**Object Oriented Programming Concepts:** object-oriented paradigm,   
methods, classes and instances, data types, arrays, control structures, looping statements, type conversion and casting, I/O – Reading and Writing Files, Constructors, overloading constructors, encapsulation, polymorphism, inheritance, Dynamic Method Dispatch

**INTRODUCTION TO JAVA:**

Java is an object oriented, general purpose language. It was invented by James Gosling and developed at Sun Microsystems, USA in 1991. Java is simple, portable, powerful and reliable.

**Features of JAVA:**

* Object oriented
* Compiled and interpreted
* Platform independent and portability
* Robust and secure
* Distributed
* Simple, small and familiar
* Multithreaded and interactive
* High performance
* Dynamic and extensible

**Object oriented :**

Java is a pure object oriented language. Everything in java is an object. Set of classes in java is simply referred as packages.

**Compile and interpreted:**

Every programming language can be either compiled or interpreted. But java combines both together. At first java compiler translates the source code into bytecode. This bytecode can be directly translated into machine code by the help of interpreter.

**Platform independent and portability:**

The most important feature of java language is, it is platform-neutral. Java is the first programming language which is not depend on any particular operating system or hardware.

Programs implemented in java can be executed at any platform and in anywhere.

This portability feature makes java becomes more popular for programming on Internet which interconnects different types of systems.

**Robust and secure:**

Java is robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in java ensures that programmers can not gain access to memory locations without proper authorization.

**Distributed:**

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet easily. This enables multiple programmers at multiple remote locations work together on a single project.

**Simple, small and familiar:**

Java is a small and simple language. Java does not use pointers, preprocessor header files, goto statements and many things. It also eliminates operator overloading and multiple inheritance.

Familiarity is another striking feature of java. To make the language familiar, it was modeled on C and C++ languages. Java uses many constructs of C and C++ and hence it looks like simplified version of C++.

**Multithreaded and interactive :**

Java handles multiple tasks simultaneously. We not need to wait for the application to finish one task before beginning another. For example, we can listen an audio while working with an Internet page and at the same time we can download an application from the network.

**High performance:**

By having the compiled and interpretation feature of java, the speed is high compared to C / C++.

It is also designed to reduce overheads during runtime. Multithreading also enhances the overall execution speed of java programs.

**Dynamic and extensible**

Java is capable of dynamically linking with new class libraries, methods and objects. Hence it becomes a dynamic language.

Java programs support functions written in other languages like C and C++. These functions are known as native methods. This facility enables the programmer to use the efficient functions available in those languages. Native methods are linked dynamically.

**ADDITIONAL FEATURES OF J2SE 5.0:**

* Ease of development
* Scalability and performance
* Monitoring and manageability
* Core XML support
* Supplementary character support
* JDBC RowSet.

Comparison with C and C++ :

|  |  |
| --- | --- |
| **Java and C** | Java does not have the C keywords goto, sizeof and typedef. |
| Java does not contain the data types struct, union and enum |
| Java does not define the type modifiers auto, extern, register, signed and unsigned |
| Java does not support an explicit pointer type. |
| Java does not have a preprocessor and therefore we cannot use #define, #include statements. |
| Java does not support defining variable arguments to functions. |
| Functions with no arguments must be declared with empty parenthesis and not with the void keyword as done in c |
| Java adds new operators such as instanceof and >>> |
| Java adds labeled break and continue statements. |

|  |  |
| --- | --- |
| **Java and C++** | Java does not support operator overloading |
| Java does not have template classes as in C++ |
| Java does not support multiple inheritances. This is done by “interface” |
| Java does not support global variables. Every variable and method is declared within a class. |
| Java does not use pointers |
| Java has replaced the destructor function with a finalize() function. |
| There are no header files in java. |

**Java and World Wide Web:**

Java is used in distributed environments such as internet. Before java the WWW was limited to the display of still images and texts. The incorporation of java into web pages has made it capable of supporting animation, graphics, games and a wide range of special effects. With the support of java, the web has become more interactive and dynamic.

**Web browsers:**

Web browsers are used to navigate through the information found on the internet.

They allow us to retrieve the information from the internet and display it using hypertext markup language(HTML). Some example browsers are

Hot java Netscape navigator Internet explorer

**Java support systems:**

The system requirements for delivering information on the internet are,

1. Internet connection – local computer should be connected to the Internet
2. Web server – A program that accepts request for information and sends the required documents.
3. Web browser – a program that provides success to WWW and runs java applets.
4. HTML – a language for creating hypertext for the web
5. APPLET tag – for placing java applets in HTML document.
6. Java code – java code is used for defining java applets.
7. Byte code – compiled java code.

**Java environment:**

Java environment includes the Java development kit(JDK) and Java Standard Library (JSL).

**Java development kit (JDK):**

JDK comes with the collection of tools that are used for developing and running java programs. It include,

**appletviewer :**

It enables us to run java applets

**java :**

java interpreter, which runs applets and applications by reading and interpreting bytecode files.

**javac:**

The java compiler, which translates java source code to byte code files that the interpreter can understand.

**javadoc:**

Creates HTML format documentation from java source code files.

**javah:**

Produces header files for use with native methods.

**javap:**

Java disassembler, which enables us to convert bytecode files into a program description.

**jdb:**

Java debugger, which help us to find errors in our programs.

**Java standard Library (JSL):**

The java standard library includes hundreds of classes and methods grouped into six functional packages.

Language support package : A collection of classes and methods required for

implementing basic features of java.

Utilities package : A collection of classes to provide utility functions such as date

and time.

Input / Output package : A collection of classes required for input /output manipulation.

Networking packages : A collection of classes for communicating with other computers

through internet.

AWT packages : The Abstract Window Tool kit package contains classes that implements platform independent graphical user interface.

Applet package : This includes a set of classes that allows us to create java applets.

**Java virtual machine (JVM):**

It is a program that interprets the intermediate java byte code and generates the desired output.

Real machine

|  |  |
| --- | --- |
| Operating system | |
| Java Virtual Machine | |
| Java Object Framework | |
| Compiler | Interpreter |
| User Application Interface | |

|  |
| --- |
| User |

Layer of interaction

The JVM is intermediary between the operating system and java object framework.

**CLASSES:**

A class is a user defined data type. Once the class type has been defined, we can create objectsof class type. In java the class objects are termed as **instances** of the classes, the data items or variables are referred as **fields** and the functions are referred as **methods**

General form of class definition is,

classclassname

{

statements;

}

The general form of inherited class is,

classclassname extends superclassname

{

fields declaration;

methods declaration;

}

There should not be a semicolon at the end of the java classes.

The keyword **extends** indicates that the properties of the superclass are extended / inherited to a derived class. Variables and methods of a class are declared inside the body.

**A simple java program:**

class first // declares a class, everything must be placed inside a class.

**{ class**is a keyword**. First** is a java identifier.

public static void main(String args[])

{

System.out.println(“welcome to java”);

}

}

**Explanation:**

public static void main(String args[])

it is a main method. Every java program must include the main() method. This is the starting point of the interpreter. A java application can have any number of class but only one of them must have a main method to initiate the execution.

**Public:**

Public is an access specifier that declares the main method as unprotected and therefore making accessible to all other classes.

**Static:**

The main() must always be declared as static. It makes the main method as belongs to the entire class and not a part of any objects of the class.

**Void:**

The type modifier void states that the main method does not return any value.

All parameters to a method are declared inside a pair parentheses. Here String args[] declares a parameter named args which contains an array of objects of a class type **String**

**The output line:**

System.out.println(“welcome to java”);

**Fields declaration:**

Data items are encapsulated in a class by placing data fields inside the body of the class definition. The variables are called instance variables because whenever an object of the class is instantiated.

**Example:**

class circle

{

float pi;

int r;

}

Here the class circle has one integer and one float type instance variables.

**METHODS DECLARATION:**

We must define methods for manipulating the data items declared in the class. Without the methods the class and its instance variables are invalid. Methods are declared inside the body of the class immediately after the declaration of instance variables.

The general form of method declaration is,

typemethodname(parameter-list)

{

method-body;

}

where,

type – the type of the value the method returns

methodname – name of the method

parameter-list – list of parameters

method-body – body of the method

**Example:**

classrectangle

{

intlength,breadth;

voidgetdetails(int a, int b)

{

length = a;

breadth=b;

}

}

**OBJECT :**

In java object creates a block of memory to store all the instance variables.

Creating an object means instantiating an object for the class. In java objects are created using the new operator. The new operator creates an object of the specified class.

rectangle rect1; //declare the object

rect1 = new rectangle(); //instantiate the object.

It also can be written as a single line,

rectangle rect1=new rectangle()’

**ACCESSING CLASS MEMBERS:**

Each object containing its own set of variables. All variables must be assigned values before they are used. The instance variables can be accessed by the concerned object with the help of *dot* operator.

objectnsme.variablename=value;

objectname.methodname(parameter-list);

example:

rect1.length = 20; rect1.width=10;

rect1.rectArea (); //calling the method using the object

**Data types:**

Java has its own character set. It is used to process different types of data like integer numbers, characters and strings.

**Constants:**

In java constant is a fixed value that never changes during the execution of a program. Java supports the following constant types.

1. Integer constant

There are three types of integer constant.

1. Decimal integer constant:

It contains set of digits from 0 to 9. Ex: 546, -87, 0.

1. Octal integer constant:

It contains any combination of digits form 0 to 7, with a preceding 0.

Ex: 0 05 087 0548

1. Hexa decimal integer constant:

It contains a sequence of digits preceded by OX. Here digits from 0 to 9 and alphabets from A to F are allowed to represent a hexadecimal integer constant. A to F represents the numbers from 10 to 15.

Ex: OX6D OX4B OXA

1. Floating point constants:

Floating point or real constants are used to represent the values with fractional point.

Ex: 3.14 -5.6 -2.39 0.762

1. Single character constant:

It contains a single character enclosed with a single quote marks.

Ex: ‘a’ ‘y’ ‘5’ ‘ ‘

Here the character constant ‘5’ is not the same as number 5.

The ‘ ‘ constant represents a blank space.

1. String Constant:

It is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces.

Ex: “java” “5000” “X+Y”

**Mixing Numeric Data Types:**

* Java will automatically convert int expressions to double values without loss of information

int i = 5;

double x = i + 10.5;

* To convert double expressions to int requires a *typecasting* operation and truncation will occur

i = (int) (10.3 \* x)

* To *round-up* instead of truncating add 0.5

i = (int) (10.3 \* x + 0.5)

**Mixed Mode Operations and Strings:**

* It is important to remember that “13” and 13 are not the same

Examples

System.out.println(“4” + “5”) // prints 45

System.out.println(“4” + 5) // prints 45

System.out.println(4 + 5) // prints 9

**Characters as Integers:**

* It is legal to assign a char to an int variable

int i = ‘a’; // assigns 97 to i

* It is legal to assign an int to an char variable

char c = 97; // assigns ‘a’ to c

* It is possible to perform arithmetic on char variables

charch = ‘a’;

ch = ch + 1; // assigns ‘b’ to ch

**Variables:**

A variable may take different values at different times during the execution of a program.

A variable name should be meaningful.

Variable names may consists of alphabets, digits, underscore ( \_ ) and dollar characters.

Variables must not begin with a digit.

Uppercase and lowercase are distinct ie the variable college is not the same as COLLEGE.

It should not be a keyword.

White spaces are not allowed.

Variable names can be of any length.

Examples:

Length height breadth total.

Declaration of variables:

Variables are the names of storage locations. Variables must be declared in order to use them in a program.

Declaration does the three thins,

1. It tells the compiler, what the variable name is.
2. It specifies what type of data the variable will hold.
3. The place of declaration decides the scope of the variable.

Syntax:

Data\_type variable1, variable2,…variableN;

Example:

int I;

float f1,f2;

double d;

char c1,c2;

**ARRAYS:**

An Array is a collection of related data items that shares a common name. a value in array is identified by index or subscript enclosed in a square brackets with array name.

An array can be of any variable type like integer, float, char and so on.

arrays can be classified into

* One dimensional arrays
* Two dimensional arrays
* Multi-dimensional arrays

Example:

int salary[10];

It creates an integer array with 10 variable sizes and currently it represents the salary of 10th employee.

**One – dimensional array:**

The collection of data items can be stored under a one variable name using only one subscript is called as one dimensional array. For example if we want to represent a set of five numbers (40, 25, 37, 18, 12) by an array variable number then it is represented as,

int number[]=new int[5];

now the computer allocates five storage locations as shown below,

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

number[0]

number[1]

number[2]

number[3]

number[4]

The values to the array contents can be assigned as follows.

number[0]=40

number[1]=25

number[2]=37

number[3]=18

number[4]=12

Hence the array numbers store the values as

|  |
| --- |
| 40 |
| 25 |
| 37 |
| 18 |
| 12 |

number[0]

number[1]

number[2]

number[3]

number[4]

**Creating an array:**

Arrays must be declared and created in the memory before they are used.

**Creation of array involves the three steps,**

1. Declare the array
2. Create memory locations
3. Put values into the memory locations.

**Declaration of arrays:**

Arrays in java may be declared in two forms,

**Form1:**

typearrayname[];

**Form 2:**

type[] arrayname;

**Examples:**

int number[];

floataverage[];

int[] count;

float [] marks;

We should not enter the size of the arrays in the declaration.

**Creation of arrays:**

After the declaration of array, we need to create it in the memory. Java permits us to create arrays using new operator only.

Syntax:

arrayname = new type[size];

examples:

number = new int[5];

average = new float[10];

these lines will creates the necessary memory locations for the above said arrays.

It is also possible to combine the two steps – declaration and creation into one as shown below,

int number[]=new int[5];

**Initialization of arrays:**

The final step is to put values into the created array. This process is known as initialization. This is done using the array subscripts.

arrayname [subscript]=value;

example:

number[0]=40;

number[1]=25;

…………….

Number[4]=12;

It is also possible to initialize arrays directly as shown,

typearrayname[]={list of value};

example:

int number[]={40,25,37,18,12};

**Two dimensional arrays:**

Two dimensional arrays are used in situation where a table of values needs to be stored in an array. These can be defined in the same fashion as in one dimensional arrays, except a separate pair of square brackets are required for each subscript.

2D arrays are stored in a row – column matrix, where the left index indicates the row and the right indicates the column.

example:

intmyArray[][];

myArray[][]=new int[3][4];

or

intmyArray[][]=new int[3][4];

ex:

classmultable

{

final static int rows=0;

final static int columns=0;

public static void main(String args[])

{

int product[][]=new int[rows][columns];

int row, column;

System.out.println(“multiplication table”);

System.out.println(“”);

inti,j;

for(i=10;i<rows;i++)

{

for(j=10;j<columns;j++)

{

Product[i][j]=i\*j;

System.out.println(“”+product[i][j]);

}

System.out.println(“”);

}

}

}

**Operators in JAVA:**

An operator is a symbol that operates on one or more arguments to produce a result.

Java provides a rich set of operators to manipulate variables.

**Operands**

Operands are the values on which the operators act upon.

An operand can be:

**A numeric variable** – integer or floating point

**Any primitive type variable** - numeric and boolean

**Reference variable** to an object

**A literal** - numeric value, boolean value, or string.

**An array element**, "a[2]“

**Types of Operators:**

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment Decrement Operators
6. Bitwise Operators
7. Ternary Operators
8. Comma Operators
9. Instanceof Operators

**Arithmetic Operators:**

The arithmetic operators are used to construct mathematical expressions as in algebra.

Their operands are of numeric type.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| + | Addition |
| - | Subtraction (also unary minus) |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| += | Addition assignment |
| -= | Subtraction assignment |
| \*= | Multiplication assignment |
| /= | Division assignment |
| %= | Modulus assignment |
| -- | Decrement |

Example:

public class Example

{

public static void main(String[] args)

{

int j, k, p, q, r, s, t;

j = 5;

k = 2;

p = j + k;

q = j - k;

r = j \* k;

s = j / k;

t = j % k;

System.out.println("p = " + p);

System.out.println("q = " + q);

System.out.println("r = " + r);

System.out.println("s = " + s);

System.out.println("t = " + t);

}

}

The result is,

>java Example

p = 7

q = 3

r = 10

s = 2

t = 1

>

**Relational Operators:**

* A relational operator compares two values and determines the relationship between them.

For example, != returns true if its two operands are unequal.

* Relational operators are used to test whether two values are equal, whether one value is greater than another, and so forth.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Checks if the value of two operands are equal or not, if yes then condition becomes true. |
| != | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. |

Example:

**Public** LessThanEx

{

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

{

int x = 5; int y = 10;

if(x < y)

{

**System**.out.println("x is less than y");

}

}

}

**Logical operators:**

These logical operators work only on boolean operands. Their return values are always boolean.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | Logical AND operator. If both the operands are non zero then then condition becomes true. |
| || | Logical OR Operator. If any of the two operands are non zero then then condition becomes true. |
| ! | Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. |

**Example:**

**public class** ANDOperatorEx

{

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

{

charans = 'y';

int count = 1;

if(ans == 'y' & count == 0)

{

**System**.out.println("Count is Zero.");

}

if(ans == 'y' & count == 1)

{

**System**.out.println("Count is One.");

}

if(ans == 'y' & count == 2)

{

**System**.out.println("Count is Two.");

} } }

Assignment Operators:

The assignment statements has the following syntax:

**<variable> = <expression>**

Examples:

Assigning values:

inta,b;

a=2; // 2 is assigned to variable a

b=5; // 5 is assigned to variable b

Assigning references:

student s1=new student(); //new object s1 is created

student s2=s1; //assigning the reference of s1 in s2

**Increment& Decrement Operators:**

The increment and decrement operators adds and degrees an integer variable by one.

increment operator: ++

decrement operator: --

**Common**  **Shorthand**

**a = a + 1; a++; or ++a;**

**a = a - 1; a--; or --a;**

**public class Example**

**{**

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

**{**

int j, p, q, r, s;

j = 5;

p = ++j; // j = j + 1; p = j;

System.out.println("p = " + p);

q = j++; // q = j; j = j + 1;

System.out.println("q = " + q);

System.out.println("j = " + j);

r = --j; // j = j -1; r = j;

System.out.println("r = " + r);

s = j--; // s = j; j = j - 1;

System.out.println("s = " + s);

**}**

**}**

The result is,

**p = 6**

**q = 6**

**j = 7**

**r = 6**

**s = 6**

Bitwise Operators:

Bitwise operators operate on individual bits of integer (int and long) values.

If an operand is shorter than an int, it is promoted to int before doing the operations.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| a & b | and | 1 if both bits are 1. |
| a | b | or | 1 if either bit is 1. |
| a ^ b | xor | 1 if both bits are different. |
| ~a | not | Inverts the bits. |
| n << p | left shift | Shifts the bits of n left p positions. Zero bits are shifted into the low-order positions. |
| n >> p | right shift | Shifts the bits of n right p positions. If n is a 2's complement signed number, the sign bit is shifted into the high-order positions. |
| n >>> p | right shift | Shifts the bits of n right p positions. Zeros are shifted into the high-order positions. |

class Test {

public static void main(String args[]) {

int a = 60; /\* 60 = 0011 1100 \*/

int b = 13; /\* 13 = 0000 1101 \*/

int c = 0;

c = a & b; /\* 12 = 0000 1100 \*/

System.out.println("a & b = " + c );

c = a | b; /\* 61 = 0011 1101 \*/

System.out.println("a | b = " + c );

c = a ^ b; /\* 49 = 0011 0001 \*/

System.out.println("a ^ b = " + c );

c = ~a; /\*-61 = 1100 0011 \*/

System.out.println("~a = " + c );

c = a << 2; /\* 240 = 1111 0000 \*/

System.out.println("a << 2 = " + c );

c = a >> 2; /\* 215 = 1111 \*/

System.out.println("a >> 2 = " + c );

c = a >>> 2; /\* 215 = 0000 1111 \*/

System.out.println("a >>> 2 = " + c );

} }

**Output:**

**a & b = 12**

**a | b = 61**

**a ^ b = 49**

**~a = -61**

**a << 2 = 240**

**a >> 15 a >>> 15**

**Ternary Operators:**

Java has a short hand way by using **?:** the ternary conditional operator for doing **if**statement.

**int answer**;

**if (** a **>**b **)**

**{** answer**=** 1;  **}**

**else**

**{** answer**= -**1;  **}**

It can be written tersely with the ternary operator as:**int answer =** a **>**b **?**1 **: -**1;

**Comma Operators:**

Java uses comma operatorwithin it’s for loop.

**public class**CommaOperator {  
  **public static void**main(String[] args) {  
    **for**(**int**i = 1, j = i + 10; i < 5; i++, j = i \* 2)

 {  
      System.out.println("i= " + i + " j= " + j);  
    }  
  }  
}

**Instanceof Operators:**

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type).

InstanceOf operator is wriiten as:

**Syntax:**

**( Object reference variable ) instanceOf (class/interface type)**

class Vehicle {}

public class Car extends Vehicle

{

public static void main(String args[])

{

Vehicle a = new Car();

boolean result = **a instanceof Car**;

System.out.println( result);

}

}

Control statements:

**Selection Statements**

* + **if and if...else**
  + **Nested if Statements**
  + **Using switch Statements**
  + **Conditional Operator**

**Repetition Statements**

* + **Looping: while, do, and for**
  + **break and continue**

**if Statements:**

**Syntax:**

**if (Expression)**

**{**

**statement(s);**

**}**

**Example:**

**if ((i >= 0) && (i <= 10))**

**{**

**System.out.println("i is an “ + “integer between 0 and 10");**

**}**

**if...else Statement:**

**Syntax:**

**if (Expression)**

**{**

**statement(s); //for true case**

**}**

**else**

**{**

**statement(s); //for false case**

**}**

**Example:**

**if (age >= 18)**

**{**

**System.out.println("Eligible to cast vote ");**

**}**

**else**

**{**

**System.out.println("Not Eligible");**

**}**

**Conditional Operator:**

**if (x > 0) y = 1**

**else y = -1;**

**is equivalent to**

**y = (x > 0) ?1 : -1;**

**switch Statements:**

**switch (option)**

**{**

**case 1: statements;**

**break;**

**case 2: statements;**

**break;**

**case 3: statements;**

**break;**

**default: System.out.println(“Wrong number of years, enter 7, 15, or 30");**

**}**

**While:**

**Syntax:**

**while(condition)**

**{**

**// loop-body;**

**}**

**Do while:**

**Syntax:**

**do**

**{**

**// Loop body;**

**}**

**while (continue-condition)**

**for:**

**Example:**

**int i;**

**for (i = 0; i<100; i++)**

**{**

**System.out.println("Welcome to Java! ” + i);**

**}**

**Continue:**

**Continue makes the loop to skip the current execution and continues with the next iteration.**

**Example:**

**for(int i=0;i<50;i++)**

**{**

**if(i%13==0){**

**continue;**

**}**

**System.out.println(“Value of i:”+i);**

**}**

**CONSTRUCTORS:**

Constructor is a special type method that enables an object to initialize itself when it is created. Constructors have the same name as the class name. They do not have return type even void.

Example:

class rectangle

{

int length;

int width;

rectangle(int x, int y) // constructor method

{

length=x;

width=y;

}

intrectarea()

{

return(length \* width);

}}

**METHODS OVERLOADING:**

Creating methods with same name but having different parameter lists and definitions. This is known as method overloading. It is useful to perform similar tasks but using different input parameters. This process is known as polymorphism.

Example:

class area

{

float length;

float breadth;

area(float x, float y) // constructor 1

{

length=x;

breadth=y;

}

area(float x) // constructor 2

{

length=breadth=x;

}

voidcalcarea()

{

return (length \* breadth);

}}

The constructor method area() is overloaded here.

**STATIC MEMBERS:**

To define a member that is common to all objects and accessed without using a particular object. i.e., the static variable and the static methods are belongs to the entire class not for the particular object.

Example:

staticint value; // static variable

staicintcalc(int x, int y); // static method

**NESTING OF METHODS:**

Usually a method of a class can be called only by an object of that class. Sometimes a method can also be called by its name by another method of the same class. This is known as nesting of methods.

class nesting

{

intm,n;

nesting (int x, int y)

{

m=x;

n=y;

}

int largest()

{

if(m>=n)

return (m);

else

{

return(n);

}

void display()

{

int large=largest();

system.out.println(“largest value=”+large);

}

}

classnestingtest

{

public static void main(string args[])

{

nesting nest=new nesting(50,40);

nest.display();

}

}

**Type Casting**

[**https://www.geeksforgeeks.org/type-conversion-java-examples/**](https://www.geeksforgeeks.org/type-conversion-java-examples/)

**Streams &I/O interfaces**

The java.io package contains all the classes need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters, etc.

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.

## Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

## Character Streams

Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader**and **FileWriter.**

## Standard Streams

All the programming languages provide support for standard I/O where user's program can take input from a keyboard and then produce output on the computer screen.

**Standard Input:** This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.

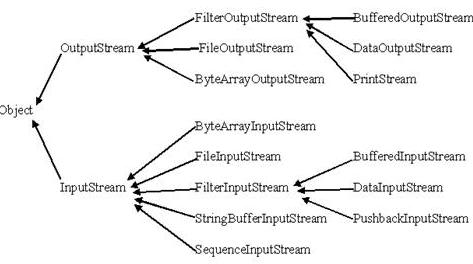
**Standard Output:** This is used to output the data produced by the user's program and usually a computer screen is used to standard output stream and represented as **System.out**.

**Standard Error:** This is used to output the error data produced by the user's program and usually a computer screen is used to standard error stream and represented as **System.err**.

## Reading and Writing Files:

A stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



Example program:

import java.io.\*;

classioexample

{

public static void main(String[] args)

{

try

{

File ip = new File("input.txt");

File op = new File("output.txt");

FileInputStreamfis = newFileInputStream(ip);

FileOutputStreamfos = newFileOutputStream(op);

int i;

while ((i = fis.read()) != -1)

{

fos.write(i);

}

fis.close();

fos.close();

}

catch (FileNotFoundExceptionfnfe)

{

System.err.println("FileStreamsTest: " +fnfe);

}

catch (IOExceptionioe)

{

System.err.println("FileStreamsTest: " + ioe);

}

}

}

INHERITANCE

Deriving a new class from an existing class is known as inheritance.

The existing class is known as base class or super class or parent class.

The new class is known as derived class or sub class or child class.

The derived class can inherit all the properties (methods and variables) of their parent class.

TYPES:

Single inheritance

Multiple inheritance

Multilevel inheritance

Hierarchical inheritance

Java does not support multiple inheritance directly.

It can be implemented by using the ***interfaces.***

**DEFINING A SUBCLASS:**

Syntax:

Class subclassname extends superclassname

{

Variables declaration;

Methods declaration;

}

The keyword ‘extends’ is used for extending the properties of super class to sub class. The sub class now contains its own variables and methods as well as the methods and variables of its super class.

Example:

Single Inheritance:

import java.io.\*;

class room

{

int length, breadth;

room(int a, int b)

{

length=a;

breadth=b;

}

int area()

{

return(length \* breadth);

}

}

classstudyroom extends room //inheriting from class room

{

int height;

studyroom(inta,intb,int c)

{

super(a,b); // pass the values to superclass

height=c;

}

int volume()

{

return(length\*breadth\*height);

}

}

classsingleInheritance

{

public static void main(String args[])

{

studyroom room1=new studyroom(10,12,14);

int area1=room1.area(); // super class method

int volume1=room1.volume(); // child class method

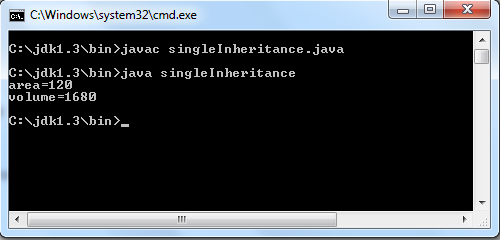
System.out.println("area="+area1);

System.out.println("volume="+volume1);

}

}

Output:



SUPER() METHOD:

A subclass constructor is used to construct the instance variables of both the subclass and the super class.

The subclass uses the keyword super to invoke the constructor method of the super class.

* Super may only be used with in a sub class constructor.
* The call to the super class constructor must appear as the first statement with in the sub class constructor.
* The parameters in the super() call must match with the variables declared in the superclass.

MULTI LEVEL INHERITANCE:

Syntax:

Class A

{

-------

-------

}

Class B extends A //level 1 inheritance

{

-------

-------

}

Class C extends B //level 2 inheritance

{

-------

-------

}

METHOD OVERRIDING:

It is possible to define a method that has the same name, same arguments and same return type in both the super class and the sub class. When that method is called the method defined in the sub class is invoked and executed instead of the method defined in the superclass.

Example:

import java.io.\*;

class base

{

int a;

base(int a)

{

this.a=a;

}

void display()

{

System.out.println("base a="+a);

}

}

class child extends base

{

int b;

child(inta,int b)

{

super(a);

this.b=b;

}

void display()

{

System.out.println("base a="+a);

System.out.println("child b="+b);

}

}

class override

{

public static void main(String args[])

{

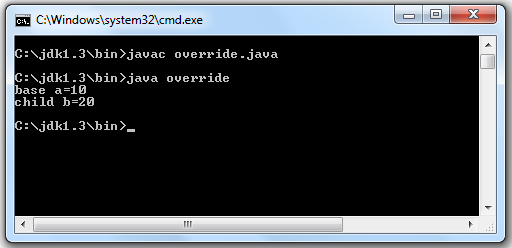
child c1=new child(10,20);

c1.display();

}

}

Output



FINAL VARIABLES & FINAL METHODS:

In java all variables and methods can be overridden in sub classes by default. To prevent the sub classes from overriding the members of the super class, we can declare them as final using the final keyword.

Example:

finalint size=100; //final variable

final void display(); //final method

{

…..

…..

}

**FINAL CLASSES:**

Final class ensures that, a class cannot be inherited to a sub class.

Example:

final class A

{

…….

……..

}

Final class prevents any unwanted extensions to another class.

**FINALIZER METHOD:**

It is similar to destructor in c++. Java calls the finalize() method whenever it is need to reclaim the space for the object.