**JAVA**

***Why Java?***

Software is a development process which converts imaginaries into reality by writing the set of programs.

Software

System Software Application Software Internet Software

(Used to Develop Functions (Used to Develop (Used to Develop

for Hardware Devices) Organizational Oriented) Distributed Applications)

In the Information Technology (IT) we aredeveloping two types of applications. They are

*a) Standalone Applications*

*b) Distributed Applications*

***Standalone Applications***

* Runs in the context of a local disk
* We can't achieve the concept of datasharing
* System software and application software comes under standalone applications
* For example C, C++, COBOL

***Distributed Applications***

* Runs in the context of a browser or in the world wide web (www)
* Can be accessed across the globe
* We can achieve the concept of data sharing
* Must use client server architecture
* Internet software is used for the distributed applications
* For example Java and Ms.Net

***History of JAVA***

Java is a general purpose, high level, structured, object-oriented and internet related programming language.Javais a distributed technology developed by the ***Sun***(Stanford UniversityNetwork) ***Micro Systems in USA*** in the year ***1991*** by the author ***James Gosling***.It was named as ***OAK****(which is a tree name)* by the author James Gosling.It is mainly invented to writethe programs with small codes to control electronic appliances like TV's, VCR's … etc.Java is a machine and platform independent programming language.In javathe source programs are converted into the byte code programs at the time of the program compilation.The byte code programs are executedin any machine at any platform(OS).In 1993 the World Wide Web (www) is implemented to transmit graphical images through internet by java.In 1994 a separate version of java called ***Hot Java***was invented to write the internet related programs called ***Applets***.In 1995 the name OAK was changed with JAVA(which is a coffee seed name) some legal problems.In 1996 it was implemented as a fullpledged object-oriented programming language.

The slogan of java is "*Write Once and Reuse/Run Anywhere (WORA)"*.

***Java Features***

A java program is designed to embedded on the internet web page is called an ***Applet****.*A java program is designed to execute on servers is called a ***Servlet****.*A java program that is called by itself is called a java ***Application Program****.*

Java has the following features and they are

***1. Compiled and Interpreted***

Java compiler generates a byte code file from the java source file at the time of the program compilation. In java the source files have an extension ***.java***and the byte code files have an extension***.class***. Java interpreter can execute the byte code files through the***JVM*** (Java Virtual Machine).

***2. Machine Independent and Portable***

Java byte code files are the machine and platform independent files, and portable than the other language files. So, the java byte code files are executed in any machine at any platform.

***3. Object-Oriented***

Java is a fully object oriented programming language. So, in java the everyprogram ismust formed by the classes.

***4. Distributed***

Java has the built-in distributed environment.So, in java the server side applications are also executed into the client side.

***5. Robust and Secure***

Java isa secured language used to transmit java programs through internet without virus effect. Java eliminates some of the unnecessary features like pre-processorstatements, pointers, goto statement …etc is called robust.

***6. High Performance***

Java has high performance than other languages. In Java the server side applications are also executed with high performance in the client side.

***7. Multi-Threaded and Interactive***

A Java program may execute several tasks at a time called multi-threaded. Interactive means the java graphical applications are transmitted from server to the clients in the high speed environment.

***8. Simple, Small and Familiar***

Java programs are written by the simple instructions and small codes,so it is familiar than other languages.

***9. Dynamic and Extensible***

Java hasthe dynamic feature to add methods, classes … etc to the existing ones. It is extensible with C and C++ language related functions and methods.

### *10. Architecture-neutral*

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

***Java Environment***

The entire Java software is divided into *J2SE* (Java 2 Standard Edition), *J2EE* (Java 2 Enterprise Edition) and *J2ME* (Java 2 Micro Edition). The entire Java environment is formed with several tools and these tools are available from the Java 2 Software Development Kit (*J2SDK or JDK*). They are

1. javac (Java Compiler) - Compiles java source code into byte code

E.g.:- javac Prog1.java

1. java (Java Interpreter)–Running java byte code application files from command prompt

E.g.:- java Prog1

1. appletviewer (for Java Applets) – Execute java applets

E.g.:-appletviewer App1.html

1. javap (Java Disassembler)–Display information about the methods, variables and parameters

present in aclass file

E.g.:- javap java.lang.Math

1. javah (for C-language header files)– Produces C header file from a java class file

E.g.:- javah Prog1

1. javadoc(for HTML Documents)– Generate HTML documentation for java program files

E.g.:- javadoc Prog1.java

1. jdb(Java Debugger)–Used for the program debugging

E.g.:- jdb Prog1

>run

***Character Set***

Character set is formed bythe alphabets, numeric and special symbols. All most all ASCII characters are available in the java character set.Java is a ***case sensitive*** programming language.

***Constants***

These are the fixed values. If a value is not changed during the execution of a program is called a constant. Constants are also called as ***literals***. In Java constants are mainly classified into 6 types and they are

1. Integer Constants
2. Floating-point Constants
3. Character Constants
4. String Constants
5. Boolean Constants
6. Back Slash Constants

***1. Integer constants***

If a numeric value must not have a decimal point"**.**",then it is called an integer constant.

E.g.:- 103, -256

***2. Floating-point Constants***

If a numeric value is must have a decimal point then it is called a floating-point constant.

E.g.:- 2.6, -0.08

***3. Character Constants***

If a single character is enclosed between the single quotes then it is called a character constant. All the backslash constants are also treated as character constants.

E.g.:-'a', '9', '+', '\n'

***4. String Constants***

If a single character or the number of characters enclosed between the double quotes then it is called a string constant. These are mainly used to write person names, places names, door numbers, dates, times … etc.

|  |
| --- |
| **Valid String Constants** |
| **"Rama"** |
| **"Vijayawada"** |
| **"52-1-4/2"** |
| **"12-Dec-2015"** |
| **"7:40AM"** |

***5. Boolean Constants***

These are the logical **true** and **false** values.

***6. Back Slash Constants***

Back slash constants are mainly used to control the program outputs into our own style. Each and every back slash constant is must start with the back slash (\). In the both C and C++ languagesthe back slash constants are also called as***escape sequence characters***.*Each escape sequence character is treated as a single character.*

|  |  |
| --- | --- |
| **Back Slash Constants**  **(or)**  **Escape SequenceCharacter** | **Meaning** |
| \b | Back Space |
| \f | Form Feed\* |
| \n | Next Line |
| \r | Carriage Return |
| \t | Tab Space |
| \\ | \ |
| \' | ' |
| \" | " |
| \0 | White Space |

***Identifiers***

Identifiers are names for the variables, arrays, classes, methods, objects, packages, interfaces, threads … etc. In Java the identifier names are must formed by following the below rules.

1. Identifier names must formed by only the alphabets, numeric, underscore (\_) and dollar ($) characters in any order.
2. In the identifier names the first character must not be a digit.
3. The maximum length of an identifier name is unlimited.

***Variables***

These are not the fixed values. A variable is a name of a memory location that holds data. A variable is a temporary storage that is used to store data temporarily.If a value is changed during the execution of a program is called a variable. Variable name is also an identifier that means the variable names are also must follow the identifier rules.

| **Valid Variable Names** | **Invalid variable Names** |
| --- | --- |
| A | 2a |
| s5 | S 5 |
| a2b | 2.a |
| T\_A | T.A |
| Basic\_Salary | Basic Salary |
| H\_R\_A | H.R.A |
| FName | Father Name |
| Sno | S.No |
| S\_no | S No |
| \_sno | 1\_sno |
| $no | 2$no |
| $\_no$ | $ no$ |

***Data types***

Before using the each and every variable in a Java program, ismust be declared by using adata type to allocate a memory location for holding the given type data. In Java the data types are mainly classified into 2 types. They are

I.Primitive data types.

II. Non-primitive data types.

***I. Primitive data types***

These are the standard data types in Java and these are again divided into 4 types. They are

1. Integer data types

2. Floating point data types

3. Character data type

4. Boolean data type

***1. Integer data types***

These are used to handle only the integer type values.Integer data types are expressed by 4 ways. They are

a.Byte data type

b.Short data type

c.Int data type

d.Long data type

***a. Byte data type***

This data type occupies only 1 byte (8 bits) space in memory. In this data type the minimum integer value is -128 and the maximum integer value is 127.

***b. Short data type***

This data type occupies 2 bytes(16 bits) space in memory. In this data type the minimum integer value is -32,768 and maximum integer value is +32,767.

***c. Int data type***

This data type occupies 4 bytes (32 bits) space in memory. In this data type the minimum integer value is-214,74,83,648 and the maximum integer value is 214,74,83,647.

***d. Long data type***

This data type occupies 8 bytes (64 bits) space in memory. In this data type the minimum integer value is-9,223,372,036,854,775,808 and the maximum integer value is 9,223,372,036,854,775,807. A long integer value is must end with ***'l'***or***'L'***.

***2. Floating point data types***

These are used to handle only the floating point values. Theseare expressed by2 ways and they are

a. Float data type

b. Double data type

***a. Float data type***

This data type occupies 4 bytes space in memory and it can maintain only the floating point values. A floating point value is must end with ***'f'***or***'F'***.

***b. Double data type***

This data type occupies 8 bytes space in memory and it can maintain the both integer and floating point values.

***3. Character data type***

This data type handles only the character type information and occupies 2 bytes space in memory. This data type is expressed by the key word ***char***.

***4. Boolean data type***

This data type can handle only the logical ***true*** or ***false*** value. This data type occupies 1 bit space in memory. This data type is expressed bythe key word ***boolean***.

***II. Non-primitive data types***

These are user defined data types in java and these are divided into 3 types. They are

1. Arrays

2. Classes

3. Interfaces

Non-primitive data types and their uses are explained in the next topics.

***Types of Expressions / Operators***

An expression is formed by the meaningful combination of operators and operands.

E.g.: a+b + is an operator, a, b are the operands.

5\*9 \* is an operator,5,9 are the operands.

a>b > is an operator, a, b are the operands

a=b = is an operator, a, b are the operands

a+b-c\*d/e +, -, \*, / are the operators, a, b, c, d, e are the operands

In java the operators are mainly classified into 3 categories. They are

**I. Unary Operators**

**II. Binary Operators**

**III. Ternary Operators**

***I. UnaryOperators***

This is an operator, which operates one operand.

E.g.: ++a

***II. BinaryOperators***

This is an operator, which operates on two operands.

E.g.: a+b

***III. TernaryOperators***

This is an operator, which operates on three operands.

E.g.: (a>b)?100:200

In java the expressions are mainly classified into 8 types. They are

**1. Arithmetic Expressions**

**2. Relational Expressions**

**3. Logical Expressions**

**4. Assignment Expressions**

**5. Increment and Decrement Expressions**

**6. Conditional Expressions**

**7. Bitwise Expressions**

**8. Special Expressions**

***1. Arithmetic Expressions***

Arithmetic expressions are formed by using the arithmetic operators of +,-,\*,/ and %. These are mainly used to perform calculations in the programs. The arithmetic operators and their meanings are explained below.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **+** | **Addition** |
| **-** | **Subtraction** |
| **\*** | **Multiplication** |
| **/** | **Division** |
| **%** | **Modules(Remainder Value After an Integer Division)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Expression** | **Result** | **Expression** | **Result** |
| **5+2** | **7** | **5%2** | **1** |
| **5-2** | **3** | **5.6%2.1** | **1.4** |
| **5\*2** | **10** | **5.6%2** | **1.6** |
| **5/2** | **2** | **5%2.1** | **0.8** |
| **5.0/2** | **2.5** | **10%2** | **0** |
| **5/2.0** | **2.5** | **10%20** | **10** |
| **5.0/2.0** | **2.5** | **% is Suitable for floating-point values in Java** | |

The output/result will depend upon the type of the operands in an expression.

|  |  |  |
| --- | --- | --- |
| **Operand1** | **Operand2** | **Output/Result** |
| **int** | **int** | **int** |
| **float** | **int** | **float** |
| **int** | **float** | **float** |
| **float** | **float** | **float** |

E.g.:1 a+b

E.g.:2 4-12/6+8\*9

***Hierarchy of an Expression***

Arithmetic expressions are executed in a pre-defined order and that order is called the hierarchy of an expression or the ***operator priority*** or the ***operator precedence***. When two or more operators in the expression have an equal priority, then the expression will be executed from left-to-right order. But in the unary operators the execution process is from right-to-left order. The arithmetic operators and their priorities are shown below

|  |  |
| --- | --- |
| **Operator** | **Priority** |
| **( )** | **1** |
| **/,\*,%** | **2** |
| **+,-** | **3** |

E.g.:1) 10-4\*2

10-8

2

E.g.: 2) 40-3\*2/(4-3)+8\*(1-2)

40-3\*2/1+8\*(-1)

40-6-8

26

***Rules for writing the Arithmetic Expressions***

1) Two operators must not appear in side by side

E.g.: 10-\*4 - is invalid

10\*(-4) - is valid

2) Every open bracket is must matched with the closing bracket

E.g.: ((8-4\*10) – is invalid

((8-4)\*10) – is valid

3) Denominator must not be zero

E.g.: 5/0 – is invalid

|  |  |
| --- | --- |
| **Algebraic Expression** | **Arithmetic Expression** |
| **a+b** | **a+b** |
| **a-b** | **a-b** |
| **ab** | **a\*b** |
|  | **a/b** |
| **a2** | **a\*a** |
|  | **(a+b)/(c+d)** |
| **b2-4ac** | **b\*b-4\*a\*c** |
|  | **u\*t+1/2\*a\*t\*t** |

***2. Relational Expressions***

Relational expressions are formed by using the relational operators of <, >, ==, <=, >=, !=. These are mainly used to write the conditions in programs and return result with either ***true*** or ***false***. The relational operators and their meanings are explained below

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **<** | **Less Than** |
| **>** | **Greater Than** |
| **==** | **Equal To** |
| **<=** | **Less Than or Equal to** |
| **>=** | **Greater Than or Equal to** |
| **!=** | **Not Equal To** |

E.g.: 1)10>4 - true 2) 5<4- false

3)15>=20- false 4) 5<=14 -true

5) 5==4 - false 6) 5!=4 -true

***3. Logical Expressions***

Logical expressions are formed by using the logical operators of&&, ||, !. These are mainly used to combine two relational expressions and then returnits total result with either ***true*** or ***false***. These are also used to write the conditions in programs. The logical operators and their meanings are shown below.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **&&** | **And** |
| **||** | **Or** |
| **!** | **Not** |

***&& Operator***

This operator combines two relational expressions and then return result as true when the given two relational expressions are true otherwise it can return result as false.

**Truth Table**

|  |  |  |
| --- | --- | --- |
| **Exp1** | **Exp2** | **Result** |
| **T** | **T** | **T** |
| **T** | **F** | **F** |
| **F** | **T** | **F** |
| **F** | **F** | **F** |

E.g.: 1) 9>6 && 6<41 - true

2) 12<5 && 16>11 - false

***|| Operator***

This operator combines two relational expressions and then return result as false when the given two relational expressions are false, otherwise it can return result as true.

**Truth Table**

|  |  |  |
| --- | --- | --- |
| **Exp1** | **Exp2** | **Result** |
| **T** | **T** | **T** |
| **T** | **F** | **T** |
| **F** | **T** | **T** |
| **F** | **F** | **F** |

E.g.: 1) 9>6 || 6<41 - true

2) 12<5 || 16>11 - true

***! operator***

This operator can be used to return negative result of the given relational expression

**Truth Table**

|  |  |
| --- | --- |
| **Exp** | **Result** |
| **T** | **F** |
| **F** | **T** |

E.g.: 1) !(9>6) - false

2) !(12<5) - true

***4. Assignment Expressions***

These are formed by using the assignment operators of =, +=, -=, \*=, /= & %=. These are mainly used to assign a value to an identifier. = is a ***simple assignment*** operator and the all other operators are the ***compound assignment*** operators.

Syntax is

identifier assignment\_operator expression;

E.g.: a=10;

b=a;

c=a+b;

a+=10; means a=a+10;

a-=10; means a=a-10;

a\*=10; means a=a\*10;

a/=10; means a=a/10;

a%=10; means a=a%10;

***5. Increment and Decrement (Unary) Expressions***

These are formed by using the increment and the decrement operators of ++ and --. In these expressions ++ is used as an increment operator and -- is used as a decrement operator. In these expressions the default or the fixed increment or decrement value is 1. These are also known as unary expressions.

1) ++a (a=a+1) Pre-Increment

2) a++ (a=a+1) Post-Increment

3) --a (a=a-1) Pre-Decrement

4) a-- (a=a-1) Post-Decrement

|  |  |  |  |
| --- | --- | --- | --- |
| **E.g.: 1** | **E.g.: 2** | **E.g.: 3** | **E.g.: 4** |
| **x=10;**  **++x;**  **System.out.println(x);**  **System.out.println(x);** | **x=10;**  **x++;**  **System.out.println(x);**  **System.out.println(x);** | **x=10;**  **System.out.println(++x);**  **System.out.println(x);** | **x=10;**  **System.out.println(x++);**  **System.out.println(x);** |
| **Output : 11**  **11** | **Output : 11**  **11** | **Output : 11**  **11** | **Output : 10**  **11** |

***6. Conditional (Ternary) Expressions***

These are formed by using the conditional operators of ? and :. These are also known as ***ternary*** expressions. These are mainly used to test the given condition, if it is true then it will execute the? following expression, otherwise it will execute the: following expression.

Syntax is

[identifier=] (condition)?exp1: exp2;

***Note : In this material square brackets in the syntax are treated as optional except in Arrays.***

E.g.: 1 x=(10>5)?20:40;

System.out.print(x)

Output: 20

E.g.: 2 System.out.print((8<2) ? 5:7);

Output: 7

***7. Bitwise Expressions***

Bitwise expressions are formed by using the bit wise operators of &, |, ~, ^, <<, >> and >>>. These are mainly used to convert the given numeric values into the binary form and then perform the specified operation after that it can return result into the normal form. The bitwise operators and their meanings are explained below

|  |  |
| --- | --- |
| **OPERATOR** | **MEANING** |
| **&** | **Bitwise AND** |
| **|** | **Bitwise OR** |
| **~** | **Bitwise NOT** |
| **^** | **Bitwise Exclusive OR** |
| **<<** | **Bitwise Left Shift** |
| **>>** | **Bitwise Right Shift** |
| **>>>** | **Bitwise Right Shift and**  **then prefill with zeros** |

***a. & (Bitwise AND) Operator***

It can return 1 when the given two expressions bits are 1's otherwise it can return 0 at the time of the bitwise operation.

E.g.:System.out.println(45 &39);

101101 (45)

& 100111 (39)

=100101 (37)

Output : 37

***b. | (Bitwise OR) Operator***

It can return 0 when the given two expressions bits are 0's otherwise it can return 1 at the time of the bitwise operation.

101101 (45)

| 100111 (39)

=101111 (47)

E.g.: System.out.println(45 | 39);

Output : 47

***c. ~ (Bitwise NOT) Operator***

It can return negative bits at the time of the bitwise operation.

~ 0111 (7)

=1000 (-8)

E.g.: System.out.println(~7);

Output : -8

Decimal Numbers and their binary codes are shown below

| **Decimal Number** | **Binary Code** |
| --- | --- |
| **0** | **0000 0000 0000 0000 0000 0000 0000 0000** |
| **1** | **0000 0000 0000 0000 0000 0000 0000 0001** |
| **2** | **0000 0000 0000 0000 0000 0000 0000 0010** |
| **3** | **0000 0000 0000 0000 0000 0000 0000 0011** |
| **4** | **0000 0000 0000 0000 0000 0000 0000 0100** |
| **5** | **0000 0000 0000 0000 0000 0000 0000 0101** |
| **-0** | **0000 0000 0000 0000 0000 0000 0000 0000** |
| **-1** | **1111 1111 1111 1111 1111 1111 1111 1111** |
| **-2** | **1111 1111 1111 1111 1111 1111 1111 1110** |
| **-3** | **1111 1111 1111 1111 1111 1111 1111 1101** |
| **-4** | **1111 1111 1111 1111 1111 1111 1111 1100** |
| **-5** | **1111 1111 1111 1111 1111 1111 1111 1011** |
| **-6** | **1111 1111 1111 1111 1111 1111 1111 1010** |

***d. ^ (Bitwise Exclusive OR) Operator***

It can return 0 when the given two expression bits are same otherwise it can return 1 at the time of the bitwise operation.

101101 (45)

^ 100111 (39)

=001010(10)

E.g.: System.out.println(45 ^ 39);

Output : 10

***e. << (Bitwise Left Shift) Operator***

It can shift specified number of bits into left side and then the right side bits are filled by zeros at the time of the bitwise operation.

E.g.:System.out.println(45<<2);

101101 (45)

<<2

=10110100(180)

Output : 180

***f. >> (Bitwise Right Shift) Operator***

It can shift specified number of bits into right side at the time of the bitwise operation.

101101 (45)

>>2

=1011(11)

E.g.:System.out.println(45>>2);

Output : 11

***g. >>> (Bitwise Right Shift and then prefill with zeros) Operator***

It can shift specified number of bits into right side and then the left side bits are filled by zeros at the time of the bitwise operation.

101101 (45)

>>>2

=001011(11)

E.g.:System.out.println(45>>>2);

Output : 11

***8. Special Expressions***

These are mainly used to perform some special kind of operations and formed with the *instanceof* and *.(dot)* operators. These are mainly used to handle class objects and members.

***a. instanceof Operator***

It is used test whether a specified object is related to the specified class or not. It can return true when the specified object is related to the specified class otherwise it can return false.

Syntax is

object instanceof class

E.g.:

x instanceof Rama

***b. .(Dot) Operator***

It is used to access members of a class by using the specified object name. The object name and its members are separated by . (dot).

Syntax is

Object.member

E.g.:

x.a

x.sum()

# *Keywords*

These are the pre-defined words in java and these are also known as ***reserved words***. Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords const and goto are reserved, even though they are not currently used. true, false and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| assert\*\*\* | default | goto\* | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum\*\*\*\* | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp\*\* | volatile |
| const\* | float | native | super | while |

***HUNGARIAN NOTATION(Naming Conventions):***

Hungarian Notation is the naming convention followed by SUN (Stanford UniversityNetwork) micro systems to develop their predefined classes, interfaces, methods and datamembers.

***1) Hungarian rule for CLASS or INTERFACE***

If a class or interface contains more than one word then we must write all the firstletters of word must be capital.

E.g.:

System, NumberFormatException, ArrayIndexOutOfBoundsException

***2) Hungarian rule for METHOD***

If a method name contains more than one word then first word first letter is small and rest ofthe words first letters must be capital and all the others are small.

E.g.:

println(), readLine(), actionPerformed(), adjustmentValueChanged()

***3) Hungarian rule for DATA MEMBERS***

All the data members in the predefined classes and interfaces must be represented by thecapital letters. In the user defined classes the identifier names are into your own style.

E.g.:

E,PI, MAX\_VALUE, MIN\_VALUE

***Structure of a Java Program***

In Java the programs are formed with the below structure.

1. Documentation section.
2. Package statement.
3. Import statement(s).
4. Interface statement(s).
5. Class definition(s).
6. Main method class.
7. ***Documentation Section***

It is used to write comment lines inthe Java program. These lines are the non-executable lines and used only for the documentation purpose. These lines may write at anywhere in the entire program. These lines are written by using 'C' like comments with /\* and \*/ or 'C++' like comments with // and additionally Java handles a new format start with /\*\* and end with \*/. This section is suggestible.

E.g.:-1

/\* sum of 10 and 20 \*/

E.g.:-2:

// product of any 3 numbers

E.g.:-3:

/\*\* simple Interest\*/

1. ***Package statement***

A package is formed with the number of classes and the each class is formed with number of members. A package is used to handle number of classes with a single reference name. The package related each and every thing is explained in the next topics. This part is optional.

1. ***Import statements***

Import statements are mainly used to include the specified package related class or number of classes into the current program. This part is also optional.

Syntax is

import package-name.class-name;

E.g.:-1: import java.lang.\*;

E.g.:-2: import pack1.Calculator;

1. ***Interface statements***

These are useful to perform multiple inheritances in java. This part is also optional and it is explained in the next topics.

1. ***Class definitions***

A class is formed with the number of members. In a classthe variables are called *instance variables* and functions are called*methods*. Every Java program may contain number of classes. The classes and their related all concepts are explained in the next topics. This part is also optional.

1. ***Main() method class***

Every Java application program is must be formed with the main method class. Without the main method class,that is not treated asa valid Java application program. At the time of the program compilation &execution the process is start from the main() method.If we want to define a main method class in the Java application program then we must follow the below syntax and its example.

Syntax is

class class\_name

{

public static void main(String[ ] args)

{

Identifier declarations& statement(s);

}

}

In the above syntax the class\_name is must be a valid identifier name and it must be the program file name when the class access modifier is public. Public members are accessed at anywhere in the entireJava. So, in Java the main method must be public. The key word static is used to directly accessthe class members without the objectreference. For the execution purpose, the main method must be static. Generally the return data type may be any validjava data type or void. Void must not return a value from method to its calling area. But in Java the main method return data type must bevoid.

E.g.:

class Prog1

{ public static void main(**S**tring[ ] args)

{ int a,b;

a=10;

b=20;

int c=a+b;

**S**ystem.out.println("sum="+c);

}

}

***Variable Declarations***

Before using the variables in a Java program, first we must declare those variables by using the data type to allocate memory locations for handling data. In Java the variables may be declared at anywhere in the entire program.

Syntax is

datatype variable-1[=expr],variable-2[=expr],….;

E.g.:-1

int x,y;

E.g.:-2

double a=5.7,b=6.0,c=a/b;

***System.out.print() method***

This is an output method and this method is used to display the given data on the standard output device and then places control beside the output.

Syntax is

System.out.print(item1+item2+……..);

*In the above syntax the item1 and item2 may be variables or constants or expressions.*

E.g.:-1

System.out.print("NICE");

E.g.:-2

System.out.print("sum="+c);

***System.out.println() method***

This is also an output method and this method is used to display the given data on the standard output device and then places control into the next line.

Syntax is

System.out.println(item1+item2+……..);

E.g.:-1

System.out.println("NICE");

E.g.:-2

System.out.println("sum="+c);

***MATH FUNCTIONS***

Math functions are the built-in functions in java. These are placed in the built-in class ***Math***. In java the Math class is placed in the j***ava.lang*** package. ***Java.lang is the default package in Java. That means you need not import it in the programs.***If we want to call any one of the math function, then we must give ***"Math."*** before the every function name, because the all members in the Math class are ***static***. Some of the useful Math class functions are explained below.

***1. abs( ) function***

abs stands for absolute. It converts the given negative numeric value into the positive numeric value. If the given value is positive then it can return the same value.

Syntax is

Math.abs(n)

E.g.:

System.out.println(Math.abs(-10.0));

Output:

10.0

***2. sqrt() function***

sqrt stands for square root. It can return square root value of the given numeric value.

Syntax is

Math.sqrt(n)

E.g.:

System.out.println(Math.sqrt(100.0));

Output:

10.0

***3. pow() function***

pow stands for power. It can return the result of a value to the power of another value.

Syntax is

Math.pow(number,power)

E.g.:

System.out.println(Math.pow(5.0,2.0));

Output:

25.0

***4. sin() function***

It can returntrigonometrical sine value of the given numeric value in radians.

Syntax is

Math.sin(n)

E.g.:

System.out.println(Math.sin(30.0));

Output:

-0.9880

***5. cos() function***

It canreturn trigonometricalcosine value of the given numeric value in radians.

Syntax is

Math.cos(n)

E.g.:

System.out.println(Math.cos(60.0));

Output:

-0.9524

***6. tan() function***

It can return trigonometrical tangent value of the given numeric value in radians.

Syntax is

Math.tan(n)

E.g.:

System.out.println(Math.tan(45.0));

Output:

1.619775

***7. ceil() function***

ceil stands for ceiling. It can return next integer value of the given floating point numeric value. If the given value is an integer then it can return the same as value.

Syntax is

Math.ceil(n)

E.g.:

System.out.println(Math.ceil(2.4));

Output:

3.0

***8. floor() function***

floor stands for flooring. It can return previous integer value of the given floating point numeric value. If the given value is an integer then it can return the same value.

Syntax is

Math.floor(n)

E.g.:

System.out.println(Math.floor(2.4));

Output:

2.0

***9. log() function***

It can return natural logarithmic value of the given numeric value with base e(2.718281828459045).

Syntax is

Math.log(n)

E.g.:

System.out.println(Math.log(10.0));

Output:

2.3025

***10. log10() function***

It can return natural logarithmic value of the given numeric value with base 10.

Syntax is

Math.log10(n)

E.g.:

System.out.println(Math.log10(10.0));

Output:

1.0

***11. exp() function***

exp stands for exponent. It can return result of an exponent value 2.718281828459045 to the power of specified value.

Syntax is

Math.exp(n)

E.g.:

System.out.println(Math.exp(1.0));

Output:

2.718281828459045

***12. max() function***

It can return maximum value from the given two numeric values.

Syntax is

Math.max(n1,n2)

E.g.:

System.out.println(Math.max(5,9));

Output:

9­

***13. min() function***

It can return minimum value from the given two numeric values.

Syntax is

Math.min(n1,n2)

E.g.:

System.out.println(Math.min(5,9));

Output:

5

***14. random() function***

It can return zigzagged numbers.

Syntax is

Math.random()

E.g.:

System.out.println(Math.random());

Output:

0.6543210

***15. round() function***

It can return nearest integer value of the given floating point numeric value.

Syntax is

Math.round(n)

E.g.:

System.out.println(Math.round(5.7));

Output:

6

***16. toDegrees() function***

It converts the given radians value into degrees.

Syntax is

Math.toDegrees(n)

E.g.:

System.out.println(Math.toDegrees(22.0/7.0));

Output:

180

***17. toRadians() function***

It converts the given degrees value into radians.

Syntax is

Math.toRadians(n)

E.g.:

System.out.println(Math.toRadians(180.0));

Output:

3.141592

***18. signum() function***

It can return 1 for positive numbers, -1 for negative numbers and 0 for the 0 value.

Syntax is

Math.signum(n)

E.g.:

System.out.println(Math.signum(-10));

Output:-

-1.0

***19. PI***

It can return mathematical PI value.

Syntax is

Math.PI

E.g.:

System.out.println(Math.PI);

Output:-

3.141592

***20. E***

It can return exponent value.

Syntax is

Math.E

E.g.:

System.out.println(Math.E);

Output:-

2.718281828459045

***Type Casting***

Type casting means to convert a value from one data type to another. This concept is also called the ***type conversion*** or ***type coersion***.

Syntax is

(conversion-type) expression

E.g.:

int a, b;

float c;

a=5;

b=2;

c= (float)a/(float)b;

System.out.print(c);

Output:-

2.5

***Recursion***

When a method is called by it-self is known as recursion. This concept is very useful to simplify the program and then increases the program's efficiency. In this process we must specify an end-point, otherwise it can generate an infinite looping.

E.g.:

class Prog1

{ public static void main(String[] args)

{ int f;

f=fact(5);

System.out.print("Factorial of 5 is : "+f);

}

static int fact(int x)

{ if(x==0) return 1;

else return x\*fact(x-1);

}

}

Output:-

Factorial of 5 is : 120

***Command Line Arguments***

When we pass values from command prompt into a java program, then those values are called command line arguments. These arguments are read only by the main method. In the main method the arguments are must be string array type.

***WRAPPER CLASSES***

Wrapper classes are used to convert object values from one data type to another. Primitive data types are the standard data types in Java. But, the primitive data types are not used to convert objecttypes into the standard data types. In java the wrapper classes are placed in the ***java.lang*** package. The primitive data types and their related wrapper classes are shown below.

|  |  |
| --- | --- |
| **Primitive Data Type** | **Wrapper Class** |
| **boolean**  **char**  **int**  **float**  **double**  **long**  **byte**  **short** | **Boolean**  **Character**  **Integer**  **Float**  **Double**  **Long**  **Byte**  **Short** |

***String to numeric type conversion methods***

If we want to convert a stringtype value (object) into another (primitive)data type then we must follow the belowmethods.

***1. parseByte() method***

It converts a string into the bytedata type. It is placed in the wrapper class ***Byte.***

Syntax is

Byte.parseByte(string)

E.g.:

byte a;

a=Byte.parseByte(args[0]);

***2. parseShort() method***

It convertsa string into the shortdata type. It is placed in the wrapper class ***Short.***

Syntax is

Short.parseShort(string)

E.g.:

short a;

a=Short.parseShort(args[0]);

***3. parseInt() method***

It convertsa string into the integer data type. It is placed in the wrapper class ***Integer.***

Syntax is

Integer.parseInt(string)

E.g.:

int a;

a=Integer.parseInt(args[0]);

***4. parseLong() method***

It convertsa string into the longdata type. It is placed in the wrapper class ***Long.***

Syntax is

Long.parseLong(string)

E.g.:

long a;

a=Long.parseLong(args[0]);

***5. parseFloat() method***

It convertsa string into the floatdata type. It is placed in the wrapper class ***Float.***

Syntax is

Float.parseFloat(string)

E.g.:

float x;

x=Float.parseFloat(args[0]);

***6. parseDouble() method***

It convertsa string into the doubledata type. It is placed in the wrapper class ***Double.***

Syntax is

Double.parseDouble(string)

E.g.:

double x;

x=Double.parseDouble(args[0]);

***7. parseBoolean() method***

It convertsa string into the booleandata type. It is placed in the wrapper class ***Boolean.***

Syntax is

Boolean.parseBoolean(string)

E.g.:

boolean x;

x=Boolean.parseBoolean(args[0]);

***DataInputStream class***

This is an input stream related class in Java and it is place in the ***java.io*** package. This class maintainsthe input related java methods. If we want to create an object to the DataInputStream class to handle its methods, then we must follow the below syntax and its example.

Syntax is

DataInputStream object = new DataInputStream(input\_device);

E.g.:

DataInputStream dis = new DataInputStream(System.in);

If we want to use this class in a java program then that program must imports the *java.io.DataInputStream* class.

***readLine() method***

This method is placed in the DataInputStream class and this method is used to read a string from the input device.

Syntax is

inputstream\_object.readLine()

E.g.:

String x;

x=dis.readLine();

If we want to use this method in a java program then the method in a program must throws an *IOException*, because the readLine() method internally throws an IOException.IOException is a class and it is placed in the ***java.io*** package. If we want to use the IOException class in a java program, then we must import *java.io.IOException* class into the current program.

***BufferedReader class***

This is an input stream related class in Java and it is placed in the ***java.io*** package. This class maintainsthe input related java methods. If we want to create an object to the BufferedReader class to handle its methods, then we must follow the below syntax and its example.

Syntax is

BufferedReader object = new BufferedReader(new InputStreamReader(input\_device));

E.g.:

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

If we want to use this class in a java program then that program must import the *java.io.BufferedReader* and*java.io.InputStreamReader* classes.

***read() method***

This method is placed in the BuffredReader class and this method is used to read a single characterfrom the input device.

Syntax is

inputstream\_object.read()

E.g.:

char x;

x=br.read();

If we want to use this method in a java program then the method of a program must throws an *IOException*, because the read() method internally throws an IOException class.

***readLine() method***

This method is placed in the BuffredReader class and this method is used to read a string from the standard input device.

Syntax is

inputstream\_object.readLine()

E.g.:

String x;

x=br.readLine();

If we want to use this method in a java program then the method of a program must throws an *IOException*, because the readLine() method internally throws an IOException class.

***Scanner Class***

It is used to read data from the input and it can breaks data into tokens by the specified delimiter. It is placed in the ***java.util*** package. If we want to create an object, then we must follow the below syntax and its example

Syntax is

Scanner object=new Scanner(input\_device);

E.g.:

Scanner sc=new Scanner(System.in);

Scanner class has several methods to read data. They are

***next() method***

It reads the next token from the input as a string type.

***nextLine() method***

It reads the nextline from the input as a string type.

***nextByte() method***

It reads the next token from the input as byte type.

***nextShort() method***

It reads the next token from the input as short type.

***nextInt() method***

It reads the next token from the input as integer type.

***nextLong() method***

It reads the next token from the input as long type.

***nextFloat() method***

It reads the next token from the input as float type.

***nextDouble() method***

It reads the next token from the input as double type.

***if statement***

This is a conditional control statement and this statement is also called a ***branching statement***. This statement is used to take a decision by using the given condition. That means if the given condition is true then it will execute only the condition following statement, otherwise it will execute only the else following statement.

Syntax is

if (condition) statement;

[else statement;]

E.g.:

if (a>b) System.out.println("a is big");

else System.out.println("b is big");

***while statement***

This is a conditional control statement and this statement is also called a***looping statement***. This statement is used to repeat an execution process of the given statement until the given condition becomes false.

Syntax is

while(condition)

statement;

E.g.:

x=1;

while(x<=5)

{ System.out.println(x);

x=x+1;

}

Output:

1 2 34 5

***do-while statement***

This is also a conditional control statement and this statement is also called a ***looping statement***. This statement is used to repeat the do and while between statement until the given condition becomes false. But, this statement executes the given statement at least one time.

Syntax is

do

statement;

while(condition);

E.g.:

x=1;

do

{ System.out.println(x);

x=x+1;

}while(x<=5);

Output: -

12345

***for statement***

This is also a conditional control statement and this statement is also called a ***looping statement***. This statement is used to repeat the given statement until the given condition becomes false.

Syntax is

for(exp1,exp2,..;exp3;exp4,exp5…)

statement;

In the above syntax exp1 and exp2 are the initial expressions, exp3 is a conditional expression, exp4 and exp5 are the increment or decrement expressions.

For statement is executed by following the below steps

*Step1: Executes the given all initial expressions*

*Step2: Test the given condition. If it is true then control changes to the next step otherwise stops the for statement*

*Step3: Execute the given statement*

*Step4: Execute the given all increment (or) decrement expressions*

*Step5: Repeat step2 again*

E.g.:

for(x=1;x<=5;++x)

System.out.println(x);

Output:

12345

***Break Statement***

This statement terminates the current loop and then changes control to the loop following statement. It is used only in the while, do-while, for and switch statements.

***Continue Statement***

This statement changes control to the current loop beginning.It is used only in the while, do-while, and for statements.

***Nested Loops***

When a looping structure is placed inside another looping structure, then it is called a nested looping structure. In this concept the first loop is called an *outer loop* and it's inside loop is called an *inner loop*. When an inner loop condition is false, then each time the control will transferred to its outer loop. When an outer loop condition is true then each time the control will transfer to its inner loop and it will be started newly.

***Switch Statement***

This is also a conditional control statement and this statement is also called a ***multiple branching statement***. This statement is used to select a single case from the given number of cases by using the given expression value is matched to the case expression. When there is no case is matched to the given expression then it will execute the default case.

Syntax is

switch(expression)

{

case exp1: statement(s);

break;

case exp2: statement(s);

break;

case exp3: statement(s);

break;

**:**

**:**

**:**

**:**

**:**

default: statement(s);

break;

}

E.g.:

switch(x)

{ case 1: System.out.println("Rama");

break;

case 2:System.out.println("Sita");

break;

case 3: System.out.println("NICE");

break;

default:System.out.println("Wrong choice");

break;

}

***ARRAYS***

Generally a single variable is used to handle only a single value at a time. But, if we want to handle more than one value with the single identifier name then we must use arrays. In an array the each allocated memory location is called an *element*. The each and every element is identified by a value called the *subscript or index*. The subscript is must enclosed between the separate pair of square brackets. In Java also the 'n' elements array is divided into 'n+1' elements and the last element maintains a null character to identify an array ending. In an array the all elements should maintain the same data type. In Java arrays are mainly classified into two types and they are

1. Single Dimensional Arrays

2. Multi-Dimensional Arrays

***Single Dimensional Arrays***

When an arrayhaving only a single subscript then that type of array is called a single dimensional array or a one dimensional array. If we want to handle an array in java, we must follow the below three steps.

Step1: Declare an array

Step2: Create memory locations to the array elements

Step3: Initialize the array elements

***Step 1. Declare an Array***

If we want to handle an array in a Java program, then first we must declare an array by using the below syntax and its example.

Syntax is

data\_type identifier[];

(or)

data\_type[] identifier;

E.g.:

int x[];

(or)

int[] x;

After the above declaration, a null pointer object (reference) is created to handle the array elements.

***Step 2. Create memory locations to the array elements***

If we want to create memory locations to the array elements,then we must follow the below syntax and its example.

Syntax is

array\_identifier= new data\_type[size];

E.g.:

x=new int[5];

After the above example it can allocate 5 different integer type memory locations as array elements with the names x[0], x[1], x[2], x[3] and x[4].

The above array declaration and creation of memory locations to the array elements,follows a direct single syntax and that is shown below.

*Syntax is*

data\_type identifier[]=new data\_type[size];

(or)

data\_type[] identifier=new data\_type[size];

*E.g.:*

int x[]= new int[5];

(or)

Int[] x= new int[5];

***Step 3. Initialize the array elements***

After the above 2 steps, if we want to initialize the array elements with the specified values then we must follow the below two different syntaxes and their examples.

Syntax1 is

Array-element= expression;

E.g.:

x[0]=10;

x[1]=20;

x[2]=x[0]+x[1];

x[3]=x[2];

Syntax2 is

datatype identifier[]={value1,value2,…..};

E.g.:

int x[]={1,5,9,4,12};

After the above example it can allocate 5 different elementswith the names x[0], x[1], x[2], x[3] and x[4]. The first element x[0] is initialized with '1',the second element x[1] is initialized with 5, the third element x[2] is initialized with 9, the fourth element x[3] is initialized with 4 and the fifth element x[4] is initialized with 12.

***Multi-Dimensional Arrays***

When an array having more than one subscript then it is called a multi-dimensional array,when an array having two subscripts, then it is called a two-dimensional array,when an array having three subscripts then it is called a three-dimensional array … etc. In java two dimensional arrays are mainly used in the matrix related operations. In the multidimensional arrays the each and every subscript is must enclosed between the separate pair of square brackets. If we want to declare a multidimensional array with a single step then we must follow the below syntax and its example.

Syntax is

data\_type identifier[][][ ] … =new datatype[size1][size2][size3] …;

E.g.:

int x[][]=new int[2][3];

After the above example it can allocate 6 elements in 2 rows and 3 columns,in the below format.

x[0][0] x[0][1] x[0][2]

x[1][0] x[1][1] x[1][2]

***CHARACTER ARRAYS***

When an array is declared by the char data type then it is called a character array. In a character array each element handlesonly a single character. If we want to declare a character array then we must follow the below syntax and its example.

Syntax is

Char[ ] identifier = new char[size];

E.g.:

Char[ ] x = new char[5];

After the above example it can allocate 5 elements with the names x[0], x[1], x[2], x[3] and x[4]. The each element canbe used to handle only a single character.

***STRINGS***

In Java the character arrays and strings are different than each other. String is a built-in class in java and it is placed in the ***java.lang***package. String is an instantiated object of the String class. If we want to declare thestring type identifier (create an object to the string class)in java then we must follow the below syntax and its example

Syntax is

String identifier (or) object; (or)String object=new String();

E.g.:

String s;(or)String s=new String();

After the above example it creates a string object with the name 's'is used to handle a string with unlimited length.

***String Arrays***

Strings are also used to create array objects to handle several strings with the same name. If we want to declare a string array then we must follow the below syntax and its example

Syntax is

String object[ ] = new String[size];

E.g.:

String x[ ] = new String[5] ;

After the above example it can allocate 5 different string type elements with the names x[0], x[1], x[2], x[3] and x[4]. The each string type element is used to handle a string.

***STRING FUNCTIONS***

These are mainly used to perform operations by using the given string type values. The string functions are placed in the String class and the string class is placed in the ***java.lang***package. Some of the useful string functions are explained below.

***1. toUpperCase() function***

It converts the given string into the upper case. If the given string is in upper case then it returns the same string.

Syntax is

string.toUpperCase()

E.g.:

String s= "NICE";

System.out.println(s.toUpperCase());

O/P:

NICE

***2. toLowerCase() function***

It converts the given string into the lower case. If the given string is in lower case then it returns the same string.

Syntax is

string.toLowerCase()

E.g.:

String s= "NICE";

System.out.println(s.toLowerCase());

O/P:

nice

***3. substring() function***

It returns part of the string from the specified startingcharacter position up to the previous character of the specified ending character position. In strings the character count is started from '0'(zero).

Syntax is

string.substring(start\_char\_pos,end\_char\_pos)

E.g.:

String s= "computers"

System.out.println(s.substring(3,6));

O/P:

put

***4. length() function***

It returns total number of character count from the given string.

Syntax is

string.length()

E.g.:

String s= "computers";

System.out.println(s.length());

O/P:

9

***5. valueOf() function***

It converts the numeric type data into the string type.

Syntax is

valueOf(num)

E.g.:

Int x= 56;

System.out.println(String.valueOf(x));

O/P:

"56"

***6. concat() function***

concat stands for concatenate. It is used to add two strings and returns total string for result.

Syntax is

string.concat(string)

E.g.:

String s= "SAI";

System.out.println(s.concat("RAM"));

O/P:

SAIRAM

***7.charAt() function***

It returns a character of the specified position from the given string.

Syntax is

string.charAt(n);

E.g.:

String s= "computers";

System.out.println(s.charAt(3));

O/P:

p

***8.compareTo()function***

It is used to compare the given two strings and returnszero when the given two strings are equal.

Syntax is

string.compareTo(string)

E.g.:

String s= "RAMA";

System.out.println(s.compareTo("rama"));

O/P:

-32

***9. compareToIgnoreCase()function***

It is used to compare the given two strings and returnszero when the given two strings are equal. But, this function ignores case sensitive at the time of testing.

Syntax is

string.compareToIgnoreCase (string)

E.g.:

String s= "RAMA";

System.out.println(s.compareToIgnoreCase ("rama"));

O/P:

0

***10. equals() function***

It is used to test the given two strings and return true when the given two strings are equal, otherwise it returns false.

Syntax is

string1.equals(string2)

E.g.:

String s="NICE";

System.out.println(s.equals("nice"));

O/P:

false

***11. equalsIgnoreCase()function***

It is used to test the given two strings and returnstrue when the given two strings are equal, otherwise it returns false. But, this function ignores case sensitive at the time of testing.

Syntax is

string1.equalsIgnoreCase(string2)

E.g.:

String s= "RAMA";

System.out.println(s.equalIgnoreCase("rama"));

O/P:

true

***12. replace() function***

It replaces a specified character with the another character in the string.

Syntax is

string.replace(search\_char,replace\_char);

E.g.:

String s= "RAMA";

System.out.println(s.replace('A' , 'X'));

O/P:

RXMX

***StringBuffer Class***

It is also same like as a string class. It is mainly used to handle strings in the created objects. String and StringBuffer are not the same. If we want to create an object for the string buffer class then we must follow the below syntax and its examples.

Syntax is

StringBuffer object = new StringBuffer(string);

E.g.:

StringBuffer s = new StringBuffer("NICE");

After the above example a string buffer object is created with***S*** and it can be assigned by the string "NICE".

***StringBuffer Functions***

***1. length()function***

It returns total number of characters count from the given string buffer.

Syntax is

stringbuffer\_object.length()

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

System.out.print(s.length());

O/P:

9

***2. deleteCharAt()function***

It removes a specified positioned character from the string buffer object.

Syntax is

stringbuffer\_object.deleteCharAt(num)

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

s.deleteCharAt(3);

System.out.print(s);

O/P:

COMUTERS

***3. insert()function***

This function insertsa specified character at a specified position into the string buffer.

Syntax is

stringbuffer\_object.insert(n,char)

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

s.insert(4, 'x');

System.out.print(s);

O/P:

COMPxUTERS

***4. append() function***

It can be used to add a string into the specified existing string buffer object.

Syntax is

Stringbuffer\_object.append(string)

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

s.append("rama");

System.out.print(s);

O/P:

COMPUTERrama

***5. charAt()function***

It returns a specified positioned character from the string buffer object.

Syntax is

stringbuffer object.charAt(n)

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

System.out.print(s.charAt(4));

O/P:

U

***6. reverse()function***

It can return reverse string form the given string buffer.

Syntax is

stringbuffer\_object.reverse()

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

s.reverse()

System.out.print(s);

O/P:

SRETUPMOC

***7. substring()function***

It returns part of the string from the specified startingcharacter position up to the previous character of the specified ending character position. In stringbufferalso the character count is started from '0'(zero).

Syntax is

stringbuffer\_object.substring(n,n1)

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

s.substring(3,6)

System.out.print(s);

O/P:-

PUT

***8. toString()function***

It converts stringbuffer into the string type.

Syntax is

stringbuffer\_object.toString()

E.g.:

StringBuffer s = new StringBuffer("COMPUTERS");

String s1=s.toString()

System.out.print(s1);

O/P:-

COMPUTERS

***What is OOPS***

OOPS stands for Object Oriented Programming System. It is a methodology or a paradigm to design program using classes and objects. It can simplify the process of software development and maintenance. A class is formed by the several members called instance variables and methods. Object is a small entity and it has state and behavior.

***Benefits OOPS***

* OOPS facilitates creating reusable code that can eventually save a lot of work
* Data security is enforced
* Inheritance saves time
* Inheritance permits to derive new classes from old ones
* User defined data types can be easily constructed
* A feature called polymorphism permits to create multiple definitions for functions
* Large complexity in the software development can be easily managed

***Classes***

Everything in Java is related to the object oriented programs and the object oriented programs are controlled by using the classes. Class is a user defined data type. Class is mainly used to maintain several members. The members may be the identifiers or functions. In a class the identifiers are called the ***data members or instance variablesor Properties*** and the functions are called the ***member functions or methods***. If we want to declare a class in Java then we must follow the below syntax and its example.

Syntax is

class class-name [extends super-class-name]

{ Identifier declarations;

Method declarations;

}

In the above syntax the class name is must be a valid identifier. The super class name is the base class (old class) name used in the inheritance. An identifier declaration means the variable and array like declarations. The method declaration meansthe function definitions.

E.g.:class Rama

{ int a,b,c;

void sum()

{a=10;

b=20;

c=a+b;

System.out.println("Sum="+c);

}

}

In the above example Rama is the class name. In the class Rama the members are a, b, c and sum(). But a, b and c are called instance variables and the sum() is called a method.

After the above class declaration it should not directly allocate memory locations to the class members. If we want to allocate memory locations to the class members, then first we must create an object tothe class by following the below syntax and its example.

Syntax is

class-name object-name;

E.g.:

Rama r;

After the above object creation it can create only a pointer reference and it should not instantiate memory locations to the object. If we want to create memory location to the above object,then we must follow the below syntax and its example.

Syntax is

object-name= new class-name();

E.g.:

r = new Rama();

The object creation and its instantiation followa direct single syntax and that is shown below.

Syntax is

class-name object = new class-name();

E.g.:

Rama r = new Rama();

If we want to access the above created object members then we must follow the below syntax and its example.

Syntax is

object-name.member

E.g.:-1

r.a

E.g.:-2

r.sum()

***A Method With Parameters***

When a method in a class having parameters,then that method is called a parameterized method. Parameters are also called arguments. Each parameter is must be declared with itsown data type. Parameters are mainly used to accept values from method calling area into the method.

***Encapsulation***

When the data members (instance variables) and member functions (methods) are placed in the single entity of a class,then it is called***encapsulation***.

***getter() and setter() methods***

The getter() and setter() methods are mainly used to update and retrieve values of the varaibles in a class by the secured manner. These are very useful to access the private variables of a class only from the getter() and setter() methods.

***Abstraction***

Abstraction means to enableaccessibilityto the essential features and disable accessibilityto the unnecessaryfeatures of a class.

In this feature the private members are not accessed at outside the class and only the public members are accessed at anywhere in java.

***Access Specifiers***

Java has 4 types of access specifiers to specify restrictions to the members of a class. They are

1. private

2. public

3. protected

4. No Modifier

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***PRIVATE*** | ***NO MODIFIER*** | ***PROTECTED*** | ***PUBLIC*** |
| ***Same Class*** | *Yes* | *Yes* | *Yes* | *Yes* |
| ***Same Package Subclass*** | *No* | *Yes* | *Yes* | *Yes* |
| ***Same Package Non-Subclass*** | *No* | *Yes* | *Yes* | *Yes* |
| ***Different Package Subclass*** | *No* | *No* | *Yes* | *Yes* |
| ***Different Package Non-Subclass*** | *No* | *No* | *No* | *Yes* |

***Private***

Private members are accessed only within the current class.

***Public***

Public members are accessed at anywhere in the entire java.

***Protected***

Protected members are accessed at anywhere in the entire java program except the different package related non-subclass.

***No Modifier***

The no-modifier access specifier is used to access the specified members only in the current class, same package subclass and the same package non-subclasses. These members are not accessed at the different package related subclass and the non-subclasses.

***Array Objects***

If we want to create array objects in java, then we must follow the below syntax and its example.

Syntax is

class-name[ ] object-name= new class-name[size];

E.g.:

Rama[ ] r = new Rama[5];

After the above example, it can create 5 different pointer objects for the class *rama* with the references r[0], r[1], r[2], r[3] and r[4]. After that we must initialize each object by following the below syntax and is example.

Syntax is

object-name= new class-name();

E.g.:

r[0] = new Rama()

***Constructors***

* + - * Constructor is a method (member function).
      * It has same name of the class name
      * Constructors must not have the return data type
      * Constructors are automatically executed (invoked) at the time of an object is created to the class
      * Constructors are mainly used to perform initial operations

In Java the constructors are mainly classified into 2 types and they are

1. Default Constructors
2. ParameterizedConstructors
3. Copy Constructors

***1. Default Constructors***

When a constructor should not have parameters then that constructor is called a default constructor.

***2. Parameterized Constructors***

When a constructor should have parameters then that constructor is called a parameterized constructor.

***3. Copy Constructors***

When a constructor which takes the object of the same type as input is known ascopy constructor.

***Inheritance***

Reusability of existing class properties into the other class is called ***inheritance*** *(or) to derive new class from the old class is* called ***inheritance***. In this concept the old class is called the base class or ***super class*** or parent class and the new class is called the derived class or ***sub class*** or child class. This concept is very useful to simplify the programs when two or more classes can havesome common properties. In Java the inheritance is mainly classified into 5 types and they are

1. Single Inheritance

2. Multi-level Inheritance

3. Hierarchical Inheritance

4. Multiple Inheritance

5. Hybrid Inheritance

***Single Inheritance***

When a single super class is inherited into the single sub class then it is called single inheritance.

***Multi-level Inheritance***

When a super class is inherited into the sub class and that newly formed sub class is inherited intoanother sub class, this process maycontinue upto the number of levels. This type of inheritance is called multilevel inheritance.

***Hierarchical Inheritance***

When a super class is inherited into two or more sub classes then it is called hierarchical inheritance.

***Multiple Inheritance***

When two or more super classes are inherited into a single sub class then it is called multiple inheritance. Java is not support for the multiple inheritance with classes. But in Java the ***interfaces*** are used to perform the multiple inheritance.

**A** **B**

**C**

***Hybrid Inheritance***

It is formed with the combination of hierarchical and multiple inheritances. In Java the ***interfaces*** are used to perform the hybrid inheritance.

**D**

If we want to derive a sub class in java then we must follow the below syntax and its example.

Syntax is

class sub-class-name extends super-class-name

{ identifier declarations;

method declarations;

}

E.g.: class Sita extends Rama

{ int x;

void display()

{x = 10;

System.out.println(x);

}

}

***super and this Keywords***

When the super and sub classes can have members with the same name, then only the sub class members are accessed directly without any reference in the sub class. But the keyword *super* is used to refer a specified memberfrom the super class. The keyword *this* is used to refer a member from the current(sub) class. If we want to refer a super class member then we must use ***super.member***. If we want to refer asub class member then we must use ***this.member***.

***Static data members& methods***

* When we declare a data member with the keyword static, then it is called a staticdata member.
* Static data members can create only a single copy of common memory location for all objectsof a class.
* The static memory allocation is done, only when the first object is created to the class.
* When we declare a method with the keyword static, then it is called a static method.
* If we want to accessanexternal member from a static method, then thatmember is must be static.
* Static members are directly accessed in the other classes without creating an object to the class, they can directly referred by the class-name.

E.g.: static int a, b, c;

static void sum()

{

:

:

:

}

***Data Hiding***

Private members are accessed only in the current class, but the public members are accessed at any-where in the entire Java program. In this concept the private members visibility is not available at its instantiated class. This concept is called the data hiding or *information hiding.*

***Method Overloading***

When a class is having several methods with same name and different number or type of arguments then it is called ***method overloading***. In the method overloading only a specified method is executed by using the given parameters are matched to the specified method. This concept is very useful to execute several methods with the same name.

***ConstructorOverloading***

When several constructors in a class can have different number or type of arguments then it is called ***constructor overloading***. In the constructor overloading only a specified constructor is executed by using the given parameters are matched to the specified constructor. This concept is very useful to execute a constructor with different number or type of parameter values.

***Method Overriding***

When thesuper and sub classes can have methods with the same name, number& type of arguments, then the super class methods are over written by the sub class methods at time of an object creation. This concept is called the method overriding.*Final and static methods are not overridden.*

***Abstract Classes&Methods***

* When a class is declared by the keyword abstract then it is called an abstract class.
* Abstract class is not instantiated (i.e. it is not accepting to create an object).
* When a method is declared by the keyword abstract, then that method is called an abstract method.
* If we want to putan abstract method in a class, then that class is must be an abstract class.
* Abstract methods must not have body (function definition).
* Abstract specifier is not allowed for the data members, constructors and static methods.
* Abstract class may contain abstract methods or the normal methods.
* Abstract methods are must be defined (implemented)in their sub classes.

It is mainly used to specify a sub class must do (i.e which methods must be implemented).

***Note:- In an abstract class we must write only declarations for the abstract methods.That means we should not write definitions for the abstract methods in the abstract class.***

***Final Classes and Members***

* When a class is declared by the keyword final, then it is called a final class.
* Final classes are not inherited.
* When a member is declared by the keyword final, then it is called a final member, the data members are called final data members and the methods are called final methods.
* The final data members are must be initialized at the time of its declaration.
* The final methods are not modified or override in any time.

E.g.:1 final class C1

{ :

:

}

E.g.:2

final int x=10;

E.g.:3

final void show()

{

System.out.print("This is from show method");

}

***Interfaces***

* Interface is also syntactically same like as a class.
* But it is automatically an abstract class.
* Generally a class can handle data members and methods.But interface can have only the method declarations and final data members.
* Interface can specify what a class must do, but not how it does.
* In an interface the methods are automatically abstract and public and the data members are automatically public and final.
* If we want to call the interface methods from aJava program then first we must implement the interface methods in a subclass with the *public* access specifier.That means without implementing the interface methods, the methods are not accessed in the Java program.
* In an interface the methods must not be static and constructors.
* Java is notacceptedto create an object to the interface.
* Interfaces are mainly used to implement the multiple and hybrid inheritances in java.

If we want to declare an interface then we must follow the below syntax and its example.

Syntax is

access-specifier interface interface-name [extends interface1,interface2,…]

{

final members and the method declarations;

}

E.g.:

interface Rama

{int x=100;

void sum();

void product(int x, int y);

void avg(float a, float b, float c);

}

After the above example if we want to use the interface methods, then first we must implement interface methods in a class by following the below syntax and its example

Syntax is

class class-name [extends super-class] implements interface1, interface2,……..

{

Interface method definitions& the class members;

}

E.g.:

class Krishna implements Rama

{ public void sum()

{

System.out.println("From sum");

}

public void product(int x, int y)

{

System.out.println(x\*y);

}

public void avg(float a, float b, float c)

{

System.out.println((a+b+c)/3.0);

}

}

***Exceptions***

Exception means the error handling statements in Java. Exceptions are mainly used to control errors in the Java programs. In Java the errors are mainly classified into two types and they are

1. Compile time errors

2. Run time errors

***Compile Time Errors***

These are raisedat the time of program compilation. That means these are raisedonly when the program statements having syntax errors. These are rectified by the programmer manually by correcting the syntax errors.

***Run Time Errors***

These are raised at the time of program execution. These are raised when the program having logical errors. These are controlled by using the exceptions. Exceptions are mainly classified into 2 types. They are

1. Built-in Exceptions

2. User Defined Exceptions

***Built-in Exceptions***

These are directly available from the java software. Some of the useful built-in exceptions are explained below.

* **NumberFormatException:-** It is raised when the given number is invalid.
* **InputMismatchException:-** It is raised when the given input is not matched.
* **ArithmeticException:-** It is raised when an arithmetic expression having an error.
* **IOException:-** It is raised when the input or output statement having an error.
* **Exception:-** It is raised when any type of error is occurred.

If we want to control exceptions in the java programs then we must use the try and catch blocks by following the below syntax and its example.

Syntax is

try

{

statement(s);

}

catch(exception\_name1 e)

{

statement(s);

}

catch(exception\_name2 e)

{

statement(s);

}

:

:

:

finally

{

statement(s);

}

In the above syntax try block having the normal program statements. When an error is raised in the try block then its related catch block is executed to control errors. Exception\_name1 and exception\_name2 are the exceptions. Some of the most commonly used exceptions are explained above.The finally block is executed after the above try and catch blocks execution.

E.g.:

try

{

c=a/b;

}

catch(ArithmeticException e)

{

System.out.println("Denominator is Zero");

}

***User-Defined Exceptions***

The user written own type exceptions are called user defined exceptions. User-defined exception is a class andit must extended by the built-in class***Exception***. The Exception class is placed in the ***java.lang*** package.

E.g.:

class Ex1 extends Exception

{ Ex1(String s)

{

super(s);

}

}

If we want to call the user defined exception, then we must use the ***throw*** statement in try block by creating an object for the user defined exception by following the below example.

E.g.:

throw new Ex1("Denominator is Zero");

***getMessage() method***

It is used to read an error message from theException class.

E.g.: catch(Ex1 e)

{

System.out.println(e.getMessage());

}

***Multi-Threading***

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a *thread,* and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

There are two distinct types of multitasking like process-based and thread-based. The process-basedmultitasking is the feature that allows your computer to run two or more programs concurrently. For example, process-based multitasking enables you to run the Java compiler at the same time that you are using a text editor. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler. In a *thread-based* multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously.

In a movie the sound and visual effects are played in parallel. But in the normal Java programs a process is continued after the before process is done. But multi-threading is used to perform several processes at parallel.

If we want to create a thread, then the class must extended by the built-in class ***Thread.*** The class Thread is placed in the ***java.lang*** package. A thread class is must override by the ***run()***method. *The run method must be public and void*. If we want to declare a thread class in java then we must follow the below syntax and its example.

Syntax is

class class-name extends Thread

{

public void run()

{

statement(s);

}

}

E.g.:

class C1 extends Thread

{ public void run()

{ for(int c=0;c<=100;++c)

System.out.println("From class C1, the value of C is : "+c);

}

}

In Java the threads are controlled by the below 6 methods. These methods are placed in the built in class ***Thread***. They are

***1. start() Method***

It is used to start a specified thread.

Syntax is

thread\_object.start();

E.g.:

C1 t1=new C1();

t1.start();

***2. stop() Method***

It can be used stop an execution process of a specified thread.

Syntax is

thread\_object.stop();

E.g.:

t1.stop();

***3. sleep() Method***

It can temporarily stops an execution process of a thread into the specified number of milli seconds (1000 ms=1sec). This method must throw an InterruptedException.

Syntax is

thread\_object.sleep(n);

E.g.:

t1.sleep(1000);

***4. suspend() Method***

It canbe used to suspend a thread up to we resume the thread.

Syntax is

Thread.suspend();

E.g.:

Thread.suspend();

***5. resume() Method***

It can continuous an execution process a specified suspended thread.

Syntax is

Thread.resume();

E.g.:

Thread.resume();

***6. destroy() Method***

It can be used to remove a specified thread object from memory.

Syntax is

thread\_object.destroy();

E.g.:

t1.destroy();

***Runnable Interface***

A Runnable interface is one used to create a thread in java. It is placed in *java.lang* package. A thread can be created in two ways and using Runnable interface is one of them. A Runnable interface would have an abstract instance of the method run() which needs to be implemented in the class which wants to create a Thread.

Syntax is

class class-name implements Runnable

{

public void run()

{

statement(s);

}

}

E.g.:

class C1 implements Runnable

{ public void run()

{ for(int c=0;c<=100;++c)

System.out.println("From class c1, the value of C is : "+c);

}

}

After the above class, we must create an object to the class and then create a thread object for the Runnable interface by following the below syntax and its example.

Syntax is

Thread object=new Thread(Runnable\_interface\_object);

E.g.:

C1 o1=new C1();

Thread t1=new Thread(o1);

t1.start();

***Thread Synchronization***

When we start two or more threads in a program, there may be a situation when multiple threads try to access the same resources and finally they can produce unforeseen result due to concurrency issue. So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time.

Syntax is

synchronized(object\_identifier) { }

E.g.:

synchronized(t1) { … }

***wait()***

Tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

***notify()***

Wakes up the first thread that called wait() on the same object.

***Thread Priorities***

Thread priorities are used to control the execution speed of a thread. Thread priorities are control by the Thread class static final variables MAX\_PRIORITY, MIN\_PRIORITY and NORM\_PRIORITY. These priorities are applied to a thread with setPriority() method is available in the Thread class.

***setPriority()Method***

It sets priority of a thread into our own style with the priority static final variables.

Syntax is

Thread\_object.setPriority(int priority);

E.g.:

t1.setPriority(Thread.MAX\_PRIORITY);

***Packages***

A **package** is a container for classes. A package helps to o**rganize**your classes into a folder structure and make it easy to locate and use them. More importantly, it helps to improve **re-usability.**

A package is a separate work space area and it is formed with the number of classes. It is very useful to handle number of classes with the single reference name. A package is a Java version of the library. A package refers simply to a group of related class files in the same directory and having in each class file a package *directive* with that directory name at the top of the file.

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching or locating and usage of classes, interfaces and enumerations etc.A Package can be defined as a grouping of related types(classes, interfaces, enumerations) providing access protection and name space management.

Some of the existing (built-in) packages in Java are:

* **java.lang** - bundles the fundamental classes
* **java.io** - classes for input , output functions are bundled in this package

Since the package creates a new namespace there won't be any name conflicts with names in the other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

In a package the classes and their members must be public,otherwise the classes and their membersare not accessed at outside the package.

## *Creating a package:*

When creating a package, you should choose a name for the package and put a **package** statement with that name at the top of every source file that contains the classes, interfaces and enumerations types that you want to include in the package.

***Package Statement***

The **package** statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file. If a package statement is not used in the class or interface or enumeration then the class or interfacesor enumeration types will be put into an unnamed package.

## E.g.: package pack1;

## *Import Statement*

If a class wants to use another class in the same package, the package name does not need to be used.If a class wants to use a class from another package, the import statement is essential.

E.g.: importpack1.Operations;

A class file may contain any number of import statements. The import statements must appear after the package statement and before the class declaration.

If we want to create a package then we must follow the below steps.

1. Write the below java program.

package pack1;

public class Calc1

{ int a=45,b=23,c;

public void add()

{ c=a+b;

System.out.println("Addition="+c);

}

public void sub()

{ c=a-b;

System.out.println("Subtraction="+c);

}

}

1. Save the above program in the source folderby the name Calc1.java.
2. Compile the above program with the below command

javac -d . Calc1.java

After compilation of the above program, it creates a new folder named pack1and then put the class file Calc1 into that location.

Write the below program to call the above package *pack1* related class Calc1and their methodsadd() and sub().

import pack1.Calc1;

class Prog1

{ public static void main(String[] args)

{ Calc1 o=new Calc1();

o.add();

o.sub();

}

}

1. Save the above program in the source folder (parent folder of pack1), compile and execute the program with the below commands.

javac Prog1.java

java Prog1

Note:- If we want to access the package pack1's related class Calc1 from its parent folder, then the classpath declaration is not necessary,otherwise we must set a class path to the parent folder of the package pack1 by giving the below command at command prompt

E.g.:

set classpath=%classpath%;z:\java

***INNER CLASSES***

When a class has a single or multiple classes its inside, then those inside classes are called inner classes. Inner class related members can be access only by its outer class. The main class of an inner class is called an outer class. This concept is also known as ***nested classes***.

E.g.:

class Rama //outer class

{

------

------

------

------

class Sai //inner class

{

------

------

------

------

}

}

In the above example *Rama* is an outer class of the class *Sai*. *Sai* is an inner class of the class R*ama*. This type of concept is very useful to protect members in the inner class.If we want to create an object to the inner class Sai, then we must follow the below example.

Rama r=new Rama();

Rama.Sai s=r.new Sai();

***POLYMORPHISM***

* This is a crucial concept in OOPS.
* A single name with multiple forms is known as polymorphism.
* It is the ability of an object to take many forms.

It is considered into 2 types. They are

1. Compile-Time polymorphism(***Static binding or method overloading***)
2. Run-Time polymorphism(***Dynamic binding or method overriding***)

***COMPILE-TIMEPOLYMORPHISM***

* This is also known as static binding.
* It is implemented only by the method overloading.
* The flow of control is decided at the time of program compilation.

***RUN-TIME POLYMORPHISM/ Dynamic Method Dispatch***

* This is also known as dynamic binding.
* It is implemented only by the method overriding.
* The flow of control is decided at the time of the program execution.

***DATA FILES***

***File***

Eachinformationwasstored on a disk is called a file. Each and every file can be identified by a name and that name is called the *file name*. Each and every file name may have some extra characters its ending separated by dot (**.**) called an *extension*. Extension is mainly used to identify the file type. In the ms-dos operating system the file name should not exceed more than 8 characters and an extension should not exceed more than 3 characters. But in the windows operating system the file name and extension must not exceed more than 255 characters.In java the files are mainly classified into two types and they are

1. Program files
2. Data files

***Program files***

These are mainly used to maintain only the programs in Java and the programs may be either application programs or applets or servlets. In Java the all types of program files havean extension of*.java.*

***Data files***

Data files are mainly used to handle the given data. In a data file the data willbe arranged into the number of rows and columns. In a data file each row is called a *record* and each column is called a *field*.

A data file is formed with the number of records. A record is formed with the number of related data items and a data item is formed with the number of characters.

In Java the data files are controlled by the program files. In Java the data files are not maintain the default extension. But generally we use***.dat***as a data file extension. In Java the data files are mainly classified into 2 types and they are

i. Stream Oriented data files

ii. Random Access data files

***i. Stream Oriented Data Files***

A Stream can be defined as a sequence of data. In a stream oriented data file the both reading and writing operations are performed into the sequential order. So these files are also called *sequential data files*. A stream can be acts like an interface between the program and the storage device (disk). The streams are mainly classified into 2 types and they are input stream and output stream. Input stream can be used for the reading operations and the output stream can be used for the writing operations.

Program **Input/Output Stream**

Disk

The stream oriented data files are controlled by 2 types of streams and they are

1. Byte code stream
2. Character code stream

***Byte Code Stream***

It is mainly used to store the given data into bytes (8-bits). There are many classes related to the byte streams like ***FileInputStream*** and the ***FileOutputStream***.

***Character Code Stream***

It is used to handle data into characters (16-bit unicode). There are many classes related to the character streams like ***FileReader*** and the ***FileWriter***.But internally FileReader uses FileInputStream and the FileWriter uses FileOutputStream. *But the major difference is the FileReader reads 2 bytes of data and the FileWriter writes 2 bytes of data at a time.*The character code stream is available only from the Java ver 1.1.

***DataOutputStream Class***

It is used to write data on the specified output stream. It is placed in the java.io package. If we want to create an object to the DataOutputStream class, then we must follow the below syntax.

Syntax is

DataOutputStream object=new DataOutputStream(output\_stream);

E.g.:-1

DataOutputStream dos=new DataOutputStream(System.out);

E.g.:-2

File f = new File("address.dat");

FileOutputStream fos =new FileOutputStream(f);

DataOutputStream dos= new DataOutputStream(fos);

The below explained methods are available in the ***DataOutputStream*** class.

***1. writeInt() Method***

It is used to store only the given integer type datain the output stream.

Syntax is

data-output-stream-object.writeInt(int);

E.g.:

dos.writeInt(10);

***2. writeShort() Method***

It is used to store only the short integer data into an output stream.

Syntax is

data-output-stream-object.writeShort(short);

E.g.:

dos.writeShort(10);

***3. writeByte() Method***

It is used to store only the byte type data into an output stream.

Syntax is

data-output-stream-object.writeByte(byte);

E.g.:

dos.writeByte(10);

***4. writeLong() Method***

It is used to store only the long type data into an output stream.

Syntax is

data-output-stream-object.writeLong(long);

E.g.:

dos.writeLong(10);

***5. writeFloat() Method***

It is used to store only the floating point data into an output stream.

Syntax is

data-output-stream-object.writeFloat(float);

E.g.:

dos.writeFloat(10.1);

***6. writeDouble() Method***

It is used to store only the double type data into an output stream.

Syntax is

data-output-stream-object.writeDouble(double);

E.g.:

dos.writeDouble(10.0);

***7. writeChar() Method***

It is used to store only the character type data into an output stream.

Syntax is

data-output-stream-object.writeChar(char);

E.g.:

dos.writeChar('x');

***8. writeBoolean() Method***

It is used to store only the boolean type data into an output stream.

Syntax is

data-output-stream-object.writeBoolean(boolean);

E.g.:

dos.writeBoolean(true);

***9. writeUTF() Method***

UTF stands for Unicode Text Format. It is used to store onlythe string type data into an output stream.

Syntax is

data-output-stream-object.writeUTF(string);

E.g.:

dos.writeUTF("Rama");

***DataInputStream Class***

It is used to read data from the specified input stream. It is placed in the java.io package. If we want to create an object to the DataInputStream class, then we must follow the below syntax.

Syntax is

DataInputStream object=new DataInputStream(input\_stream);

E.g.:-1

DataInputStream dis=new DataInputStream(System.in);

E.g.:-2

File f = new File("address.dat");

FileInputStream fis =new FileInputStream(f);

DataInputStream dis= new DataInputStream(fis);

The below explained methods are available in the ***DataInputStream*** class.

***1. readLine() Method***

It is used to read an entire line from an input stream as a string.

Syntax is

data-input-steam-object.readLine()

E.g.:

dis.readLine();

***2. readInt() Method***

It is used to read only an integer datafrom an input stream.

Syntax is

data-input-stream-object.readInt();

E.g.:

dis.readInt();

***3. readShort() Method***

It is used to read only a shorttype datafrom an input stream.

Syntax is

data-input-stream-object.readShort();

E.g.:

dis.readShort();

***4. readByte() Method***

It is used to read only the byte type data from an input stream.

Syntax is

data-input-stream-object.readByte();

E.g.:

dis.readByte();

***5. readLong() Method***

It is used to read only the long type data from an input stream.

Syntax is

data-input-stream-object.readLong();

E.g.:

dis.readLong();

***6. readFloat() Method***

It is used to read only a float type datafrom an input stream.

Syntax is

data-input-stream-object.readFloat();

E.g.:

dis.readFloat();

***7. readDouble() Method***

It is used to read only the double type datafrom an input stream.

Syntax is

data-input-stream-object.readDouble();

E.g.:

dis.readDouble();

***8. readChar() Method***

It is used to read only the charactertype data from an input stream.

Syntax is

data-input-stream-object.readChar();

E.g.:

dis.readChar();

***9. readBoolean() Method***

It is used to read only the boolean type datafrom an input stream.

Syntax is

data-input-stream-object.readBoolean();

E.g.:

dis.readBoolean();

***9. readUTF() Method***

It is used to readonly the string type data from an input stream.

Syntax is

data-input-stream-object.readUTF();

E.g.:

dis.readUTF();

***File Class***

It is used to handle a data file namewith extensionand path.This class is used for creating files and directories, searching and deleting files … etc. In Java the data files related allclasses are placed in the ***java.io*** package.

Syntax is

File object = new File("data\_file\_name.extension");

E.g.:

File f = new File("address.dat");

***FileOutputStream Class***

It is used to open a data file to perform writing operations. Output operations means to store or append data into a specified file.

Syntax is

FileOutputStream object = new FileOutputStream(file-class-object or "file-name.extension"[,append]);

In the above syntax append may be either *true* or *false*. The default value for append is false. If it is true then it is used to add new records into the specified file.Otherwise it can store new records by overwriting the existing records.

E.g.:-1

File f = new File("address.dat");

FileOutputstream fos= new FileOutputStream(f);

E.g.:-2

FileOutputStream fos= new FileOutputStream("address.dat");

E.g.:-3

FileOutputStream fos= FileOutputStream("address.dat",true);

***FileInputStream Class***

It is used to open a data file to perform reading operations.

Syntax is

FileInputStream object = new FileInputStream(file-class-object or "filename.extension");

E.g.:-1

File f = new File("address.dat");

FileInputStream fis = new FileInputStream(f);

E.g.:-2

FileInputStream fis = new FileInputStream("address.dat");

***close() Method***

It closes the currently opened specified data file.

Syntax is

filestreamobject.close();

E.g.:-

fos.close();

***Exceptions in Files***

***IOException***

It is raised when an input or output operation related error is occurred in the program.

***FileNotFoundException***

It is raised when the specified file is not found at the specified path.

***EOFException***

EOF stands for End Of File. It is raised when the specified file is end.

***ii. Random Access Data Files***

The RandomAccessFile class is placed in the ***java.io*** package. The random access data files are also called the *direct access files*. That means these files should not handle the data input and output streams.

***RandomAccessFile Class***

It is used to declare a random access data file related data file name and its mode.

Syntax is

RandomAccessFile object = new RandomAccessFile("filename.ext" or file-object, "access mode");

In the above syntax the access mode may be either 'rw' or 'r'. The access mode 'rw' is used for the reading, writing and append operations. But the access mode 'r' is only used for the reading operation.

E.g.:

RandomAccessFile raf = new RandomAccessFile("ranadd.dat", "rw");

***close() Method***

It closes the currently opened specified data file.

Syntax is

Random\_access\_file\_object.close();

E.g.:

raf.close();

The reading and writing operations are also same like as the stream oriented data files. But we must use the random access file object instead of data input stream or the data output stream object.

***length() Method***

This method is a member in the RandomAccessFile class. It is used to returnnumbers of bytes areoccupied by the specified random access file.

Syntax is

random\_accessfile\_object.length();

E.g.:

System.out.println(raf.length());

***seek() Method***

This method is a member in the RandomAccessFile class. This method is used to transfer control to a specified byte position into the randomaccessfile.

Syntax is

random\_accessfile\_object.seek(int);

E.g.:-1

raf.seek(0);

E.g.:-2

raf.seek(raf.length());

***AutoBoxing and UnBoxing***

The automatic conversion of primitive data types into its equivalent wrapper type is known as *boxing* and opposite operation is known as *unboxing*. This is the new feature of Java5.

E.g.:-1

Int a=5;

Integer a1=new Integer(a); //autoboxing

Integer a2=5; //autoboxing

E.g.:-2

Integer i=new Integer(50);

Int a=i; //unboxing

***Enum Types***

An enum type is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.In the Java programming language, you define an enum type by using the enum keyword. For example, you would specify a daysof theweek enum type as

E.g.:

enum days

{

Sunday, Monday, Tuesday, Wednesday,Thursday,Friday,Saturday

}

class Prog1

{ public static void main(String[] args)

{ days d;

d=days.Thursday;

System.out.println("the value of d is : "+d);

System.out.println("the value of "+d+" is : "+d.ordinal());

}

}

***Enumeration Interface***

Enumeration interface generates a series of elements, one at a time. Successive calls to the nextElement method return successive elements of the series.

*E.g.:*

import java.util.Vector;

import java.util.Enumeration;

public class Enum1

{ public static void main(String[] args)

{ Vector dayNames = new Vector();

dayNames.add("Sunday");

dayNames.add("Monday");

dayNames.add("Tuesday");

dayNames.add("Wednesday");

dayNames.add("Thursday");

dayNames.add("Friday");

dayNames.add("Saturday");

Enumeration days=dayNames.elements();

while (days.hasMoreElements()){

System.out.println(days.nextElement());

}

}

}

***Collections FrameWork***

It provides many interfaces and classes to perform searching, sorting, insertion, manipulation, deletion … etc. The colletion framework related all classes and interfaces are placed in the ***java.util*** package.

*\*\* There are two ways to retrieve the list from a collection. They are \*\**

1. *Iterator*
2. *for each*

***1) Iterator Interface***

It can provide facility of iterating the elements in forward direction only. It hasthe below methods,

***hasNext() :*** It can test whether the iterator has more elements or not. If it has more elements, then it can return true, otherwise it can return false

***next() :*** It can return current element and moves pointer to the next element.

***remove() :*** It can remove last element returned by the iterator.

E.g.:/\*\*iterator\*/

import java.util.ArrayList;

import java.util.Iterator;

class Prog1

{ public static void main(String[] args)

{ ArrayList al=new ArrayList();

al.add("NICE");

al.add(123);

al.add(true);

Iterator i=al.iterator();

while(i.hasNext()) System.out.println(i.next());

}

}

***2) for each :*** It can be used to retrieve elements from a collection one by one.

E.g.:-1 /\*\*foreach\*/

import java.util.ArrayList;

class Prog2

{ public static void main(String[] args)

{ ArrayList<String> al=new ArrayList();

al.add("NICE");

al.add("123");

al.add("true");

for(String x:al) System.out.println(x);

}

}

E.g.:-2 /\*\*foreach\*/

class Prog3

{ public static void main(String[] args)

{ int a[]={10,20,30,40,50};

for(int x:a) System.out.println(x);

}

}

***ArrayList Class***

It is a dynamic array used for storing elements. It may contain duplicate elements and maintains insertion order, it can allow random access by its index.Insertion and deletion operation is slow because of shifting the existing data. It has the below methods.

***add() :*** It is used to insert an element at a specified index position.

***addAll() :*** It is used to add all the elements from a collection at the end of the existing list.

***clear() :*** It is used to remove all the elements from the array list.

***size() :*** It can return number of elements count from the array list.

***isEmpty() :*** It can be used to test whether anarray list is either empty or not. If it is empty, then it can retrun true as result, otherwise it can return false as result.

***toArray() :*** It is used to return an array containing all of the elements in the array list.

***indexOf() :*** It can be used to return index of the first occurance of the specified element from the list.

***lastIndexOf() :*** It can be used to return index of the last occurance of the specified element from the list.

***Iterator() :*** It can return elements from the list.

E.g.: /\*\*foreach with ArrayList\*/

import java.util.ArrayList;

class Prog4

{ public static void main(String[] args)

{ ArrayList al=new ArrayList();

al.add("NICE");

al.add(123);

al.add(true);

for(Object x:al) System.out.println(x);

}

}

***LinkedList Class***

It is a doubly linked list used for storing elements. It may contain duplicate elements, maintains insertion order, insertion and deletion operation is fast because of no shifting is needed.

***add() :*** It is used to insert an element at a specified index position.

***addAll() :*** It is used to add all the elements from a collection at the end of the existing list.

***addFirst() :*** It is used to insert an element at the beginning of the existing list.

***addLast() :*** It is used to insert an element at the ending of the existing list.

***size() :*** It can return number elements count from the list.

***remove() :*** It can be used to remove first occurance of the specified element from the list.

***getFirst() :*** It can return first element from the list.

***getLast() :*** It can return last element from the list.

***removeFirst() :*** It can remove first element from the list.

***removeLast() :*** It can remove last element from the list.

***get() :*** It can return an element from the list by its index.

***set() :*** It is used to assign an element in the list by its index.

***indexOf() :*** It can be used to return index of the first occurance of the specified element from the list.

***lastIndexOf() :*** It can be used to return index of the last occurance of the specified element from the list.

***pop() :*** It can remove and return top element from the list.

***peek() :*** It can return top element from the list.

***Differences between ArrayList and LinkedList***

|  |  |
| --- | --- |
| ***ArrayList*** | ***LinkedList*** |
| Internmally uses dynamic array | Internally uses doubly linked list |
| Manipulation is very slow | Manipulation is faster |
| Better for storing & accessing data | Better for manipulating data |

E.g.: /\*\*LinkedList\*/

import java.util.LinkedList;

class Prog5

{ public static void main(String[] args)

{ LinkedList ll=new LinkedList();

ll.add("NICE");

ll.add(123);

ll.add(true);

ll.add(123);

for(Object x:ll) System.out.println(x);

}

}

***List Interface***

It is a subinterfce of Collection. It is used to insert and delete elements in the index basis.It has the below methods.

***add() :*** It is used to insert an element at a specified index position.

***addAll() :*** It is used to insert a collection from a specified index position.

***remove() :*** It can be used to remove a specified element from the list and return the deleted element.

***isEmpty() :*** It can be used to test whether an array list is either empty or not. If it is empty, then it can retrun true as result, otherwise it can return false as result.

***set() :*** It is used to assign an element in the list by its index.

***get() :*** It can return an element from the list by its index.

***hasNext() :*** It can test whether the iterator has more elements or not. If it has more elements, then return true, otherwise it can return false.

***next() :*** It can return next element and moves pointer to the next element.

***hasPrevious() :*** It can test whether the iterator has more previous elements or not. If it has more previous elements, then return true, otherwise it can return false.

***previous() :*** It can return previous element and moves pointer to the previous element.

***indexOf() :*** It can be used to return index of the first occurance of the specified element from the list.

***lastIndexOf() :*** It can be used to return index of the last occurance of the specified element from the list.

E.g.: /\*\*List\*/

import java.util.ArrayList;

import java.util.List;

class Student

{ int rno; String name;

Student(int x,String y)

{ rno=x;

name=y;

}

}

class Prog6

{ public static void main(String[] args)

{ List<Student> lst=new ArrayList<Student>();

Student st1=new Student(100,"Rama");

Student st2=new Student(200,"Sita");

Student st3=new Student(150,"Gopal");

lst.add(st1);

lst.add(st2);

lst.add(st3);

for(Student s:lst) System.out.println(s.rno+" "+s.name);

}

}

***HashSet Class***

It is used to create a collection and uses hash table for storage. It uses the mechanism called **hashing**. It contains unique elements only. It has the below methods.

***add() :*** It is used to add an element, if it is not in the hashset.

***isEmpty() :*** It can return true when the set contains no elements.

***remove() :*** It can be used to remove a specified element from the list if it is present in the hashset.

**size*() :*** It can return number elements in the hashset.

***clear() :*** It is used to remove all the elements from the hashset.

***Iterator() :*** It can return elements from the hasset.

E.g.: /\*\*HashSet\*/

import java.util.HashSet;

class Prog7

{ public static void main(String[] args)

{ HashSet hs=new HashSet();

hs.add("NICE");

hs.add(123);

hs.add(true);

hs.add(123);

hs.add("NICE");

for(Object x:hs) System.out.println(x);

}

}

***TreeSet Class***

It uses a tree for storage. It contains unique elements. Access and retrieval time is very fast. It maintains list in the ascending order. It has the below methods.

***add() :*** It is used to add an element, if it is not in the list.

***addAll() :*** It is used to add all elements from a collection to the list.

***isEmpty() :*** It can return true when the set contains no elements.

***remove() :*** It can be used to remove a specified element from the list if it is present in the list.

***clear() :*** It can be used to remove all elements from the list.

***first() :*** It can return first(lowest) elements from the list.

***last() :*** It can return last(highest) elements from the list.

**size*() :*** It can return number elements in the list.

**lower*() :*** It can return greatest element less than the given element.

**higher*() :*** It can return least element greater than the given element.

*E.g.:* /\*\*TreeSet\*/

import java.util.TreeSet;

class Prog8

{ public static void main(String[] args)

{ TreeSet ts=new TreeSet();

ts.add("NICE");

ts.add("123");

ts.add("true");

ts.add("123");

ts.add("NICE");

for(Object x:ts) System.out.println(x);

}

}

***HashTable Class***

It is used to map keys to the values. It inherits Dictionary class. It is an array of List. It contain values based on the key.It can contain only the unique elements. It may not have any null value or key. It has the below methods.

***put() :*** It is used to add an element value and its key.

***getkey() :*** It can return current key.

***getValue() :*** It can return current value.

***clear() :*** It can be used to reset the hash table.

***isEmpty() :*** It can return true when the set contains no elements.

***remove() :*** It can be used to remove a specified element from the hash table if it is present in the list.

**size*() :*** It can return number entries in the hash table.

**hashCode*() :*** It can get the hashcode value from the Map interface.

**equals*() :*** It can compares the specified object with the Map for equality.

E.g.: /\*\*Hashtable\*/

import java.util.Hashtable;

import java.util.Map;

class Prog9

{ public static void main(String[] args)

{ Hashtable<Integer,String> ht=new Hashtable<Integer,String>();

ht.put(1,"NICE"); ht.put(3,"computers");

ht.put(2,"Vijayawada");

for(Map.Entry m:ht.entrySet())

{

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

***VectorClass***

It is a dynamic array and it is similar to the ArrayList.It's size may be varied at any time by the ***Vector(int size,int incr)*** constructor. It has the below methods.

***addElement() :*** It can add an element to the end of the vector.

***insertElementAt() :*** It can insert an element at a specified index position.

***removeElementAt() :*** It can remove an element at a specified index position.

***remove() :*** It can remove an element at a specified index position.

***setSize() :*** It can set size of the vector.

***size() :*** It can return number of elements count from the vector.

***removeAllElements() :*** It can remove all elements from the vector and sets its size to zero.

***isEmpty() :*** It can test whether the vector is either empty or not.

***toArray() :*** It can return an array containing all of the elements from the vector.

***indexOf() :*** It can return index of the first occurance of the specified element from the vector.

***lastIndexOf() :*** It can return index of the last occurance of the specified element from the vector.

***Iterator() :*** It can return elements from the vector.

***firstElement() :*** It can return first element from the vector.

***lastElement() :*** It can return last element from the Vector.

E.g.: /\*\*vector\*/

import java.util.Vector;

class Prog10

{ public static void main(String[] args)

{ Vector v=new Vector();

v.addElement("Sunday");

v.addElement("Monday");

v.addElement("Tuesday");

v.addElement("Wednesday");

v.addElement("Thursday");

v.addElement("Friday");

v.addElement("Saturday");

for(Object x:v) System.out.println(x);

}

}

***Stack Class***

It is a subclass of the Vector class. It implements a last-in, first-out stack. It has the below methods.

***push() :*** It can push an element into the stack.

***pop() :*** It can return top element from the stack and removes it.

***peek() :*** It can return top element from the stack, but does not remove it.

***search() :***It can search an element from the stack.

E.g.: /\*\*Stack\*/

import java.util.Stack;

import java.util.EmptyStackException;

class Prog11

{ public static void main(String[] args)

{ Stack s=new Stack();

s.push(10);

s.push(15);

s.push(5);

s.push(8);

s.push(12);

try

{ while(true)

{

System.out.println(s.pop());

}

}

catch(EmptyStackException e) { }

}

}

***Collections.sort() Method***

It can be used to sort the elements of a list.

E.g.:

/\*\*sort method\*/

import java.util.ArrayList;

import java.util.Collections;

class Prog12

{ public static void main(String[] args)

{ ArrayList al=new ArrayList();

al.add("NICE");

al.add("Rama");

al.add("Sai");

al.add("Krishna");

Collections.sort(al);

for(Object x:al) System.out.println(x);

}

}

***StringTokenizer Class***

It can be used to break a string into tokens.

E.g.: /\*\*Stringtokenizer\*/

import java.util.StringTokenizer;

class Prog13

{ public static void main(String[] args)

{ StringTokenizer st=new StringTokenizer("NICE COMPUTERS VIJAYAWADA");

while(st.hasMoreTokens())

{

System.out.println(st.nextToken());

}

}

}

***DateClass***

It can maintain the date and time in milliseconds since January 1, 1970, 00:00:00 GMT.

E.g.: /\*\*Date\*/

import java.util.Date;

class Prog14

{ public static void main(String[] args)

{ Date dt=new Date();

System.out.println("Current Date & Time is : "+dt);

}

}

Output:

Current Date & Time is : Wed Aug 29 17:19:54 IST 2018

***APPLETS***

Applet is a small java program that is embedded and ran in some other java interpreter program,such as a java technology enabled browser or sun's ***appletviewer***. Internet related programs in Java are called applets. There is no main() method in an applet.An applet uses AWT (Abstract window tool kit) package for graphics.

In Java the Applet class is placed in the ***java.applet***package.In an applet the graphical operations are controlled by the ***Graphics*** class and it is placed in the ***java.awt*** (Abstract Window Toolkit) package. When an applet is stored in the local computer, then it is called a***local applet***.If we want to access the local applets, then we need not have an internet connection. When an applet is not stored in the local computer, then it is called a***Remote Applet***. It we want to call remote applets then we must require an internet or an intranet(Local Area Network)connection. In java the applets are controlled by four methods. They are

* + - * 1. **init()**
        2. **start()**
        3. **stop()**
        4. **destroy()**
        5. **paint()**

These all methods are placed in the Applet class. The life cycle of an applet is explained below

***init() Method***

This methodis automatically executed, only once at the time of an applet loading. It is intended for whatever initialization is needed for an applet.

***start() Method***

This method is automatically called after the init() method. It is also called whatever user returns to the page containing the applet after visiting other pages.

***stop() Method***

This method is automatically called whenever the user moves away from the page containing applets. This method can be used to stop an animation.

***destroy() Method***

This methodis called automatically when the browser shuts down normally. This method can de-allocate the applet'sresources.

***paint() Method***

If we want to display text or graphical images on the applet, then we must use the paint() method. The paint() method is placed in the ***java.awt.Component***class.It is immediately invoked after the start() method. The paint() method must be public and void.

E.g.: public void paint(Graphics g){ }

***drawString() Method***

This method is placed in the ***Graphics*** class, and Graphics class is placed in the ***java.awt*** package. This method displays the given string at a specified location on the applet's graphics screen.

Syntax is

graphics\_object.drawString(string, int x\_pos,int y\_pos);

E.g.:

g.drawString("NICE COMPUTERS",100,100);

***Image Class***

It is placed in the java.awt package, it can represent a graphical image.

E.g.:

Image img=getImage(getDocumentBase(),"pict1.gif");

***getDocumentBase() Method***

It can return URL of the document in which the applet is embedded.

E.g.:

file:/Z:/jv7pm/java/applets/

***getCodeBase() Method***

It can return base URL of the document in which the applet is embedded.

E.g.:

file:/Z:/jv7pm/java/applets/App4.java

***drawImage() Method***

It is a ***Graphics*** class method and it can display animage on the applet screen at a specified location.

Syntax is

graphics\_object.drawImage(image\_object, int x\_pos,int y\_pos,image\_observer);

E.g.:

g.drawImage(img,10,10,this);

The execution process of anapplet in browser is explained below.

1. Browser visits page containing an applet
   1. Browser calls an init method on that applet, once
   2. Browser calls start method on that applet
2. Browser goes away from that page
   1. Browser calls stop method on that applet
3. Browser comes back to that page
   1. Browser calls start method again on that applet
4. Browser shuts down
   1. Browser calls destroy method on the applet, once

The life cycle diagram of an applet is shown below.

init()

start() stop()

paint()

destroy()

If we want to write a program with applets in java,then we must follow the below steps.

* 1. The user defined applet class is must imports both the java.awt.Graphics and java.applet.Applet packages.
  2. The user defined applet class is must be public, otherwise it is not accessed from the browser or from thesun's appletviewer.
  3. The user defined applet class is must extended by the built-in class Applet.
  4. The user defined applet class is must implement paint() or at least any one of it's life cycle method.
  5. Applet class is called by the HTML script with the <applet> tag.

Applets are executed by the sun's appletviewer or any one of the web browser like InternetExplorer, MozillaFireFox,Google Chrome, Safari or Opera … etc.

E.g.: /\*\*Applet\*/

Import java.applet.Applet;

Import java.awt.Graphics;

Public class App1 extends Applet

{ public void paint(Graphics g)

{

g.drawString("Welcome to my first Applet",50,100);

}

}

After writing the above applet, we must save that applet with the filename & extension of ***App1.java***. After that, we must compile it by the below command.

javac App1.java

If we want to callthe above applet then we must write the below***HTML***code.

<html>

<head>

<title>Sample Applet</title>

</head>

<body bgcolor="lightyellow">

<appletcode="App1.class"width="500"height="500">

</applet>

</body>

</html>

After writing the above code, wemust save that code with the file name& extension of***App1.html***.

If we want to execute theabove applet with appletviewer, we must use the below command.

*appletviewer App1.html*

If we want to execute an applet through the browser (E.g:. internet explorer,Mozilla Firefox, Google Chrome, Safari, Opera,…etc), we should give double click on the html file (or) open html file through the browser.

***Note: If we want to execute an applet in browser, we must change java security level.***

Start**⇨**All Programs**⇨**Java**⇨**Configure Java**⇨**Security**⇨**Medium**⇨**Ok

If we want to write both the applet class and html code into a single program, we must follow the below example.

E.g.: /\*\*Applet\*/

/\*\*<applet code="App2.class" width="500" height="500"></applet>\*/

Import java.applet.Applet;

Import java.awt.Graphics;

Public class App2 extends Applet

{ public void paint(Graphics g)

{

g.drawString("Welcome to my first Applet",50,100);

}

}

Save & compile the above program and execute it, with the below command.

*appletviewer App1.java*

***Passing values from HTML script's Applet tag to an applet***

If we want to pass values from HTML script's Applet tag to an applet then we must use the parameter tag within the applet tag by using the below syntax and its example.

Syntax is

<applet……>

<param name="parameter\_name" value="parameter\_value">

:

:

:

</applet>

E.g.:

<applet code="App1" width="500" height="500">

<param name= "uname" value= "RAMA">

<paramname= "pass" value= "Krishna">

</applet>

***getParameter() Method***

It is used to read a specified parameter value from HTML script's applet tag into an applet.

Syntax is

getParameter("parameter-name");

E.g.:

String s;

s=getParameter("uname");

***setBackground() Method***

It can change background color of the applet screen. In an applet the colors are controlled by theColor class and it is placed in the ***java.awt*** package.

Syntax is

setBackground(color);

E.g.:

setBackground(Color.BLUE);

***setForeground() Method***

It can be used to set foreground color on the applet screen.

Syntax is

setForeground(color);

E.g.:

setForeground(Color.RED);

**Advantages**

* Automatically integrated with HTML, hence resolved virtually all installation issues
* Can be accessed from various platforms and various java-enabled web browsers
* Can provide dynamic, graphics capabilities and visualizations
* Implemented in Java, an easy-to-learn object oriented programming language
* Alternative to HTML GUI design
* Safe! Because of the security built into the core Java language and the applet structure, you don't have to worried about bad code causing damage to someone's system
* Can be launched as a standalone web application independent of the host web server

**Disadvantages**

* Applets can't run any local executable programs
* Applets can't access any host other than the originating server
* Applets can't read/write to local computer's file system
* Applets can't find any information about the local computer
* All java-created pop-up windows carry a warning message
* Stability depends on stability of the client's web server
* Performance directly depend on client's machine

**GRAPHICS**

In Java the graphic methods are placed in the ***Graphics*** class. The Graphics class is placed in the ***java.awt*** package. Graphics class has several methods to draw lines, rectangles, ovals, polygons, arcs … etc. The most frequently used Graphicsclass methods are explained below.

***drawLine() Method***

It can draw a line image by using the given starting and ending pixel positions.

Syntax is

graphics\_class\_object.drawLine(int x1, int y1, int x2, int y2);

E.g.:

g.drawLine(10,10,100,100);

***drawRect() Method***

It can draw a rectangle image by using the giventop-left cornerpixel, width and height values.

Syntax is

graphics\_class\_object.drawRect(int x, int y, int width, int height);

E.g.:

g.drawRect(100,100,100,100);

***fillRect() Method***

I can draw a filled rectangle image by using the given top-left cornerpixel, width and height values.

Syntax is

graphics\_class\_object.fillRect(int x, int y, int width, int height);

E.g.:

g.fillRect(100,100,30,50);

***drawOval() Method***

It can draw a circle or ellipse image by using the giventop-left cornerpixel, x-axis and y-axis radius values.

Syntax is

graphics\_class\_object.drawOval(int x, int y, int x-radius, int y-radius);

E.g.:

g.drawOval(100,100,30,50);

***fillOval() Method***

It can draw a filled circle or ellipse image by using the giventop-left cornerpixel, x-axis and y-axis radius values.

Syntax is

graphics\_class\_object.fillOval(int x, int y, int x-radius, int y-radius);

E.g.:

g.fillOval(100,100,30,50);

***drawArc() Method***

It can draw an arc image by using the giventop-left corner pixel, x-axis radius, y-axis radius, starting angle and number of degrees to draw an arc values.

Syntax is

graphics\_class\_object.drawArc(int x,int y, int x-radius,int y-radius,

int st\_angle\_in\_degrees,int number\_of\_degrees\_to\_draw\_an\_arc);

E.g.:

g.drawArc(100,100,100,100,180,180);

***fillArc() Method***

It can draw a filled arc image by using the giventop-left corner pixel, x-axis radius, y-axis radius, starting angle and the number of degrees to draw an arc values.

Syntax is

graphics\_class\_object.fillArc(int x, int y, int x-radius, int y-radius, int st-angle\_in\_degrees, int number\_of\_degrees\_to\_draw\_an\_arc);

E.g.:

g.fillArc(100,100,100,100,180,180);

***drawRoundRect() Method***

It can draw a rounded rectangle image by using the given starting pixel position, width, height, corner1 and corner2 values.

Syntax is

graphics\_class\_object.drawRoundRect(int x, int y, int width, int height, int corner1,

int corner2);

E.g.:

g.drawRoundRect(10,10,100,100,20,20);

***fillRoundRect() Method***

It can draw a filled rounded rectangle image by using the given starting pixel position, width, height, corner1 and corner2 values.

Syntax is

graphics\_class\_object.fillRoundRect(int x, int y, int width, int height, int corner1,

int corner2);

E.g.:

g.fillRoundRect(10,10,100,100,20,20);

***drawPolygon() Method***

It can draw a polygon image by using the given x co-ordinate pixel positions array, y co-ordinate pixel positions arrays and the number of points value.

Syntax is

graphics\_class\_object.drawPolygon(int x-points\_array,int y-points\_array,int num\_points);

E.g.:

int x[]={100,200,300};

int y[]={110,10,110};

g.drawPolygon(x, y, 3); (200,10)

(100,110) (300,110)

***fillPolygon() Method***

It can draw a filled polygon image by using the given x co-ordinate pixel positions array, y co-ordinate pixel positions arrays and the number of points value.

Syntax is

graphics\_class\_object.fillPolygon(int x-points\_array,int y-points\_array,int num\_points);

E.g.:

int x[]={100,200,300};

int y[]={110,10,110};

g.fillPolygon(x, y, 3);

***setColor() Method***

It can be used to set foreground colorfor the graphical images.

Syntax is

graphics\_class\_object.setColor(color);

E.g.:

g.setColor(Color.RED);

***AWT***

AWT stands for Abstract Window Tool kit. It is mainly used to improve applets with windows, panes, frames,dialog boxes, labels, text fields,check boxes, check box groups, menus, scroll bars, eventlisteners, layout managers ... etc into our own style. The abstract window toolkit related all classes are placed in the***java.awt*** package.

In the abstract window toolkit the main super class name is the Component and its subclass name is the Container. The Container class is a super class for the Window and Panel classes. The Window class is a super class for the Frame and Dialog classes. The Panel class is a super class for the Applet class. The AWT architecture is shown below

***ComponentClass***

This is a super class for the Container class and it can implement java.awt.image.ImageObserver,java.awt.MenuContainer,java.io.Serializable interfaces. This is a super class of the Label, TextFiled, Button, Checkbox … etc classes.

***ContainerClass***

This is a sub class of the Component class and it is a super class of the Window, Panel clases.

***Panel Class***

This is a container, and it has no border, title bar and menubars. It can hold componets on it.

Syntax is

Panel object=new Panel();

E.g.:

Panel p=new Panel();

***WindowClass***

This is a container, and it has no border and menubars, so use Frame orDialog to create a new window.

***Frame Class***

Frame is a subclass of the window class. It has border, title bar and menubars.

Syntax is

Frame object=new Frame(["title\_string"]);

E.g.: - 1

Frame f=new Frame();

E.g.: - 2

Frame f=new Frame("NICE");

The most important Frame class methods are explained below.

1. ***setSize() Method***

It can be used to set frame width and height with the specified number of pixels.

Syntax is

frame\_object.setSize(int width, int height);

E.g:

Frame f=new Frame();

f.setSize(100,100);

1. ***setTitle() Method***

It can be used to set title of a specified frame with the given string.

Syntax is

frame\_object.setTitle(string);

E.g.:

f.setTitle("NICE COMPUTERS");

1. ***setVisible() Method***

It can be used to display or hide a frame with true or false values.

Syntax is

frame\_object.setVisible(boolean);

E.g.:

f.setVisible(true);

1. ***setIconImage() Method***

It can be used to set icon of the frame with the given image.

Syntax is

frame\_object.setIconImage(image\_object);

E.g.:

Image img=getImage(getCodeBase(),"winter.jpg");

f.setIconImage(img);

***DialogClass***

It is same like as a Frame class, but it does not have the maximize and minimize buttons. It can be used to take some form of input from the user.

Syntax is

Dialog object=new Dialog(frame\_class\_object,["title\_string"]);

E.g.:

Dialogd=new Dialog(f,"Save Dialog");

The most important Dialog class methods are explained below.

1. ***setSize() Method***

It can be used to set dialogwidth and height with the specified number of pixels.

Syntax is

dialog\_object.setSize(int width, int height);

E.g:

Dialogd=new Dialog(f,"Save Dialog");

d.setSize(100,100);

1. ***setTitle() Method***

It can be used to set title of a specified dialog with the given string.

Syntax is

dialog\_object.setTitle(string);

E.g.:

d.setTitle("NICE COMPUTERS");

1. ***setVisible() Method***

It can be used to display or hide a dialog with true or false values.

Syntax is

dialog\_object.setVisible(boolean);

E.g.:

d.setVisible(true);

1. ***setIconImage() Method***

It can be used to set icon of the dialog with the given image.

Syntax is

dialog\_object.setIconImage(image\_object);

E.g.:

Image img=getImage(getCodeBase(),"winter.jpg");

d.setIconImage(img);

***CanvasClass***

This is sub class of the Component class. It represents a blank rectangle area.

Syntax is

Canvas object=new Canvas();

E.g.:

Canvas c=new Canvas();

***add() Method***

It is placed in the Container class. It can be used to add a component on the Panel, Applet or a Frame.

Syntax is

[applet/frame/panel\_object.]add(component);

E.g.:

Frame f=new Frame();

Label l1=new Label(NICE");

f.add(l1);

***repaint() Method***

It is placed in the Component class. It can be used to execute the paint method.

Syntax is

repaint();

E.g.:

repaint();

***Label Class***

It can be used to put single line headingson the frames, panels, applets … etc.

Syntax is

Label object=new Label(string);

E.g.: - 1

Label l1=new Label();

E.g.: - 2

Label l1=new Label("NICE");

The most important Label class methods are explained below.

1. ***setText() Method***

It can change the label text.

Syntax is

label\_object.setText(string);

E.g.:

Label l1=new Label();

l1.setText("Enter User Name");

1. ***getText() Method***

It can return text from the label.

Syntax is

label\_object.getText();

E.g.:

Label l1=new Label("Enter User Name");

String s=l1.getText();

1. ***setAlignment() Method***

Itcan align label into left, right orcenter positions. In this method *how* may be either Label.LEFT, Label.RIGHT or Label.CENTER.

Syntax is

label\_object.setAlignment(how);

E.g.:

l1.setAlignment(Label.LEFT);

***TextField Class***

It can be used to accept single line text as input.

Syntax is

TextField object=new TextField([size]);

E.g.:

TextField t1=new TextField(10);

The most important TextField class methods are explained below.

1. ***setText() Method***

It can change the textfield text.

Syntax is

textfield\_object.setText(string);

E.g.:

t1.setText("Rama");

1. ***getText() Method***

It can return current text from the textfield.

Syntax is

textfield\_object.getText();

E.g.:

String x;

x=t1.getText();

***Button Class***

It can be used to display a button and then perform actions by its push.

Syntax is

Button object=new Button(string);

E.g.:

Button b1=new Button("Button1");

The most important Button class methods are explained below.

1. ***setLabel() Method***

It can change the Button control's label with the given string.

Syntax is

button\_object.setLabel(string);

E.g.:

b1.setLabel("Submit");

1. ***getLabel() Method***

It can return current label from the Button control.

Syntax is

button\_object.getLabel();

Ex:-

String x;

x=b1.getLabel();

***Checkbox Class***

It displays a checkbox, that is used to selector deselect an item with ***on***or ***off*** states.

Syntax is

Checkbox object=new Checkbox(["label"][,boolean][,checkbox\_group]);

In the above syntax label specifies the checkbox value, boolean specifies the checkbox state &checkbox\_group is anobject of the CheckboxGroupclass. It is used to controlthe number of check boxes into a single group and select only a one checkbox at a time.

E.g.: -1

Checkbox c1=new Checkbox();

E.g.: -2

Checkbox c1=new Checkbox("Java");

E.g.: -3

CheckboxGroup cg=new CheckboxGroup();

Checkbox c1=new Checkbox("C",false,cg);

Checkbox c2=new Checkbox("C++",false,cg);

Checkbox c3=new Checkbox("Java",true,cg);

Checkbox c4=new Checkbox("Oracle",false,cg);

The most important Checkbox class methods are explained below.

1. ***setLabel() Method***

It can change the Checkbox control's label.

Syntax is

checkbox\_object.setLabel(string);

E.g.:

c1.setLabel("Java");

1. ***getLabel() Method***

It can return current label from the Checkbox control.

Syntax is

checkbox\_object.getLabel();

E.g.:

String x;

x=c1.getLabel();

1. ***setState() Method***

It can change the Checkbox control's state with either ***on*** or ***off*** with**true** or **false** values.

Syntax is

checkbox\_object.setState(boolean);

E.g.:

c1.setState(true);

1. ***getState() Method***

It can return current state from the Checkbox control.

Syntax is

checkbox\_object.getState();

E.g.:

Boolean x;

x=c1.getState();

***CheckboxGroup Class***

It can handle number of checkboxes into a group and select only a single checkbox at a time.

Syntax is

CheckboxGroup object=new CheckboxGroup();

E.g.:

CheckboxGroup cg=new CheckboxGroup();

Checkbox c1=new Checkbox("C",false,cg);

Checkbox c2=new Checkbox("C++",false,cg);

Checkbox c3=new Checkbox("Java",true,cg);

Checkbox c4=new Checkbox("Oracle",false,cg);

***Choice Class***

It can handle number of optionslike a popup menu andselect only a single optionat a time.

Syntax is

Choice object=new Choice();

E.g.:

Choice c1=new Choice();

The most important Choice class methods are explained below.

1. ***add() Method***

It can be used to add a new option to the Choice control.

Syntax is

choice\_object.add(string);

E.g.:

c1.add("Java");

1. ***addItem() Method***

It can be used to add a new option to the Choice control.

Syntax is

choice\_object.addItem(string);

E.g.:

c1.addItem("Java");

1. ***getSelectedItem()Method***

It can return current selected option from the Choice control.

Syntax is

choice\_object.getSelectedItem();

E.g.:

String s;

s=c1.getSelectedItem();

***List Class***

It is also same like as aChoice control, but it displays the number of values at a time. It can select a single or the number of options at a time.

Syntax is

List object=new List([int items-count][,boolean multi-select]);

In the above syntax items-count is used to display no.of options at a time, and the multi-select can be used to select a single or multiple options at a time.

E.g.: -1

List lst1=new List();

E.g.: -2

List lst1=new List(4,true);

The most important List class methods are explained below.

1. ***add() Method***

It can be used to add a new option to the List control.

Syntax is

list\_object.add(string);

E.g.:

lst1.add("Java");

1. ***addItem() Method***

It can be used to add a new option to the List control.

Syntax is

list\_object.addItem(string);

E.g.:

lst1.addItem("Java");

1. ***getSelectedItem()Method***

It can return current selected option from the List control.

Syntax is

list\_object.getSelectedItem();

E.g.:

String s;

s=lst1.getSelectedItem();

1. ***getSelectedItems()Method***

It can return current selected multiple options from the List control.

Syntax is

list\_object.getSelectedItems();

E.g.:

String[] s;

s=lst1.getSelectedItems();

***TextArea Class***

It can accept multi-line text as input.

Syntax is

TextArea object=new TextArea(String text,int num\_rows,int num\_cols);

E.g.:

TextArea ta1=new TextArea("NICE Computers",5,20);

The below methods are used to control the TextArea.

1. ***setText()Method***

It can change the text\_area control's text with the given string.

Syntax is

text\_area\_object.setText(string);

E.g.:

ta1.setText("NICE Computers");

1. ***getText()Method***

It can return total text from the text\_area control.

Syntax is

text\_area\_object.getText();

E.g.:

String s;

s=ta1.getText();

1. ***getSelectedText()Method***

It can return selected text from the text\_area control.

Syntax is

text\_area\_object.getSelectedText();

E.g.:

String s;

s=ta1.getSelectedText();

***Menus***

In the application development almost all applicationsare designed by the menus. A menubar can handle several menus. A menu can handle several menu items. The menus and their items structure is shown below.

Menus

FILE EDIT Menu bar

NEW

Menu items OPEN

SAVE

SAVE AS

The organization of MenuBar, Menu & MenuItem classes are shown below.

***MenuBar Class***

It can handle several menus and it is a sub class of the MenuComponent.

E.g.:

MenuBar mbar=new MenuBar();

***Menu Class***

It can be used to create menus for the menubar, without menus nothing is displayed on the menubar. Menu class is a sub class of the MenuItem class. It is a pulldown menu component.

Syntax is

Menu object=new Menu(string);

E.g.:

Menu m1=new Menu("File");

If we want to add a menu object on the menu bar, we must usethe add() method with below syntax & its example.

Syntax is

menubar-object.add(menu-object);

E.g.:

mbar.add(m1);

***MenuItemClass***

A MenuItem class is a sub class of the MenuComponent class. A menuitem is used to handle a labled command and is displayed on the menu. Without the menu items nothing is displayed on the pull down menu.

Syntax is

MenuItem object=new MenuItem(string);

E.g.:

MenuItem m11=new MenuItem("New");

If we want to add a menu item on thepull down menu, we must follow the below syntax and its example of the add() method.

Syntax is

menu-object.add(menuitem-object);

E.g.:

m1.add(m11);

***setMenuBar()Method***

The menubar is not added into the applet window. If we want to add a menubaron the frame, then we must use the setMenuBar() method.

E.g:

f.setMenuBar(mbar);

***PopupMenuClass***

It is a sub class of the Menuclass. It can create a popup menu to show on the specified component at a specified location.

Syntax is

PopupMenuobject=new PopupMenu();

E.g.:

PopupMenu pm=new PopupMenu();

If we want to add a menu item on thepopupmenu, we must follow the below syntax and its example of the add() method.

Syntax is

popupmenu-object.add(menuitem-object);

E.g.:

pm.add(m11);

***show()Method***

It can display a popup menu at a specified location on the spcifiede component.

Syntax is

popupmenu\_object.show(component\_object,int x,int y);

E.g:

pm.show(f,100,200);

***Scrollbar Class***

It can be used to addvertical and horizontal scrollbars. It is a sub class of the component class.

Syntax is

Scrollbar object=new Scrollbar(style,int init-value,int thumb,int min-value,int max-value);

In the above syntax the style may be either Scrollbar.HORIZONTAL or Scrollbar.VERTICAL. Scrollbar.VERTICAL is used to create a vertical scrollbar and Scrollbar.HORIZONTAL is used to create a horizontal scrollbar.

E.g.:

Scrollbar h=new Scrollbar(Scrollbar.HORIZONTAL,0,1,0,100);

The most important scrollbar methods are explained below.

***setValue() Method***

It can be used to set current value of the scrollbar.

Syntax is

scrollbar-object.setValue(int value);

E.g.:

hs.setValue(100);

***getValue() Method***

It can return current value from the scrollbar.

Syntax is

scrollbar-object.getValue();

E.g:

hs.getValue();

***Font Class***

It can be used to create a new font object by the specified font name, style and its size. It can be applied for the any type of component.

Syntax is

Font object=new Font(fontname,style,size);

In the above syntax the style may be either Font.PLAIN, Font.BOLD, Font.ITALIC or Font.UNDERLINE.

E.g.:

Font f=new Font("Arial",Font.BOLD,20);

***setFont()Method***

It is used to apply the font object settings to a component.

Syntax is

component-object.setFont(font);

E.g.:

b1.setFont(f);

**Event Handling**

***Event***

Event is a class in Java. An event can be used to change state of an object. The handling process of events in Java is called the event handling. In java the events are placed in the java.awt.event package. The mechanism of handling events is known as event handling.The most commonly used events are

1) ActionEvent

2) MouseEvents

3) MouseWheelEvent

4) KeyEvent

5) ItemEvent

6) AdjustmentEvent

7) WindowEvent

***ActionEvent Class***

Action events are mainly used to control the actions of the buttons& menu items. Action events are controlled by an interface called the **ActionListener**. If we want to create action events then we must follow the below steps.

* 1. Import java.awt.event.ActionEvent class.
  2. Import java.awt.event.ActionListener interface.
  3. The class is must implemented by an interface named ActionListener
  4. If we want to register an action event tothe specified object, we must add that object with the addActionListener() method.
  5. An interface ActionListener has a method called **public void actionPerformed(ActionEvent)**,so we must implement that method.

***MouseEvent Class***

Mouse events are the mouse related operations for the specified objects. Mouse events are controlled by the 2 different interfaces called **MouseListener** and **MouseMotionListener**. If we want to control mouse events then we must follow the below steps.

* 1. Import java.awt.event.MouseEvent class.
  2. Import java.awt.event.MouseListener interface.
  3. Import java.awt.event.MouseMotionListener interface.
  4. The class is must implemented by an interface named MouseListener and or MouseMotionListener.
  5. If we want to register a mouse event tothe specified object, we must add that object with the addMouseListener()or addMouseMotionListener()method.
  6. An interface MouseListener has5 methods and MouseMotionListener has 2 methods, so we must implement those all methods.

An interface MouseListener has the below methods.

a) public void mouseClicked(MouseEvent)

b) public void mousePressed(MouseEvent)

c) public void mouseReleased(MouseEvent)

d) public void mouseEntered(MouseEvent)

e) public void mouseExited(MouseEvent)

An interface MouseMotionListener has the below methods.

a) public void mouseDragged(MouseEvent)

b) public void mouseMoved(MouseEvent)

***MouseWheelEvent Class***

Mouse wheel events are the mouse wheel related operations for the specified objects. Mouse wheel events are controlled by an interface named**MouseWheelLlistener**. If we want to control mousewheel events then we must follow the below steps.

1. Import java.awt.event.MouseWheelEvent class.
2. Import java.awt.event.MouseWheelListener interface.
3. The class is must implemented by an interface named MouseWheelListener.
4. If we want to register a mouse wheel event to the specified object, we must add that object with the addMouseWheelListener() method.
5. An interface MouseWeelListener has a method public void mouseWheelMoved(MouseWheelEvent), so we must implement that method.

***KeyEvent Class***

Key events are the keyboard related operations for the specified objects.Key events are controlled by the **KeyListener** interface.If we want to control key events then we must follow the below steps.

1. Import java.awt.event.KeyEvent class.
2. Import java.awt.event.KeyListener interface.
3. The class is must implemented by an interface named KeyListener.
4. If we want to register a key event to the specified object, we must add that object with the addKeyListener() method.
5. An interface KeyListener has 3 methods, so we must implement those all methods.

An interface KeyListener has the below methods.

a) public void keyTyped(KeyEvent)

b) public void keyPressed(KeyEvent)

c) public void keyReleased(KeyEvent)

***ItemEvent Class***

Item events are mainly used to control item states of the specified objects like check boxes& radio buttons. Item events are controlled by an interface named**ItemListener**. If we want to control item events then we must follow the below steps.

1. Import java.awt.event.ItemEvent class.
2. Import java.awt.event.ItemListener interface.
3. The class is must implemented by an interface named ItemListener.
4. If we want to register anitem event to the specified object, we must add that object with the addItemListener() method.
5. An interface ItemListener has a method**public void itemStateChanged(ItemEvent)**, so we must implement that method.

***AdjustmentEvent Class***

Adjustment events are mainly used to control the scrollbar events. The adjustment events are controlled by an interface named**AdjustmentListener**. If we want to control adjustment events then we must follow the below steps.

1. Import java.awt.event.AdjustmentEvent class.
2. Import java.awt.event.AdjustmentListener interface.
3. The class is must implemented by an interface named AdjustmentListener.
4. If we want to register anadjustment event to the specified object, we must add that object with the addAdjustmentListener() method.
5. An interface AdjustmentListener has a method**public void adjustmentValueChanged(AdjustmentEvent)**, so we must implement that method.

***WindowEvent Class***

Window events are mainly used to control the action of a window. The window events are controlled by aninterface named **WindowListener**. If we want to control the window events then we must follow the below steps.

1. Import java.awt.event.WindowEvent class.
2. Import java.awt.event.WindowListener interface.
3. The class is must implemented by an interface named WindowListener.
4. If we want to register a windowevent to the specified object, we must add that object with the addWindoListener() method.
5. An interface WindowListener has 7method, so we must implement those all methods.

An interface WindowListener has the below methods.

a) public void windowOpened(WindowEvent)

b) public void windowClosing(WindowEvent)

c) public void windowClosed(WindowEvent)

d) public void windowIconified(WindowEvent)

e) public void windowDeiconified(WindowEvent)

f) public void windowActivated(WindowEvent)

g) public void windowDeactivated(WindowEvent)

***LAYOUT MANAGERS***

Layout managers are used to arrange the components into own style. The most commonly used layouts are

1. Null Layout
2. FlowLayout
3. BorderLayout
4. GridLayout
5. GridBagLayout
6. CardLayout

***null Layout***

it specifies, no layout. It is used to add components into our own style with specified size and location.

***setLayout() Method***

It can set a specified layout on the specified component.

Syntax is

setLayout(layout-object or null);

E.g.:

setLayout(null);

***setBounds() Method***

It can be used to adda specified component on the null layoutobject.

Syntax is

null\_layout\_object.setBounds(int x,int y,int width,int height);

E.g.:

b1.setBounds(100,50,300,20);

***FlowLayout***

This is a default layout in the Java applets. It is used to add controls into side by side means from left to right and then top to bottom. In this layout a fixed length is set to the controls and put a horizontal and vertical space between them.

Syntax is

FlowLayout object=new FlowLayout(int how,int hspace,int vspace);

In the above syntax "how" specifies, how to align objects inthe FlowLayout. The how may be either FlowLayout.LEFT or FlowLayout.RIGHT or FlowLayout.CENTER. FlowLayout.LEFT is used to arrange controls from the left side, FlowLayout.RIGHT can be used to arrange controls from right side and FlowLayout.CENTER is used to set controls on center. "hspace" and "vspace" specifies space between controls in the horizontal and vertical directions.

E.g.: - 1

FlowLayout fl=new FlowLayout();

E.g.:-2

FlowLayout fl=new FlowLayout(FlowLayout.LEFT);

E.g.:-3

FlowLayout fl=new FlowLayout(FlowLayout.RIGHT,20,20);

***GridLayout***

It can arrange the components into the number of rows and columns in a grid structure. It is very useful to design calculator like applications.

Syntax is

GridLayout object=new GridLayout(int rows,int columns,int hspace, int vspace);

E.g.:-1

GridLayout gl=new GridLayout();

E.g.:-2

GridLayout gl=new GridLayout(2,3);

E.g.:-3

GridLayout gl=new GridLayout(2,3,10,10);

***GridBagLayout***

It can arrange components into the center of both the vertical and horizontal directions in a window. It is very useful to set controls into the center of the window.

Syntax is

GridBagLayout obect=new GridBagLayout();

E.g.:

GridBagLayout gbl=new GridBagLayout();

***BorderLayout***

It can be used to set objects into 5 different locations ateast,north,west,south and center.

Syntax is

BorderLayout object=new BorderLayout(int hspace,int vspace);

E.g.: - 1

BorderLayout bl=new BorderLayout();

E.g.:-2

BorderLayout bl=new BorderLayout(10,10);

If we want to add a component to the above layout we must use the below add method.

Syntax is

add(component,position);

In the above syntax the position may be either BorderLayout.EAST, BorderLayout.NORTH,BorderLayout.WEST,BorderLayout.SOUTH or BorderLayout.CENTER.

E.g.:

Button b1=new Button("Button1");

add(b1,BorderLayout.NORTH);

***CardLayout***

It can be used to arrange an each component as a card. It is very useful to display each card at a time by the user request.In the CardLayout each card is palced a panel called a deck. Only a one card is displayed at a time.

Syntax is

CardLayout object=new CardLayout( int hspace,int vspace);

E.g.:-1

CardLayout cl=new CardLayout();

E.g.:-2

CardLayout cl=new CardLayout(10,10);

If we want to add a cardon the panel, we must follow the below syntax and its example.

Syntax is

panel-object.add(component,name);

E.g.:

p.add(winpan,"Windows");

The below methods are used to show a card at a time.

***first() Method***

It displaysthe first card on the cardlayout.

Syntax is

card\_layout\_object.first(container);

E.g.:

cl.first(p);

***last() Method***

It displays the last card on the cardlayout.

Syntax is

card\_layout\_object.last(container);

E.g.:

cl.last(p);

***next() Method***

It displays the next card on the cardlayout.

Syntax is

card\_layout\_object.next(container);

E.g.:

cl.next(p);

***previous() Method***

It displays the previous card on the cardlayout.

Syntax is

card\_layout\_object.previous(container);

E.g.:

cl.previous(p);

***show() Method***

It displays a specified card on the card layout.

Syntax is

card\_layout\_object.show(container,name);

E.g.:

cl.show(p, "Windows");

***SWINGS***

It is built on top of AWT API (Application Programming Interface). It is a part of Java Foundation Classes(JFC). It can provide platform independant and the lightweight components. Swing is an advanced package than AWT. Like AWT it is used to create labels, text fields, buttons, checkboxes, radio buttons … etc. Other than those components, it can create tabbed pane, scroll pane, trees, tables … etc. Swings are placed ina package named**javax.swing**. Like an Applet class the swing applicationsare controlled by a class named JApplet and it is place in the **javax.swing** package.

In the JApplet class, the default layout is null, so first we must set FlowLayout in the JApplet by following the be low examples.

E.g.: - 1

FlowLayout fl=new FlowLayout();

setLayout(fl);

E.g.: - 2

setLayout(new FlowLayout());

***JFrame Class***

JFrame is a subclass of the frame class. It has border, title bar and menubars.

Syntax is

JFrame object=new JFrame(["title\_string"]);

E.g.: - 1

JFrame f=new JFrame();

E.g.: - 2

JFrame f=new JFrame("NICE");

The most important Frame class methods are explained below.

1. ***setSize() Method***

It can be used to set frame width and height with the specified number of pixels.

Syntax is

jframe\_object.setSize(int width, int height);

E.g:

JFrame f=new JFrame();

f.setSize(100,100);

1. ***setTitle() Method***

It can be used to set title of a specified frame with the given string.

Syntax is

jframe\_object.setTitle(string);

E.g.:

f.setTitle("NICE COMPUTERS");

1. ***setVisible() Method***

It can be used to display or hide a frame with true or false values.

Syntax is

jframe\_object.setVisible(boolean);

E.g.:

f.setVisible(true);

1. ***setDefaultCloseOperation()Method***

It can close the frame, when we click on the close button of the frame.

Syntax is

jframe\_object.setDefaultCloseOperation(jframe\_event);

E.g.:

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

***JDialogClass***

It is same like as a JFrame class, but it does not have the maximize and minimize buttons. It can be used to take some form of input from the user.

Syntax is

JDialog object=new JDialog(jframe\_class\_object,["title\_string"]);

E.g.:

JDialogd=new JDialog(f,"Save Dialog");

The most important JDialog class methods are explained below.

1. ***setSize() Method***

It can be used to set jdialogwidth and height with the specified number of pixels.

Syntax is

jdialog\_object.setSize(int width, int height);

E.g:

JDialogd=new JDialog(f,"Save Dialog");

d.setSize(100,100);

1. ***setTitle() Method***

It can be used to set title of a specified jdialog with the given string.

Syntax is

jdialog\_object.setTitle(string);

E.g.:

d.setTitle("NICE COMPUTERS");

1. ***setVisible() Method***

It can be used to display or hide a jdialog with true or false values.

Syntax is

jdialog\_object.setVisible(boolean);

E.g.:

d.setVisible(true);

***JPanel Class***

This is a container, and it has no border, title bar and menubars. It can hold componets on it. It is a sub class of the JComponent class.

Syntax is

JPanel object=new JPanel();

E.g.:

JPanel p=new JPanel();

***ImageIcon Class***

It can be used to handle a specified image to put on acomponent as an icon. It is very useful to display components into the GUI style.

Syntax is

ImageIcon object=new ImageIcon(string filename);

E.g.:

ImageIcon i1=new ImageIcon("Mouse.gif");

***JLabel Class***

It can be used to display a read only text. This is a sub class of the JComponent class.

Syntax is

JLabel object=new JLabel(string label,icon,int horizontal\_alignment);

In the above syntax the alignment may be either JLabel.CENTER, JLabel.LEFT or JLabel.RIGHT.

E.g.:

JLabel l1=new JLabel("Mouse",i1,JLabel.CENTER);

***JTextField Class***

It can be used to accept a single line text from the input like an AWTTextField. This is a sub class of the JTextComponent class.

Syntax is

JTextField object=new JTextField(string default\_text,int size);

E.g.:

JTextField t1=new JTextField("NICE", 20);

***JPasswordField Class***

It can be used to accept a single line text as password format from the input. This is a sub class of the JTextField class.

Syntax is

JPasswordField object=new JPasswordField(string default\_text,int size);

E.g.:

JPasswordField p1=new JPasswordField("NICE", 20);

***JButton Class***

It can be used to create a labeled button with the specified image icon and or a label. It can perform an action when the button is pressed. This is a sub class of the AbstractButton class.

Syntax is

JButton object=new JButton(string label,icon);

E.g.:

JButton b1=new JButton("Button1",i1);

JButton class has the below methods to control an imageicon.

***setPressedIcon() Method***

It displays a specified icon at the time of the component is in the pressed state.

Syntax is

component-object.setPressedIcon(icon);

E.g.:

b1.setPressedIcon(i1);

***setDisabledIcon() Method***

It displays a specified icon at the time of the specified component is in the disabled state.

Syntax is

component-object.setDisabledIcon(icon);

E.g.:

b1.setDisabledIcon(i1);

***setSelectedIcon() Method***

It displays a specified icon at the time of a specified component is in the selected state.

Syntax is

component-object.setSelectedIcon(icon);

E.g.:

b1.setSelectedIcon(i1);

***setRolloverIcon() Method***

It display a specified icon at the time of the specified component is in the rollover state.

Syntax is

component-object.setRolloverIcon(icon);

E.g.:

b1.setRolloverIcon(i1);

***JCheckBox Class***

It can be used to select and option with on or off states with true or false values. This is a sub class of the JToggleButton class and it is sub class of the AbstractButton class.

Syntax is

JCheckBox object=new JCheckBox(string label,icon,boolean state);

E.g.:

JCheckBox c1=new JCheckBox("Java",i1,true);

***JRadioButton Class***

It can be used to create a radio button by selecting only a one option at a time.This is a sub class of the JToggleButton class and it is sub class of the AbstractButton class. It should be added in ButtonGroup to select only one button at a time.

Syntax is

JRadioButton object=new JRadioButton(string label,icon,boolean state);

E.g.:

JRadioButton rb1=new JRadioButton("Java",i1,true);

***ButtonGroup class***

It can be used to create a button group object to add the specified number of radio buttons into a group. In a button group,we can select only a one radio button at a time.

Syntax is

ButtonGroup object=new ButtonGroup();

E.g.:

ButtonGoup bg=new ButtonGroup();

If we want to add a radio button into the button group, we must follow the below add method syntax and its example.

Syntax is

buttongroup-object.add(radiobutton-object);

E.g.:

bg.add(rb1);

***JComboBox Class***

It can create a choice control with several items and select only a single item at a time. It is a sub class of the JComponent class.

Syntax is

JComboBox object=new JComboBox();

E.g.:

JComboBox cb1=new JComboBox();

***addItem() Method***

If we want to add an option into the JComboBox control, we must follow the below syntax and its example.

Syntax is

jcombobox-object.addItem(string);

E.g.:

cb1.addItem("Java");

***removeItem() Method***

If we want to remove an item from the JComboBox control, we must follow the below syntax and its example.

Syntax is

jcombobox-object.removeItem(string);

E.g.:

cb1.removeItem("Java");

***JList Class***

It can create a choice control with several items and select one or more item at a time. It is a sub class of the JComponent class.

Syntax is

JList object=new JList(string[]);

E.g.:

String[] s={"C","C++","Java","MS.Net","Oracle"};

JListl=new JList(s);

***setSelectedIndex() Method***

It can be used to change an index of the selected item.

Syntax is

jlist-object.setSelectedIndex(int);

E.g.:

l.setSelectedIndex(1);

***getSelectedIndex() Method***

It can return an index of the current selected item.

Syntax is

jlist-object.getSelectedIndex();

E.g.:

int x=l.getSelectedIndex();

***getSelectedValue() Method***

It can return value of the current selected item.

Syntax is

jlist-object.getSelectedValue();

E.g.:

String x=l.getSelectedValue().toString();

***JOptionPane Class***

It can displaya message dialog box, confirm dialog box andan input dialog box. It is a sub class of the JComponent class.

Syntax is

JOptionPane object=new JOptionPane();

E.g.:

JOptionPane op=new JOptionPane();

***showMessageDialog() Method***

It can display a message dialog box.

Syntax is

joptionpane-object.showMessageDialog(parent\_component,message);

E.g.:

op.showMessageDialog(f,"Welcome to NICE COMPUTERS");

***showConfirmDialog() Method***

It can display a confirm dialog box with Yes, No & Cancel options. It can return 0 for Yes, 1 for No and 2 for Cancel.

Syntax is

joptionpane-object.showConfirmDialog(parent\_component,message);

E.g.:

int x=op.showConfirmDialog(f,"Are you sure?");

***showInputDialog() Method***

It can display an input dialog box to request from the user. It can return string type result.

Syntax is

joptionpane-object.showInputDialog(parent\_component,message);

E.g.:

String x=op.showinputDialog(f,"Enter User Name");

***JScrollBar Class***

It can create a horizontal or a vertical scrollbar. It is a sub class of the JComponent class.

Syntax is

JScrollBar object=new JScrollBar(int orientation,int value,int extent,int min,int max);

In the above syntax the orientation may be either 0 or 1. 0 specifies horizontal scroll bar and 1 specifies the vertical scroll bar.

E.g.:

JScrollBar hsb=new JScrollBar(0,5,2,1,100);

***setValue() Method***

It can be used to set current value of the scrollbar.

Syntax is

jscrollbar-object.setValue(int);

E.g.:

hsb.setValue(10);

***getValue() Method***

It can return current value from the scrollbar.

Syntax is

jscrollbar-object.getValue();

E.g.:

Int x=hsb.getValue();

***JMenuBar Class***

It can handle several menus and it is a sub class of the JComponent.

E.g.:

JMenuBar mbar=new JMenuBar();

***JMenu Class***

It can be used to create menus for the menubar, without menus nothing is displayed on the menubar. Menu class is a sub class of the JMenuItem class. It is a pull down menu component.

Syntax is

JMenu object=new JMenu(string);

E.g.:

JMenu m1=new JMenu("File");

If we want to add a menu object on the menu bar, we must usethe add() method with below syntax & its example.

Syntax is

jmenubar-object.add(jmenu-object);

E.g.:

mbar.add(m1);

***JMenuItem Class***

A MenuItem class is a sub class of the AbstractButton class. A menuitem is used to handle a labled command and is displayed on the menu. Without the menu items nothing is displayed on the pull down menu.

Syntax is

JMenuItem object=new JMenuItem(string);

E.g.:

JMenuItem m11=new JMenuItem("New");

If we want to add a menu item on thepull down menu, we must follow the below syntax and its example of the add() method.

Syntax is

jmenu-object.add(jmenuitem-object);

E.g.:

m1.add(m11);

***JCheckBoxMenuItem Class***

JCheckBoxMenuItem class is a sub class of the JMenuItem class. A JCheckBoxMenuItem is used to handle a checkbox command and is displayed on the menu.

Syntax is

JCheckBoxMenuItem object=new JCheckBoxMenuItem(string);

E.g.:

JCheckBoxMenuItem m11=new JCheckBoxMenuItem("Wrap Text");

***setJMenuBar()Method***

The menubar is not added into the japplet widow. If we want to add a menubaron the jframe, then we must use the setJMenuBar() method.

E.g:

f.setJMenuBar(mbar);

***JPopupMenu Class***

It is a sub class of the Menu class. It can create a popup menu to show on the specified component at a specified location.

Syntax is

JPopupMenuobject=new JPopupMenu();

E.g.:

JPopupMenu pm=new JPopupMenu();

If we want to add a menu item on thepopup menu, we must follow the below syntax and its example of the add() method.

Syntax is

jpopupmenu-object.add(jmenuitem-object);

E.g.:

pm.add(m11);

***show()Method***

It can display a popup menu at a specified location on the spcifiede component.

Syntax is

jpopupmenu\_object.show(component\_object,int x,int y);

E.g:

pm.show(f,100,200);

***JSeparator Class***

It can be used to generate a divider line in either horizontal or vertical direction.It is sub class of the JComponent class.

Syntax is

JSeparatorobject=new JSeparator(int orientation);

E.g.:

JSeparatorsp=new JSeparator(0);

***JProgressBar Class***

It can be used to display the progress of the task, either horizontal or vertical direction. It is sub class of the JComponent class.

Syntax is

JProgressBarobject=new JProgressBar(int orientation,int min,in max);

E.g.:

JProgressBarpb=new JProgressBar(0,1,100);

***setValue()Method***

It can set current value of the progressbar.

Syntax is

jprogressbar\_object.setValue(int);

E.g:

pb.setValue(50);

***setStringPainted()Method***

If it is true, it can display a string on the progressbar.

Syntax is

jprogressbar\_object.setStringPainted(boolean);

E.g:

pb.setStringPainted(true);

***JColorChooser Class***

It can be used to display the progress of the task, either horizontal or vertical direction. It is sub class of the JComponent class.

Syntax is

JColorChooser object=new JColorChooser();

E.g.:

JColorChoosercc=new JColorChooser();

***showDialog()Method***

It can display the colorchooser dialog.

Syntax is

jcolorchooser\_object.showDialog(parent\_component,title,init\_color);

E.g:

cc.showDialog(f,"Select a Color",Color.RED);

***JSlider Class***

It can create a slider, to select a value from the specified range.

Syntax is

JSliderobject=new JSlider();

E.g.:

JSliders=new JSlider();

***setMinorTickSpacing()Method***

It can set minor tick spacing to the slider.

Syntax is

jslider\_object.setMinorTickSpacing(int);

E.g:

s.setMinorTickSpacing(2);

***setMajorTickSpacing()Method***

It can set major tick spacing to the slider.

Syntax is

jslider\_object.setMajorTickSpacing(int);

E.g:

s.setMajorTickSpacing(10);

***setPaintTicks()Method***

It can set tick marks on the slider.

Syntax is

jslider\_object.setPaintTicks(boolean);

E.g:

s.setPaintTicks(true);

***setPaintLabels()Method***

It can set labels on the slider.

Syntax is

jslider\_object.setPaintLabels(boolean);

E.g:

s.setPaintLabels(true);

***setPaintTrack()Method***

It can set a track on the slider.

Syntax is

jslider\_object.setPaintTrack(boolean);

E.g:

s.setPaintTrack(true);

***JSpinner Class***

It displays a single line input field, to select a number from the ordered sequence.

Syntax is

JSpinnerobject=new JSpinner();

E.g.:

JSpinners=new JSpinner();

***getValue()Method***

It can return current value from the spinner object.

Syntax is

jspinner\_object.getValue();

E.g:

System.out.print(s.getValue());

f.setIconImage(img);

***JFileChooser Class***

This is a dialog box to select a file from the user. It can hold componets on it. It is a sub class of the JComponent class.

Syntax is

JFileChooser object=new JFileChooser(path);

E.g.:

JFileChooserfc=new JFileChooser();

***showOpenDialog()Method***

It can display a open dialog, used to select a file.

Syntax is

jfilechooser\_object.showOpenDialog(component);

E.g:

fc.showOpenDialog(null);

***showSaveDialog()Method***

It can display a save dialog, used to save a file.

Syntax is

jfilechooser\_object.showSaveDialog(component);

E.g:

fc.showSaveDialog(null);

***getSelectedFile()Method***

It can return a selected file from the FileChooser.

Syntax is

jfilechooser\_object.getSelectedFile();

E.g:

fc.getSelectedFile();

***JToggleButton Class***

This is a two states button to switch ON or OFF.

Syntax is

JToggleButton object=new JToggleButton(string);

E.g.:

JToggleButtontb=new JToggleButton("ON");

***isSelected()Method***

It can return the selected state of the toggle button.

Syntax is

jtogglebutton\_object.isSelected();

E.g:

tb.isSelected();

***setText()Method***

It can be used change text on the toggle button.

Syntax is

jtogglebutton\_object.setText(string);

E.g:

tb.setText("OFF");

***JToolBar Class***

This is a container, used to maintain the number of components.

Syntax is

JToolBar object=new JToolBar();

E.g.:

JToolBar tb=new JToolBar();

***add()Method***

It can be used add a component on the toolbar.

Syntax is

jtoolbar\_object.add(component);

E.g:

tb.add(new JButton("OK"));

***JTextAreaClass***

It can be used to accept multi-line text as input with the specified number of rows and columns. It is sub class of the JTextComponent class.

Syntax is

JTextArea object=new JTextArea(string text,int rows,int columns);

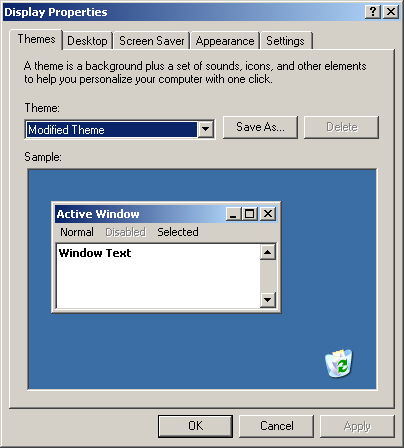
E.g.:

JTextArea ta=new JTextArea("NICE COMPUTERS",5,20);

***JTabbedPane Class***

It can be used to handle several tabs and an each tab can be used to maintain several components on it. An each tab components are placed in a separate class and that class must extended by the**JPanel**class. This is a sub class of the JComponent class. JTabbedPane is displayed into the below style.

Tabs



Syntax is

JTabbedPane object=new JTabbedPane();

E.g.:

JTabbedPane jtp=new JTabbedpane();

***addTab() Method***

If we want to add a tab on the JTabbedPane, we must follow the below syntax and its example.

Syntax is

tabbedpane-object.addTab(string label,component);

E.g.:

jtp.addTab("Buttons",new Buttons());

***JScrollPane Class***

It can used to create new scrollable objects with both the vertical and horizontal scroll bars. It is very useful to scroll objects from top to bottom and left to right positions.This is a sub class of the JComponent class.

Syntax is

JScrollPane object=new JScrollPane(panel,vscrollbar,hscrollbar);

E.g.:

JScrollPane jsp=new JScrollPane(jp,v,h);

If we want to create a vertical and horizontal scrollbar, we must follow the below syntax and its example.

Syntax is

int variable=ScrollPaneConstants.scrollbar;

In the above syntax the scrollbar may be either

HORIZONTAL\_SCROLLBAR\_ALWAYS,

VERTICAL\_SCROLLBAR\_ALWAYS,

HORIZONTAL\_SCROLLBAR\_AS\_NEEDED

orVERTICAL\_SCROLLBAR\_AS\_NEEDED.

E.g.:

int h=ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_ALWAYS;

***JTree Class***

A tree can handle several items, and an each item can handle several sub items. The items are expended or collapsed by the mouse click. The JTree class is placed in the **javax.swing.tree** package. If we want to create a top node or the other nodes of a tree, we must follow the below syntax and its example.

Syntax is

DefaultMutableTreeNode object=new DefaultMutableTreeNode(string);

E.g.:

DefaultMutableTreeNode top=new DefaultMutableTreeNode("Engineering");

If we want to create an object to the JTree class, we must follow the below syntax and its example.

Syntax is

JTree object=new JTree(top-node-object);

E.g.:

JTree jt=new JTree(top);

***JTable Class***

A table is formed by the number of rows and columns. The top row in a table is called a headings row, and the all other rows are called the data rows.

Syntax is

JTable object=new JTable(data,heading);

E.g.:

JTable jt=new JTable(data,heads);

If we want to createa heading row for the table, we must follow the below example.

E.g.:

String[] heads={"NAME", "JOB","SALARY"};

If we want to createdata rowsfor the table, we must follow the below example.

E.g.:

Object[][] data={ {"Rama", "Manager","120000"},

{"Sita","Sales Girl","20000"},

{"Raja","Clerk","30000"},

{"Nagesh","Clerk","40000"}

};

***setToolTipText() Method***

It can create a tooltip text for any JComponent.

Syntax is

jcomponent-object.setToolTiptext(string);

E.g.:

JButton b1=new JButton("Button1");

b1.setToolTipText("Click Me");

***How to create an executable JAR(Java Archive) file***

Step 1: Write a java program and compile.

Step 2: Create manifest file with any name (E.g:. myfile.mf) by the below code

Main-Class: Prog1

Step 3: Use the below jar command to create an executable jar file.

jar –cvmf myfile.mf Prog1.jar Prog1.class

Step 4: The command to execute .jar file from command prompt

java -jar Prog1.jar

Step 5: But, in GUI give double click on .jar file

**JDBC (Java Database Connectivity)**

**What is JDBC Driver ?**

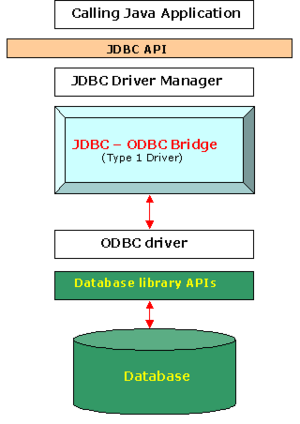
JDBC driver is a software component enabling a Java application to interact with a database. JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers.

To connect with individual databases, JDBC (the Java Database Connectivity API) requires drivers for each database. The JDBC driver gives out the connection to the database and implements the protocol for transferring the query and result between client and database.

JDBC technology drivers fit into one of four categories.

1. JDBC-ODBC Bridge Driver
2. Native-API Driver
3. Network-Protocol Driver(MiddleWare Driver)
4. Database-Protocol Driver(Pure Java Driver)

**Type 1 Driver (JDBC-ODBC Bridge Driver)**

[](http://en.wikipedia.org/wiki/File:JDBC_driver.png)

Schematic of the JDBC-ODBC bridge

The JDBC type 1 driver, also known as the JDBC-ODBC bridge driver, is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls.

The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the underlying operating system the JVM is running upon. Also, use of this driver leads to other installation dependencies; for example, ODBC must be installed on the computer having the driver and the database must support an ODBC driver. The use of this driver is discouraged if the alternative of a pure-Java driver is available. The other implication is that any application using a type 1 driver is non-portable given the binding between the driver and platform. This technology isn't suitable for a high-transaction environment. Type 1 drivers also don't support the complete Java command set and are limited by the functionality of the ODBC driver.

Sun provides a JDBC-ODBCBridgedriver:**sun.jdbc.odbc.JdbcOdbcDriver**. This driver is native code and not Java, and is closed source.

If a driver has been written so that loading it causes an instance to be created and also calls **DriverManager.registerDriver** with that instance as the parameter (as it should do), then it is in the DriverManager's list of drivers and available for creating a connection.

It may sometimes be the case that more than one JDBC driver is capable of connecting to a given URL. For example, when connecting to a given remote database, it might be possible to use a JDBC-ODBC bridge driver, a JDBC-to-generic-network-protocol driver, or a driver supplied by the database vendor. In such cases, the order in which the drivers are tested is significant because the DriverManager will use the first driver it finds that can successfully connect to the given URL.

First the DriverManager tries to use each driver in the order it was registered. (The drivers listed in jdbc.drivers are always registered first.) It will skip any drivers that are untrusted code unless they have been loaded from the same source as the code that is trying to open the connection.

It tests the drivers by calling the method Driver.connect on each one in turn, passing them the URL that the user originally passed to the method **DriverManager.getConnection**. The first driver that recognizes the URL makes the connection.

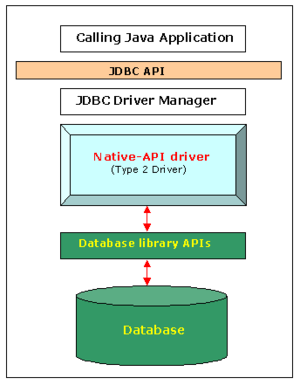
***Advantages***

* 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 Overhead 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.

**Type 2 Driver (Native-API Driver)**

[](http://en.wikipedia.org/wiki/File:Native_API_driver.png)

Schematic of the Native API driver

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

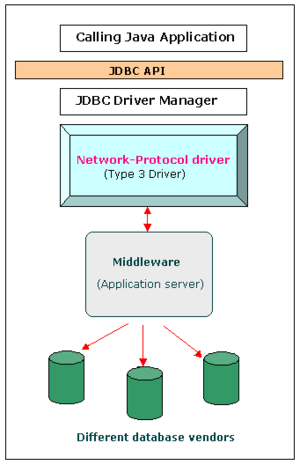
***Advantages***

* 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

**Type 3 Driver (Network-Protocol Driver/MiddleWare Driver)**

[](http://en.wikipedia.org/wiki/File:Network_Protocol_driver.png)

Schematic of the Network Protocol driver

The JDBC type 3 driver, also known as the **Pure Java Driver** for Database Middleware, is a database driver implementation which makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.

This differs from the type 4 driver in that the protocol conversion logic resides not at the client, but in the middle-tier. Like type 4 drivers, the type 3 driver is written entirely in Java. The same driver can be used for multiple databases. It depends on the number of databases the middleware has been configured to support. The type 3 driver is platform-independent as the platform-related differences are taken care of by the middleware. Also, making use of the middleware provides additional advantages of security and firewall access.

***Functions***

* Sends JDBC API calls to a middle-tier net server that translates the calls into the DBMS-specific network protocol. The translated calls are then sent to a particular DBMS.
* Follows a three tier communication approach.
* Can interface to multiple databases - Not vendor specific.
* The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
* Thus the client driver to middleware communication is database independent.

***Advantages***

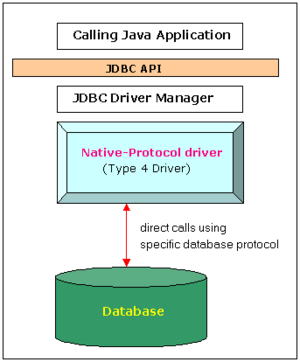
* Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client. The client need not be changed for a new database.
* The middleware server (which can be a full fledged J2EE Application server) can provide typical middleware services like caching (of connections, query results, etc.), load balancing, logging, and auditing.
* A single driver can handle any database, provided the middleware supports it.

E.g.:-IDA Server

***Disadvantages***

* Requires database-specific coding to be done in the middle tier.
* The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.

**Type 4 Driver (Database-Protocol Driver)**

[](http://en.wikipedia.org/wiki/File:Native_Protocol_driver.png)

Schematic of the Native-Protocol driver

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

Written completely in Java, type 4 drivers are thus platform independent. They install inside the Java Virtual Machine of the client. This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 3 drivers, it does not need associated software to work.

As the database protocol is vendor specific, the JDBC client requires separate drivers, usually vendor supplied, to connect to different types of databases. This type includes, for example, the widely used **Oracle thin driver**.

***Advantages***

* Completely implemented in Java to achieve platform independence.
* These drivers don't translate the requests into an intermediate format (such as ODBC).
* The client application connects directly to the database server. No translation or middleware layers are used, improving performance.
* The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.

***Disadvantages***

* Drivers are database dependent, as different database vendors use widely different (and usually proprietary) network protocols.

The first thing you need to do the following steps:

***1. Install Java and JDBC on your machine.***

To install both the Java platform and the JDBC API, download and install the latest release of JDK (Java Development Kit). When you install the JDK, you will get JDBC as well.

***2. Install a driver on your machine.***

Your driver should include instructions for installing it. For JDBC drivers written for specific DBMSs, installation consists of just copying the driver onto your machine, there is no special configuration needed. The JDBC-ODBCBridge driver is not quite as easy to set up. If you download JDK, you will automatically get the JDBC-ODBCBridge driver, which does not itself require any special configuration. ODBC, however does. If you do not already have ODBC on your machine, you will need to see your ODBC driver vendor for information on installation and configuration.

***3. Install your DBMS if needed.***

If you do not already have a DBMS installed, you will need to follow the vendor's instructions for installation. Most users will have a DBMS installed and will be working with an established database.

***Configuring Database:***

Configuring a database is not at all difficult, but it requires special permissions and is normally done by a database administrator.

First, open the **control panel**. You might find **Administrative tools** select it, again you may find shortcut for **Data Sources (ODBC)**. When you open the Data Sources icon, you'll see a **ODBC Data Source Administrator** dialog window with the number of tabs, including User DSN, System DSN, File DSN,… etc., in which DSN means **Data Source Name**, Select **User DSN** and add a new entry there, Select appropriate driver for the data source or directory where database lives. You can name the entry anything you want, assume here we are giving our data source name as **"MySource"**.

***JDBC Database Access***

JDBC was designed to keep simple things simple. This means that the JDBC API makes everyday database tasks, like simple SELECT statements, very easy.

* Import a package java.sql.\* ;This package provides you set of all classes that enables a network interface between the front end and back end database.
* DriverManager will create a Connection object.
* java.sql.Connection interface represents a connection with a specific database. Methods of connection is close(), creatStatement(), prepareStatement(), commit() and prepareCall().
* Statement interface used to interact with database via the execution of SQL statements. Methods of this interface are executeQuery(), executeUpdate(), execute() and getResultSet().
* A ResultSet is returned when you execute an SQL statement. It maintains a pointer to a row within the tabular results. Methods of this interface are next(), getBoolean(), getByte(), getDouble(), getString() close() and getInt().

***Establishing a Connection***

The first thing you need to do is establish a connection with the DBMS you want to use. This involves two

steps: (1) Loading the driver and (2) Making the connection.

***1. Loading the Driver***

Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBCBridge driver, the following code will load it

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code:

Class.forName("jdbc.DriverXYZ");

***2. Making the Connection***

The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

Connection con=DriverManager.getConnection(url,"myLogin", "myPassword");

If you are using the JDBC-ODBCBridge driver, the JDBC URL will start with jdbc:odbc:. The rest of the URL is generally your data source name or database system. So, if you are using ODBC to access an ODBC data source called "MySource" for example, your JDBC URL could be **jdbc:odbc:MySource**. In place of "myLogin" you put the name you use to log in to the DBMS, in place of "myPassword" you put your password for the DBMS. So if you loginto your DBMS with a login name of **"scott"** and a password of **"tiger"** just these two lines of code will establish a connection.

Connection con=DriverManager.getConnection("jdbc:odbc:MySource","scott","tiger");

The connection returned by the method DriverManager.getConnection() is an open connection you can use to create JDBC statements that pass your SQL statements to the DBMS. In the previous example, con is an open connection and we will use it in the forth coming examples.

***Creating JDBC Statements***

A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate.It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt.

Statement stmt=con.createStatement();

At this point stmt exists, but it does not have an SQL statement to pass on to the DBMS. We need to supply that to the method we use to execute stmt. For example, in the following code fragment, we supply executeUpdate with the SQL statement from the example above.

stmt.executeUpdate("create table emp(ename varchar2(20),job varchar2(15),salary number(8))");

Since the SQL statement will not quite fit on one line on the page, we have split it into two strings concatenated by a plus sign (+) so that it will compile.

***Executing Statements***

Statements that create a table, alter a table or drop a table are all examples of DDL statements and are executed with the method executeUpdate. The method executeUpdate is also used to execute SQL statements that update a table. In practice, executeUpdate is used far more often to update tables than it is to create them because a table is created once but may be updated many times. The method used most often for executing SQL statements is executeQuery. This method is used to execute SELECT statements, which comprise the vast majority of SQL statements.

***Entering Data into a Table***

We have shown how to create the table **EMP** by specifying the names of the columns and the data types to be stored in those columns, but this only sets up the structure of the table. The table does not yet contain any data. We will enter our data into the table one row at a time, supplying the information to be stored in each column of that row. Note that the values to be inserted into the columns are listed in the same order that the columns were declared when the table was created, which is the default order.The following code inserts one row of data.

Statement stmt=con.createStatement();

stmt.executeUpdate("insert into emp values('Rama',’Manager’,45000)");

Note that we use single quotation marks around the employee name because it is nested within double quotation marks. For most DBMSs, the general rule is to alternate double quotation marks and single quotation marks to indicate nesting.

The code that follows inserts a second row into the table **EMP**. Note that we can just reuse the Statement object stmt rather than having to create a new one for each execution.

stmt.executeUpdate("insert into emp values('Sita','Clerk',25000)");

***Getting Data from a Table***

Now that the table **EMP** has values in it, we can write a SELECT statement to access those values. The star (\*) in the following SQL statement indicates that all columns should be selected. Since there is no WHERE clause to narrow down the rows from which to select, the following SQL statement selects the whole table.

sql> select \* from emp;

***Retrieving Values from Result Sets***

We now show how you send the above SELECT statements from a program written in the Java programming language and how you get the results we showed.

JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it.

ResultSet rs=stmt.executeQuery("select ename,job from emp");

The following code accesses the values stored in the current row of rs. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs.

ResultSet rs=stmt.executeQuery("select \* from emp");

while(rs.next())

{Stringn=rs.getString("ename");

String j=rs.getString("job");

ints=rs.getInt("salary");

System.out.println(n + " " + j + " " + s);

}

***Updating Tables***

Suppose that after a period of time we want to update the **SALARY**column in the table **EMP**. The SQL statement to update one row might look like this.

stmt.executeUpdate("update emp set salary=60000 where salary=45000");

***Using try and catch Blocks***

Something else all the sample applications include is the try and catch blocks. These are the Java programming language's mechanism for handling exceptions. Java requires that when a method throws an exception, there be some mechanism to handle it. Generally a catch block will catch the exception and specify what happens (which you may choose to be nothing).In the sample code, we use two try blocks and two catch blocks.

The first try block contains the method Class.forName, from the java.lang package. This method throws a ClassNotFoundException, so the catch block immediately following it deals with that exception.

The second try block contains JDBC methods, which all throw SQLException, so one catch block at the end of the application can handle all of the rest of the exceptions that might be thrown because they will all be SQLException objects.

***Retrieving Exceptions***

JDBC lets you see the warnings and exceptions generated by the DBMS and by the Java compiler. To see exceptions, you can have a catch block to print them out. For example, the following two catch blocks from the sample code print out a message explaining the exception.

try

{

Calss.forName("sun.jdbc.odbc.JdbcOdbcDriver");

}

catch(ClassNotFoundException e)

{

System.println("Class Not Found Exception: "+e.getMessage());

}

try

{

// Code that could generate an exception goes here.

// If an exception is generated, the catch block below

// will print out information about it.

}

catch(SQLException e)

{

System.err.println("SQLException: "+e.getMessage());

}

***PreparedStatement***

A PreparedStatement is a subinterface of Statement. It can be used to execute a query with parameters. That means it can accept input parameters at runtime. All the parameters in JDBC are represented by the ?symbol.

PreparedStatement ps=con.prepareStatement("insert into emp values(?,?,?)");

After the above statement, we must supply values to the parameters with the set methods, like setString(), setInt(), setLong(), setFloat(), setDouble().

ps.setString(1,"Raja");

ps.setString(2,"Manager");

ps.setInt(3,49000);

After the above steps, we must execute the above query using the below statement.

ps.executeUpdate();

***CallableStatement***

If we want to write a function to add 2 numbers, we must follow the below example in oracle's PL/SQL.

create or replace function fun1(a number,b number) return number is

c number;

begin

c:=a+b;

return c;

end;

/

If we want to write a procedure to add 2 numbers, we must follow the below example in oracle's PL/SQL.

create or replace procedure proc1(a number,b number) is

c number;

begin

c:=a+b;

dbms\_output.put\_line('Sum='||c);

end;

/

A CallableStatement interface is used to call the stored procedures and functions.

***If we want to call a function, we must follow the below statements.***

CallableStatement cs=con.prepareCall("{?=call fun1(?,?)}");

After the above statement, we must supply values to the parameters with the set methods, like setString(), setInt(), setLong(), setFloat(), setDouble().

cs.setInt(1,25);

cs.setInt(2,45);

After the above statements, we must register an output parameter value, with the below statement.

cs.registerOutParameter(1,Types.INTEGER);

After the above steps, we must execute the above query and display its result, using the below statement.

cs.execute();

System.out.print(st.getInt(1));

***If we want to call a procedure, we must follow the below statements.***

CallableStatement cs=con.prepareCall("{call proc1(?,?)}");

After the above statement, we must supply values to the parameters with the set methods, like setString(), setInt(), setLong(), setFloat(), setDouble().

cs.setInt(1,25);

cs.setInt(2,45);

After the above steps, we must execute the above query using the below statement.

cs.execute();

***Transaction Management***

Transaction management can be used to commit or rollback transcations in RDBMS(Relational Data Base Management) by following the below statements.

con.commit();

con.rollback();

**SERVLETS**

Servlets are small programs that are executed on the server side of a web connection. Just as applets dynamically extend the functionality of a web browser, servlets dynamically extend the functionality of a web server.

In order to understand the advantages of servlets, you must have a basic understanding of how web browsers and servers cooperate to provide content to a user.

Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser.

* First, performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request.
* Second, servlets are platform-independent because they are written in Java.
* Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
* Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases or other software via the sockets and RMI mechanisms.

***The Life Cycle of a Servlet***

Three methods are central to the life cycle of a servlet. These are **init( )**, **service( )**, and **destroy( )**. They are implemented by every servlet and are invoked at specific times by the server.

* First, assume that a user enters a Uniform Resource Locator (URL) to a web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.
* Second, this HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server.
* Third, the server invokes the **init( )** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.
* Fourth, the server invokes the **service( )** method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client. The servlet remains in the server’s address space and is available to process any other HTTP requests received from clients. The **service( )** method is called for each HTTP request.
* Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the **destroy( )** method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

***Using Tomcat for Servlet Development***

To create servlets, you will need access to a servlet development environment. Tomcat is an open-source product maintained by the Jakarta Project of the Apache Software Foundation. It contains the class libraries, documentation and runtime support that you will need to create and test servlets.

If you load Tomcat in a different location, you will need to make appropriate changes to the examples. You may need to set the environmental variable ***JAVA\_HOME***to the top-level directory in which the Java Development Kit is installed.

To start Tomcat, select Configure Tomcat in the Start – All Programs menu and then press Start in the Tomcat Properties dialog. When you are done testing servlets, you can stop Tomcat by pressing Stop in the Tomcat Properties dialog. The directory C:\Tomcat 7.0\lib\ contains ***servlet-api.jar***. This JAR file contains the classes and interfaces that are needed to build servlets. To make this file accessible, update your ***CLASSPATH***environment variable so that it includes ***C:\Tomcat 7.0\lib\servlet-api.jar***Alternatively, you can specify this file when you compile the servlets. For example, the following command compiles the first servlet example.

javac HelloServlet.java –classpath "C:\Tomcat 7.0\lib\servlet-api.jar"

Once you have compiled a servlet, you must enable Tomcat to find it. This means putting it into a directory under Tomcat’s ***webapps***directory and entering its name into a ***web.xml***file. To keep things simple, the examples in this topic use the directory and *web.xml*file that Tomcat supplies for its own example servlets. Here is the procedure that you will follow.

***First, copy the servlet’s class file into the following directory***

C:\Tomcat 7.0\webapps\servlets\WEB-INF\classes

***Next, add the servlet’s name and mapping to the web.xml file in the following directory***

C:\Tomcat 7.0\webapps\servlets\WEB-INF

*For instance, assuming the first example, called* ***Serv1****, you will add the following lines in the section that defines the servlets.*

<?xml version="1.0"?>

<servlet>

<servlet-name>Serv1</servlet-name>

<servlet-class>Serv1</servlet-class>

</servlet>

*Next, you will add the following lines to the section that defines the servlet mappings.*

<servlet-mapping>

<servlet-name>Serv1</servlet-name>

<url-pattern>/Serv1</url-pattern>

</servlet-mapping>

*Follow this same general procedure for all of the examples.*

***A Simple Servlet***

To become familiar with the key servlet concepts, we will begin by building and testing a simple servlet. The basic steps are the following.

1. Create and compile the servlet source code. Then, copy the servlet’s class file to the proper directory and add the servlet’s name and mappings to the proper web.xmlfile.

2. Start Tomcat.

3. Start a web browser and request the servlet.

Let us examine each of these steps in detail.

***Create and Compile the Servlet Source Code***

To begin, create a file named ***Serv1.java***that contains the following program.

import java.io.\*;

import javax.servlet.\*;

public class Serv1 extends GenericServlet

{ public void service(ServletRequest req,ServletResponse res) throws ServletException, IOException

{ res.setContentType("text/html");

PrintWriter pw = res.getWriter();

pw.println("<B>This is my first generic Servlet!");

pw.close();

}

}

***Let’s look closely at this program.***

First, note that it imports the *javax.servlet*package. This package contains the classes and interfaces required to build servlets.

Next, the program defines *Serv1*as a subclass of GenericServlet. The GenericServletclass provides functionality that simplifies the creation of a servlet.

For example, it provides versions of init( )and destroy( ) which may be used as is. You need supply only the service( )method. Inside Serv1, the service( )method (which is inherited from GenericServlet) is overridden. This method handles requests from a client. Notice that the first argument is a *ServletRequest*object. This enables the servlet to read data that is provided via the client request. The second argument is a *ServletResponse*object. This enables the servlet to formulate a response for the client. The call to *setContentType( )*establishes the MIME type of the HTTP response. In this program, the MIME type is text/html. This indicates that the browser should interpret the content as HTML source code.

Next, the *getWriter( )*method obtains a *PrintWriter*. Anything written to this stream is sent to the client as part of the HTTP response. Then *println( )*is used to write some simple HTML source code as the HTTP response.

Compile this source code and place the ***Serv1.class***file in the proper Tomcat directory as described in the previous section. Also, add ***Serv1***to the *web.xml*file, as described earlier.

***Start Tomcat***

Start Tomcat as explained earlier. Tomcat must be running before you try to execute a servlet.

***Start a Web Browser and Request the Servlet***

Start a web browser and enter the URL shown here:

http://localhost:8080/servlets/Serv1

Alternatively, you may enter the URL shown here:

http://127.0.0.1:8080/servlets/Serv1

This can be done because 127.0.0.1 is defined as the IP address of the local machine. You will observe the output of the servlet in the browser display area. It will contain the string ***This is my first generic Servlet!***in bold type.

***The Servlet API***

Two packages contain the classes and interfaces that are required to build servlets. These are *javax.servlet*and *javax.servlet.http*. They constitute the Servlet API. Keep in mind that these packages are not part of the Java core packages. Instead, they are standard extensions provided by Tomcat. Therefore, they are not included with Java Standard Edition.

***The javax.servlet Package***

The *javax.servlet*package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes the core interfaces that are provided in this package. The most significant of this is *Servlet*. All servlets must implement this interface or extend a class that implements the interface. The *ServletRequest*and *ServletResponse*interfaces are also very important.

**Interface Description**

* Servlet : Declares life cycle methods for a servlet.
* ServletConfig : Allows servlets to get initialization parameters.
* ServletContext : Enables servlets to log events and access information about their environment.
* ServletRequest : Used to read data from a client request.
* ServletResponse : Used to write data to a client response.

The following table summarizes the core classes that are provided in the *javax.servlet*package:

**Class Description**

* GenericServlet : Implements the Servlet and ServletConfig interfaces.
* ServletInputStream : Provides an input stream for reading requests from a client.
* ServletOutputStream : Provides an output stream for writing responses to a client.
* ServletException : Indicates a servlet error occurred.
* UnavailableException : Indicates a servlet is unavailable.

Let us examine these interfaces and classes in more detail.

***The Servlet Interface***

All servlets must implement the **Servlet** interface. It declares the **init( )**, **service( )**, and **destroy( )** methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters.

The **init( )**, **service( )**, and **destroy( )** methods are the life cycle methods of the servlet. These are invoked by the server. The **getServletConfig( )** method is called by the servlet to obtain initialization parameters. A servlet developer overrides the **getServletInfo( )** method to provide a string with useful information (for example, author, version, date, copyright). This method is also invoked by the server.

**The ServletConfig Interface**

The **ServletConfig** interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

ServletContext getServletContext( ) Returns the context for this servlet.

String getInitParameter(String param) Returns the value of the initialization parameter named param.

Enumeration getInitParameterNames( ) Returns an enumeration of all initialization parameter names.

String getServletName( ) Returns the name of the invoking servlet.

**The ServletContext Interface**

The **ServletContext** interface enables servlets to obtain information about their environment.

**The ServletRequest Interface**

The **ServletRequest** interface enables a servlet to obtain information about a client request.

**The ServletResponse Interface**

The **ServletResponse** interface enables a servlet to formulate a response for a client.

**The GenericServlet Class**

The **GenericServlet** class provides implementations of the basic life cycle methods for a servlet.

**GenericServlet** implements the **Servlet** and **ServletConfig** interfaces.

**PrintWriter getWriter( ) throws IOException**

Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream( ) has already been invoked for this request.

void setContentLength(int size) Sets the content length for the response to size.

void setContentType(String type) Sets the content type for the response to type.

**The ServletInputStream Class**

The **ServletInputStream** class extends **InputStream**. It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request. It defines the default constructor. In addition, a method is provided to read bytes from the stream. It is shown here:

int readLine(byte[ ] *buffer*, int *offset*, int *size*) throws IOException

Here, *buffer* is the array into which *size* bytes are placed starting at *offset.* The method returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

**The ServletOutputStream Class**

The **ServletOutputStream** class extends **OutputStream**. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response. A default constructor is defined. It also defines the **print( )** and **println( )** methods, which output data to the stream.

**The Servlet Exception Classes**

**javax.servlet**defines two exceptions. The first is **ServletException**, which indicates that a servlet problem has occurred. The second is **UnavailableException**, which extends **ServletException**. It indicates that a servlet is unavailable.

**Reading Servlet Parameters**

The **ServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use.

This example contains two files. A web page is defined in **PostParameters.html** and a servlet is defined in **PostParametersServlet.java**.

The HTMLsource code for **PostParameters.html**is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

<html>

<body>

<center>

<form name="Form1" method="post" action="http://localhost:8080/servlets/PostParametersServlet">

<table>

<tr>

<td><B>Employee</td>

<td><input type=textbox name="e" size="25" value=""></td>

</tr>

<tr>

<td><B>Phone</td>

<td><input type=textbox name="p" size="25" value=""></td>

</tr>

</table>

<input type=submit value="Submit">

</form>

</center>

</body>

</html>

The source code for **PostParametersServlet.java** is shown in the following listing. The **service( )** method is overridden to process client requests. The **getParameterNames( )** method returns an enumeration of the parameter names. These are processed in a loop. You can see that the parameter name and value are output to the client. The parameter value is obtained via the **getParameter( )** method.

import java.io.\*;

import java.util.\*;

import javax.servlet.\*;

public class PostParametersServlet extends GenericServlet

{ public void service(ServletRequest request, ServletResponse response)throws ServletException, IOException

{

// Get print writer.

PrintWriter pw = response.getWriter();

// Get enumeration of parameter names.

Enumeration e = request.getParameterNames();

// Display parameter names and values.

while(e.hasMoreElements())

{String pname = (String)e.nextElement();

pw.print(pname + " = ");

String pvalue = request.getParameter(pname);

pw.println(pvalue);

}

pw.close();

}

}

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat (if it is not already running).

2. Display the web page in a browser.

3. Enter an employee name and phone number in the text fields.

4. Submit the web page.

After following these steps, the browser will display a response that is dynamically generated by the servlet.

***The Cookie Class***

The **Cookie** class encapsulates a cookie. A *cookie* is stored on a client and contains state information. Cookies are valuable for tracking user activities. For example, assume that a Method Description

Object getAttribute(String attr) Returns the value associated with the name passed in attr. Returns null if attr is not found.

Enumeration getAttributeNames( ) Returns an enumeration of the attribute names associated with the session.

long getCreationTime( ) Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.

String getId( ) Returns the session ID.

long getLastAccessedTime( ) Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.

void invalidate( ) Invalidates this session and removes it from the context.

boolean isNew( ) Returns true if the server created the session and it has not yet been accessed by the client.

void removeAttribute(String attr) Removes the attribute specified by attr from the session.

void setAttribute(String attr, Object val) Associates the value passed in val with the attribute name passed in attr.

**The below Methods Defined by HttpSession**

void sendError(int c) throws IOException Sends the error code c to the client.

void sendError(int c, String s) throws IOException Sends the error code c and message s to the client.

void sendRedirect(String url) throws IOException Redirects the client to url.

void setDateHeader(String field, long msec) Adds field to the header with date value equal to msec (milliseconds since midnight, January 1, 1970, GMT).

void setHeader(String field, String value) Adds field to the header with value equal to value.

void setIntHeader(String field, int value) Adds field to the header with value equal to value.

void setStatus(int code) Sets the status code for this response to code.

Assume that a user visits an online store. A cookie can save the user’s name, address and other information. The user does not need to enter this data each time he or she visits the store. A servlet can write a cookie to a user’s machine via the **addCookie( )** method of the **HttpServletResponse** interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser. The names and values of cookies are stored on the user’s machine. Some of the information that is saved for each cookie includes the following:

• The name of the cookie

• The value of the cookie

• The expiration date of the cookie

• The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user’s machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user’s machine.

The domain and path of the cookie determine when it is included in the header of an HTTP request. If the user enters a URL whose domain and path match these values, the cookie is then supplied to the Web server. Otherwise, it is not.

There is one constructor for **Cookie**. It has the signature shown here:

Cookie(String *name*, String *value*)

Here, the name and value of the cookie are supplied as arguments to the constructor. The methods of the **Cookie** class are summarized below.

Object clone( ) Returns a copy of this object.

String getComment( ) Returns the comment.

String getDomain( ) Returns the domain.

int getMaxAge( ) Returns the maximum age (in seconds).

String getName( ) Returns the name.

String getPath( ) Returns the path.

boolean getSecure( ) Returns true if the cookie is secure. Otherwise, returns false.

String getValue( ) Returns the value.

int getVersion( ) Returns the version.

void setComment(String c) Sets the comment to c.

void setDomain(String d) Sets the domain to d.

void setMaxAge(int secs) Sets the maximum age of the cookie to secs. This is the number of seconds after which the cookie is deleted.

void setPath(String p) Sets the path to p.

void setSecure(boolean secure) Sets the security flag to secure.

void setValue(String v) Sets the value to v.

void setVersion(int v) Sets the version to v.

**Using Cookies**

Now, let’s develop a servlet that illustrates how to use cookies. The servlet is invoked when a form on a web page is submitted. The example contains three files as summarized here:

File Description

***AddCookie.html*** Allows a user to specify a value for the cookie named MyCookie.

***AddCookieServlet.java*** Processes the submission of AddCookie.html.

***GetCookiesServlet.java*** Displays cookie values.

The HTML source code for **AddCookie.html**is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to **AddCookieServlet** via an HTTP POST request.

<html>

<body>

<center>

<form name="Form1" method="post"action="http://localhost:8080/servlets/AddCookieServlet">

<B>Enter a value for MyCookie:</B>

<input type=textbox name="data" size=25 value="">

<input type=submit value="Submit">

</form>

</center>

</body>

</html>

The source code for **AddCookieServlet.java** is shown in the following listing. It gets the value of the parameter named “data”. It then creates a **Cookie** object that has the name “MyCookie” and contains the value of the “data” parameter. The cookie is then added to the header of the HTTP response via the **addCookie( )** method. A feedback message is then written to the browser.

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class AddCookieServlet extends HttpServlet

{ public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

// Get parameter from HTTP request.

String data = request.getParameter("data");

// Create cookie.

Cookie cookie = new Cookie("MyCookie", data);

// Add cookie to HTTP response.

response.addCookie(cookie);

// Write output to browser.

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>MyCookie has been set to");

pw.println(data);

pw.close();

}

}

The source code for **GetCookiesServlet.java** is shown in the following listing. It invokes the **getCookies( )** method to read any cookies that are included in the HTTP GET request. The names and values of these cookies are then written to the HTTP response. Observe that the **getName( )** and **getValue( )** methods are called to obtain this information.

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class GetCookiesServlet extends HttpServlet

{public void doGet(HttpServletRequest request,HttpServletResponse response)

throws ServletException, IOException

{

// Get cookies from header of HTTP request.

Cookie[] cookies = request.getCookies();

// Display these cookies.

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>");

for(int i = 0; i < cookies.length; i++)

{String name = cookies[i].getName();

String value = cookies[i].getValue();

pw.println("name = " + name +"; value = " + value);

}

pw.close();

}

}

Compile the servlets. Next, copy them to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.

2. Display **AddCookie.html**in a browser.

3. Enter a value for **MyCookie**.

4. Submit the web page.

After completing these steps, you will observe that a feedback message is displayed by the browser.

Next, request the following URL via the browser:

http://localhost:8080/servlets-examples/servlet/GetCookiesServlet

Observe that the name and value of the cookie are displayed in the browser. In this example, an expiration date is not explicitly assigned to the cookie via the**setMaxAge( )** method of **Cookie**. Therefore, the cookie expires when the browser session ends. You can experiment by using **setMaxAge( )** and observe that the cookie is then saved to the disk on the client machine.

***Session Tracking***

HTTP is a stateless protocol. Each request is independent of the previous one. However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.

A session can be created via the **getSession( )** method of **HttpServletRequest**. An **HttpSession** object is returned. This object can store a set of bindings that associate names with objects. The **setAttribute( )**, **getAttribute( )**, **getAttributeNames( )**, and **removeAttribute( )** methods of **HttpSession** manage these bindings. It is important to note that session state is shared among all the servlets that are associated with a particular client. The following servlet illustrates how to use session state. The **getSession( )** method gets the current session. A new session is created if one does not already exist. The **getAttribute( )** method is called to obtain the object that is bound to the name “date”. That object is a **Date** object that encapsulates the date and time when this page was last accessed. (Of course, there is no such binding when the page is first accessed.) A **Date** object encapsulating the current date and time is then created. The **setAttribute( )** method is called to bind the name “date” to this object.

import java.io.\*;

import java.util.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class DateServlet extends HttpServlet

{ public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{ // Get the HttpSession object.

HttpSession hs = request.getSession(true);

// Get writer.

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.print("<B>");

// Display date/time of last access.

Date date = (Date)hs.getAttribute("date");

if(date != null)

{

pw.print("Last access: " + date + "<br>");

}

// Display current date/time.

date = new Date();

hs.setAttribute("date", date);

pw.println("Current date: " + date);

}

}

When you first request this servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the date and time when the servlet was last accessed. The second line shows the current date and time.

**JSP (Java Server Pages)**

The Servlet technology and Java Server Pages(JSP) are the two main technologies for developing java Web applications. When first introduced by Sun Microsystems in 1996, the Servlet technology was considered superior to the reigning Common Gateway Interface (CGI), because servlets stay in memory after they service the first requests. Subsequent requests for the same servlet do not require instantiation of the servlet’s class therefore enabling better response time.

Servlets are Java classes that implement the ***javax.servlet.Servlet*** interface. They are compiled and deployed in the web server. The problem with servlets is that you embed HTML in Java code. If you want to modify the cosmetic look of the page or you want to modify the structure of the page, you have to change code.

Generally speaking, this is left to the better hands (and brains) of a web page designer and not to a Java developer.

PrintWriter pw = response.getWriter();

pw.println("<html><head><title>Testing</title></head>");

pw.println("<body bgcolor=\"#ffdddd\">");

...

As seen from the example above this method presents several difficulties to the web developer.

1. The code for a servlet becomes difficult to understand for the programmer.
2. The HTML content of such a page is difficult if not impossible for a web designer to understand or design.
3. This is hard to program and even small changes in the presentation, such as the page’s background color, will require the servlet to be recompiled. Any changes in the HTML content require the rebuilding of the whole servlet.
4. It's hard to take advantage of web-page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process which is time consuming, error prone, and extremely boring.
5. In many Java servlet-based applications, processing the request and generating the response are both handled by a single servlet class.
6. The servlet contains request processing and business logic (implemented by methods) and also generates the response HTML code, are embedded directly in the servlet code.

JSP solves these problems by giving a way to include java code into an HTML page using scriptlets. This way the HTML code remains intact and easily accessible to web designers, but the page can still perform its task.

In late 1999, Sun Microsystems added a new element to the collection of Enterprise Java tools: Java Server Pages (JSP).

Java Server Pages are built on top of Java servlets and designed to increase the efficiency in which programmers and even nonprogrammers, can create web content.

A JSP page is handled differently compared to a servlet by the web server. When a servlet is deployed into a web server in compiled (byte code) form and then a JSP page is deployed in its original, human-readable form.

When a user requests the specific page, the web server compiles the page into a servlet and from there on handles it as a standard servlet.

This accounts for a small delay, when a JSP page is first requested, but any subsequent requests benefit from the same speed effects that are associated with servlets.

***JSP Processing***

Once you have a JSP capable web-server or application server, you need to know the following information about it.

* Where to place the files
* How to access the files from your browser (with an http: prefix, not as file: )

You should be able to create a simple file, such as

<HTML>

<BODY>

Hello, world

</BODY>

</HTML>

Know where to place this file and how to see it in your browser with an http://prefix. Since this step is different for each web-server, you would need to see the web-server documentation to find out how this is done. Once you have completed this step, proceed to the next.

***Your first JSP***

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to ".jsp" instead of ".html". In fact, this is the perfect exercise for your first JSP.

Take the HTML file you used in the previous exercise. Change its extension from ".html" to ".jsp". Now load the new file, with the ".jsp" extension, in your browser. You will see the same output, but it will take longer! But only the first time, if you reload it again, it will load normally.

What is happening behind the scenes is that your JSP is being turned into a Java file, compiled and loaded. This compilation only happens once, so after the first load, the file doesn't take long to load anymore. (But every time you change the JSP file, it will be re-compiled again) Of course, it is not very useful to just write HTML pages with a .jsp extension! We now proceed to see what makes JSP so useful.

***Adding dynamic content via expressions***

As we saw in the previous section, any HTML file can be turned into a JSP file by changing its extension to .jsp. Of course, what makes JSP useful is the ability to embed Java. Put the following text in a file with .jsp extension (let us call it hello.jsp), place it in your JSP directory, and view it in a browser.

<HTML>

<BODY>

Hello! The time is now <%= new java.util.Date() %>

</BODY>

</HTML>

Notice that each time you reload the page in the browser, it comes up with the current time. The character sequences <%= and %> enclose Java expressions, which are evaluated at run time. This is what makes it possible to use JSP to generate dynamic HTML pages that change in response to user actions or vary from user to user.

***JSP Elements***

In JSP elements can be divided into 4 different types. These are:

***1. Expressions***

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

Syntax of JSP expressions are:<%="Any thing" %>

JSP Expressions start with Syntax of JSP Scriptlets are with <%= and ends with %>. Between these you can put anything and that will convert to the String and that will be displayed.

Example: <%="Hello World!" %>

Above code will display 'Hello World!'

***2. Scriptlets***

In this tag we can insert any amount of valid java code and these codes are placed in \_*jspService* method by the JSP engine.

Syntax of JSP scriptlets are:<%//java codes%>

JSP scriptlets begins with <% and ends %>. We can embed any amount of java code in the JSP scriptlets. JSP Engine places these code in the \_jspService() method. Variables available to the JSP scriptlets are:

***a. Request***

Request represents the clients request and is a subclass of ***HttpServletRequest****.* Use this variable to retrieve the data submitted along the request.

E.g:

<% //java codes

String userName=null;

userName=request.getParameter("userName");

%>

***b. Response***

Response represents the server response and is a subclass of ***HttpServletResponse***.

Ex: - <% response.setContentType("text/html"); %>

***c. Session***

It represents the HTTP session object associated with the request.

Ex: - Your Session ID: <%= session.getId() %>

***d. Out***

out is an object of output stream and is used to send any output to the client.

***3. Directives***

A JSP ‘directive’ starts with <%@ characters. In the directives we can import packages, define error handling pages or the session information of the JSP page.

Syntax of JSP directives is:<%@directive attribute="value" %>

***a. page:*** page is used to provide the information about it.

E.g.:<%@page language="java" %>

***b. include:*** include is used to include a file in the JSP page.

E.g.:<%@ include file="/header.jsp" %>

***c. taglib:*** taglib is used to use the custom tags in the JSP pages (custom tags allows us to defined our own tags).

E.g.:<%@ taglib uri="tlds/taglib.tld" prefix="mytag" %>

***Page tag attributes are:***

***a. language="java"***

This tells the server that the page is using the java language. Current JSP specification supports only java language.

Example:<%@page language="java" %>

***b. extends="mypackage.myclass"***

This attribute is used when we want to extend any class. We can use comma(,) to import more than one packages.

Example:<%@page language="java" import="java.sql.\*" %>

***c. session="true"***

When this value is true session data is available to the JSP page otherwise not. By default this value is true.

Example: <%@page language="java" session="true" %>

***d. errorPage="error.jsp"***

errorPage is used to handle the un-handled exceptions in the page.

Example:<%@page session="true" errorPage="error.jsp" %>

***e. contentType="text/html;charset=ISO-8859-1"***

Use this attribute to set the mime type and character set of the JSP.

Example:<%@page contentType="text/html;charset=ISO-8859-1" %>

***4. Declarations***

This tag is used for defining the functions and variables to be used in the JSP.

Syntax of JSP declaratives are:

<%!

//java codes

%>

JSP Declaratives begins with <%! and ends with %>.We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class level and can be used anywhere in the JSP page.

E.g.:

<%@ page import="java.util.\*" %>

<HTML>

<BODY>

<%!

Date theDate = new Date();

Date getDate()

{

System.out.println( "In getDate() method" );

return theDate;

}

%>

Hello! The time is now <%= getDate() %>

</BODY>

</HTML>

***JSP Standard Tag Library (JSTL): JSP Tags***

Another important syntax element of JSP is tags. JSP tags do not use **<%**, but just the **<**character. A JSP tag is some what like an HTML tag. JSP tags can have a "start tag", a "tag body" and an "end tag". The start and end tag both use the tag name, enclosed in < and > characters. The end starts with a / character after the < character. The tag names have an embedded colon character : in them, the part before the colon describes the type of the tag. For instance:

<some:tag>

body

</some:tag>

If the tag does not require a body, the start and end can be conveniently merged together, as <some:tag/>. Here by closing the start tag with a /> instead of > character, we are ending the tag immediately and without a body. (This syntax convention is the same as XML)

Tags can be of two types: loaded from an external tag library, or predefined tags.

Predefined tags start with **jsp:** characters. For instance, jsp:include is a predefined tag that is used to include other pages.

We have already seen the include directive. jsp:include is similar. But instead of loading the text of the included file in the original file, it actually calls the included target at run-time (the way a browser would call the included target. In practice, this is actually a simulated request rather than a full round-trip between the browser and the server). Following is an example of jsp:include usage

<HTML>

<BODY>

Going to include hello.jsp...<BR>

<jsp:include page="hello.jsp"/>

</BODY>

</HTML>

***The <jsp:include> Tag***

Includes the response from a servlet or JSP page

***The <jsp:forward> Tag***

Forwards the processing of a request to a JSP page

***The <jsp:param> Tag***

This tag is used inside the <jsp:include>, <jsp:forward> and <jsp:plugin> tags. It defines and sets different parameters with their values .e.g,

<jsp:include page="JspEx12.jsp">

<jsp:param name="stu" value="xStudent" />

</jsp:include>

In the code above a parameter with the name 'stu' and a value of "xStudent" has been provided to the included JSP page; JspEx11.jsp. The JspEx12.jsp page can access the value of this parameter by :

<%

String sn = request.getParameter("stu");

%>

<BR>

Hello <%=sn%>! Welcome

***The <jsp:plugin> Tag***

Generates HTML that contains the appropriate browser-dependent elements (OBJECT or EMBED) needed to execute an applet or bean with the Java Plug-in software. This tag allows you to embed Java applets or beans in a web page.

It's syntax is as follows:

<jsp:plugintype="bean | applet" code="className"

codebase="classes path to root directory"

align="top | middle | bottom | left | right"

archives="jar files required for this app"

height="height in pixels"

width="width in pixels"

hspace="horizontal space in pixels"

vspace="vertical space"

jreversion="the JRE version required"

name="name of object"

nspluginurl="URL for Netscape plugin"

iepluginurl="URL For IE plugin">

<jsp:params>

<jsp:param name="name"value="value"/>

</jsp:params>

<jsp:fallback>

Sorry browser doesn't support Java applets.

</jsp:fallback>

</jsp:plugin>

E.g.: 1

<HTML>

<BODY >

Going to plugin an applet...<BR>

<jsp:plugin type="bean"code="pros/Count.class"width="70"height="50">

<jsp:fallback>

<B>Error:this example requires Java.</B>

</jsp:fallback>

</jsp:plugin>

</BODY>

</HTML>

E.g.:2

<HTML>

<BODY >

Going to plugin a bean...<BR>

<jsp:plugin type="applet"code="pros/Calci.class"width="100" height="50">

<jsp:fallback>

<B>Error: this example requires Java.</B>

</jsp:fallback>

</jsp:plugin>

</BODY>

</HTML>

*<jsp:plugin>*

Generates Java Plugin software like an applet

*<jsp:useBean>*

Makes a JavaBeans component

*<jsp:getProperty>*

Gets a property value from a JavaBeans component

*<jsp:setProperty>*

Sets a JavaBeans component property value

***Model View Controller***

JSP technology can play a part in everything from the simplest web application to complex enterprise applications. How large a part JSP plays differs in each case, of course. Let introduce a design model called Model-View-Controller (MVC), suitable for both simple and complex applications. MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate logic into three distinct units: the Model, the View, and the Controller. In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing.

***Business logic*** is the term used for the manipulation of an application's data, such as customer, product, and order information. ***Presentation*** refers to how the application data is displayed to the user, for example, position, font, and size. And finally, ***request processing*** is what ties the business logic and presentation parts together.

In MVC terms, presentation should be separated from the business logic. Presentation of that data (the View) changes fairly often. Just look at all the face-lifts many web sites go through to keep up with the latest fashion in web design. Some sites may want to present the data in different languages or present different subsets of the data to internal and external users

***Sharing Data between JSP Pages, Requests, and Users***

Any real application consists of more than a single page, and multiple pages often need access to the same information and server-side resources. When multiple pages process the same request (e.g., one page that retrieves the data the user asked for and another that displays it), there must be a way to pass data from one page to another.

One of the most fundamental features of JSP technology is that it allows for separation of request processing, business logic and presentation, using what's known as the Model-View-Controller (MVC) model. The different aspects of the User Info example can be categorized like this:

• Display the form for user input (presentation)

• Validate the input (request processing and business logic)

• Display the result of the validation (presentation)

If the customer wants a different look, a page author can modify the View JSP pages without touching the request processing or business logic code. Using different JSP pages as Controller and View means that more than one page is used to process a request. To make this happen, you need to be able to do two things:

• Pass control from one page to another

• Pass data from one page to another

***Passing Control from One Page to Another***

JSP page passes control to one page to other pages based on the result of the input validation. JSP supports this through the <jsp:forward> action, the <jsp:forward> action stops processing of one page and starts processing the page specified by the page attribute instead, called the *target page*. The control never returns to the original page.

The target page has access to all information about the request, including all request parameters. You can also add additional request parameters when you pass control to another page by using one or more nested <jsp:param> action elements

<jsp:forward page="userinfoinput.jsp" >

<jsp:param name="msg" value="Invalid email address" />

</jsp:forward>

Parameters specified with <jsp:param> elements are added to the parameters received with the original request. The target page, therefore, has access to both the original parameters and the new ones, and can access both types in the same way.

If a parameter is added to the request using a name of a parameter that already exists, the new value is added first in the list of values for the parameter. The page attribute value is interpreted relative to the location of the current page.

***Dealing with Syntax Errors***

The first type of error you will encounter is the one that created by simple typos—in other words, syntax errors. The JSP container needs every JSP element to be written exactly as it's defined in the specification in order to process the JSP page. When it finds something that's not right, it tells you. How easy it is to understand what it tells you depend on the type of error.

Let's first look at how Tomcat reports some typical syntax errors in JSP directives and action elements.

Improperly terminated directive

**<%@ page contentType="text/jsp" >**

The syntax error here is that the page directive on the first line isn't closed properly with %>; the percent sign is missing.

Tomcat reports the error by sending an error message to the browser. This is the default behavior for Tomcat, but it's not mandated by the JSP specification. The specification requires only that a response with the HTTP status error code (500) is returned.

The actual error message in browser window is what is called an ***exception stack trace***. When something goes really wrong in a Java method, it typically throws an exception. An exception is a special Java object, and throwing an exception is the method's way of saying it doesn't know how to handle a problem. Tomcat container does when it finds a problem with a JSP page during the translation phase; it sends the exception stack trace to the browser. The stack trace contains a message about what went wrong and where the problem occurred.

You can see the message as:

/jsp/kanth/JspErr1.jsp(0,0) Unterminated <%@ page tag

The first part of the message is the name of the JSP page. The numbers within parentheses indicate on which line and character position in the file the error as found and then the message states what the problem is.

Improperly terminated action

<jsp:param name="stu" value="xStudent">

The syntax error here is almost the same as the above but now it's the <jsp:param> action element that's not terminated properly (it's missing the closing slash required for an empty element). The message reported by Tomcat in this case is:

/jsp/kanth/JspErr2.jsp(6,42) Unterminated

Mistyped attribute name

<jsp:include paeg="jsp8.jsp”>

Tomcat reports the problem like this:

/jsp/kanth/JspErr3.jsp(4,0) Forward: Mandatory attribute page missing

In this case, the typo is in the name of a mandatory attribute, so Tomcat reports it as missing. If the type is in the name of an optional attribute, Tomcat reports it as an invalid attribute name.

Missing end quote in attribute value

<jsp:forward page="JspEx3.jsp/>

If you look carefully at the <jsp:forward> element, you see that the closing quote for the page attribute is missing. Tomcat reports the problem like this:

/jsp/kanth/JspErr4.jsp(4,19) jsp.error.attribute.unterminated

Attribute with no equal sign before value

<jsp:forward page"JspEx3.jsp" />

If you look carefully at the <jsp:forward> element, you see that the equal sign for page attribute is missing.

Tomcat reports the problem like this:

/jsp/kanth/JspErr5.jsp(4,17) jsp.error.attribute.noequal

***Debugging a JSP Application***

After you have fixed all syntax errors, pat yourself on the back and enjoy the moment. you will likely find that one or more things still don't work as you expected. Logic errors, such as not taking care of all possible input combinations, can easily slip into an application during development. Finding and correcting this type of problem is called debugging.

For applications developed in compiled languages such as Java, C, or C++, a tool called a debugger is often used in this phase. It lets you step through the program line by line or run the program until it reaches a break point that you have defined, and lets you inspect the values of all variables in the program. With careful analysis of the program flow in runtime, you can discover why it works the way it does and not the way you want it to.

There are debuggers for JSP as well, such as IBM's Visual Age for Java. Such products let you debug a JSP page exactly the same way as a program written in a more traditional programming language. A standard Java debugger, which can be found in most Java Interactive Development Environments (IDEs).

***Exception Handling in JSP Pages***

What are Exceptions?

Exceptions mean exceptional events and as we all know exceptional events can occur any where in a program e.g. you are trying to connect to a database and the database server is down, now you wouldn't have expected this to happen ;).

**How to catch Exceptions?**

You can catch exceptions in a JSP page like you would do in other Java classes. Simply put the code which can throw an exception/s between a try..catch block.

<%

try {

// Code which can throw can exception

} catch(Exception e) {

// Exception handler code here

}

%>

There is yet another useful way of catching exceptions in JSP pages. You can specify error page in the 'page' directive. Then if any exception is thrown, the control will be transferred to that error page where you can display a useful message to the user about what happened .

***Demonstration error page:***

To demonstrate the run-time exception handling feature of JSP pages, we will build three pages.

· Form.html - Display a Form to the user to enter his or her age.

· FormHandler.jsp - A JSP Page which receives this value and prints it on the user screen.

· ExceptionHandler.jsp - An exception handler JSP page which is actually an error page to which control will be passed when an exception is thrown.

***i. Form.html***

Create a new Form.html page. Copy the following code and paste it into the Form.html page:

<html>

<head>

<title> form.html</title>

</head>

<body>

<form action="FormHandler.jsp" method="post">

Enter your lab marks :

<input type="text" name="marks" />

<input type="submit" value="Submit" />

</form>

</body>

</html>

*Explanation:*

Form.html page simply displays a single input field Form to the user to enter his age in years. The name of input field where user will enter his/her internal marks is "marks". We will use this input field name "marks" in the FormHandler.jsp page to receive it's value.

***ii. FormHandler.jsp***

Create new FormHandler.jsp page. Copy and paste the following code in it.

<%@ page errorPage="ExceptionHandler.jsp" %>

<html>

<head>

<title> errorpage </title>

</head>

<body>

<%int marks;

marks = Integer.parseInt(request.getParameter("marks"));

%>

<%-- Displaying Student marks --%>

<p>You have scored : <%= marks %> marks.</p>

<p><a href="Form.html">Back</a>.</p>

</body>

</html>

*Explanation:*

Code above is rather simple. Notice the first line, the page directive. It specifies an errorPage ExceptionHandler.jsp, our exception handler JSP page.

<%@ page errorPage="ExceptionHandler.jsp" %>

Then we declare an int variable "marks". Then using the static method of Integer class we parse the entered value using Integer.parseInt() method. The value is retrieved using request.getParameter() method. The argument to request.getParameter() is the name of Form field whose value we want to retrieve.

int marks;

marks = Integer.parseInt(request.getParameter("marks"));

If all goes well and user entered an int ( e.g. 48 ) value in the input field then we display that value back to the user.

<p>Your have scored : <%= marks %> marks.</p>

Now things can go wrong and exceptional events can occur. For example, if student didn't enter a value and what if student entered his marks as String type instead of an integer? These things will be handled by the ExceptionHandler.jsp JSP page.

***iii. ExceptionHandler.jsp***

Create a new ExceptionHandler.jsp page. Copy and paste the following Ex:

<%@ page isErrorPage="true" import="java.io.\*" %>

<html>

<head>

<title>Exceptional Event Occurred!</title>

</head>

<body>

<%-- Exception Handler --%>

<font color="orange">

<%= exception.toString() %><br>

</font>

</body>

</html>

*Explanation:*

To make a JSP page exception handler ( i.e. errorPage ), you have to specify isErrorPage attribute in the page directive at the top and set it's value to true.

<%@ page isErrorPage="true" %>

When a JSP page has been declared an errorPage, it is made available an object with name of "exception" of type java.lang.Throwable. We use different methods of this exception object to display useful information to the user.

***Running the Demo Pages***

Place Form.html, FormHandler.jsp and ExceptionHandler.jsp pages in a place where your application server ( e.g. Tomcat) can find them. Never put your JSP pages in the /WEB-INF/ folder.

**NETWORKING in JAVA**

**Network :-** Network means the group of computers and their peripherals are connected through a communication channel, to share the other system disk drives, files, printers, network connections … etc.

**Socket :-** A socket identifies an end point in a network. Sockets are at the foundation of modern networking because a socket allows a single computer to serve many different clients at once, as well as to serve many different types of information.

**Port :-** The socket related tasks are accomplished through the use of a *port,* which is a numbered socket on a particular machine. A server process is said to “listen” to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique.

**Protocol :-** Socket communication takes place via a protocol. *Internet Protocol (IP)* is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.

**TCP :-***Transmission Control Protocol* (TCP) is a higher-level protocol that manages to robustly string together these packets, sorting and retransmitting them as necessary to reliably transmit data.

**UDP :-** A third protocol, *User Datagram Protocol (UDP),* sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets. Once a connection has been established, a higher-level protocol ensues, which is dependent on which port you are using.

**TCP/IP :-** TCP/IP stands for transmission control protocol and the internet protocol. TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to you if you have spent any time surfing the Internet. Port number 21 is for FTP; 23 is for Telnet; 25 is for e-mail; 43 is for whois; 79 is for finger; 80 is for HTTP; 119 is for netnews—and the list goes on. It is up to each protocol to determine how a client should interact with the port.

**HTTP :-** HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images. It is a quite simple protocol for a basic page-browsing web server. Here’s how it works. When a client requests a file from an HTTP server, an action known as a *hit,* it simply sends the name of the file in a special format to a predefined port and reads back the contents of the file. The server also responds with a status code to tell the client whether or not the request can be fulfilled and why.

**IP Address :-** A key component of the Internet is the *address*. Every computer on the Internet has one*.* An Internet address is a number that uniquely identifies each computer on the Net. Originally, all Internet addresses consisted of 32-bit values, organized as four 8-bit values. This address type was specified by IPv4 (Internet Protocol, version 4). However, a new addressing scheme, called IPv6 (Internet Protocol, version 6) has come into play. IPv6 uses a 128-bit value to represent an address, organized into eight 16-bit chunks. Although there are several reasons for and advantages to IPv6, the main one is that it supports a much larger address space than does IPv4. To provide backward compatibility with IPv4, the low-order 32 bits of an IPv6 address can contain a valid IPv4 address. Thus, IPv4 is upwardly compatible with IPv6. Fortunately, when using Java, you won’t normally need to worry about whether IPv4 or IPv6 addresses are used because Java handles the details for you.

**Domain Name :-** The name of an Internet address, called its *domain name,* describes a machine’s location in a name space. For example, **www.osborne.com** is in the *COM* domain (reserved for U.S. commercial sites); it is called *osborne* (after the company name), and *www(World Wide Web)* identifies the server for web requests. An Internet domain name is mapped to an IP address by the *Domain Naming Service (DNS).* This enables users to work with domain names, but the Internet operates on IP addresses.

In java the network related all classes are placed in the ***java.net*** package.

**InetAddress Class:-**The InetAddressclass is used to encapsulate both the numerical IP address and the domain name for that address. You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddressclass hides the number inside. InetAddresscan handle both IPv4 and IPv6 addresses.

**Factory Methods: -** The InetAddressclass has no visible constructors. To create an InetAddressobject, you have to use one of the available factory methods. Factory methods are merely a convention whereby static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer. Three commonly used InetAddressfactory methods are shown below:

static InetAddress getLocalHost( ) throws UnknownHostException

static InetAddress getByName(String *hostName*) throws UnknownHostException

static InetAddress[ ] getAllByName(String *hostName*) throws UnknownHostException

The **getLocalHost()** method simply returns the InetAddressobject that represents the local host. The **getByName( )** method returns an InetAddressfor a host name passed to it. If these methods are unable to resolve the host name, they throw an UnknownHostException.

On the Internet, it is common for a single name to be used to represent several machines. In the world of web servers, this is one way to provide some degree of scaling. The **getAllByName( )** factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. It will also throw an UnknownHostException. If it can’t resolve the name to at least one address.

InetAddressalso includes the factory method **getByAddress( )**, which takes an IP address and returns an InetAddress object. Either an IPv4 or an IPv6 address can be used. The following example prints the addresses and names of the local machine and two well-known Internet web sites:

// Demonstrate InetAddress

import java.net.\*;

classProg1

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

{

InetAddress Address = InetAddress.getLocalHost();

System.out.println("The current System Address is : " + Address);

Address = InetAddress.getByName("osborne.com");

System.out.println("osborne.com related ip address is : " + Address);

InetAddress SW[] = InetAddress.getAllByName("www.nba.com");

for (int i=0; i<SW.length; i++)

System.out.println("The www.nba.com address is : " + SW[i]);

}

}

Here is the output produced by this program.

The current System Address is : computer49/192.168.0.49

osborne.com related ip address is : osborne.com/198.45.22.173

The www.nba.com address is : www.nba.com/117.239.240.9

The www.nba.com address is : www.nba.com/117.239.240.41

**Instance Methods:-**The InetAddress class has several other methods, which can be used on the objects returned by the methods. Here are some of the more commonly used methods:

**equals() Method**:- Returns true if this object has the same Internet address as other.

E.g.: boolean equals(Object other)

**getAddress() Method**:- Returns a byte array that represents the object’s IP address in network byte order.

E.g.: byte[ ] getAddress( )

**getHostAddress() Method**:- Returns a string that represents the host address associated with the InetAddress object.

E.g.: String getHostAddress()

**getHostName() method**:- Returns a string that represents the host name associated with the InetAddress object.

E.g.: String getHostName()

**isMulticastAddress( ) Method**:- Returns true if this address is a multicast address. Otherwise, it returns false.

E.g.: boolean isMulticastAddress()

**toString( ) Method**:- Returns a string that lists the host name and the IP address for convenience.

E.g.: String toString()

**TCP/IP Sockets**:- There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients. The *ServerSocket*class is designed to be a “listener”, which waits for clients to connect before doing anything. Thus, *ServerSocket*is for servers. The *Socket*class is for clients. It is designed to connect to server sockets and initiate protocol exchanges. Because client sockets are the most commonly used by Java applications, they are examined here. The creation of a Socketobject implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

**Socket(String hostName, int port) throws UnknownHostException, IOException**

The above method creates a socket connected to the named host and port.

**Socket(InetAddress ipAddress, int port) throws IOException**

The above method creates a socket using a preexisting InetAddress object and a port.

Socketdefines several instance methods. For example, a **Socket** can be examined at any time for the address and port information associated with it, by use of the following methods:

**getInetAddress( ) Method**:- Returns the InetAddress associated with the Socket

object. It returns null if the socket is not connected.

E.g.: InetAddress getInetAddress( )

**getPort( )Method**:- Returns the remote port to which the invoking Socket object is connected. It returns 0 if the socket is not connected.

E.g.: int getPort( )

**getLocalPort( )**:- Returns the local port to which the invoking Socket object is bound. It returns –1 if the socket is not bound.

E.g.: int getLocalPort( )

You can gain access to the input and output streams associated with a Socketby use of the getInputStream( ) and getOuptutStream( ) methods, as shown here. Each can throw an IOExceptionif the socket has been invalidated by a loss of connection.

**getInputStream()**:- Returns the InputStream associated with the invoking socket.

E.g.: InputStream getInputStream( ) throws IOException

**getOutputStream()**:- Returns the OutputStream associated with the invoking socket.

Ex:- OutputStream getOutputStream( ) throws IOException

Several other methods are available, including connect(), which allows you to specify a new connection; isConnected( ), which returns true if the socket is connected to a server; isBound(), which returns true if the socket is bound to an address; and isClosed(), which returns true if the socket is closed.

// Demonstrate Sockets.

import java.net.\*;

import java.io.\*;

classProg2

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

{ int c;

// Create a socket connected to NICE.com, port 80.

Socket s = new Socket("NICE.com",80);

// Obtain input and output streams.

InputStream in = s.getInputStream();

OutputStream out = s.getOutputStream();

// Construct a request string.

String str = (args.length == 0 ? "osborne.com" : args[0]) + "\n";

// Convert to bytes.

byte buf[] = str.getBytes();

// Send request.

out.write(buf);

// Read and display response.

while((c=in.read())!=-1) System.out.print((char) c);

s.close();

}

}

**URL**:-The preceding example was rather obscure because the modern Internet is not about the older protocols such as whois, finger, and FTP. It is about WWW, the World Wide Web. The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser. One of the most important aspects of the Web is that Tim Berners-Lee devised a scaleable way to locate all of the resources of the Net. Once you can reliably name anything and everything, it becomes a very powerful paradigm. The ***Uniform Resource Locator*** (URL) does exactly that.

The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web. Within Java’s network class library, the **URL** class provides a simple, concise API to access information across the Internet using URLs. All URLs share the same basic format, although some variation is allowed. Here are two examples:

**http://www.osborne.com/ and http://www.osborne.com:80/index.htm.**

A URL specification is based on four components. The first is the protocol to use, separated from the rest of the locator by a colon (:). Common protocols are HTTP, FTP, gopher, and file, although these days almost everything is being done via HTTP (in fact, most browsers will proceed correctly if you leave off the “http://” from your URL specification). The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (//) and on the right by a slash (/) or optionally a colon (:). The third component, the port number, is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/). (It defaults to port 80, the predefined HTTP port; thus, “:80” is redundant.) The fourth part is the actual file path. Most HTTP servers will append a file named index.htmlor index.htmto URLs that refer directly to a directory resource. Thus, http://www.osborne.com/is the same as <http://www.osborne.com/index.htm>.

Java’s URLclass has several constructors; each can throw a MalformedURLException. One commonly used form specifies the URL with a string that is identical to what you see displayed in a browser:

URL(String *urlSpecifier*) throws MalformedURLException

The next two forms of the constructor allow you to break up the URL into its component

parts:

URL(String *protocolName*, String *hostName*, int *port*, String *path*) throws MalformedURLException

URL(String *protocolName*, String *hostName*, String *path*) throws MalformedURLException

Another frequently used constructor allows you to use an existing URL as a reference context and then create a new URL from that context. Although this sounds a little contorted, it’s really quite easy and useful.

URL(URL *urlObj*, String *urlSpecifier*) throws MalformedURLException

The following example creates a URL to Osborne’s download page and then examines its properties:

// Demonstrate URL.

import java.net.\*;

classProg3

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

{ URL hp = new URL("http://www.osborne.com/downloads");

System.out.println("Protocol: " + hp.getProtocol());

System.out.println("Port: " + hp.getPort());

System.out.println("Host: " + hp.getHost());

System.out.println("File: " + hp.getFile());

System.out.println("Ext:" + hp.toExternalForm());

}

}

Output:-

Protocol: http

Port: -1

Host: www.osborne

File: /downloads

Ext:http://www.osborne/downloads

Note:- if the port is –1; this means that a port was not explicitly set.

To access the actual bits or content information of a URL, create a URLConnectionobject from it, using its openConnection()method, like this:

urlc = url.openConnection()

openConnection( )has the following general form:

URLConnection openConnection( ) throws IOException

It returns aURLConnectionobject associated with the invoking URLobject. Notice that it may throw an IOException.

**URLConnection Class:-**URLConnectionis a general-purpose class for accessing the attributes of a remote resource. Once you make a connection to a remote server, you can use URLConnectionto inspect the properties of the remote object before actually transporting it locally. These attributes are exposed by the HTTP protocol specification and, as such, only make sense for URLobjects that are using the HTTP protocol.

URLConnectiondefines several methods. Here is a sampling:

**getContentLength( ) Method**:-Returns the size in bytes of the content associated

with the resource. If the length is unavailable, –1 is returned.

E.g.: int getContentLength( )

**getContentType( ) Method**:- Returns the type of content found in the resource. This is the value of the content-type header field. Returns null if the content type is not available.

E.g.: String getContentType( )

**getDate( ) Method**:- Returns the time and date of the response represented in terms of milliseconds since January 1, 1970 GMT.

E.g.: long getDate( )

**getExpiration( ) Method**:- Returns the expiration time and date of the resource represented in terms of milliseconds since January 1, 1970 GMT. Zero is returned if the expiration date is unavailable.

E.g.: long getExpiration( )

**getHeaderField(int idx) Method**:- Returns the value of the header field at index idx. (Header field indexes begin at 0.) Returns null if the value of idx exceeds the number of fields.

E.g.: String getHeaderField(int idx)

**getHeaderField(String fieldName) Method**:- Returns the value of header field whose name is specified by fieldName. Returns null if the specified name is not found.

E.g.: String getHeaderField(String fieldName) Method

**getHeaderFieldKey(int idx) Method**:- Returns the header field key at index idx. (Header field indexes begin at 0.) Returns null if the value of idx exceeds the number of fields.

E.g.: String getHeaderFieldKey(int idx)

**getHeaderFields( ) Method**:- Returns a map that contains all of the header fields

and values.

E.g.: Map<String, List<String>>getHeaderFields( )

**getLastModified( ) Method**:- Returns the time and date, represented in terms of milliseconds since January 1, 1970 GMT, of the last modification of the resource. Zero is returned if the last-modified date is unavailable.

E.g.: long getLastModified( )

**getInputStream( ) Method**:- Returns an InputStream that is linked to the resource. This stream can be used to obtain the content of the resource.

E.g.: InputStream getInputStream( ) throws IOException

Notice that URLConnectiondefines several methods that handle header information. A header consists of pairs of keys and values represented as strings. By using getHeaderField(), you can obtain the value associated with a header key. By calling getHeaderFields(), you can obtain a map that contains all of the headers. Several standard header fields are available directly through methods such as getDate()and getContentType().

The following example creates a URLConnectionusing the openConnection()method of a URLobject and then uses it to examine the document’s properties and content:

// Demonstrate URLConnection.

import java.net.\*;

import java.io.\*;

import java.util.Date;

classProg4

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

{ int c;

URL hp = new URL("http://www.internic.net");

URLConnection hpCon = hp.openConnection();

// get date

long d = hpCon.getDate();

if(d==0) System.out.println("No date information.");

else System.out.println("Date: " + new Date(d));

// get content type

System.out.println("Content-Type: " + hpCon.getContentType());

// get expiration date

d=hpCon.getExpiration();

if(d==0)System.out.println("No expiration information.");

else System.out.println("Expires: " + new Date(d));

// get last-modified date

d=hpCon.getLastModified();

if(d==0) System.out.println("No last-modified information.");

else System.out.println("Last-Modified: " + new Date(d));

// get content length

int len = hpCon.getContentLength();

if(len == -1) System.out.println("Content length unavailable.");

else System.out.println("Content-Length: " + len);

if(len != 0)

{ System.out.println("=== Content ===");

InputStream input = hpCon.getInputStream();

int i = len;

while (((c = input.read()) != -1)) System.out.print((char) c);

input.close();

}

else System.out.println("No content available.");

}

}

The program establishes an HTTP connection to www.internic.netover port 80. It then displays several header values and retrieves the content. Here are the first lines of the output (the precise output will vary over time).

Date: Thu Jun 08 14:41:35 CDT 2006

Content-Type: text/html

No expiration information.

Last-Modified: Wed Oct 05 19:49:29 CDT 2005

Content-Length: 4917

=== Content ===

<html>

<head>

<title>InterNIC | The Internet's NetworkInformationCenter</title>

<meta name="keywords"

content="internic,network information, domain registration">

<style type="text/css">

<!--

p, li, td, ul { font-family: Arial, Helvetica, sans-serif}

-->

</style>

</head>

**HttpURLConnection:-**Java provides a subclass of URLConnectionthat provides support for HTTP connections. This class is calledHttpURLConnection. You obtain an HttpURLConnectionin the same way just shown, by calling openConnection()on a URLobject, but you must cast the result to HttpURLConnection. (Of course, you must make sure that you are actually opening an HTTP connection.) Once you have obtained a reference to an HttpURLConnectionobject, you can use any of the methods inherited from URLConnection. You can also use any of the several methods defined by HttpURLConnection. Here is a sampling:

**boolean getFollowRedirects() Method**:- Returns true if redirects are automatically followed and false otherwise. This feature is on by default.

E.g.: static boolean getFollowRedirects()

**getRequestMethod( ) Method**:- Returns a string representing how URL requests are made. The default is GET. Other options, such as POST, are available.

E.g.: String getRequestMethod( )

**getResponseCode() method**:- Returns the HTTP response code. –1 is returned if no response code can be obtained. An IOException is thrown if the connection fails.

E.g.: int getResponseCode( ) throws IOException

**getResponseMessage( ) Method**:- Returns the response message associated with the response code. Returns null if no message is available. An IOException is thrown if the connection fails.

E.g.: String getResponseMessage( ) throws IOException

**setFollowRedirects(boolean how) Method**:- If how is true, then redirects are automatically followed. If how is false, redirects are not automatically followed. By default, redirects are automatically followed.

E.g.: static void setFollowRedirects(boolean how)

**setRequestMethod(String how)**:- Sets the method by which HTTP requests are made to that specified by how. The default method is GET, but other options, such as POST, are available. If how is invalid, a ProtocolException is thrown.

E.g.: void setRequestMethod(String how) throws ProtocolException

The following program demonstrates HttpURLConnection. It first establishes a connection to www.google.com. Then it displays the request method, the response code and the response message. Finally, it displays the keys and values in the response header.

// Demonstrate HttpURLConnection.

import java.net.\*;

import java.io.\*;

import java.util.\*;

class net5

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

{ URL hp = new URL("http://www.google.com");

HttpURLConnection hpCon = (HttpURLConnection) hp.openConnection();

// Display request method.

System.out.println("Request method is " + hpCon.getRequestMethod());

// Display response code.

System.out.println("Response code is " + hpCon.getResponseCode());

// Display response message.

System.out.println("Response Message is " + hpCon.getResponseMessage());

}

}

The output produced by the program is shown here. (Of course, the exact response returned by www.google.comwill vary over time.)

Request method is GET

Response code is 200

Response Message is OK