
file.seek(position);
file.write(data.getBytes());
file.close();
}
}
```
The myFile.TXT contains text "I love my country and my people"

9. Calender in Java :

Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.
As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

- Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
- Calendar.getInstance(TimeZone zone)
- Calendar.getInstance(Locale aLocale)
- Calendar.getInstance(TimeZone zone, Locale aLocale)

Java program to demonstrate getInstance() method:

```
// Date getTime(): It is used to return a
// Date object representing this
// Calendar's time value.

import java.util.*;
public class Calendar1 {
    public static void main(String args[])
    {
        Calendar c = Calendar.getInstance();
        System.out.println("The Current Date is:" +
c.getTime());
    }
}
```

10. Jbutton, Jtextfield, Jlabel and Listeners in Java

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

**Public class** Jbutton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

| Constructor | Description |
|---|---|
| JButton() | It creates a button with no text and icon. |
| JButton(String s) | It creates a button with the specified text. |
| JButton(Icon i) | It creates a button with the specified icon object. |

---

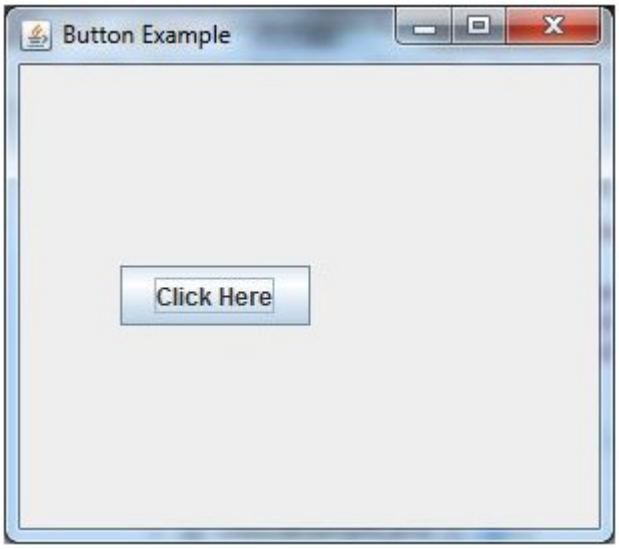
Commonly used Methods of AbstractButton class:

| Methods | Description |
|---|---|
| void setText(String s) | It is used to set specified text on button |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the specified Icon on the button. |
| Icon getIcon() | It is used to get the Icon of the button. |
| void setMnemonic(int a) | It is used to set the mnemonic on the button. |
| void addActionListener(ActionListener a) | It is used to add the action listener to this object. |

```
Import javax.swing.*;
public class ButtonExample {
```

```
public static void main(String[] args){
Jframe f=new Jframe("Button Example");
Jbutton b=newJButton("Click Here");
b.setBounds(50,100,95,30);
f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:




**Jtextfield :**

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

**public class** JtextField **extends** JtextComponent **implements** SwingConstants

Commonly used Constructors:

| Constructor | Description |
|---|---|
| JTextField() | Creates a new TextField |

| | |
|---|---|
| JTextField(String text) | Creates a new TextField initialized with the specified text. |
| JTextField(String text, int columns) | Creates a new TextField initialized with the specified text and columns. |
| JTextField(int columns) | Creates a new empty TextField with the specified number of columns. |

---

Commonly used Methods:

| Methods | Description |
|---|---|
| void addActionListener(ActionListener l) | It is used to add the specified action listener to receive action events from this textfield. |
| Action getAction() | It returns the currently set Action for this ActionEvent source, or null if no Action is set. |
| void setFont(Font f) | It is used to set the current font. |
| void removeActionListener(ActionListener l) | It is used to remove the specified action listener so that it no longer receives action events from this textfield. |

```
Import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
{
Jframe f= new Jframe("TextField Example");
JtextField t1,t2;
t1=new JtextField("Welcome to ADAMASUNIV.");
t1.setBounds(50,100,200,30);
t2=new JtextField("AWT Tutorial");
t2.setBounds(50,150,200,30);
f.add(t1); f.add(t2);
```

```
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

**Jlabel:**

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

**public class** Jlabel **extends** Jcomponent **implements** SwingConstants, Accessible

Commonly used Constructors:

| Constructor | Description |
|---|---|
| JLabel() | Creates a JLabel instance with no image and with an empty string for the title. |
| JLabel(String s) | Creates a JLabel instance with the specified text. |
| JLabel(Icon i) | Creates a JLabel instance with the specified image. |
| JLabel(String s, Icon i, int horizontalAlignment) | Creates a JLabel instance with the specified text, image, and horizontal alignment. |

---

Commonly used Methods:

| Methods | Description |
|---|---|
| String getText() | t returns the text string that a label displays. |
| void setText(String text) | It defines the single line of text this component will display. |
| void setHorizontalAlignment(int alignment) | It sets the alignment of the label's contents along the X axis. |

| | |
|---|---|
| Icon getIcon() | It returns the graphic image that the label displays. |
| int getHorizontalAlignment() | It returns the alignment of the label's contents along the X axis. |

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
{
Jframe f= new Jframe("Label Example");
JLabel l1,l2;
l1=new Jlabel("First Label.");
l1.setBounds(50,50, 100,30);
l2=new Jlabel("Second Label.");
l2.setBounds(50,100, 100,30);
f.add(l1); f.add(l2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:

**Listeners:**

The Event listener represent the interfaces responsible to handle events. Java provides us various Event listener classes but we will discuss those which are more frequently used. Every method of an event listener method has a single argument as an object which is subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListner interface

It is a marker interface which every listener interface has to extend.This class is defined in java.util package.

Class declaration

Following is the declaration for **java.util.EventListener** interface:

```
public interface EventListener
```

AWT Event Listener Interfaces:

Following is the list of commonly used event listeners.

| Sr. No. | Control & Description |
|---|---|
| 1 | ActionListener<br><br>This interface is used for receiving the action events. |
| 2 | ComponentListener<br><br>This interface is used for receiving the component events. |
| 3 | ItemListener<br><br>This interface is used for receiving the item events. |
| 4 | KeyListener<br><br>This interface is used for receiving the key events. |
| 5 | MouseListener |

| | |
|---|---|
| | This interface is used for receiving the mouse events. |
| 6 | TextListener<br><br>This interface is used for receiving the text events. |
| 7 | WindowListener<br><br>This interface is used for receiving the window events. |
| 8 | AdjustmentListener<br><br>This interface is used for receiving the adjusmtent events. |
| 9 | ContainerListener<br><br>This interface is used for receiving the container events. |
| 10 | MouseMotionListener<br><br>This interface is used for receiving the mouse motion events. |
| 11 | FocusListener<br><br>This interface is used for receiving the focus events. |

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exists as convenience for creating listener objects.

AWT Adapters:

Following is the list of commonly used adapters while listening GUI events in AWT.

| **Sr. No.** | **Adapter & Description** |
|---|---|

| | |
|---|---|
| 1 | FocusAdapter<br><br>An abstract adapter class for receiving focus events. |
| 2 | KeyAdapter<br><br>An abstract adapter class for receiving key events. |
| 3 | MouseAdapter<br><br>An abstract adapter class for receiving mouse events. |
| 4 | MouseMotionAdapter<br><br>An abstract adapter class for receiving mouse motion events. |
| 5 | WindowAdapter<br><br>An abstract adapter class for receiving window events. |

11. Security issues of Applet

One of the most important features of Java is its security model. It allows untrusted code, such as applets downloaded from arbitrary web sites, to be run in a restricted environment that prevents that code from doing anything malicious, like deleting files or sending fake email. The Java security model has evolved considerably between Java 1.0 and Java 1.2 and is covered in detail in Java in a Nutshell. To write applets, you don't need to understand the entire Java security model. What you do need to know is that when your applet is run as untrusted code, it is subject to quite a few security restrictions that limit the kinds of things it can do. This section describes those security restrictions and also describes how you can attach a digital signature to applets, so that users can treat them as trusted code and run them in a less restrictive environment. The following list details the security restrictions that are typically imposed on untrusted applet code. Different web browsers and applet viewers may impose slightly different security restrictions and may allow the end user to customize or selectively relax the restrictions. In general, however, you should assume that your untrusted applet are restricted in the following ways: Untrusted code cannot read from or write to the local filesystem. This means that untrusted code cannot:

a. Read files List directories Check for the existence of files
b. Obtain the size or modification date of files
c. Obtain the read and write permissions of a file
d. Test whether a filename is a file or directory
e. Write files
f. Delete files
g. Create directories
h. Rename files
i. Read or write from FileDescriptor objects

12. Difference between Java Applet and Java Application

Java Application is just like a Java program that runs on an underlying **operating system** with the support of a **virtual machine**. It is also known as an application program. The **graphical user interface** is not necessary to execute the java applications, it can be run with or without it.
**Java Applet** is an applet is a Java program that can be embedded into a **web page**. It runs inside the **web browser** and works on the client-side. An applet is embedded in an **HTML page** using the APPLET or OBJECT tag and hosted on a web server. Applets are used to make the website more dynamic and entertaining.
The difference between Application and Applet:

| **Parameters** | **Java Application** | **Java Applet** |
|---|---|---|
| Definition | Applications are just like a Java program that can be executed independently without using the web browser. | Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution. |
| main () method | The application program requires a main() method for its execution. | The applet does not require the main() method for its execution instead init() method is required. |
| Compilation | The “javac” command is used to compile application programs, which are then executed using the “java” command. | Applet programs are compiled with the “javac” command and run using either the “appletviewer” command or the web browser. |
| File access | Java application programs have full access to the local file system and network. | Applets don’t have local disk and network access. |
| Access level | Applications can access all kinds of resources available on the system. | Applets can only access browser-specific services. They don’t have access to the local system. |
| Installation | First and foremost, the installation of a Java application on the local computer is required. | The Java applet does not need to be installed beforehand. |
| Execution | Applications can execute the programs from the local system. | Applets cannot execute programs from the local machine. |
| Program | An application program is needed to perform some tasks directly for the user. | An applet program is needed to perform small tasks or part of them. |
| Run | It cannot run on its own; it needs | It cannot start on its own, but it can be |

| Parameters | Java Application | Java Applet |
|---|---|---|
| | JRE to execute. | executed using a Java-enabled web browser. |
| Connection with servers | Connectivity with other servers is possible. | It is unable to connect to other servers. |
| Read and Write Operation | It supports the reading and writing of files on the local computer. | It does not support the reading and writing of files on the local computer. |
| Security | Application can access the system's data and resources without any security limitations. | Executed in a more restricted environment with tighter security. They can only use services that are exclusive to their browser. |
| Restrictions | Java applications are self-contained and require no additional security because they are trusted. | Applet programs cannot run on their own, necessitating the maximum level of security. |

13. File handling in Java

In Java, with the help of File Class, we can work with files. This File Class is inside the java.io package. The File class can be used by creating an object of the class and then specifying the name of the file.

**Why File Handling is Required?**

- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file.

```
// Importing File Class
import java.io.File;

class GFG {
   public static void main(String[] args)
   {

      // File name specified
      File obj = new File("myfile.txt");
        System.out.println("File Created!");
   }
}
```

Output:

```
File Created!
```

Read from a File: We will use the Scanner class in order to read contents from a file. Following is a demonstration of how to read contents from a file in Java :

```
// Import the File class
import java.io.File;

// Import this class for handling errors
import java.io.FileNotFoundException;

// Import the Scanner class to read content from text files
import java.util.Scanner;

public class GFG {
   public static void main(String[] args)
   {
      try {
         File Obj = new File("myfile.txt");
         Scanner Reader = new Scanner(Obj);
         while (Reader.hasNextLine()) {
            String data = Reader.nextLine();
            System.out.println(data);
         }
         Reader.close();
      }
      catch (FileNotFoundException e) {
         System.out.println("An error has occurred.");
         e.printStackTrace();
      }
   }
}
```

Output:

An error has occurred.

14. Features of JDBC / ODBC Drivers:

The difference between JDBC and ODBC are mentioned below:

| ODBC | JDBC |
|---|---|
| ODBC Stands for Open Database Connectivity. | JDBC Stands for java database connectivity. |
| Introduced by Microsoft in 1992. | Introduced by SUN Micro Systems in 1997. |
| We can use ODBC for any language like C,C++,Java etc. | We can use JDBC only for Java languages. |
| We can choose ODBC only windows platform. | We can Use JDBC in any platform. |
| Mostly ODBC Driver developed in native languages like C,C++. | JDBC Stands for java database connectivity. |
| For Java applications it is not recommended to use ODBC because performance will be down due to | For Java application it is highly recommended to use JDBC because there |

internal conversion and applications will becomeare no performance & platform dependent platform Dependent. problem.
ODBC is procedural. JDBC is object oriented.

15. Event in Java:

Java Event classes and Listener interfaces

| Event Classes | Listener Interfaces |
|---|---|
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

Steps to perform Event Handling
Following steps are required to perform event handling:

1.Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

**a. Button**

- public void addActionListener(ActionListener a){}

**b. MenuItem**

- public void addActionListener(ActionListener a){}

**c. TextField**

- public void addActionListener(ActionListener a){}
- public void addTextListener(TextListener a){}

**d. TextArea**

- public void addTextListener(TextListener a){}

**e. Checkbox**

- public void addItemListener(ItemListener a){}

**f. Choice**

- public void addItemListener(ItemListener a){}

**g. List**

- public void addActionListener(ActionListener a){}
- public void addItemListener(ItemListener a){}

---

Java Event Handling Code

We can put the event handling code into one of the following places:

1.Within class

2.Other class

3.Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class Aevent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
```

```
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

16. Layout Management in GUI programming :

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1.java.awt.BorderLayout

2.java.awt.FlowLayout

3.java.awt.GridLayout

4.java.awt.CardLayout

5.java.awt.GridBagLayout

6.javax.swing.BoxLayout

7.javax.swing.GroupLayout

8.javax.swing.ScrollPaneLayout

9.javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

**1.public static final int NORTH**

**2.public static final int SOUTH**

**3.public static final int EAST**

**4.public static final int WEST**

**5.public static final int CENTER**

Constructors of BorderLayout class:

**a. BorderLayout():** creates a border layout but with no gaps between the components.

**b. BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.