Project On : CALCULATOR

Developed By

Name: NISHA RANI

Roll No: 213175

PUNJABI UNIVERSITY PATIALA

CERTIFICATE

This is to certify that this Report titled ADDRESS BOOK embodies original work done by NISHA RANI in Partial Fulfillment of their course required at BCA

System Analysis & Configuration

System Summary:	**Processor: Pentium 4 (Intel)**
	OS : Windows XP
	RAM : 256 MB
	HD : 40 GB

Project Contents

JAVA™

1. INTRODUCTION TO JAVA

Java is an object oriented programming language that was designed to meet the need for a platform independent language. Java is used to create applications that can run on a single computer as well as a distributed network. Java is both a language and a technology used to develop stand –alone and internet-based applications.

With the increasing use of internet, Java has become a widely used programming language. Java software works everywhere, from the smallest devices, such as microwave ovens and remote controls to supercomputers. Java programs are independent of the type of computer, telephone, television, or the operating system these devices run on. The java programs work on any type of compatible device that supports java. The first version that was released in the market was 1.02. Java seduces programmers with its friendly syntax, object-oriented features, memory management, and best of all- the promise of portability.

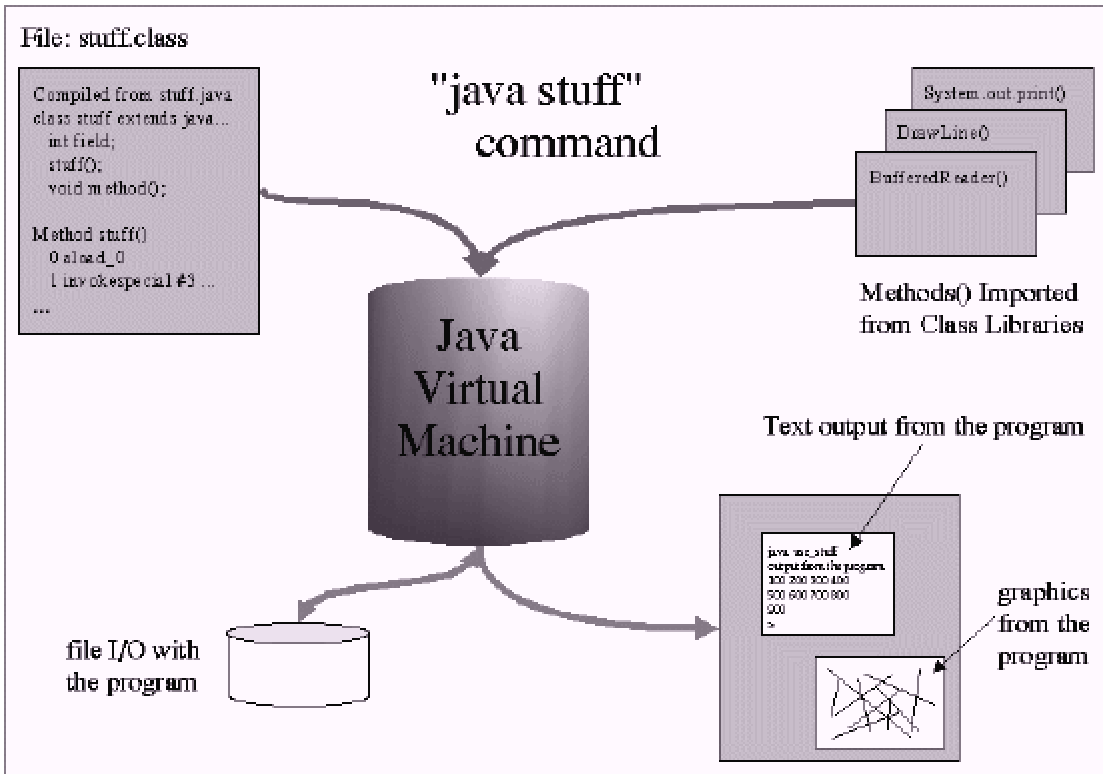
Java 1.02 was slow and had 250 classes in it. The big thing in this version of the java was Applets. Then a newer version of java that was java 1.1 was introduced. It was faster, more capable, friendlier and had better GUI code than its earlier version. It had 500 classes in it. Then java2(versions 1.2-1.4) was introduced which is much faster then its earlier versions. It has 2300 classes in it. It comes in three flavors: Micro Edition (J2ME), Standard Edition (J2SE) and Enterprise Edition (J2EE). It becomes the **language of choice** for new enterprise (especially web-based) and mobile applications. Then java 5.0(versions 1.5 and up) was introduced. It has 3500 classes in it. It is **more powerful, easier to develop with** then its previous editions. Java 5.0 is also known as “Tiger” (its marketing name).

2. NEED FOR JAVA

The java language contains built-in-support for the World Wide Web (WWW), which is a service of the internet to retrieve information in the form of Web pages. The primary motive behind developing java language was the need for a portable and platform-independent language that could be used to produce code that would run on a variety of control processing unit (CPU) under different environments. We can use java to develop network-oriented programs because networking features are built in features in java. **Java** is **simple**, **object-oriented**, **compiled and interpreted**, **portable**, **distributed** and a **secure** language.

JAVA VIRTUAL MACHINE (JVM)

The JVM forms the base for the java platform and is convenient to use on various hardware-based platforms.


File: stuff.class
"java stuff" command
System.out.print()
DrawLine()
BufferedReader()
Methods() Imported from Class Libraries
Java Virtual Machine
Text output from the program
graphics from the program
file I/O with the program

```
Compiled from stuff.java
class stuff extends java...
   int field;
   stuff();
   void method();

Method stuff()
   0 aload_0
   1 invokespecial #3 ...
...
```

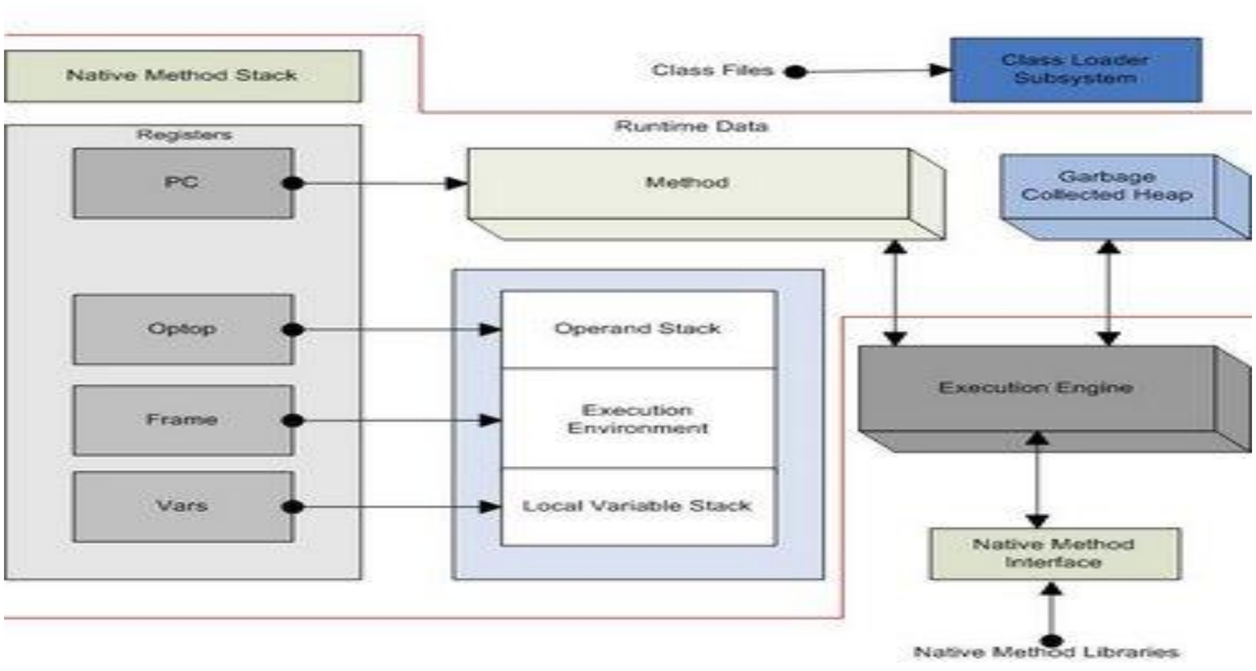
```
java xxx_stuff
output from the program
100 200 300 400
500 600 700 800
900
>
```

Components of the JVM

JVM for different platforms uses different techniques to execute the Bytecode. The major components of JVM are:

1. Class loader
2. Execution engine
3. Just In Time (JIT) compiler.

JVM Internal Architecture



1. Class Loader: *The class loader loads the class files, which are required by a program running in the memory*. The classes are loaded dynamically when required by the running program. A JVM can have following types of class loaders:

1.1 Primordial Class Loader: Loads the Java API classes required by the running Java program.

1.2 Class Loader Objects: Loads the classes of the Java application program. An application can create class loaders at runtime for loading the classes of the application.

2. Execution Engine: *The Java execution engine is the component of the JVM that runs the Bytecode one line after another.* The execution engines implemented by different vendors use different techniques to run the Bytecode. The Java execution engine converts the Bytecode to the machine object code and runs it.
3. JIT Compiler: *The JIT compiler is used for compiling the Bytecode into executable code.* The JVM runs the JIT compiled code without interpreting because the JIT-compiled code is in the machine code format. Running the JIT-compiled code is faster than running the interpreted code because it is compiled and does not require being run, line after line.

<u>3. BASIC CONCEPTS OF OOPS</u>

1. ENCAPSULATION: "Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics." It implies that the non-essential details of an object are hidden from the user and an access is provided to its essential details. Therefore, it is also known as information hiding.

In object-oriented methodology, need of encapsulation arises because the emphasis is on designing classes in such a manner that the classes share data and methods among themselves. Encapsulation is the feature that provides security to the data and the methods of a class.

2. ABSTRACTION: "An abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer."

Abstraction refers to the attributes of an object that clearly democrats it from other objects. The concept of abstraction is implemented in object-oriented programming by creating classes. All the attributes of the objects of the classes are defined in the class. However we cannot store any data in a class because creating a class does not allocate any memory space to the class. To store data, we need to create objects of the class, which have memory allocated as soon as it is created. Classes form the templates for creating objects. In addition to this, abstraction enables to provide a restricted access to data. In object-oriented programming, abstraction means ignoring the non-essential details of an object and concentrating on its essential features.

3. INHERITANCE: In object-oriented methodology, inheritance enables to extend the functionality of an existing class. We can create a class that inherits the attributes and behavior of another class. In addition, the new class can consist of a few new attributes and behaviors that are specific to the class. In terms of classes and objects, attributes refer to the data and behavior refer to the methods. There are various types of inheritance like Single inheritance, Multiple inheritance, Multilevel inheritance, Hybrid inheritance, Multipath inheritance and Hierarchical inheritance.

4. POLYMORPHISM: *Polymorphism* is derived from two latin words- poly, which means many, and morph, which means forms. Any thing that exists in more than one form is known as a polymorph.

In object-oriented methodology, polymorphism is the feature that enables us to assign a different meaning or usage to an entity in different contexts. The entity can be a variable, method ,or an object. Polymorphism enables an entity to have more than one form depending upon the context in which it is used. Polymorphism enables one entity to be used as a general category for different types of actions.

EXCEPTION HANDLING IN JAVA

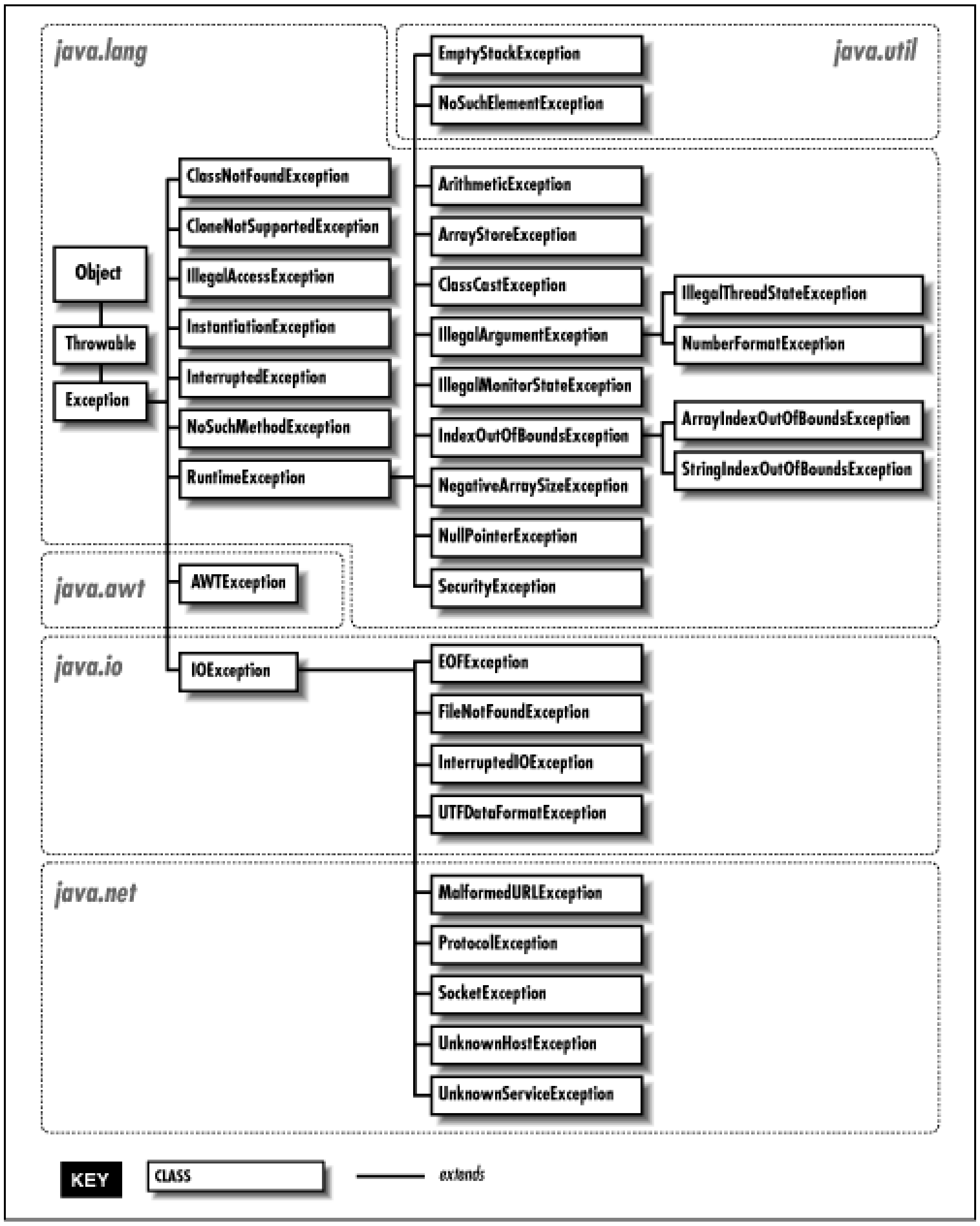
The term exception in java indicates an exceptional event. It can be defined as an abnormal event that occurs during program execution and disrupts the normal flow of instructions. The abnormal event can also be an error in the program.

Errors in a java program are categorized into two groups: compile-time errors and run-time errors. Compile- time errors occur when we do not follow the syntax of a programming language. The compiler detects the syntax error of the program while compiling the program.

Run time errors occur during the execution of a program. These can occur due to Running out of memory, Resource allocation errors, Inability to find files, or Problem in network connectivity.

EXCEPTION CLASSES: To handle exceptions, the java run-time system searches for an exception-handler. In java, a **catch statement** is an exception handler that is used to handle an exception. The search for an exception handler begins with the method in which the exception is raised. If no appropriate exception-handler is found, the java run-time system searches the exception-handler in the next higher method hierarchy. The type of exception handled by the exception-handler should match the type of exception thrown. The java run-time system proceeds with the normal execution of the program after an exception gets handled. If no appropriate exception-handler is found by the java run-time system, the program is terminated.

In java, Throwable class is the superclass of all the exception classes. Object class is the base class of the exception hierarchy. The Exception class and the Error class are two subclasses of the Throwable class. Exception handling can be implemented by using try, catch, throw, throws and finally keywords available in java.



AWT (ABSTRACT WINDOW TOOLKIT)

Java provides the AWT control components, which contains classes that enable us to create standard components, such as buttons, labels, and text fields in java. A java component enables us to accept input from an end user. We can position AWT components in containers using the different layout managers.

An AWT control is a component that enables end users to interact with applications created in java. All AWT controls in java are subclasses of the Component class. the Component class provides the add() method to add AWT components, such as an applet or a window, to containers. The various classes that we can use for creating AWT controls are:

1. **TextField Class:** Text Fields are user interface components that accept text input from end user. A text field enables us to type text in a **single line.**

2. **TextArea Class:** Text areas are also used to accept text input from an end user, but it enables us to type text in **multiple lines**.

3. **Button Class:** Buttons are AWT controls that are used for handling events. Java provides the Button class to create AWT button components.

4. **List Class:** List control is a scrollable list of text items that enables us to select either one item or multiple items. We can create a list using the List class in java.

5. **CheckBox Class:** Check boxes are the components that exist in dual state, checked and unchecked. We can change the state of a check box by clicking the check box. We can create a check box by using CheckBox class.

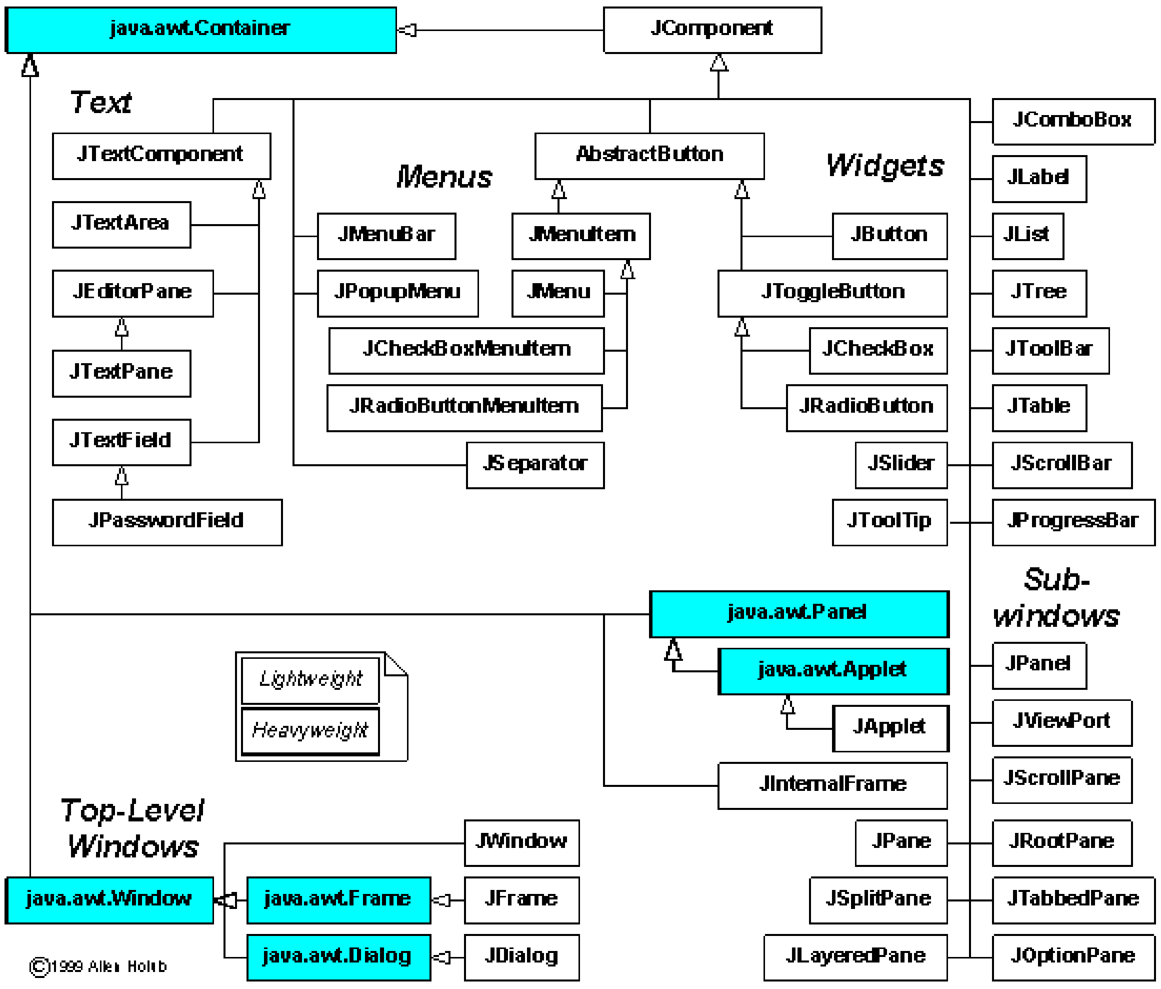
6. **Label Class:** Labels are used for displaying a single line of text in a container. We can create a label by using Label class.

THE SWING COMPONENTS

Swing components also provide GUI environment to java in addition to applets. Swing components are a collection of lightweight visual components that provide a replacement for heavyweight AWT components. The major difference between the lightweight visual components and heavyweight visual components is that lightweight components have transparent pixels while the latter are opaque.

Swing components contain the Pluggable Look And Feel (PL&F) feature that allows applications to have the same behavior on various platforms.

SWING COMPONENT CLASS HIERARCHY



The JComponent class is the root of the Swing hierarchy, which is an extension of the AWT container class. the class hierarchy of the Swing components is categorized into:

1. **Top-level Swing Containers:** Acts as a container for placing the intermediate-level and atomic swing components, such as panels,

frames, buttons, and check boxes. The various top-level swing containers are:

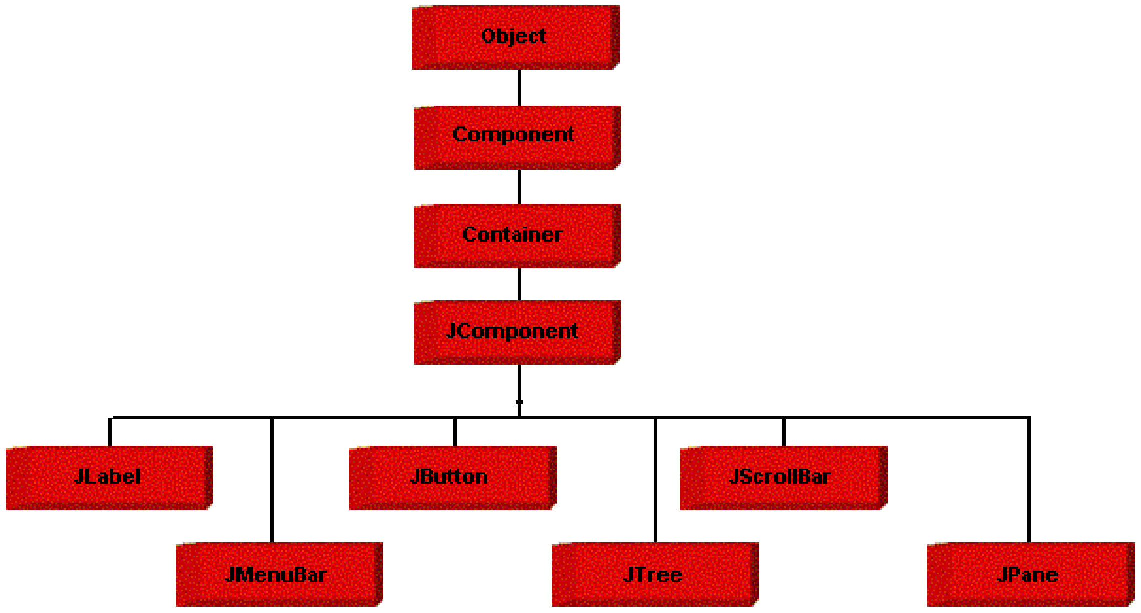
A. JFRAME
B. JAPPLET
C. JDIALOG

2. **Intermediate-level Swing Containers:** Placed on the top-level containers and contains atomic components, such as buttons, check boxes, and labels. The various intermediate-level swing components are:

A. JPanel
B. **JtabbedPane**
C. **JscrollPane**
D. **JToolBar**

3. **Atomic Components:** Placed on the intermediate-level swing containers. Atomic components are used to accept input from a user. The various atomic components are:

A. JTextField
B. **JComboBox**
C. **JCheckBox**
D. **JButton**
E. **JLabel**
F. **JTable**



EVENT-HANDLING IN JAVA

An object that describes a change of state in a source component is called an event. The source components can be the elements of the Graphical User Interface (GUI). When we interact with these source components, their state changes, and this change of state causes various events to occur. For example, **pressing a mouse button generates an event. The operating system traps the event and the data associated with it, such as the time of event occurrence and type of event occurred. The event type can be a mouse event, window event, or a mouse motion event**. **The operating system then passes this data to the application to which the event belongs.**

In java events are supported by the classes and interfaces defined in the java.awt.event package. The event-driven program contains objects with methods that control the various events and circumstances.

SOURCES OF EVENTS: Event sources are the Abstract Window Toolkit (AWT) GUI components, such as buttons, choice lists, and scroll bars. An event source can generate various types of events depending upon the change of state of the event source.

EVENT LISTENERS AND EVENT HANDLERS: **An event listener listens for a specific event and is notified when that specific event occurs. An event listener registers with one or more event sources to receive notifications about specific types of events and processes the events. An event-handler is called by the event-listener whenever a specific event occurs.**

In java, event listeners are interfaces and the event-handlers are the methods declared in the event listener interface that is implemented by the application. The methods that are defined in the set of interfaces and classes are found in the java.awt.event package.

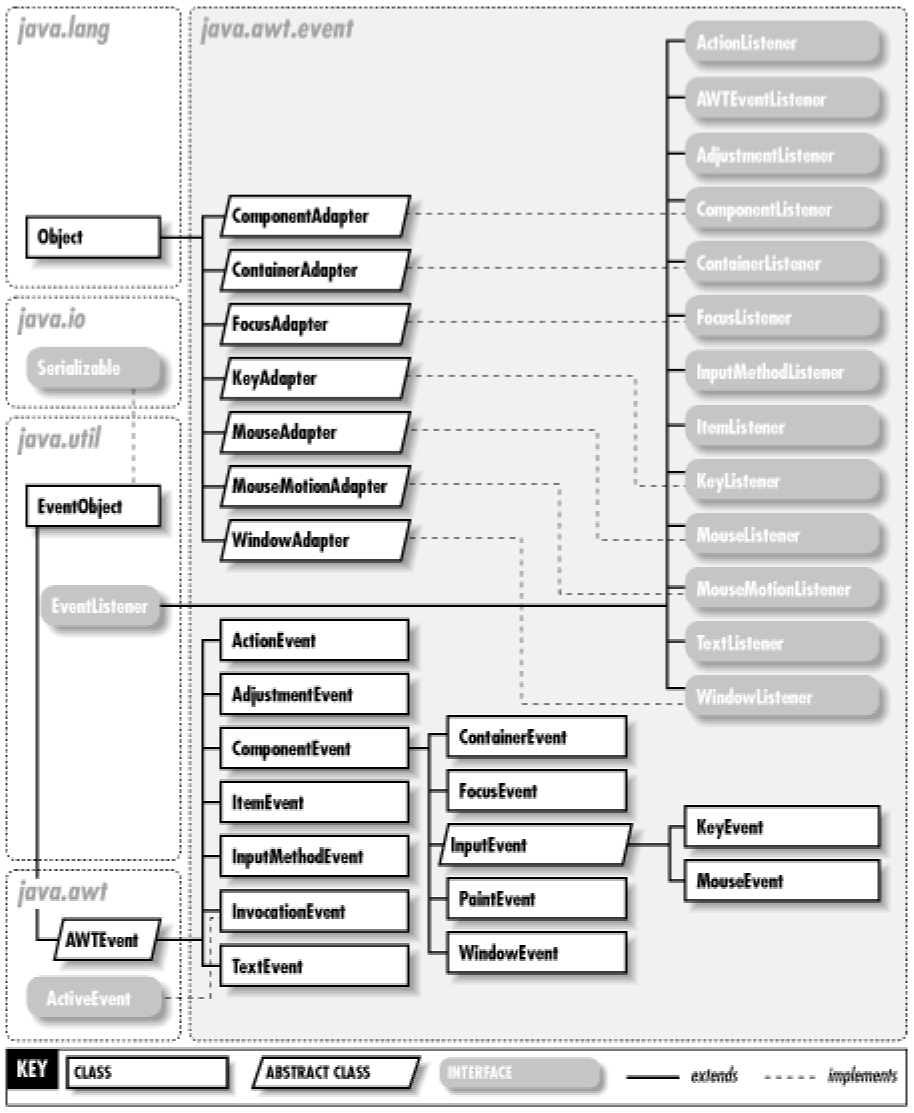
For example, the *MouseMotionListener* interface defines two methods to receive notification when a mouse is moved or dragged.

An event source registers event listeners, which receive the notifications about a specific type of event generation. The syntax of a method that registers an event listener with an event source is:

public void addTYPEListener (TYPEListener obj)

In the preceding syntax, TYPE is a type of event that an event listener handles, such as an action event or an item event and obj is a reference to the event listener.

For example, **the** *addActionListener ()* **method is used to register an action event listener and the** *addMotionListener ()* **is used to register a mouse motion listener.**

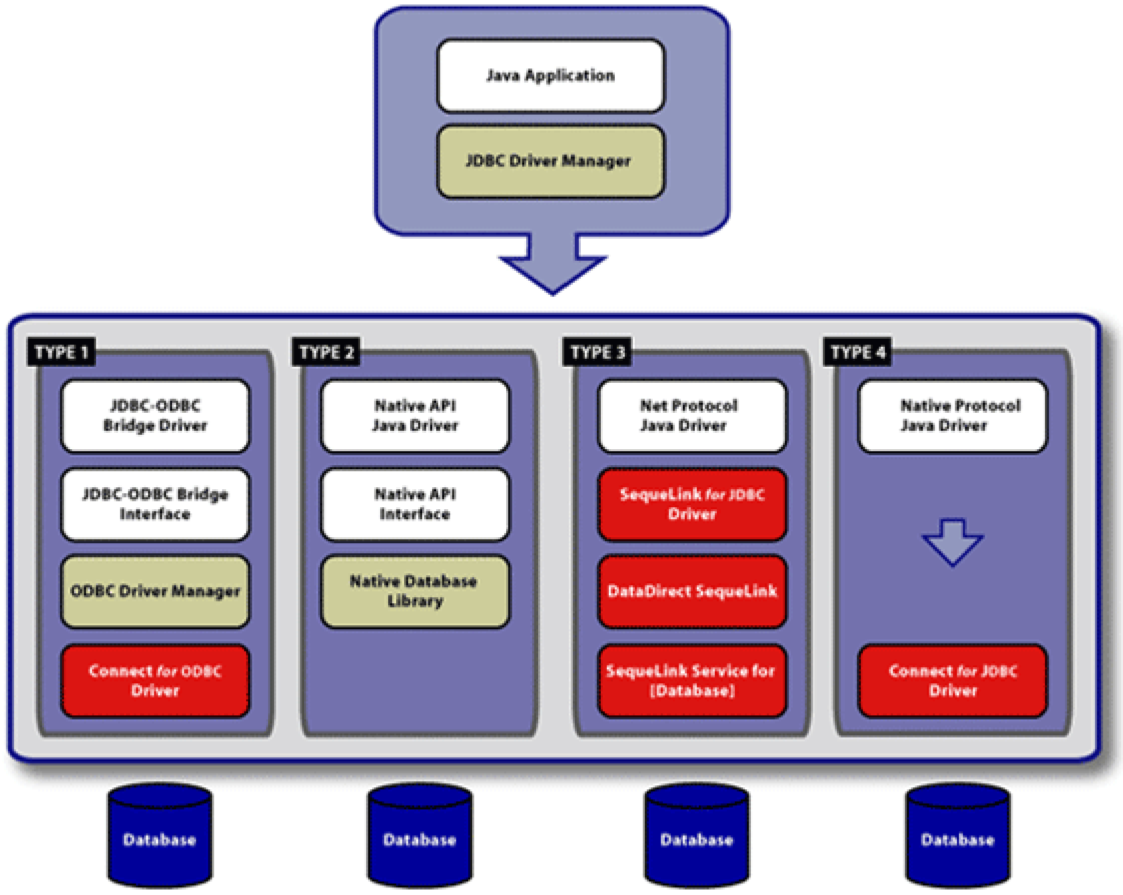

java.lang
Object
java.io
Serializable
java.util
EventObject
EventListener
java.awt
AWTEvent
ActiveEvent
java.awt.event
ComponentAdapter
ContainerAdapter
FocusAdapter
KeyAdapter
MouseAdapter
MouseMotionAdapter
WindowAdapter
ActionListener
AWTEventListener
AdjustmentListener
ComponentListener
ContainerListener
FocusListener
InputMethodListener
ItemListener
KeyListener
MouseListener
MouseMotionListener
TextListener
WindowListener
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
InputMethodEvent
InvocationEvent
TextEvent
ContainerEvent
FocusEvent
InputEvent
PaintEvent
WindowEvent
KeyEvent
MouseEvent
KEY
CLASS
ABSTRACT CLASS
INTERFACE
extends
implements

JAVA DATA BASE CONNECTIVITY (JDBC)

Java applications cannot directly communicate with a database to submit data and retrieve the results queries. This is because a database can interpret only SQL statements and not java language statements. For this reason, you need a mechanism to translate java statements into SQL statements. The JDBC architecture provides the mechanism for this kind of translation. The JDBC architecture can be classified into two layers:

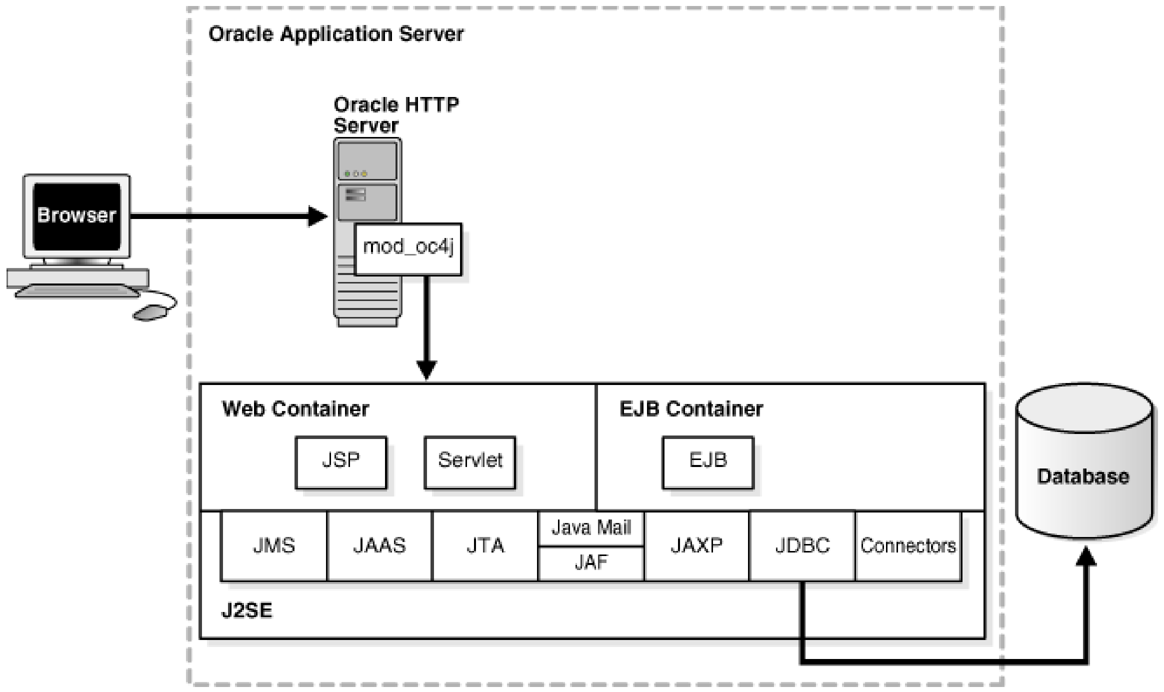
1. **JDBC Application Layer:** signifies a Java application that uses the JDBC API to interact with the JDBC drivers. A JDBC driver is software that a Java application uses to access a database. The JDBC driver manager of JDBC API connects the Java application to the server.

2. JDBC driver Layer: Acts as an interface between a Java application and a database. This layer contains a driver, such as a SQL Server driver or an Oracle driver, which enables connectivity to a database. A driver sends the request of a Java application to the database. After processing the request, the database sends the response back to the driver. The driver translates and sends the response to the JDBC API. The JDBC API forwards it to the Java application.



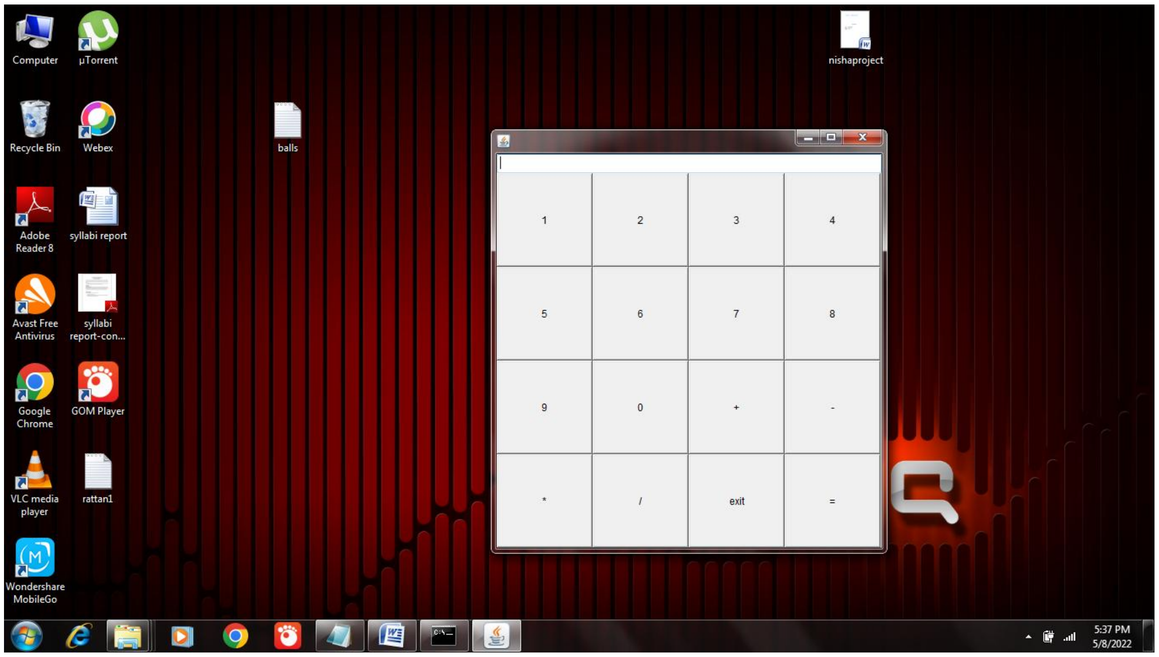
THE JDBC-ODBC BRIDGE DRIVER

The JDBC-ODBC Bridge driver is called the Type-1 driver. The JDBC-ODBC Bridge driver converts JDBC calls to Open Database Connectivity (ODBC) calls. ODBC is an open standard API to communicate with databases. The JDBC-ODBC bridge driver enables a Java application to use any database that supports ODBC driver. A Java application cannot interact directly with the ODBC driver. For this reason, the application uses the JDBC-ODBC Bridge driver that works as an interface between the application and the ODBC driver. To use the JDBC-ODBC Bridge driver you need to have the ODBC driver installed on the client computer. The JDBC-ODBC Bridge driver is usually used in stand-alone applications. The following figure shows the working of JDBC-ODBC Driver:


Oracle Application Server
Oracle HTTP Server
Browser
mod_oc4j
Web Container
JSP
Servlet
EJB Container
EJB
JMS
JAAS
JTA
Java Mail
JAF
JAXP
JDBC
Connectors
J2SE
Database

PROJECT DETAILS

1) Opening Page of our calculator



Coding of the project

```
import java.awt.*;
import java.awt.event.*;
class ss extends Frame implements ActionListener{
int a=0,b=0,c=0,i=0;
TextField tf;
Button b1,b2,b3,b4,b5,b6,b7,b8,b9,b0;
Button badd,bsub,bmul,bdiv,bexit,bequal;
ss(){
  setLayout(new BorderLayout());
  setSize(300,300);
Panel p1 = new Panel(new GridLayout(1,1));
Panel p2 = new Panel(new GridLayout(4,4));
  tf=new TextField(10);
  p1.add(tf);
b1=new Button("1");
b2=new Button("2");
b3=new Button("3");
b4=new Button("4");
b5=new Button("5");
b6=new Button("6");
b7=new Button("7");
b8=new Button("8");
b9=new Button("9");
b0=new Button("0");


badd=new Button("+");
bsub=new Button("-");
bmul=new Button("*");
bdiv=new Button("/");
bexit=new Button("exit");
bequal=new Button("=");

p2.add(b1);
p2.add(b2);
```

```
p2.add(b3);
p2.add(b4);
p2.add(b5);
p2.add(b6);
p2.add(b7);
p2.add(b8);
p2.add(b9);
p2.add(b0);
p2.add(badd);
p2.add(bsub);
p2.add(bmul);
p2.add(bdiv);
p2.add(bexit);
p2.add(bequal);
  add(p1,BorderLayout.NORTH);
  add(p2,BorderLayout.CENTER);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
b8.addActionListener(this);
b9.addActionListener(this);
b0.addActionListener(this);
badd.addActionListener(this);
bsub.addActionListener(this);
bmul.addActionListener(this);
bdiv.addActionListener(this);
bexit.addActionListener(this);
bequal.addActionListener(this);
  }
public void actionPerformed(ActionEvent ae){
  String s=ae.getActionCommand();
```

```
if(s.equalsIgnoreCase("exit"))
  System.exit(0);
if(s.equalsIgnoreCase("1"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("2"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("3"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("4"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("5"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("6"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("7"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("8"))
  tf.setText(tf.getText()+s);
if(s.equalsIgnoreCase("0"))
  tf.setText(tf.getText()+s);


if(s.equalsIgnoreCase("+"))
{        i=1;
  a=Integer.parseInt(tf.getText());
  tf.setText("");
}
if(s.equalsIgnoreCase("-"))
{        i=2;
  a=Integer.parseInt(tf.getText());
  tf.setText("");
}
if(s.equalsIgnoreCase("*"))
{        i=3;
  a=Integer.parseInt(tf.getText());
  tf.setText("");
}
if(s.equalsIgnoreCase("/"))
{        i=4;
  a=Integer.parseInt(tf.getText());
  tf.setText("");
  }
if(s.equalsIgnoreCase("="))
  {
  b=Integer.parseInt(tf.getText());
switch(i){
case 1:
  c=a+b;
  break;
case 2:
  c=a-b;
  break;
case 3:
  c=a*b;
  break;
case 4:
  c=a/b;
  break;
  }
tf.setText(""+c);
  }
}
}

class calculater{
public static void main(String args[]){
ss a1= new ss();
a1.setVisible(true);
}}
```

BIBLOGRAPHY

2 *HEAD FIRST JAVA 2nd EDITION, KATHY SIERRA & BERT BATES*

2. *JAVA FUNDAMENTALS (PART-1)-NIIT COURSEWARE (AAE LEVEL 2)*

3. *JAVA FUNDAMENTALS (PART-2)-NIIT COURSEWARE(AAE LEVEL 2)*

4. *PROGRAMMING IN JAVA-NIIT COURSEWARE (AAE LEVEL 2)*

5. *A COMPLETE REFERENCE TO JAVA, ROBERT SHIELD.*

6. *MICROSOFT SQL SERVER 2000, ALLAN HIRT, CATHAN COOK, KIMBERLEY TRIPP, FRANK McBATH*

7. *LEARN MICROSOFT SQL SERVER 2000, JOSE A. RAMATHO*

8. *www.java.sun.com*

9. *www.w3schools.org*

10. *www.google.co.in*

11. *www.javaranch.com*