Programming in Java

[1. Elements of java Language 1](#_Toc458790745)

[2. Object Oriented Programming in java 5](#_Toc458790746)

[3. Exception Handling 11](#_Toc458790747)

[4. Streams (File Handling) : 15](#_Toc458790748)

[5. Applet and Applications: 20](#_Toc458790749)

[Swing: 23](#_Toc458790750)

[6. Network Programming 28](#_Toc458790751)

[7. Java Database Connectivity(JDBC) 33](#_Toc458790752)

[JDBC drivers 34](#_Toc458790753)

# Elements of java Language

**Introduction about Java**

Java is a Object oriented, multi threaded, platform independent programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

The java source program is compiled into java virtual machine code called byte code. With an interpreter each java bytecode, instructions is parsed and run on the computer. The compilations of the program happen just once but interpretation occurs each time the program is executed.

**History of Java**

James Gosling and Patrick Naughton initiated the Java language project in June 1991 for use in one of his many set-top box projects. Java was originally designed for interactive television, but it was too advanced. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random word

**Sample Java Program**

*class Demo{*

*public static void main(String args[]){*

*System.out.println(“First Java”);*

*}*

*}*

Steps to save , compile and run java program.

i.Save above code with the name “Demo.java”. Extension of file must be java.

ii. Comile with javac command. Eg, javac Demo.java

iii.After successful compilation java compiler will generats the .class file. Class file is collection of byte code.

iv. To run java program use java command. Eg, java Demo. Here Demo is class file.

**Benefits of Java**

* It is "simple, object-oriented and familiar"
* It is "robust and secure"
* It is "architecture-neutral and portable"
* It execute with "high performance"
* It is "interpreted, threaded, and dynamic"

**Data Types**

Types of data that the variable will store. Data types also decide the memory occupied by variable. There are two categories for data type in java programming, which are primitive and reference types. There are eight types of primitive data:

**Data Type Size(bit) Wrapper class**

byte 8bit Byte

short 16 Short

int 32 Integer

float 32 Float

long 64 Long

double 64 Double

boolean Boolean

char 8 Character

Note: 8 bit = 1 byte

Java provides 2 types of reference type, which are:

i.Array

ii.Class

**Variables**

Variables are data or values stored that can be changed during the execution of the program; they must be initialized before being used. We can assign value to a variable at the time of declaration or at any point later on after the variable is being declared.

Syntax:

*type identifier; or type identifier=value;*

**Assignment and Initialization**

To assign a value to a variable is assignment. To assign the value to variable “=” operators are used. While assigning the value to a variable type of data must match with variables datatype.To Initialize variables, objects are initialization . Eg,

Int a=10;

Demo obj=new Demo();

String arr[]=new String[2];

**Operators**

Operators are symbols used with data or variables to create mathematical or logical expressions. Operates are used to performs mathematical operations between two or more than two operands.

There are 5 different types of operators used in java programming:

* Arithmetic Operator [+,-,\*,/,%]
* Relational or Comparison operators [<,>,==,<=,>=,!=]
* Logical operator [||, &&,!]
* Assignment operator [=,+=,-=,\*=,/=,%=]
* Increment/decrement operator (unary operator) [++,--]

**Strings**

A sequence of characters, which is enclosed within double quotation marks, is known as a string. The string is a special type classified as a reference and each String in java in implemented as object of type String.The class String is defined in java.lang.String which automatically imported into every program.

Eg,

String name=”ram”;

**Control Flow**

Control flow statements govern the flow of control on program during execution. That is the order in which statements are executed in a running program. There are three main categories of control flow:

* Selection statements [if,if-else,else if, switch] : That allows the program to choose between alternative actions during execution.
* Iteration statements [while,do while,for]: Allows a block of statements to be executed repeatedly. A Boolean condition called a loop condition is commonly used to determine when to terminate the loop.
* Transfer Statements [break,continue,return, try-catch-finally]: Use for transferring the control in a program.

**Methods (User Defined Function)**

Set of codes used to perform certain task is method. As java in pure object oriented programming language we always need to declare method inside the class.There are four type of methods.

i.method that doesn’t returns value. Eg, void add()

ii.Methods that returns value. Eg, int add()

iii.Methods that doens;t takes any arguments. Eg, void add()

iv.Methods that takes arguments. Eg, void add(int a,int b)

Methods can be declared in a two way inside class. Instance method and static method.

Instance method belongs to the object of class and always needs object to access it.Static method belongs to class and we can able to access it without using the object of class. Instance of instance methods are created for each objects where as instance of static methods are created only once.

**Array**

An array is a class structures that defines an indexed collection of a fixed number of homogeneous data elements. This means that all elements in the array have the same data types. In java array are objects, Arrays can be of primitives data types or reference types. Each array object has a final field called length which specifies the size of array.

Syntax

<Type>[] <identifier>;

int a[]=new int[3];

int[] b;

int c[]={1,2,3};

Pizza obj[]=new Pizza[2];

length: pre defined field, which returns length of array.

Multidimentional Array

int a[][]=new int[2][2];

int b[][]={{1,3},

{3,4}

};

Array Iteration

*for(Pizza p:obj){*

*System.out.println(p.price);*

*}*

# Object Oriented Programming in java

OOPS is a conceptual approach of designing program, with this approach data and program(procedure) can be packaged into single unit called object. Object oriented programming is an extension of procedural programming, It involves creating program components as objects t that are similar to concrete objects on the real world that are manipulated to achieve a desired results. When writing object-oriented programs, it involves both creating objects and creating applications that use those objects.

Oops is based on a three basic concept

i.Inheritance: Inheriting the property of one class by another class is Inheritance. We use extends key word in java to inherit property of a class. It allows new class to be derived from existing class. The sub class(child class) can add new behavior and properties, and under certain circumstances modify its inherited behavior.

ii.Encapsulation: Encapsulation is a mechanism that binds together attributes and methods into a single object, and keeps both safe from interference or misuse, that is the details of the object are encapsulated or hidden, from the user. We use access qualifies to achieve Encapsulation in Java. Encapsulation hides the details of the object, it sometimes is called information hiding.

iii. Polymorphism: Polymorphism is a feature that allows one interface to be used for a general class of action, that is , the ability of an object to take many forms or identities. Overloading and Overriding is example of Polymorphism in java.

**Class**

Group of data types and methods to perform certain task is class. Class is essential to implement object oriented programming concept in java. class contains static as well as instance methods and data types.To access the properties of class an object is must but we can able to access static property without using object. There are two types of classes available in java, they are:

Pre-Defined class: class already defined on a system. Eg, String,File,Thread

User-Defined class: class defined by user.

Creating class:

Syntax:

Class <Name of class>

Eg,

Class Student{

String name;

Int roll;

Void show(){

System.out.println(“Name of Student:”+name);

}

}

//Creating Object of class

Student obj=new Student();

//Accessing the properties of class

Obj.show();

**Constructor**

Used to initialized the properties of class. Doesn’t have return type. Name is same as of class. Called at a time of object creation. The main purpose of constructors is to set the initial state of an object when the object is created by using the new operator.

**Method overloading**

Each method has a signature, which comprises the name of the method and the types and order of the parameters in the formal parameters list. Several method implementations may have the same name, along as the method signatures differ, this is called method overloading.

**This keyword**

This keyword is Used to access the Global property of class. It always refers to current object.We can’t able to access the static property using this.

**Inheritance**

\*In java for inheritance we use extends keyword

\* Java does not support multiple inheritance. This simplifies the language and aids in preventing potential errors and [anti-pattern](http://en.wikipedia.org/wiki/Anti-pattern) design.

Syntax

*class <subclass> extends <parent class>*

**super**:

Always refer to base class. The super key word provides a reference to the current object as an instance of its superclass.

**Overriding**

Overriding is a concept to redefine method of super class. To override the method of super class sub class must define the method with same signature. Overriding is an example of polymorphism.

Eg,

*package packageA;*

*public class RnD extends ISYS{*

*String projects[];*

*public RnD(){*

*projects=new String[]{"ISYSE","DXEG"};*

*}*

*public RnD(String location){*

*super(location);*

*}*

*public void showProject(){*

*System.out.println("Projects:");*

*for(String p:projects){*

*System.out.println("\*"+p);*

*}*

*}*

*public void printInfoRnD(){ //overriding*

*System.out.println("RnD");*

*super.printInfoRnD();*

*}*

*public static void main(String args[]){*

*RnD childObj=new RnD();*

*childObj.printInfoRnD();*

*childObj.showProject();*

*ISYS parentObj=childObj;*

*parentObj.printInfoRnD(20);*

*}*

*}*

**Interface**

Interface is collection of abstract functions which must be defined by its sub class.

\* Only abstract and public modifiers

\* A class can neither narrow the accessibility of an interface method nor specify new exceptions in the method's throw clause

\* Use implements key word to inherit from interface, must define all methods of interface

\* An interface can extends other interfaces, using extends clause

\* Interface can extend any number of other interfaces.

syntax

*interface <name>{*

*}*

To Implement interface

Syntax

*class <subclass> implements <interface name>{*

*}*

Eg,

*package packageA;*

*public interface ISYSDAO {*

*public final static int MAX\_ROW=10;*

*public void displayData();*

*}*

*package packageA;*

*public class ISYSDAOOracleImpl implements ISYSDAO{*

*@Override*

*public void displayData() {*

*for(int i=0;i<MAX\_ROW;i++){*

*System.out.println(i);*

*}*

*}*

*public static void main(String args[]){*

*ISYSDAO dao=new ISYSDAOOracleImpl();*

*dao.displayData();*

*}*

*}*

**Nested class/Inner class**

A class that is declared within another class or interface is called nested class.

*package packageA;*

*public class NestedDemo {*

*class StudentInfo{*

*String name;*

*int age;*

*}*

*void showStudentInfo(){*

*for(StudentInfo data:getDatas()){*

*System.out.println("Name:"+data.name);*

*System.out.println("Age:"+data.age);*

*}*

*}*

*StudentInfo[] getDatas(){*

*StudentInfo[] datas=new StudentInfo[2];*

*datas[0]=new StudentInfo();*

*datas[0].name="Kalu";*

*datas[0].age=33;*

*datas[1]=new StudentInfo();*

*datas[1].name="Ramu";*

*datas[1].age=30;*

*return datas;*

*}*

*public static void main(String args[]){*

*NestedDemo obj=new NestedDemo();*

*obj.showStudentInfo();*

*}*

*}*

**Access Protection Mechanism:**

An Access specifiers defines the circumstances under which a class or class members in program can be accessed. It is used to control the visibility of properties of class. There are four types of access specifier in java.

Private: Access only by defining class

Protected: Access by defining class and its sub-class

Default: Access within the defining package

Public: Access every where

**Abstract class**

Class which does not complete in itself.

\* use abstract keyword to create abstract class

\* abstract class can define both abstract as well as normal methods

\* abstract class can't be instantiated

Syntax

*abstract class <Class name>{*

*public void show(){}*

*abstract public void shape();*

*}*

**Final class**

Class that can't be inherited. We can declare a class as final by using final key word.

syntax

*final class <class name>{}*

**Package**

Encapsulation mechanism that can be used to group related classes, interface and sub packages. Package declaration should always be first statement in java program. To access the class inside package we must use import key word. The (.) notation is used to uniquely identify package members in the package hierarchy.

Defining package

syntax

*package <package name>;*

Eg,

package com.ISys.RnD;

Using package

\*use import keyword

syntax

*import <fully qualified type name>;*

*import com.ISys.RnD.\*; (All classes inside this package)*

*import com.ISys.RnD.ISYSE; (ISYSE is class)*

# Exception Handling

While running program there might be possibility of getting errors. To trap this error, provide proper information about the error and make program run smoothly is Exception handling. The mechanism for handling exceptions is embedded in the tr-catch-finally construct, wich has the following form:

try{

Statement;

}catch(Exception){

Statement;

}finally{

Statement;

}

Exception thrown during execution of the try block can be caught and handled in a catch block. A finally block in guaranteed it be executed, regardless of the cause of exit from the try block or whether any cant block was executed.

All Exceptions derived from java.lang.Trowable class

Two main subclass of Throwable are

a. Exception

b. Error

\*Throwable provides following method to query an exception

i.*String getMessage()* :Detail Error message

ii.*void printStactTrace()* :Detail Error message along with error occurred line and other related information

**Types of exception**

\*RuntimeException : Occurs during execution of program

Eg,NullPointerException,ClassCastException,NumberFormatException,ArithMeticException,ArrayIndexOutOfBoundsException

\*Normal

IOException,SQLException,AWTException,FileNotFoundException

Checked & UnChecked Exception

\*Except for RuntimeException,Error and their subclasses all other exceptions are checked exception. We must explicitly deal with

exception. UnChecked Exception will be handled by complier.

*try,catch,finally*

*syntax*

*try{*

*block;*

*}catch(<Exception> <obj>){*

*block;*

*}finally{*

*block;*

*}*

**Some of exceptions related class:**

IOException : Exception occurs during I/O Operation

ArrayIndexOutOfBoudsException : While trying to access array with invalid index.

ClassCastException : while trying to cast between wrong classes.

FileNotFoundExcepion : If given file was not found while input or output operation.

SQLException : Exception related to Sql.Occures during SQL operation

Exception : Exception is super class of all exception.

**throw statement**

\*To explicitly throw an Exception

syntax

*throw <object reference of exception>;*

**throws clause**

Method can use throws clause to unhandle checked exception.

Eg,

*package packageA;*

*public class ExceptionDemo {*

*public void divide(int a,int b){*

*int res=0;*

*try{*

*res=a/b;*

*System.out.println("Result:"+res);*

*}catch(ArithmeticException e){*

*System.out.println("Arithmetic Exception occure,please provide correct value"+e.getMessage());*

*e.printStackTrace();*

*}catch(Exception e){*

*System.out.println("Other Exception occured...");*

*}*

*finally{*

*res=0;*

*System.out.println("Finally block");*

*}*

*}*

*public void divideA(int a,int b){*

*if(b==0){*

*throw new ArithmeticException();*

*}*

*int res=a/b;*

*System.out.println("Result:"+res);*

*}*

*public void divideB(int a,int b)throws Exception{*

*if(b==0){*

*throw new Exception();*

*}*

*int res=a/b;*

*System.out.println("Result:"+res);*

*}*

*public static void main(String args[]){*

*ExceptionDemo obj=new ExceptionDemo();*

*obj.divide(2,2);*

*obj.divide(2,0);*

*obj.divideA(2,0);*

*//obj.divideB(2,3);//Error this line can't compile either use try catch or throws*

*}*

*}*

**Casting**:

Converting a value of one type into another type. Casting can be applied to primitive values as well as references. Casting between primitives data types and reference types in not permitted. Boolean value can’t be cast to other data values and vice versa.

Syntax:

(<type>) <expression>

Eg,

int a=20;

long b=2222;

b=a;

a=(int)b;

**Reflection:**

Provides ability to find information about class, methods and properties during runtime.

Class c=Class.forName()

Package package = aClass.getPackage();

Class superclass = aClass.getSuperclass();

Class[] interfaces = aClass.getInterfaces();

Constructor[] constructors = aClass.getConstructors();

Method[] method = aClass.getMethods();

Field[] method = aClass.getFields();

# Streams (File Handling) :

To bring data into a program, a Java program opens a stream to a data source, such as a file or remote socket, and reads the information serially. On the flip side, a program can open a stream to a data source and write to it in a serial fashion. Whether we were reading from a file or from a socket, the concept of serially reading from, and writing to different data sources is the same. This process in java is called Stream Handling. There are two types of stream in java:

i.Byte Stream: To read and write byte data. Eg, Image file,sound file etc. InputStream and OutputStream are classes used to read and write byte data.

ii.Character Stream: To read and write character base data. Eg, Plain Text file. Reader and Writer are the classes used to read and write character data.

**Handling Files:**

File handling is concept to read data from file and write data into file. We take help of different classes inside java.io package to read and write from various files. File handling also provides us capability to get information about the file. Some of classes used for file handling are:

i.File : To point to the file and to get information about the file.

ii.FileReader : To read character data from file.

iii.BufferedReader : To read character data into buffer.

iv.FileInputStream : To open file in input mode and read the content of file in byte format.

v.FileOutputStream : To open file in output mode and writer the content in byte format.

vi.PrintWriter : To print chacater type of data into various source, such as file,socket or console.

Example:

**import** java.io.\*;

**public** **class** FileDemo {

**public** **static** **void** main(String args[])**throws** Exception{

FileDemo obj=**new** FileDemo();

obj.fileWrite(“abc.txt”);

obj.fileReader(“abc.txt”);

}

**public** **void** fileWrite(String fileName)**throws** Exception{

File f=**new** File(fileName);

PrintWriter pw=**new** PrintWriter(**new** FileOutputStream(f, **true**));

pw.println("This is File Writing");

pw.close();

}

**public** **void** fileReader(fileName)**throws** Exception{

BufferedReader br=**new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(**new** File(fileName))));

**while**(**true**){

String line=br.readLine();

**if**(**null**==line){

**break**;

}

System.*out*.println("Line::"+line);

}

}

}

**Zip file Stream:**

Java provides a facility to read from zip file and create a zip file. This concept of zip file reading and writing is Zip File Stream. From a java.util.zip.ZipFile object, we can get access to the name and data of each entry in the zip bundle.

ZipFile : Represents the zip file

Entries() -> Returns the collection of Zip File

ZipEntry: Represents single file in zip

ZipInputStream: To read from zip file

ZipOutputStream: to write zip file

nextEntry(ZipFile) : to add new ZipFile

Eg, (Reading Zip file)

**import** java.util.\*;

**import** java.util.zip.\*;

**import** java.io.\*;

**class** ZipFileReadDemo{

**public** **static** **void** main(String[] args) {

**try** {

File file = **new** File("zipfile.zip");

ZipFile zipFile = **new** ZipFile(file);

// create a directory named the same as the zip file in the

// same directory as the zip file.

File zipDir = **new** File(file.getParentFile(), "zipfile");

zipDir.mkdir();

Enumeration<ZipEntry> entries =(Enumeration<ZipEntry>) zipFile.entries();

**while**(entries.hasMoreElements()) {

ZipEntry entry = entries.nextElement();

String nme = entry.getName();

// File for current file or directory

File entryDestination = **new** File(zipDir, nme);

// This file may be in a subfolder in the Zip bundle

// This line ensures the parent folders are all

// created.

entryDestination.getParentFile().mkdirs();

// Directories are included as seperate entries

// in the zip file.

**if**(!entry.isDirectory()) {

*generateFile*(entryDestination, entry, zipFile);

}

}

}

**catch**(IOException e) {

e.printStackTrace();

}

}

**private** **static** **void** generateFile(File destination, ZipEntry entry, ZipFile owner) **throws** IOException {

InputStream in = **null**;

OutputStream out = **null**;

**try** {

InputStream rawIn = owner.getInputStream(entry);

in = **new** BufferedInputStream(rawIn);

FileOutputStream rawOut = **new** FileOutputStream(destination);

out = **new** BufferedOutputStream(rawOut);

// pump data from zip file into new files

**byte**[] buf = **new** **byte**[1024];

**int** len;

**while** ((len = in.read(buf)) > 0) {

out.write(buf, 0, len);

}

}

**finally** {

**if**(in != **null**) {

in.close();

}

**if**(out != **null**) {

out.close();

}

}

}

}

Eg,(Creating Zip file)

**import** java.io.\*;

**import** java.util.zip.ZipEntry;

**import** java.util.zip.ZipOutputStream;

**public** **class** ZipFileCreatorDemo {

**public** **static** **void** main(String args[]){

ZipOutputStream out = **null**;

InputStream in = **null**;

**try** {

File outputFile = **new** File("myZipFile1.zip");

OutputStream rawOut = **new** BufferedOutputStream(**new** FileOutputStream(outputFile));

out = **new** ZipOutputStream(rawOut);

String files[]={"abc.txt","abc1.txt"};

**for**(String f:files){

File inputFile1 = **new** File(f);

// optional - manages amount of compression

// out.setLevel(java.util.zip.Deflator.BEST\_COMPRESSION);

InputStream rawIn = **new** FileInputStream(inputFile1);

in = **new** BufferedInputStream(rawIn);

// entry for our file

// should be root/sub/sub2/myFile.txt if we want it

// 3 folders deep

ZipEntry entry = **new** ZipEntry(f);

// notify output stream of entry.

out.putNextEntry(entry);

// pump data from file into zip file

**byte**[] buf = **new** **byte**[1024];

**int** len;

**while** ((len = in.read(buf)) > 0) {

out.write(buf, 0, len);

}

}

}**catch**(IOException e) {

e.printStackTrace();

}

**finally** {

**try** {

**if**(in != **null**) {

in.close();

}

**if**(out != **null**) {

out.close();

}

}

**catch**(IOException ignored) { }

}

}

}

**Object Streams:**

Object stream is used to read and write data into object format. Using object stream we can able to write java object into file and read the object from file. Following are the class and its function used to work with object stream.

ObjectOutputStream : class to write object data.

writeOject(Object)

ObjectInputStream : class to read object data.

readObject()

Eg,

**//Read Write object**

**public** **void** objectWrite()**throws** Exception{

FileOutputStream fos = **new** FileOutputStream(**new** File("t.tmp"));

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

MyPojo obj=**new** MyPojo();

obj.setName("Java");

oos.writeObject(obj);

obj=**new** MyPojo();

obj.setName("Oracle");

oos.writeObject(obj);

oos.close();

}

//Read Object

**public** **void** objectRead()**throws** Exception{

FileInputStream input=**new** FileInputStream(**new** File("t.tmp"));

ObjectInputStream ios=**new** ObjectInputStream(input);

**while**(**true**){

**try**{

MyPojo obj=(MyPojo)ios.readObject();

System.*out*.println("Read Object:"+obj.getName());

}**catch**(EOFException e){

**break**;

}

}

}

# Applet and Applications:

**Fundamental Concept of Applet:**

Beside using java to develop stand-alone application, java can also used to develop applets.Using applet we can able to develop a java program to create Graphics User Interface(GUI) based application. Applets are programs that are called from within another applicaton. It can be run within a page on the internet or within another program called appletviewer, which comes with the java development kit. An applet must be called from within another document written in html, which is used to create web pages for the internet. In Applet there are four life cycle methods .

i.init()

ii.start()

iii.stop()

iv.destroy()

**Simple Applets:**

*import java.applet.\*;*

*public class SampleApplet extends Applet*

*{*

*public void init(){*

*System.out.println("Applet initilized");*

*}*

*public void start(){*

*System.out.println("Applet Started");*

*}*

*public void stop(){*

*System.out.println("Applet Stopped");*

*}*

*public void destroy(){*

*System.out.println("Applet destroyed");*

*}*

*}*

*/\**

*<applet code=”SampleApplet.class” width=”100” height=”100”></applet>*

*\*/*

**Testing Applet:**

There are two ways to test the applet.

i.by using appletviewer command, provided by JDK

ii.By using Web browser

While creating and testing applet, the following have to be done:

* Write the applet in the java programming language, and save it with a .java file extension.
* The applet program must be complied into bytecode using javac command.
* Write a HTML document that includes a statement to call our complied java class.
* Load the HTML document into a Web browser or run the appletviewer program.

**Converting Application to applet:**

* Create a class that must extends from Applet class
* The class must be public
* Define required Applet lifecycle method
* Create HTML or add applet tag at end of class
* Load the HTML document into a Web browser or run the appletviewer program.

**Applets HTML tag and attributes:**

In Order to run an applet, a HTML document has to be created. Within the HTML document , we will have to add an <applet> and </applet> tag pair. Following are the attributes of Applet Html tag:

Code: name of the compiled class(applet) which we are calling

Width: the width of applet on the screen.

Height: the height of applet on the screen.

Align: Specifies the alignment of an applet according to surrounding elements

Alt: Specifies an alternate text for an applet

Archive: Specifies the location of an archive file

Codebase: Specifies a relative base URL for applets specified in the code attribute

hspace: Defines the horizontal spacing around an applet

name: Defines the name for an applet (to use in scripts)

vspace: Defines the vertical spacing around an applet

**Unsigned applets *cannot* perform the following operations (Applet Restriction)**:

* They cannot access client resources such as the local filesystem, executable files, system clipboard, and printers.
* They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).
* They cannot load native libraries.
* They cannot change the SecurityManager.
* They cannot create a ClassLoader.
* They cannot read certain system properties. See [System Properties](http://docs.oracle.com/javase/tutorial/deployment/doingMoreWithRIA/properties.html) for a list of forbidden system properties.

**Pop-up window in applet:**

To Create pop up from applet we should use following code:

try  
    {  
      AppletContext a = getAppletContext();  
      URL url = new URL(“”);  
      a.showDocument(url,"\_blank");  
    }catch(Exception e){}

**6.Graphics and Image animation/multimedia:**

**Graphics Programming:**

**Graphics Object**

**Graphics:**

The Graphics class in java has a rich variety of graphics function. All the output into the window take place through a graphics context. We can easily draw and color simple objects like line, circles,rectangle,polygons etc very easily. The position of these objects Is relative to the of left hand corner of the screen(0,0).

**Functions to draw various shapes:**

drawLine: Allows us to draw horizontal, vertical and diagonal lines. It takes in four parameters to draw any of these lines. Eg, drawLine(int xpos,in typos,int x1pos,int y1post)

drawRect: Allows us to draw rectangle or square.It takes in four parameters to draw any of these shapes.Eg, drawRect(int xpos,int ypos,int width,int height)

drawOval:Allows us to draw a circle or eclipse. It takes in four parameters to draw the shape. They are the x and y co-ordinates of the striating and the width and heights of the circle.Eg, drawOval(int xpos,int ypos,int width,int height).

drawArc: Allows us to draw an arc.It takes in six parameters to draw an arc.They are the x and y co-ordinates of the strating, the width and length of the arc, the starting position of the arc and the angle of the arc. Eg, drawArc(int xpos,int ypos,int width,int height,int starting angle,int degree).

drawRoundRect: Allows us to draw a round rectangle. It takes in six parameters to draw a rectangle. They are x and y co-ordinates of the starting, width and height of rectangle and width and height of edge curve. Eg, drawRoundRect(int xpos,int ypos,int width,int height,int roundwidth,int roundheight)

drawPolygon: Allows us to draw polygon. It takes in three parameters to draw polygon. They are arrays of x co-ordinates, y co-ordinates and number of points.Eg, drawPolygon(int xpos[],int ypos[],int number of points)

drawImage: Allows us to draw image on window. We can use an instance of the image class to draw images. Methods like getImage() and drawImage() are used to get images and display them on the screen.Eg, drawImage(Image img,int xpos,int ypos,Object parent)

**5.Events , Handling Events and AWT/Swing:**

## Swing:

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check box and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables and lists. Swing components are present inside javax.swing package.

**JFrame:**

Frame is used to create GUI based stand alone application.It is a top-level window with a title, a border, a close button. Usually it contains Panels. Its derived from a window and has a borderlayout by default.

**JPanel:** Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

The default layout manager for a panel is the FlowLayout layout manager.

Eg,

JPanel p=new JPanel();

p.add(new JButton(“OK”));

This will add Button in panel.

**Layout Management**

Layout Determines the size and position of the components within a container.Layout is used to define the placement of component inside Container. BorderLayout is default layout of frame, where as Flowlayout is default layout of Applet.we can able to set the layout of container by using function setLayout(). Following is types of layout:

**BorderLayout:** A BorderLayout places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area.

**BoxLayout:** The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components**.**

**CardLayout:** The CardLayout class lets you implement an area that contains different components at different times. A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group o f components) the CardLayout displays.

**FlowLayout:** FlowLayout is the default layout manager for every JPanel. It simply lays out components in a single row, starting a new row if its container is not sufficiently wide

**GridBagLayout:**GridBagLayout is a sophisticated, flexible layout manager. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.

**GridLayout:**GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

**GroupLayout:**GroupLayout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently**.**

**SpringLayout:**SpringLayout is a flexible layout manager designed for use by GUI builders. It lets you specify precise relationships between the edges of components under its control. For example, you might define that the left edge of one component is a certain distance (which can be dynamically calculated) from the right edge of a second component

**Component**

**JTextField:** A TextField object is a text component that allows for the editing of a single line of text. . Every time the user types a key in the text field, one or more key events are sent to the text field. A KeyEvent may be one of three types: keyPressed, keyReleased, or keyTyped.

Eg, JTextField tf=new JTextField(int number of character)

**JButton:** This class creates a labeled button. The application can cause some action to happen when the button is pushed.If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method.

Eg,

JButton b=new JButton("Ok");

**JTextArea:**

A TextArea object is a multi-line region that displays text. It can be set to allow editing or to be read-only.

Eg,

JTextArea ta=new JTextArea(int rows,int cols)

**JCheckBox:**

A check box is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."

Eg,

JCheckBox c=new JCheckBox(Sring text);

**JRadioButton**

A radiobutton is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a radiobutton changes its state from "on" to "off," or from "off" to "on.". Only single selection is allowed when radio button is associated with ButtonGroup.

JRadioButton male=new JRadioButton(“Male”);

JRadioButton female=new JRadioButton(“Female”);

ButtonGroup bg=new ButtonGroup();

bg.add(male);

bg.add(female);

**JComboBox**

A component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request. If we make the combo box editable, then the combo box includes an editable field into which the user can type a value.

Eg,

JComboBox jb=new JComboBox();

Jb.addItem(Object item)

**Event Handling:**

Event Handling is a concept to trap different actions and behavior performed by user, such as clicking a button, moving mouse, closing window these all are event. Through event handing we can able to trap those actions and performed a desire task.

**AWT Event Hierarchy:**

**Semantics and Low level Event:**

**1)Semantic events**

ActionEvent

AdjustmentEvent

ItemEvent

TextEvent

These classes are used for high level semantic events, to represent user interaction with a GUI component. For example, clicking a button, selecting a menu item, selecting a checkbox, scrolling and changing text in the text field or text area all result in individual semantic events

**2) Low level Event**

ComponentEvent

ContainerEvent

FocusEvent

KeyEvent

MouseEvent

PaintEvent

**WindowEvent**

These classes are used to represent low level input or window operations. Several low level events can constitute a single semantic event. For example, clicking on a button is actually a sequence of mouse movements to position the cursor followed by pressing and releasing a mouse button.

**Event Delegation Module**

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

There are three participants in event delegation model in Java;

- Event Source – the class which broadcasts the events. Eg, Button,Mouse  
- Event Listeners – the classes which receive notifications of events. Eg, ActionListener,MouseListener  
- Event Object – the class object which describes the event. Eg, ActionEvent,MouseEvent

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

**Multicasting Event**

Multicast events are quite useful when we have to work with many windows in our application and we just want to perform the same action or a group of actions in a number of windows in our application at the same time.

For example, if the user have opened many windows in the application, we might want to provide a command that closes all the windows at once.

**Dialog box:** A Dialog window is an independent subwindow meant to carry temporary notice apart from the main Application Window. Most Dialogs present an error message or warning to a user, but Dialogs can present images, directory trees, or just about anything compatible with the main Application that manages them.

# Network Programming

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols:

**TCP(Transmission Control Protocol):**

TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

**UDP(User Datagram Protocol):**

UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

**Socket Programming**

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

* The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
* The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
* After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
* The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
* On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

**URL**

URL stands for Uniform Resource Locator and represents a resource on the World Wide Web, such as a Web page or FTP directory.

A URL can be broken down into parts, as follows:

protocol://host:port/path?query#ref

Examples of protocols include HTTP, HTTPS, FTP, and File. The path is also referred to as the filename, and the host is also called the authority.

The following is a URL to a Web page whose protocol is HTTP:

http://www.amrood.com/index.htm?language=en#j2se

Notice that this URL does not specify a port, in which case the default port for the protocol is used. With HTTP, the default port is 80.

URL Class’s Method:

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | **public String getPath()** Returns the path of the URL. |
| 2 | **public String getQuery()** Returns the query part of the URL. |
| 3 | **public String getAuthority()** Returns the authority of the URL. |
| 4 | **public int getPort()** Returns the port of the URL. |
| 5 | **public int getDefaultPort()** Returns the default port for the protocol of the URL. |
| 6 | **public String getProtocol()** Returns the protocol of the URL. |
| 7 | **public String getHost()** Returns the host of the URL. |
| 8 | **public String getHost()** Returns the host of the URL. |
| 9 | **public String getFile()** Returns the filename of the URL. |
| 10 | **public String getRef()** Returns the reference part of the URL. |
| 11 | **public URLConnection openConnection() throws IOException** Opens a connection to the URL, allowing a client to communicate with the resource. |

**Example**

The following URLDemo program demonstrates the various parts of a URL. A URL is entered on the command line, and the URLDemo program outputs each part of the given URL.

// File Name : URLDemo.java

import java.net.\*;

import java.io.\*;

public class URLDemo

{

public static void main(String [] args)

{

try

{

URL url = new URL("http://www.amrood.com/index.htm?language=en#j2se");

System.out.println("URL is " + url.toString());

System.out.println("protocol is " + url.getProtocol());

System.out.println("authority is " + url.getAuthority());

System.out.println("file name is " + url.getFile());

System.out.println("host is " + url.getHost());

System.out.println("path is " + url.getPath());

System.out.println("port is " + url.getPort());

System.out.println("default port is " + url.getDefaultPort());

System.out.println("query is " + url.getQuery());

System.out.println("ref is " + url.getRef());

}catch(IOException e)

{

e.printStackTrace();

}

}

}

**URLConnections Class :**

The openConnection() method returns a **java.net.URLConnection**, an abstract class whose subclasses represent the various types of URL connections.

For example:

* If we connect to a URL whose protocol is HTTP, the openConnection() method returns an HttpURLConnection object.
* If we connect to a URL that represents a JAR file, the openConnection() method returns a JarURLConnection object.
* etc...

The URLConnection class has many methods for setting or determining information about the connection.

**Example:**

The following URLConnectionDemo program connects to a URL entered from the command line.

If the URL represents an HTTP resource, the connection is cast to HttpURLConnection, and the data in the resource is read one line at a time.

// File Name : URLConnDemo.java

import java.net.\*;

import java.io.\*;

public class URLConnDemo

{

public static void main(String [] args)

{

try

{

URL url = new URL("http://www.amrood.com");

URLConnection urlConnection = url.openConnection();

HttpURLConnection connection = null;

if(urlConnection instanceof HttpURLConnection)

{

connection = (HttpURLConnection) urlConnection;

}

else

{

System.out.println("Please enter an HTTP URL.");

return;

}

BufferedReader in = new BufferedReader(

new InputStreamReader(connection.getInputStream()));

String urlString = "";

String current;

while((current = in.readLine()) != null)

{

urlString += current;

}

System.out.println(urlString);

}catch(IOException e)

{

e.printStackTrace();

}

}

}

# Java Database Connectivity(JDBC)

**JDBC** is a Java-based data access technology (Java Standard Edition platform) from Sun Microsystems, Inc.. It is an acronym as it is unofficially referred to as **Java Database Connectivity**, with DB being universally recognized as the abbreviation for **database**. This technology is an [API](http://en.wikipedia.org/wiki/Application_programming_interface) for the [Java programming language](http://en.wikipedia.org/wiki/Java_%28programming_language%29) that defines how a client may access a [database](http://en.wikipedia.org/wiki/Database). It provides methods for querying and updating data in a database. JDBC is oriented towards [relational databases](http://en.wikipedia.org/wiki/Relational_database_management_system). A JDBC-to-[ODBC](http://en.wikipedia.org/wiki/Open_Database_Connectivity) bridge enables connections to any ODBC-accessible data source in the [JVM](http://en.wikipedia.org/wiki/JVM) host environment.

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as [SQL](http://en.wikipedia.org/wiki/SQL)'s CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additional+ly, [stored procedures](http://en.wikipedia.org/wiki/Stored_procedures) may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

* [Statement](http://download.oracle.com/javase/7/docs/api/java/sql/Statement.html) – the statement is sent to the database server each and every time.
* [PreparedStatement](http://download.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html) – the statement is cached and then the execution path is pre determined on the database server allowing it to be executed multiple times in an efficient manner.
* [CallableStatement](http://download.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html) – used for executing [stored procedures](http://en.wikipedia.org/wiki/Stored_procedures) on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.

Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the [javax.sql](http://download.oracle.com/javase/7/docs/api/javax/sql/package-summary.html).

JDBC connections are often managed via a connection pool rather than obtained directly from the driver. Examples of connection pools include [BoneCP](http://jolbox.com), [C3P0](http://sourceforge.net/projects/c3p0) and [DBCP](http://commons.apache.org/dbcp)

## JDBC drivers

JDBC drivers are client-side [adapters](http://en.wikipedia.org/wiki/Adapter) (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

**Types**

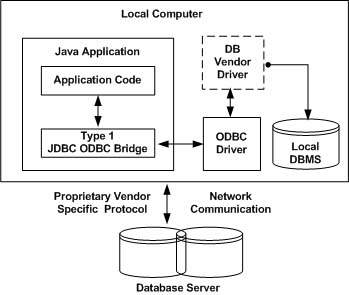
There are commercial and free drivers available for most relational database servers. These drivers fall into one of the following types:

* [Type 1](http://en.wikipedia.org/wiki/JDBC_driver#Type_1_Driver_-_JDBC-ODBC_bridge) that calls native code of the locally available ODBC driver.
* [Type 2](http://en.wikipedia.org/wiki/JDBC_driver#Type_2_Driver_-_Native-API_Driver_specification) that calls database vendor native library on a client side. This code then talks to database over network. Eg, oracle OCI Driver
* [Type 3](http://en.wikipedia.org/wiki/JDBC_driver#Type_3_Driver_-_Network-Protocol_Driver), the pure-java driver that talks with the server-side middleware that then talks to database.
* [Type 4](http://en.wikipedia.org/wiki/JDBC_driver#Type_4_Driver_-_Native-Protocol_Driver), the pure-java driver that uses database native protocol. Eg, oracle thin driver.

**JDBC-ODBC-BRIDGE Driver**

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



**Advantages**

Almost any database for which an ODBC driver is installed can be accessed, and data can be retrieved.

**Disadvantages**

Performance overhead since the calls have to go through the JDBC bridge to the ODBC driver, then to the native db connectivity interface (thus may be slower than other types of drivers).

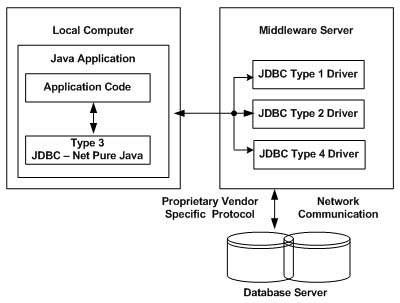
The ODBC driver needs to be installed on the client machine.

Not suitable for applets, because the ODBC driver needs to be installed on the client.

**Java Native Driver**

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead. For EG, OCI Driver (Oracle Callable Interface)



**Advantages**

As there is no implementation of JDBC-ODBC bridge, its considerably faster than a type 1 driver.

**Disadvantages**

The vendor client library needs to be installed on the client machine.

Not all databases have a client side library.

This driver is platform dependent.

This driver supports all java applications except applets.

import java.sql.\*;

public class CreateExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String user = "your\_username";

String password = "your\_password";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

String sql = "INSERT INTO employees (name, age) VALUES (?, ?)";

PreparedStatement preparedStatement = connection.prepareStatement(sql);

String name = "John Doe";

int age = 30;

preparedStatement.setString(1, name);

preparedStatement.setInt(2, age);

int rowsInserted = preparedStatement.executeUpdate();

if (rowsInserted > 0) {

System.out.println("New employee record created successfully!");

} else {

System.out.println("Failed to create a new employee record.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

import java.sql.\*;

public class ReadExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String user = "your\_username";

String password = "your\_password";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

String sql = "SELECT \* FROM employees";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

int age = resultSet.getInt("age");

System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

import java.sql.\*;

public class UpdateExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String user = "your\_username";

String password = "your\_password";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

String sql = "UPDATE employees SET name = ?, age = ? WHERE id = ?";

PreparedStatement preparedStatement = connection.prepareStatement(sql);

String newName = "Jane Smith";

int newAge = 35;

int employeeIdToUpdate = 1;

preparedStatement.setString(1, newName);

preparedStatement.setInt(2, newAge);

preparedStatement.setInt(3, employeeIdToUpdate);

int rowsUpdated = preparedStatement.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Employee record updated successfully!");

} else {

System.out.println("Failed to update the employee record.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

import java.sql.\*;

public class DeleteExample {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database\_name";

String user = "your\_username";

String password = "your\_password";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

String sql = "DELETE FROM employees WHERE id = ?";

PreparedStatement preparedStatement = connection.prepareStatement(sql);

int employeeIdToDelete = 1;

preparedStatement.setInt(1, employeeIdToDelete);

int rowsDeleted = preparedStatement.executeUpdate();

if (rowsDeleted > 0) {

System.out.println("Employee record deleted successfully!");

} else {

System.out.println("Failed to delete the employee record.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**Happy Coding ☺**

***NOTE:* For example code please refer to program written in a class. If you want to score more marks then please do some brain storming by making hands dirty on Google (www.google.com)**