**History of Java**

**Java history** is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt

4) After that, it was called **Oak** and was developed as a part of the Green project.



Figure: OAK Tree

#### Why Oak name for java language?

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

#### Why Java name for java language?

7) **Why did they choose java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996).

### Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

# Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

### Simple

|  |
| --- |
| According to Sun, Java language is simple because: |
| syntax is based on C++ (so easier for programmers to learn it after C++). |
| removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. |
| No need to remove unreferenced objects because there is Automatic Garbage Collection in java. |

### Object-oriented

|  |
| --- |
| Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. |
| Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules. |
| Basic concepts of OOPs are: |
| 1. Object 2. Class 3. Inheritance 4. Polymorphism 5. Abstraction 6. Encapsulation |

### Platform Independent

|  |
| --- |
| A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components:   1. Runtime Environment 2. API(Application Programming Interface) |

|  |
| --- |
| Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA). |

### Secured

|  |
| --- |
| Java is secured because: |
| * No explicit pointer * Programs run inside virtual machine sandbox. |

|  |  |
| --- | --- |
|  | how java is secured |

|  |
| --- |
| * **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources. * **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects. * **Security Manager-** determines what resources a class can access such as reading and writing to the local disk. |
| These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc. |

### Robust

|  |
| --- |
| Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points make java robust. |

### Architecture-neutral

|  |
| --- |
| There are no implementation dependent features e.g. size of primitive types is set. |

### Portable

|  |
| --- |
| We may carry the java bytecode to any platform. |

### High-performance

|  |
| --- |
| Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++) |

### Distributed

|  |
| --- |
| We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet. |

### Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

# Simple Program of Java

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

### Requirement for Hello Java Example

|  |
| --- |
| For executing any java program, you need to   * install the JDK if you don't have installed it. * set path of the jdk/bin directory. * compile and run the java program |

### Creating hello java example

Let's create the hello java program:

class Simple{

public static void main(String args[]){

 System.out.println("Hello Java");

}

}

save this file as Simple.java

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

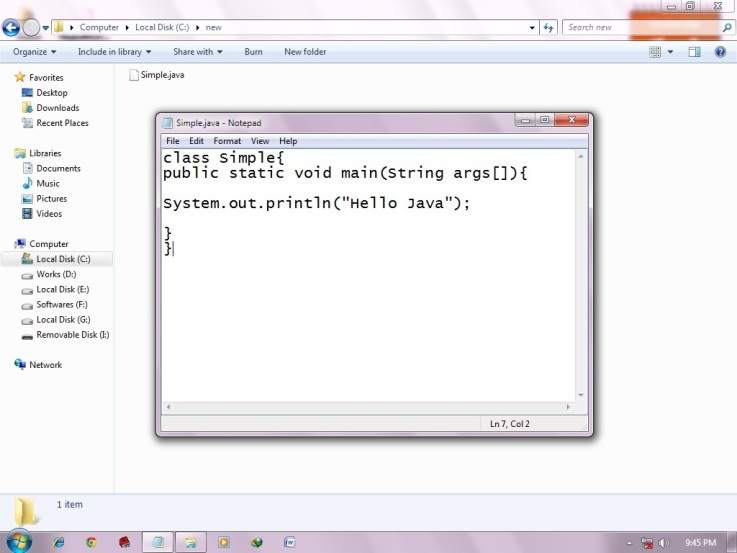
**Output:**Hello Java

### Understanding first java program

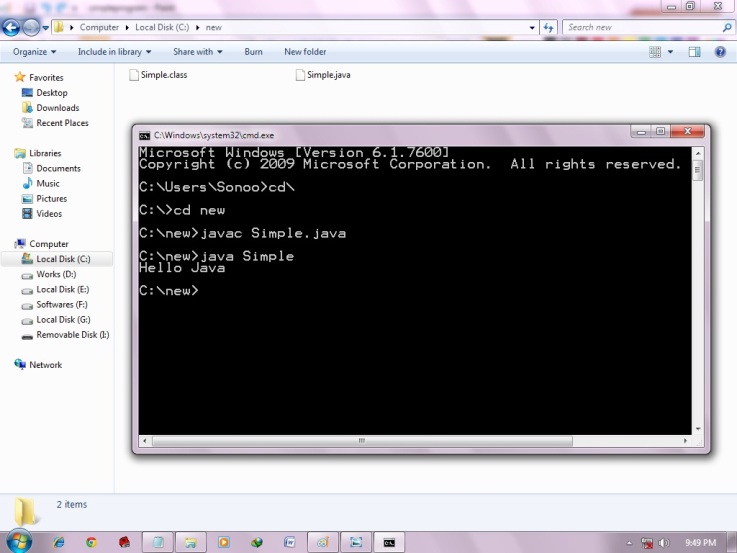
Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

* **class** keyword is used to declare a class in java.
* **public** keyword is an access modifier which represents visibility, it means it is visible to all.
* **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
* **void** is the return type of the method, it means it doesn't return any value.
* **main** represents startup of the program.
* **String[] args** is used for command line argument. We will learn it later.
* **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

To write the simple program, open notepad by **start menu -> All Programs -> Accessories -> notepad** and write simple program as displayed below:



|  |
| --- |
| As displayed in the above diagram, write the simple program of java in notepad and saved it as Simple.java. To compile and run this program, you need to open command prompt by **start menu -> All Programs -> Accessories -> command prompt**. |



|  |
| --- |
| To compile and run the above program, go to your current directory first; my current directory is c:\new . Write here: |

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

## How many ways can we write a java program

There are many ways to write a java program. The modifications that can be done in a java program are given below:

**1) By changing sequence of the modifiers, method prototype is not changed.**

Let's see the simple code of main method.

1. static public void main(String args[])

**2) subscript notation in java array can be used after type, before variable or after variable.**

Let's see the different codes to write the main method.

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])

**3) You can provide var-args support to main method by passing 3 ellipses (dots)**

Let's see the simple code of using var-args in main method. We will learn about var-args later in Java New Features chapter.

1. public static void main(String... args)

**4) Having semicolon at the end of class in java is optional.**

Let's see the simple code.

1. class A{
2. static public void main(String... args){
3. System.out.println("hello java4");
4. }
5. };

## Valid java main method signature

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])
4. public static void main(String... args)
5. static public void main(String[] args)
6. public static final void main(String[] args)
7. final public static void main(String[] args)
8. final strictfp public static void main(String[] args)

## Invalid java main method signature

1. public void main(String[] args)
2. static void main(String[] args)
3. public void static main(String[] args)
4. abstract public static void main(String[] args)

### Resolving an error "javac is not recognized as an internal or external command" ?

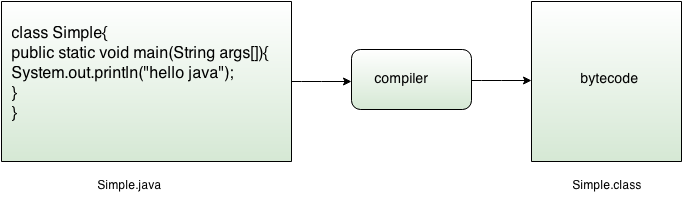
If there occurs a problem like displayed in the below figure, you need to set path. Since DOS doesn't know javac or java, we need to set path. Path is not required in such a case if you save your program inside the jdk/bin folder. But its good approach to set path.

# Internal Details of Hello Java Program

In the previous page, we have learned about the first program, how to compile and how to run the first java program. Here, we are going to learn, what happens while compiling and running the java program. Moreover, we will see some question based on the first program.

### What happens at compile time?

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



### What happens at runtime?

|  |
| --- |
| At runtime, following steps are performed: |
| what happens at runtime when simple java program runs |

|  |
| --- |
| **Classloader:** is the subsystem of JVM that is used to load class files. |
| **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects. |
| **Interpreter:** read bytecode stream then execute the instructions. |

### Q)Can you save a java source file by other name than the class name?

|  |
| --- |
| Yes, if the class is not public. It is explained in the figure given below: |
| how to save simple java program by another name |

|  |  |
| --- | --- |
| **To compile:** | javac Hard.java |
| **To execute:** | java Simple |

### Can you have multiple classes in a java source file?

|  |
| --- |
| Yes, like the figure given below illustrates: |
| how to contain multiple class in simple java program |

# How to set path in Java

The path is required to be set for using tools such as javac, java etc.

If you are saving the java source file inside the jdk/bin directory, path is not required to be set because all the tools will be available in the current directory.

But If you are having your java file outside the jdk/bin folder, it is necessary to set path of JDK.

There are 2 ways to set java path:

1. temporary
2. permanent

### 1) How to set Temporary Path of JDK in Windows

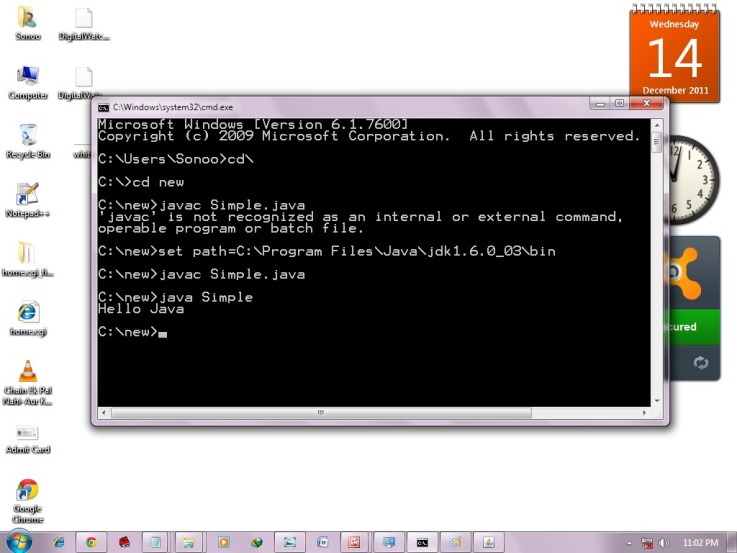
To set the temporary path of JDK, you need to follow following steps:

* Open command prompt
* copy the path of jdk/bin directory
* write in command prompt: set path=copied\_path

### For Example:

set path=C:\Program Files\Java\jdk1.6.0\_23\bin

Let's see it in the figure given below:



### 2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

* Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

### For Example:

|  |
| --- |
| **1)Go to MyComputer properties** |
| how to set path in java |
| **2)click on advanced tab** |
| how to set path in java |
| **3)click on environment variables** |
| how to set path in java |
| **4)click on new tab of user variables** |
| how to set path in java |
| **5)write path in variable name** |
| how to set path in java |
| **6)Copy the path of bin folder** |
| how to set path in java |
| **7)paste path of bin folder in variable value** |
| how to set path in java |
| **8)click on ok button** |
| how to set path in java |
| **9)click on ok button** |
| how to set path in java |

Now your permanent path is set.You can now execute any program of java from any drive.

# Difference between JDK, JRE and JVM

|  |
| --- |
| Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.  If you want to get the detailed knowledge of Java Virtural Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM. |
| JVM  |  | | --- | | JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. | | JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent. | | The JVM performs following main tasks:   * Loads code * Verifies code * Executes code * Provides runtime environment | |  |  JRE  |  | | --- | | JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM.It physically exists.It contains set of libraries + other files that JVM uses at runtime. | | Implementation of JVMs are also actively released by other companies besides Sun Micro Systems. | | jre |  JDK  |  | | --- | | JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools. | | jdk | |

# Internal Details of JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent).

### What is JVM?

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, and instance of JVM is created.

### What it does?

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

### Internal Architecture of JVM

|  |
| --- |
| Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc. |



### 1) Classloader:

Classloader is a subsystem of JVM that is used to load class files.

### 2) Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap:

It is the runtime data area in which objects are allocated.

### 4) Stack:

|  |
| --- |
| Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return. |
| Each thread has a private JVM stack, created at the same time as thread. |
| A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes. |

### 5) Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack:

It contains all the native methods used in the application.

### 7) Execution Engine:

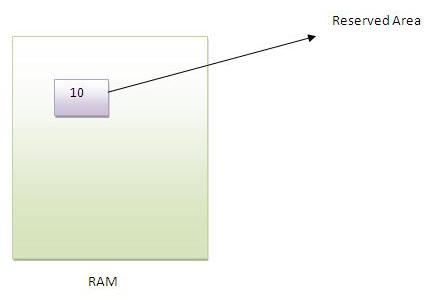
|  |
| --- |
| It contains: |
| **1) A virtual processor** |
| **2) Interpreter:**Read bytecode stream then execute the instructions. |
| **3) Just-In-Time(JIT) compiler:**It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU. |

# Variable and Datatype in Java

In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive.

### Variable

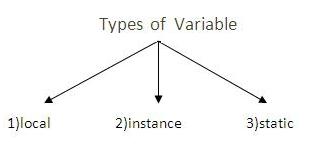
Variable is name of reserved area allocated in memory.



1. int data=50;//Here data is variable

### Types of Variable

|  |
| --- |
| There are three types of variables in java   * local variable * instance variable * static variable |



#### Local Variable

|  |
| --- |
| A variable that is declared inside the method is called local variable. |

#### Instance Variable

|  |
| --- |
| A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static. |

#### Static variable

|  |
| --- |
| A variable that is declared as static is called static variable. It cannot be local. |

|  |
| --- |
| We will have detailed learning of these variables in next chapters. |

#### Example to understand the types of variables

1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class

### Data Types in Java

|  |
| --- |
| In java, there are two types of data types   * primitive data types * non-primitive data types |



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

# Operators in java

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

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

**Java Programming Exercises**

# Java Programs

Java programs are frequently asked in the interview. These programs can be asked from control statements, array, string, oops etc. Let's see the list of java programs.

## 1) [Fibonacci series](http://www.javatpoint.com/fibonacci-series-in-java)

Write a java program to print fibonacci series without using recursion and using recursion.

**Input:** 10

**Output:** 0 1 1 2 3 5 8 13 21 34

## 2) [Prime number](http://www.javatpoint.com/prime-number-program-in-java)

Write a java program to check prime number.

**Input:** 44

**Output:** not prime number

**Input:** 7

**Output:** prime number

## 3) [Palindrome number](http://www.javatpoint.com/palindrome-program-in-java)

Write a java program to check palindrome number.

**Input:** 329

**Output:** not palindrome number

**Input:** 12321

**Output:** palindrome number

## 4) [Factorial number](http://www.javatpoint.com/factorial-program-in-java)

Write a java program to print factorial of a number.

**Input:** 5

**Output:** 120

**Input:** 6

**Output:** 720

## 5) [Armstrong number](http://www.javatpoint.com/armstrong-number-in-java)

Write a java program to check Armstrong number.

**Input:** 153

**Output:** Armstrong number

**Input:** 22

**Output:** not Armstrong number

## 6) [Bubble Sort](http://www.javatpoint.com/bubble-sort-in-java)

Write a java program to sort an array elements using bubble sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

## 7) [Selection Sort](http://www.javatpoint.com/selection-sort-in-java)

Write a java program to sort an array elements using selection sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

## 7) [Insertion Sort](http://www.javatpoint.com/insertion-sort-in-java)

Write a java program to sort an array elements using insertion sort algorithm.

**Input:** 18 9 33 4 84 32

**Output:** 4 9 18 32 33 84

**Getting User Input in Java**

* 1. Using BufferedReader Class
  2. Using DataInputStream Class
  3. Using Scanner class
  4. Using JOptionPane Class

1. **Using BufferedReader Class to get User Input**

We are providing you a java application that will communicate with the user at the command line and returns the user input.

We have prompt the user to enter the name by using **System.out.print()** method to keep the cursor on the same line. Then we have used the **System.in** object, along with the **InputstreamReader** and **BufferedReader** classes in order to read the user input. The **br.readline()** method reads the name from the command line. After pressing the enter key, you will get the user input.

**Here is the code of GetUserInput.java**

import java.io.\*;

public class GetUserInput {

    public static void main (String[] args) {

       System.out.print("Enter your name and press Enter: ");

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

       String name = null;

       try {

         name = br.readLine();

       } catch (IOException e) {

         System.out.println("Error!");

         System.exit(1);

       }

       System.out.println("Your name is " + name);

}

}

1. **Using DataInputStream Class to get input**

###### Example on Keyboard Input DataInputStream

Following program uses **readLine()** method of DataInputStream to read data from the keyboard.

import java.io.\*;

public class KeyboardReading1

{

  public static void main(String args[]) throws IOException

  {

    DataInputStream dis = new DataInputStream(System.in);

    System.out.println("Enter your name: ");

    String str1 = dis.readLine();

    System.out.println("I know your name is " + str1);

    System.out.println("Enter a whole number: ");

    String str2 = dis.readLine();

    int x = Integer.parseInt(str2);

    System.out.println("Enter a double value: ");

    String str3 = dis.readLine();

    double y = Double.parseDouble(str3);

    if(x > y)

      System.out.println("First number " +x + " is greater than second number " + y);

    else

      System.out.println("First number " +x + " is less than second number " + y);

    dis.close();

  }

}

The **readLine()** method of **DataInputStream** reads a line at a time and returns as a string, irrespective of what the line contains. Depending on the input value, the string is to be parsed into an **int** or **double** etc. This is extra work in the keyboard reading when compared to C/C++. In C/C++, parsing is done implicitly by %d and %f etc.

**Note:** This program compilation raises deprecation warning (observe the screen shot) as readLine() method of DataInputStream is deprecated, but the program works.

**What is "System.in"?**

" **in**" is an object of " **InputStream**" class defined in System class (like "out" is an object of PrintStream class defined in System class). It is declared as static and final object. The **in** object is connected implicitly to the " **standard input stream**" of the underlying OS.

DataInputStream dis = new DataInputStream(System.in);

**System.in** is a byte stream which is automatically connected to the keyboard reading mechanism of operating system. It is a low-level byte stream. The low-level byte stream is passed to high-level byte stream DataInputStream following the **rules of chaining**. It is chained to DataInputStream to facilitate the reading of user input with it 's readLine() method.

Like **System.in** is connected to the keyboard mechanism of OS, the **System.out** is connected to the standard output mechanism of OS.

1. **Using Scanner Class to get User Input**

The **Scanner** class is placed in **java.util** package (not in java.io) to facilitate to take input for DS (Data structure) elements. The advantage of Scanner class is parsing is not required for keyboard input data which is required with DataInputStream (as in DataInpuStream Example) and BufferedReader (as in BufferedReader Example).

import java.util.Scanner;

public class KeyboardReading3

{

  public static void main(String args[])

  {

    Scanner scan = new Scanner(System.in);

                                                  // reads a whole line

    System.out.println("Enter your name:");

    String s1 = scan.nextLine();

    System.out.println("Your name is " + s1);

                                                 // reads only one word

    System.out.println("City name:");

    String s2 = scan.next();

    System.out.println("City is " + s2);

                                                  // reading an integer

    System.out.println("Enter whole number:");

    int x = scan.nextInt();

                                                  // reading a double

    System.out.println("Enter double value:");

    double y = scan.nextDouble();

    System.out.println("Number are " + x + " and " + y + " and their product is " + (x\*y));

                          // all data entering at a time in a single line

    System.out.println("Enter name, age, maraks, height (floating-point) and \npromoted or not (boolean) (giving whitespace):");

    String name = scan.next();

    byte age = scan.nextByte();

    short marks = scan.nextShort();

    float height = scan.nextFloat();

    boolean promoted = scan.nextBoolean();

    System.out.println("The values entered: " + name + ", " + age + ", " + marks + ", " + height + " and " + promoted);

    scan.close();

  }

}

This way of taking keyboard input is much easier as **parsing is not required** by the programmer. The **Scanner** class includes special methods like **nextLine()**, **next()**, **nextByte()**, **nextShort()**, **nextFloat()** and **nextBoolean()** etc. which parses to the appropriate data type implicitly. These methods are used in the above program.

The nextLine() method reads whole line that may contain whitespaces in the middle of the input. The **next()** method reads one word only delimited by whitespace. For example, if the user enters "Hello World", the next() method reads only "Hello" leaving "World".

1. **Using JOptionPane Class to get User Input**

JOptionPane is a easy way to do dialog boxes, messages or inputs. You have to import it at the beginning of the program: import javax.swing.JOptionPane;

import javax.swing.JOptionPane; //import class JOptionPane   
  
public class Addition {   
public static void main ( [String](http://www.google.com/search?q=allinurl%3AString+java.sun.com&bntl=1) args[] )   
{   
[String](http://www.google.com/search?q=allinurl%3AString+java.sun.com&bntl=1) firstNumber, //first string entered by user   
         secondNumber; // second string entered by user   
  
int number1,     // first number to add   
    number2,    //second number to add   
    sum;                //sum of number1 and number2   
  
//read in the first number from user as a string   
firstNumber = [JOptionPane](http://www.google.com/search?q=allinurl%3AJOptionPane+java.sun.com&bntl=1).showInputDialog ( "Enter first integer" );   
  
//read in the second number from user as a string   
secondNumber = [JOptionPane](http://www.google.com/search?q=allinurl%3AJOptionPane+java.sun.com&bntl=1).showInputDialog ( "Enter second interger" );   
  
//convert numbers from type String to type int   
number1 = [Integer](http://www.google.com/search?q=allinurl%3AInteger+java.sun.com&bntl=1).parseInt ( firstNumber);   
number2 = [Integer](http://www.google.com/search?q=allinurl%3AInteger+java.sun.com&bntl=1).parseInt ( secondNumber);   
  
//add the numbers   
sum = number1 + number2;   
  
//display the results   
[JOptionPane](http://www.google.com/search?q=allinurl%3AJOptionPane+java.sun.com&bntl=1).showMessageDialog ( null, "The sum is " + sum);   
[System](http://www.google.com/search?q=allinurl%3ASystem+java.sun.com&bntl=1).exit ( 0 ); //ends the program   
}   
}

**NOTE:**   
JOptionPane can ONLY be used to input string, so if you want to use numbers, you have to create separate variables to change it into integers, or real numbers.

**Control Statements in Java Programming**

A program executes from top to bottom except when we use control statements, we can control the order of execution of the program, based on logic and values.

In Java, control statements can be divided into the following three categories:

* Selection Statements
* Iteration Statements
* Jump Statements

**Selection Statements**

Selection statements allow you to control the flow of program execution on the basis of the outcome of an expression or state of a variable known during runtime.

Selection statements can be divided into the following categories:

* The if and if-else statements
* The if-else statements
* The if-else-if statements
* The switch statements

**The if statements**

The first contained statement (that can be a block) of an if statement only executes when the specified condition is true. If the condition is false and there is not else keyword then the first contained statement will be skipped and execution continues with the rest of the program. The condition is an expression that returns a boolean value.

**Example:**

import java.util.Scanner;

public class IfDemo

{

public static void main(String[] args) {

int age;

Scanner inputDevice = new Scanner(System.in);

System.out.print("Please enter Age: ");

age = inputDevice.nextInt();

if(age > 18)

System.out.println("above 18 ");

}

}

**The if-else statements**

In if-else statements, if the specified condition in the if statement is false, then the statemet after the else keyword (that can be a block) will execute.

**Example:**

import java.util.Scanner;

public class IfElseDemo

{

public static void main( String[] args )

{

int age;

Scanner inputDevice = new Scanner( System.in );

System.out.print( "Please enter Age: " );

age = inputDevice.nextInt();

if ( age >= 18 )

System.out.println( "above 18 " );

else

System.out.println( "below 18" );

}

}

**The if-else-if statements**

This statement following the else keyword can be another if or if-else statement.  
  
That would looks like this:

**if(condition)  
   statements;  
else if (condition)**

**statements;  
else if(condition)**

**statement;  
else  
   statements;**  
  
Whenever the condition is true, the associated statement will be executed and the remaining conditions will be bypassed. If none of the conditions are true then the else block will execute.

**Example:**

import java.util.Scanner;

public class IfElseIfDemo

{

public static void main( String[] args )

{

int age;

Scanner inputDevice = new Scanner( System.in );

System.out.print( "Please enter Age: " );

age = inputDevice.nextInt();

if ( age >= 18 && age <=35 )

System.out.println( "between 18-35 " );

else if(age >35 && age <=60)

System.out.println("between 36-60");

else

System.out.println( "not matched" );

}

}

**The Switch Statements**

The switch statement is a multi-way branch statement. The switch statement of Java is another selection statement that defines multiple paths of execution of a program. It provides a better alternative than a large series of if-else-if statements.

**Example:**

import java.util.Scanner;

public class SwitchDemo

{

public static void main( String[] args )

{

int age;

Scanner inputDevice = new Scanner( System.in );

System.out.print( "Please enter Age: " );

age = inputDevice.nextInt();

switch ( age )

{

case 18:

System.out.println( "age 18" );

break;

case 19:

System.out.println( "age 19" );

break;

default:

System.out.println( "not matched" );

break;

}

}

}

An expression must be of a type of byte, short, int or char. Each of the values specified in the case statement must be of a type compatible with the expression. Duplicate case values are not allowed. The break statement is used inside the switch to terminate a statement sequence. The break statement is optional in the switch statement.

**Iteration Statements**

Repeating the same code fragment several times until a specified condition is satisfied is called iteration. Iteration statements execute the same set of instructions until a termination condition is met.  
  
Java provides the following loop for iteration statements:

* The while loop
* The for loop
* The do-while loop
* The for each loop

**The while loop**

It continually executes a statement (that is usually be a block) while a condition is true. The condition must return a boolean value.

**Example:**

public class WhileDemo {

public static void main( String[] args ) {

int i = 0;

while ( i < 5 )

{

System.out.println( "Value :: " + i );

i++;

}

}

}

**The do-while loop**

The only difference between a while and a do-while loop is that do-while evaluates its expression at the bottom of the loop instead of the top. The do-while loop executes at least one time then it will check the expression prior to the next iteration.

**Example**

public class DoWhileDemo {

public static void main( String[] args ) {

int i = 0;

do{

System.out.println( "value :: " + i );

i++;

}

while ( i < 5);

}

}

**The for loop**

A for loop executes a statement (that is usually a block) as long as the boolean condition evaluates to true. A for loop is a combination of the three elements initialization statement, boolean expression and increment or decrement statement.

**Syntax:  
  
for(<initialization>;<condition>;<increment or decrement statement>){**

**<block of code>**

**}**  
  
The initialization block executes first before the loop starts. It is used to initialize the loop variable.  
  
The condition statement evaluates every time prior to when the statement (that is usually be a block) executes, if the condition is true then only the statement (that is usually a block) will execute.  
  
The increment or decrement statement executes every time after the statement (that is usually a block).  
  
**Example:**

public class DoWhileDemo {

public static void main( String[] args ){

int i;

for(i=0; i<5;i++){

System.out.println( "value :: " + i );

}

}

}

**Jump Statements**

Jump statements are used to unconditionally transfer the program control to another part of the program.  
Java provides the following jump statements:

* break statement
* continue statement
* return statement

**Break Statement**

The break statement immediately quits the current iteration and goes to the first statement following the loop. Another form of break is used in the switch statement.

The break statement has the following two forms:

* Labeled Break Statement
* Unlabeled Break Statement

Unlabeled Break Statement: This is used to jump program control out of the specific loop on the specific condition.  
  
**Example:**

public class UnLabeledBreakDemo{

public static void main( String[] args ) {

for ( int var = 0; var < 5; var++ ) {

System.out.println( "Var is : " + var );

if ( var == 3 )

break;

}

}

}

**Labeled Break Statement:** This is used for when we want to jump the program control out of nested loops or multiple loops.

**Example:**

public class LabeledBreakDemo{

public static void main( String[] args ) {

Outer: for ( int var1 = 0; var1 < 5; var1++ ) {

for ( int var2 = 1; var2 < 5; var2++ ) {

System.out.println( "var1:" + var1 + ", var2:" + var2 );

if ( var1 == 3 )

break Outer;

}

}

}

}

**Continue Statement**

The continue statement is used when you want to continue running the loop with the next iteration and want to skip the rest of the statements of the body for the current iteration. The continue statement has the following two forms:

* Labeled Continue Statement
* Unlabeled Continue Statement

**Unlabeled Continue Statement:** This statement skips the current iteration of the innermost for, while and do-while loop.

**Example 1:** Unlabeled Continue

public class UnlabeledContinueDemo{

public static void main( String[] args ) {

for ( int var1 = 0; var1 < 4; var1++ ) {

for ( int var2 = 0; var2 < 4; var2++ ) {

if ( var2 == 2 )

continue;

System.out.println( "var1:" + var1 + ", var2:" + var2 );

}

}

}

}

**Example 2: L**abeled Continue

public class LabeledContinueDemo {

public static void main( String[] args ) {

Outer: for ( int var1 = 0; var1 < 5; var1++ ) {

for ( int var2 = 0; var2 < 5; var2++ ) {

if ( var2 == 2 )

continue Outer;

System.out.println( "var1:" + var1 + ", var2:" + var2 );

}

}

}

}

**Return Statement**

The return statement is used to immediately quit the current method and return to the calling method. It is mandatory to use a return statement for non-void methods to return a value.

**Example:**

public class ReturnDemo {

public static void main( String[] args ) {

ReturnDemo returnDemo = new ReturnDemo();

System.out.println( "No : " + returnDemo.returnCall() );

}

int returnCall() {

return 5;

}

}

**Java - Methods**

Definition 1:

A Java method is a collection of statements that are grouped together to perform an operation.

Definition 2:

A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name.

**Defining Methods:**

Method definition consists of a method header and a method body. The same is shown below:

modifier returnType nameOfMethod (Parameter List) {

// method body

}

**Example:**

public static int methodName(int a, int b) {

// body

}

Here,

* **public static** : modifier.
* **int**: return type
* **methodName**: name of the method
* **a, b**: formal parameters
* **int a, int b**: list of parameters

## Example:

Here is the source code of the above defined method called max(). This method takes two parameters num1 and num2 and returns the maximum between the two:

public static int MaxFunction(int n1, int n2) {

int max;

if (n1 > n2)

max = n1;

else

max = n2;

return max;

}

## Method Calling:

For using a method, it should be called. There are two ways in which a method is called i.e. method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when:

* return statement is executed.
* reaches the method ending closing brace.

Think of a method as a subprogram that acts on data and often returns a value.

When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code. As the following figure shows, the methods are called in main method.

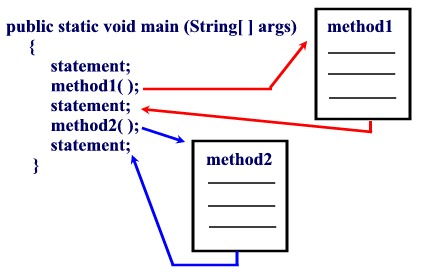


Figure: Method Calling

**Types of Methods**

There are two basic types of methods:

|  |
| --- |
| **Built-in:**  Build-in methods are part of the compiler package, such as System.out.println( ) and  System.exit(0). |
| **User-defined:** User-defined methods are created by you, the programmer. These methods take-on names that you assign to them and perform tasks that you create. |

**Working Exaple:**

public class MyMethod{

public static int sum(){

int a,b,c;

a=5;

b=10;

c=a+b;

return c;

}

public static int square(int x){

int sq=x\*x;

System.out.println(sq);

return 0;

}

public static void main(String args[]){

int d=sum();

System.out.println(d);

square(52);

}

}

## Overloading Methods

The Java programming language supports *overloading* methods, and Java can distinguish between methods with different *method signatures*. This means that methods within a class can have the same name if they have different parameter lists.

**Example:**

public class DataArtist {

...

public void draw(String s) {

...

}

public void draw(int i) {

...

}

public void draw(double f) {

...

}

public void draw(int i, double f) {

...

}

}

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different.

**Argument lists could differ in:**

1. Number of parameters.  
2. Data type of parameters.  
3. Sequence of Data type of parameters.

**Method overloading** is also known as **Static Polymorphism**.

**Working Example:**

public class Calculation{

public static void sum(int a,int b){

System.out.println(a+b);

}

public static void sum(int a,int b,int c){

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

}

public static void main(String args[]){

sum(10,10,10);

sum(20,20);

}

}

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

**Working Example 2:**

public class Calculation2{

public static void sum(int a,int b){

System.out.println(a+b);

}

public static void sum(double a,double b){

System.out.println(a+b);

}

public static void main(String args[]){

sum(10.5,10.5);

sum(20,20);

}

}

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

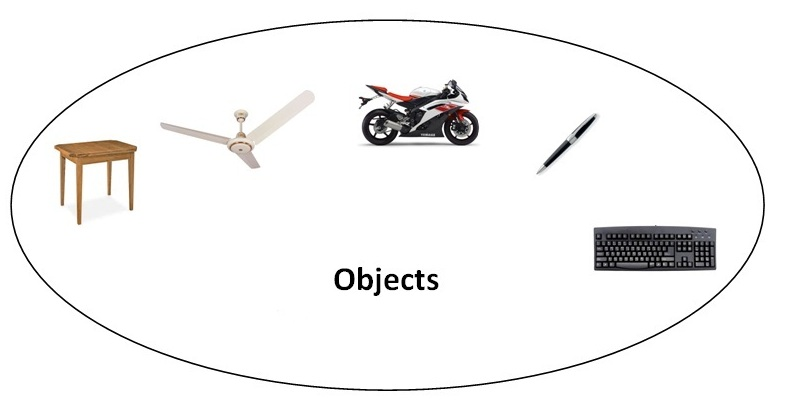
# Java OOPs Concepts

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

#### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

****

#### Class

**Collection of objects** is called class. It is a logical entity.

#### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

#### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

#### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

#### Encapsulation

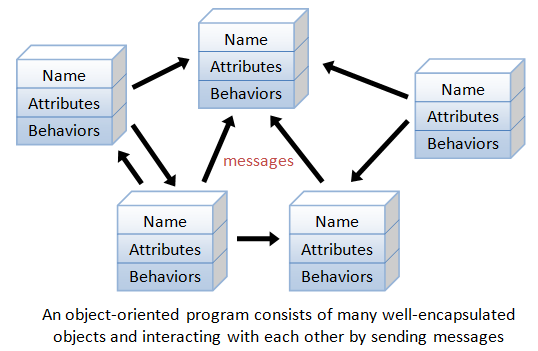
**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

### Advantage of OOPs over Procedure-oriented programming language

|  |  |
| --- | --- |
| 1) OOP makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. | |
| 2) OOP provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere. | |
| 3) OOP provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language. | |
| Global Data | Object Data |

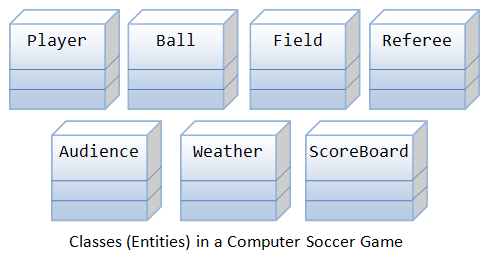
Object-oriented programming (OOP) languages are designed to overcome these problems.

1. The basic unit of OOP is a *class*, which encapsulates both the *static properties* and *dynamic operations* within a "box", and specifies the public interface for using these boxes. Since classes are well-encapsulated, it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.
2. OOP languages permit *higher level of abstraction* for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++ and C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.



As an example, suppose you wish to write a computer soccer games (which I consider as a complex application). It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" appear in the soccer games.

* Player: attributes include name, number, location in the field, and etc; operations include run, jump, kick-the-ball, and etc.
* Ball:
* Reference:
* Field:
* Audience:
* Weather:



Most importantly, some of these classes (such as Ball and Audience) can be reused in another application, e.g., computer basketball game, with little or no modification

##### Benefits of OOP

The procedural-oriented languages focus on procedures, with function as the basic unit. You need to first figure out all the functions and then think about how to represent data.

The object-oriented languages focus on components that the user perceives, with objects as the basic unit. You figure out all the objects by putting all the data and operations that describe the user's interaction with the data.

Object-Oriented technology has many benefits:

* Ease in software design as you could think in the problem space rather than the machine's bits and bytes. You are dealing with high-level concepts and abstractions. Ease in design leads to more productive software development.
* Ease in software maintenance: object-oriented software are easier to understand, therefore easier to test, debug, and maintain.
* Reusable software: you don't need to keep re-inventing the wheels and re-write the same functions for different situations. The fastest and safest way of developing a new application is to reuse existing codes - fully tested and proven codes

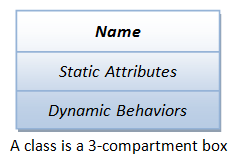
#### Class & Instances

In Java, a class is a definition of objects of the same kind. In other words, a class is a blueprint, template, or prototype that defines and describes the static attributes and dynamic behaviors common to all objects of the same kind.

An instance is a realization of a particular item of a class. In other words, an instance is an instantiation of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".

The term "object" usually refers to instance. But it is often used loosely, which may refer to a class or an instance.

#### A Class is a 3-Compartment Box encapsulating Data and Operations

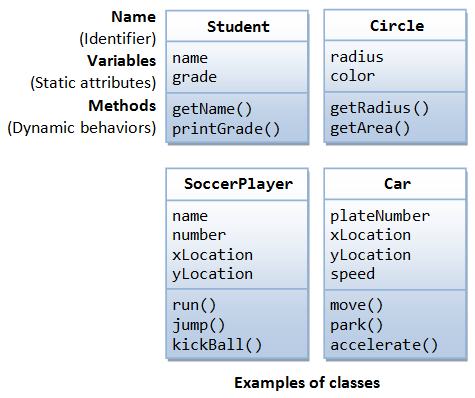
****

A class can be visualized as a three-compartment box, as illustrated:

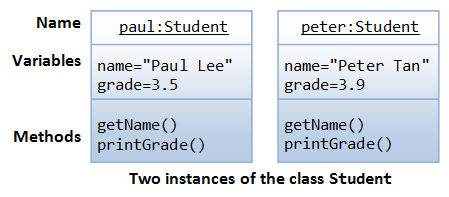
1. *Name* (or identity): identifies the class.
2. *Variables* (or attribute, state, field): contains the *static attributes* of the class.
3. *Methods* (or behaviors, function, operation): contains the *dynamic behaviors* of the class.

In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

The followings figure shows a few examples of classes:

****

The following figure shows two instances of the class Student, identified as "paul" and "peter".

****

Working Example of Class and Object:

class MyObject // Start of Class Definition{

int a, b;

public void Addition(){

int c=a+b;

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

}

}//End of class Definition

class MainClass{

public static void main(String args[]){

MyObject mo=new MyObject();//Creation of Instance Variable mo or Object mo

mo.a=10;

mo.b=25;

mo.Addition();//Accessing Class Member Function using Object

}

}

**Constructor in Java**

Constructor in Java is block of code which is executed at the time of Object creation. But other than getting called, Constructor is entirely different than methods and has some specific properties like name of constructor must be same as name of Class. Constructor also can not have any return type, constructor’s are automatically chained by using this keyword and super. Since Constructor is used to create object, object initialization code is normally hosted in Constructor. Similar to method you can also overload constructor in Java.

**Characteristics of Java Constructor**

1. First and most important rule of declaring constructor is that name of constructor in Java must be exactly same with the class on which you declare constructor.
2. Another important rule of declaring constructor is that constructor in Java doesn't have return type.

1. A Constructor is initialized up on the creation of object.

**Example 1:**

class ConstructorDemo{

public ConstructorDemo(){

System.out.println("No argument inside constructor");

}

public ConstructorDemo(String name){

System.out.println("One argument inside constructor in Java with name: " + name);

}

public static void main(String args[]){

ConstructorDemo d = new ConstructorDemo(); //calling no argument constructor in java

ConstructorDemo e = new ConstructorDemo("Testing"); //calling one argument constructor in java

}

}

**Example 2:**

class Example{

private int var;

public Example(){

//code for default one

var = 10;

String msg="Constructor";

System.out.println(msg);

}

public Example(int num) { //code for parameterized one

var = num;

}

public int getValue() {

return var;

}

public static void main(String args[]) {

Example ob = new Example();

int c=ob.getValue();

System.out.println("var is: "+c);

}

}