Java

Java is a **programming language** and a **platform**.

Platform Any hardware or software environment in which a program runs, known as a platform. Since Java has its own Runtime Environment (JRE) and API, it is called platform.

There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Java Applications

There are mainly 4 type of applications that can be created using java:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere (WORA)**, providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

Java Version History

There are many java versions that has been released.

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 (14th March,2014)

Features of Java

There is given many features of java. They are also known as java buzzwords.

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 behaviour.

Object-oriented programming(OOPs) is a methodology that simplify 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.

- 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 makes java robust.

Architecture-neutral

There is 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.

Sample Java Program

```
class Demo
 {
  public static void main(String []args)
{
 System.out.println("Hello world");
}
```

}

• **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.

• **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.

• **System.out.println()** is used print statement.

Basic Syntax

About Java programs, it is very important to keep in mind the following points.

Case Sensitivity − Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.

Class Names − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: class DemoProgram

Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: public void myMethodName()

package names - It should be in lowercase letter e.g. java, lang, sql, util etc.

constants name : It **s**hould be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Program File Name − Name of the program file should exactly match the class name.

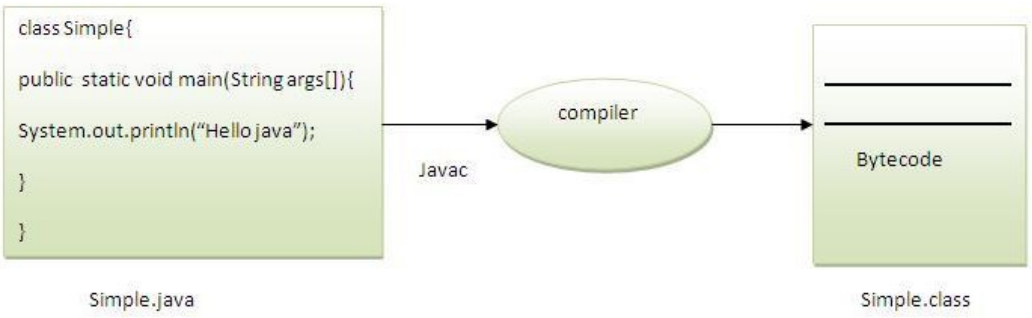
When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

Example:

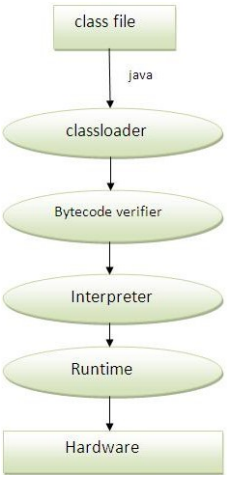
Assume 'DemoProgram' is the class name. Then the file should be saved as 'DemoProgram.java'

public static void main(String args[]) − Java program processing starts from the main() method which is a mandatory part of every Java program.

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



At runtime, following steps are performed:



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.

PATH and **CLASSPATH** parameters and create necessary directories(folders) where we will be storing all our program files

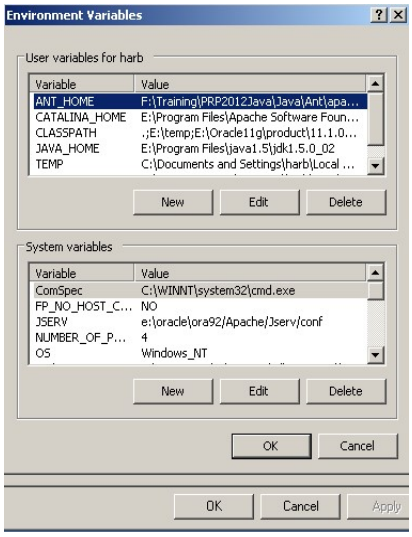
PATH is an environmental variable in DOS(Disk Operating System), Windows and other operating systems like Unix.

PATH tells the operating system which directories(folders) to search for executable files, in response to commands issued by a user .

It is a convenient way of executing files without bothering about providing the absolute path to the folder, where the file is located.

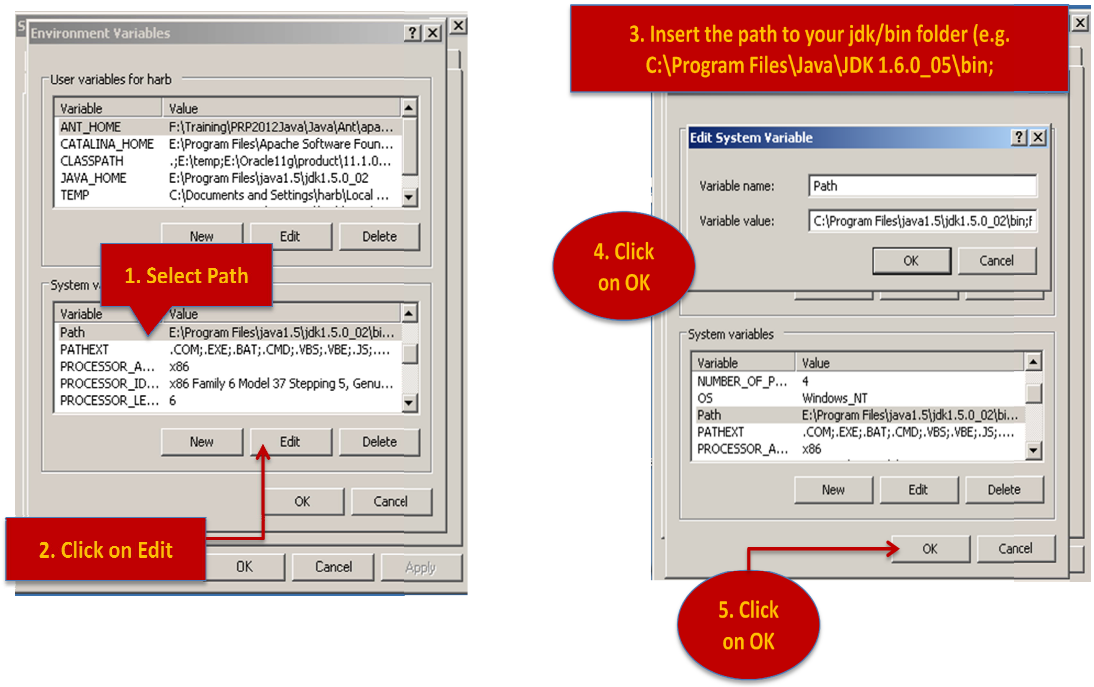
How to set PATH ?

- Right Click My Computer
- Select Properties
- You will get to see the Properties Page of My Computer
- Select Advanced Tab
- Select Environment Variables
- You will see Environment Variables Page as displayed here



When the Environment Variables page is displayed on your screen, you can see two options. User variables for <username> and System variables. The system variables once declared are available for every user who uses this system, where as User variables are declared to be used by the individual user.

Here PATH is already declared as a system variable. In the next page, we will see how to edit the existing PATH variable and add jdk folder to it.



Never overwrite the existing PATH value. When you click on Edit, you will get the see the Edit System Variable page which consists of Variable name and Variable value. You will see Variable value portion highlighted. Click somewhere within Variable value text box. It is no more highlighted. Click on Home button from your keyboard. It will take the cursor to the first position. Insert the path to JDK folder here.

CLASSPATH

CLASSPATH is a parameter which tells the JVM or the Compiler, where to locate classes that are not part of Java Development ToolKit(JDK).

CLASSPATH is set either on command-line or through environment variable.

CLASSPATH set on command-line is temporary in nature, while the environment variable is permanent.

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

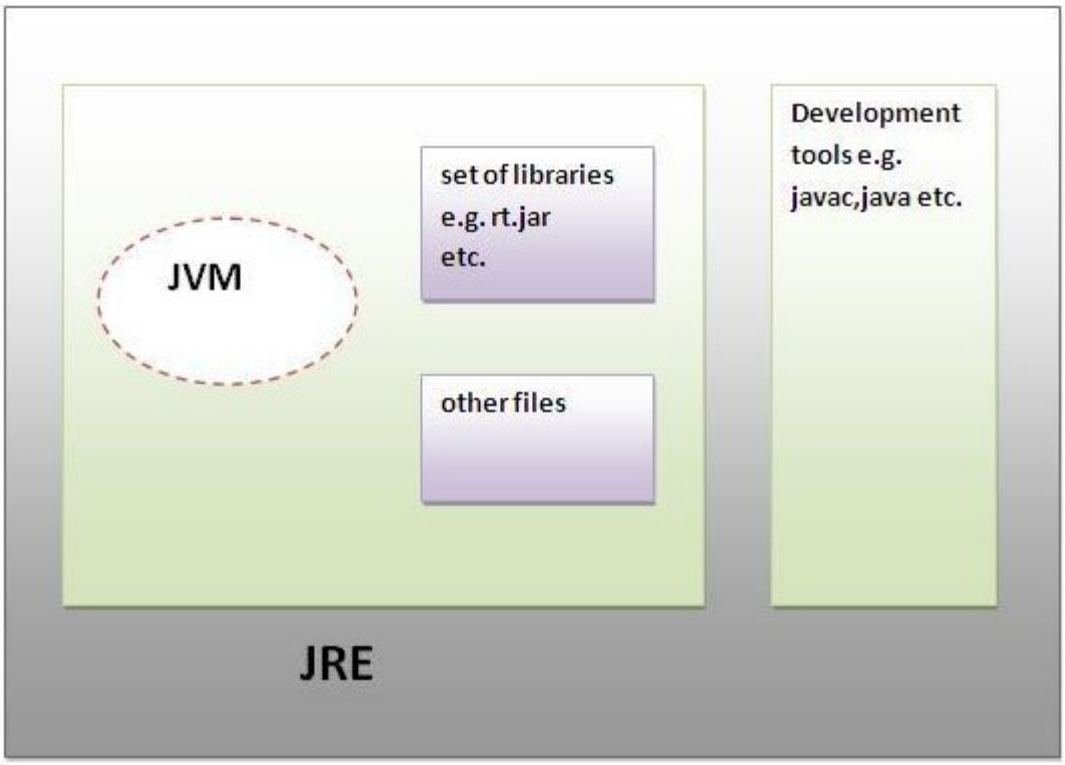
JRE

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.

JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



JDK

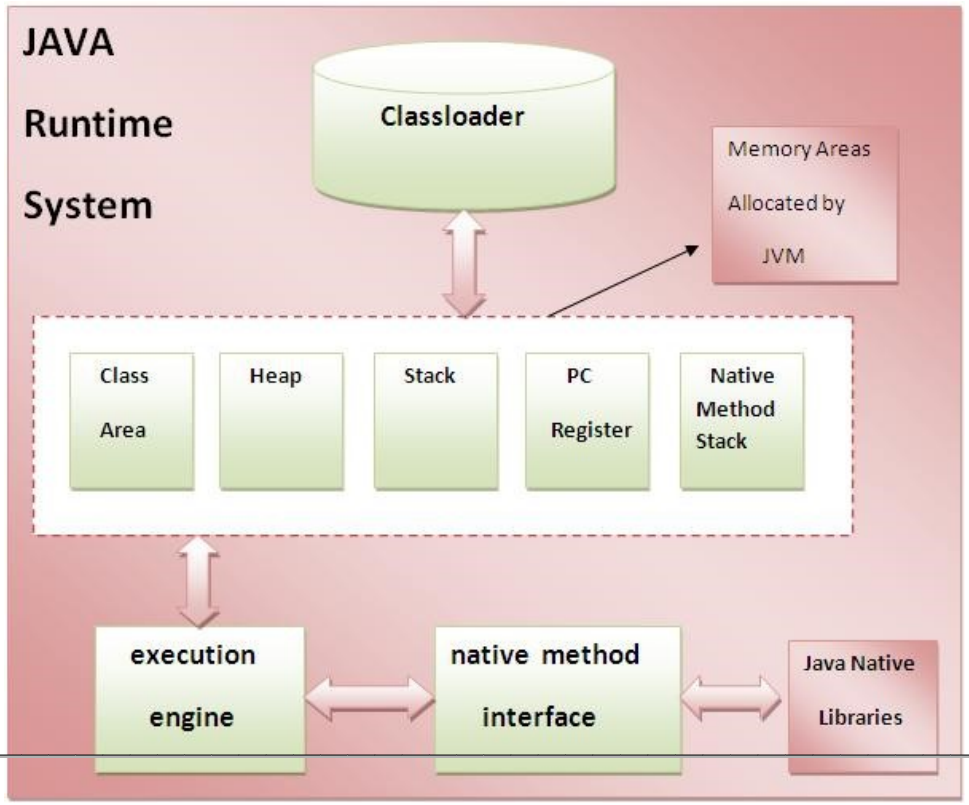
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap

Internal Architecture of JVM



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 Regiser:

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.

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

Object − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

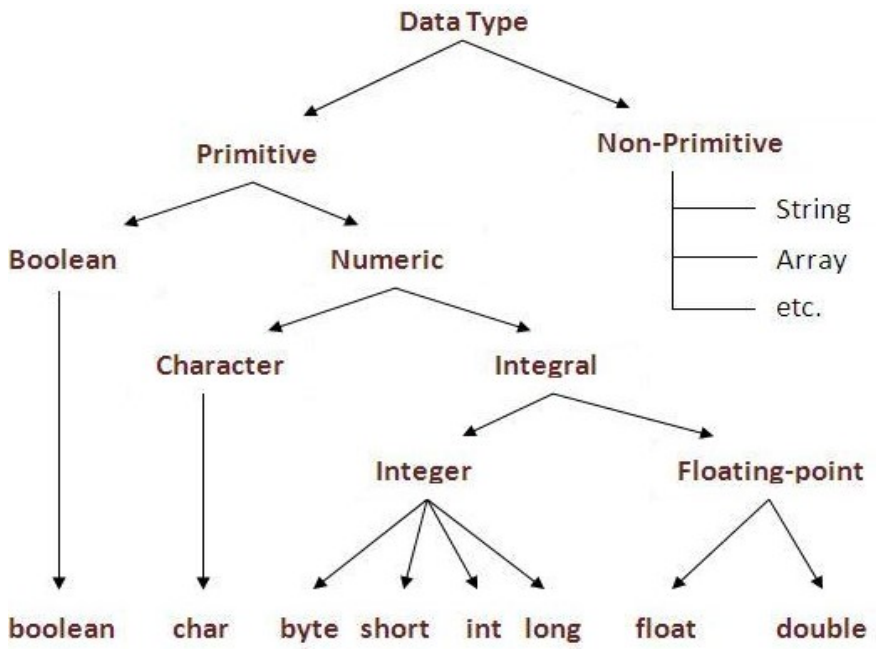
Methods − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

Instance Variables − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Data Types in Java

In java, there are two types of data types

- **primitive data types**
- **non-primitive data types (or) Reference type / Object Data type**



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

Reference Datatypes

Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Student, etc.

- *Class objects and various type of array variables come under reference datatype.*
- *Default value of any reference variable is null.*
- *A reference variable can be used to refer any object of the declared type or any compatible type*.

Example: Animal animal = new Animal("Lion");

Java Keywords

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

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

Java Identifiers

All Java components require names. Names used for classes, variables, and methods are called identifiers.

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

Java Variables :

Variables are nothing but reserved memory locations to store values. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, we can store integers, decimals, or characters in these variables.

1. Local Variables
2. Class Variables (Static Variables)
3. Instance Variables (Non-static Variables)

1. Local variables − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

- *Local variables are declared in methods, constructors, or blocks.*
- *Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.*
- *Access modifiers cannot be used for local variables.*
- *Local variables are visible only within the declared method, constructor, or block.*
- *Local variables are implemented at stack level internally.*
- *There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.*

2. Class variables − Class variables are variables declared within a class, outside any method, with the static keyword.

- *Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.*
- *There would only be one copy of each class variable per class, regardless of how many objects are created from it.*
- *Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.*
- *Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.*
- *Static variables are created when the program starts and destroyed when the program stops.*
- *Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.*
- *Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.*
- *Static variables can be accessed by calling with the class name ClassName.VariableName.*
- *When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.*

3. Instance variables − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

- *Instance variables are declared in a class, but outside a method, constructor or any block.*
- *When a space is allocated for an object in the heap, a slot for each instance variable value is created.*
- *Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.*

- *Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.*

- *Instance variables can be declared in class level before or after use.*

- *Access modifiers can be given for instance variables.*

- *The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.*

- *Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.*

- *Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.*

Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of Method overloading is it increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Modifiers are keywords that you add to those definitions to change their meanings. Java language has a wide variety of modifiers, including the following −

1. Java Access Modifiers

2. Non Access Modifiers

Access Control Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are −

- **Default** *: Visible to the package. No modifiers are needed.*
- **Private** *: Visible to the class only.*
- **Public** *: Visible to the world.*
- **Protected** *: Visible to the package and all subclasses.*

Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

- *The static modifier for creating class methods and variables.*
- *The final modifier for finalizing the implementations of classes, methods, and variables.*
- *The abstract modifier for creating abstract classes and methods.*
- *The synchronized and volatile modifiers, which are used for threads.*

Java - Basic Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

There are following relational operators supported by Java language.

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.

!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the	A >> 2 will give 15 which is 1111

	right operand.	
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Conditional Operator (? :)

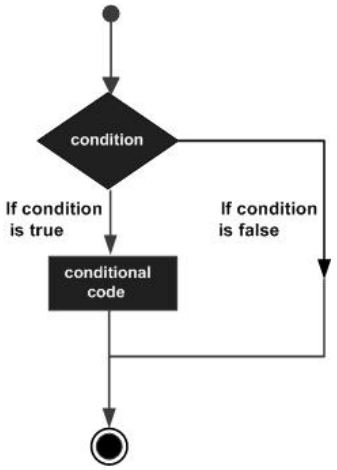
Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable.

instanceof Operator

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)

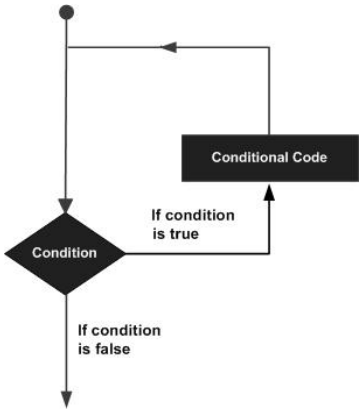
Flow of Controls

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



Sr.No.	Statement	Description
1	if statement	An if statement consists of a boolean expression followed by one or more statements.
2	if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
3	nested if statement	We can use one if or else if statement inside another if or else if statement(s).
4	switch statement	A switch statement allows a variable to be tested for equality against a list of values. Programming languages provide various control structures that allow for more complicated execution paths.

A **loop** statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Java programming language provides the following types of loop to handle looping requirements.

Sr.No.	Loop	Description
1	while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop	Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Java supports the following control statements.

Sr.No.	Control Statement	Description
1	break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement	Causes the loop to skip the remainder of its body and immediately

retest its condition prior to reiterating. Enhanced for loop in Java

As of Java 5, the enhanced for loop was introduced. This is mainly used to traverse collection of elements including arrays.

Syntax

Following is the syntax of enhanced for loop −

```
for(declaration : expression) {
  // Statements
}
```

- **Declaration** − The newly declared block variable, is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

- **Expression** − This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

Constructor

Constructor is a special type of method that is used to initialize the object.

Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor

Rules for creating constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name

2. Constructor must have no explicit return type

There are two types of constructors:

1. default constructor (no-arg constructor)

2. parameterized constructor

1) Default Constructor

A constructor that have no parameter is known as default constructor.

If there is no constructor in a class, compiler automatically creates a default constructor.

Note

Default constructor provides the default values to the object like 0, null etc. depending on the type.

2) Parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Parameterized constructor is used to provide different values to the distinct objects.

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Differences between constructor and method

Constructor	Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't	Method is not provided by compiler in

have any constructor.	any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

static keyword

The static keyword is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

---

1) static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

  Advantage of static variable

  It makes your program memory efficient (i.e it saves memory).

2) static method

If you apply static keyword with any method, it is known as static method

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.

2. this and super cannot be used in static context.

3)static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

this keyword

Usage of this keyword

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

Arrays

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable −

Syntax

dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[]; // works but not preferred way.

Note − The style dataType[] arrayRefVar is preferred. The style dataType arrayRefVar[] comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example

double[] myList;   // preferred way.

or

double myList[];   // works but not preferred way.

Creating Arrays

We can create an array by using the new operator with the following syntax −

Syntax

arrayRefVar = new dataType[arraySize];

The above statement does two things −

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively we can create arrays as follows −

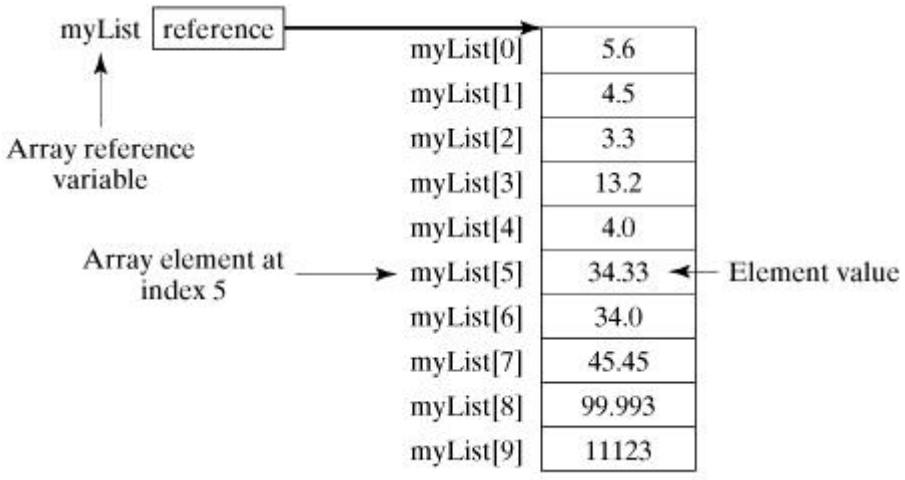
dataType[] arrayRefVar = {value0, value1, ..., valuek};

The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList −

double[] myList = new double[10];



Copying an array

We can copy an array to another by the arraycopy method of System class.

Syntax of **arraycopy** method

public static void arraycopy(

Object src, int srcPos,Object dest, int destPos, int length

)

String Handling in Java

String

Generally string is a sequence of characters. But in java, string is an object. String class is used to create string object.

There are two ways to create String object:

1. **By string literal**
2. **By new keyword**

1) String literal

String literal is created by double quote.

For Example:

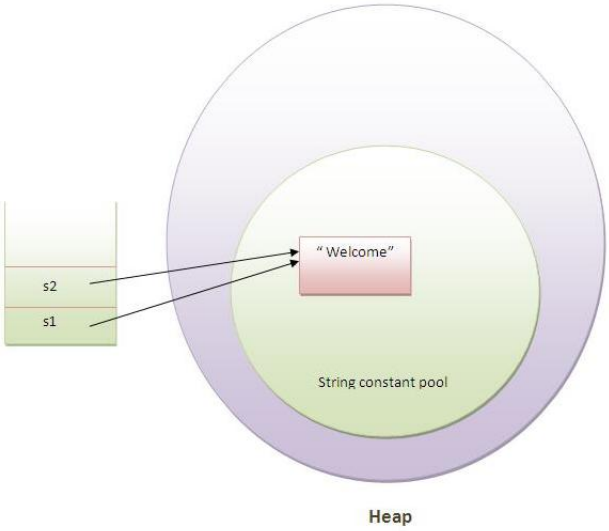
```
String s="Hello";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.

For example:

```
String s1="Welcome";
String s2="Welcome";//no new object will be created
```

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).



By new keyword

String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new String object in normal(nonpool) Heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in Heap(nonpool).

Immutable String in Java

In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

String comparison in Java

We can compare two given strings on the basis of content and reference.

It is used in authentication (by equals() method), sorting (by compareTo() method), reference matching (by == operator) etc.

There are three ways to compare String objects:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) By equals() method

equals() method compares the original content of the string.It compares values of string for equality. String class provides two methods:

• public boolean equals(Object another){} compares this string to the specified object.

• public boolean equalsIgnoreCase(String another){} compares this String to another String, ignoring case.

2) By == operator

The = = operator compares references not values.

3) By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than.

Suppose s1 and s2 are two string variables.If:

• s1 == s2 :0

• s1 > s2 :positive value

• s1 < s2 :negative value

String class has a variety of methods for string manipulation.

1. public char charAt(int index)

This method requires an integer argument that indicates the position of the character that the method returns. This method returns the character located at the String's specified index. Remember, String indexes are zero-based—

for example,

```
String x = "airplane";

System.out.println( x.charAt(2) ); // output is 'r'
```

2. **public String concat(String s)**

This method returns a String with the value of the String passed in to the method appended to the end of the String used to invoke the method—

for example,

String x = "book";

System.out.println(x.concat(" author")); // output is "book author"

The overloaded + and += operators perform functions similar to the concat()method—for example,

3. public boolean equalsIgnoreCase(String s)

This method returns a boolean value (true or false) depending on whether the value of the String in the argument is the same as the value of the String used to invoke the method. This method will return true even when characters in the String objects being compared have differing cases—

for example,

String x = "Exit";

System.out.println(x.equalsIgnoreCase("EXIT")); // is "true"

System.out.println(x.equalsIgnoreCase("tixe")); // is "false"

4. public int length()

This method returns the length of the String used to invoke the method—for example,

String x = "01234567";

System.out.println(x.length()); // returns "8"

5. public String replace(char old, char new)

This method returns a String whose value is that of the String used to invoke the method, updated so that any occurrence of the char in the first argument is replaced by the char in the second argument—

for example,

String x = "oxoxoxox";

System.out.println(x.replace('x', 'X')); // output is "oXoXoXoX"

6. public String substring(int begin)/ public String substring(int begin, int end)

The substring() method is used to return a part (or substring) of the String used to invoke the method. The first argument represents the starting location (zero-based) of the substring. If the call has only one argument, the substring returned will include the characters to the end of the original String. If the call has two arguments, the substring returned will end with the character located in the nth position of the original String where n is the second argument.

String x = "0123456789"; // the value of each char is the same as its index!

System.out.println(x.substring(5)); // output is "56789"

System.out.println(x.substring(5, 8)); // output is "567"

7. public String toLowerCase()

This method returns a String whose value is the String used to invoke the method, but with any uppercase characters converted to lowercase—

for example,

String x = "A New Java Book";

System.out.println(x.toLowerCase()); // output is "a new java book"

8. public String toUpperCase()

This method returns a String whose value is the String used to invoke the method, but with any lowercase characters converted touppercase—

for example,

String x = "A New Java Book";

System.out.println(x.toUpperCase()); // output is"A NEW JAVA BOOK"

9. public String trim()

This method returns a String whose value is the String used to invoke the method, but with any leading or trailing blank spaces removed—

for example,

String x = " hi ";

System.out.println(x + "x"); // result is" hi x"

System.out.println(x.trim() + "x"); // result is "hix"

10. public char[ ] toCharArray( )

This method will produce an array of characters from characters of String object. For example

String s = “Java”;

Char [] arrayChar = s.toCharArray(); //this will produce array of size 4

11. public boolean contains(“searchString”)

This method returns true of target String is containing search String provided in the argument.

For example-

String x = “Java is programming language”;

System.out.println(x.contains(“Amit”)); // This will print false

System.out.println(x.contains(“Java”)); // This will print true

12. boolean startsWith(String prefix)

Tests if this string starts with the specified prefix.

StringBuffer class:

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.

Commonly used Constructors of StringBuffer class:

1. StringBuffer(): creates an empty string buffer with the initial capacity of 16.

2. StringBuffer(String str): creates a string buffer with the specified string.

3. StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

Commonly used methods of StringBuffer class:

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str)**: is used to replace the string from specified startIndex and endIndex.

4. **public synchronized StringBuffer delete(int startIndex, int endIndex)**: is used to delete the string from specified startIndex and endIndex.

5. **public synchronized StringBuffer reverse()**: is used to reverse the string.

6. **public int capacity()**: is used to return the current capacity.

StringBuilder class:

The StringBuilder class is used to create mutable (modifiable) string. The StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK1.5.

Commonly used Constructors of StringBuilder class:

1. StringBuilder(): creates an empty string Builder with the initial capacity of 16.

2. StringBuilder(String str): creates a string Builder with the specified string.

3. StringBuilder(int length): creates an empty string Builder with the specified capacity as length.

toString() method

If we want to represent any object as a string, toString() method comes into existence.

The toString() method returns the string representation of the object.

If we print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of the toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

Inheritance (IS-A) vs. Composition (HAS-A) Relationship

One of the advantages of an Object-Oriented programming language is code reuse. There are two ways we can do code reuse either by the

- implementation of inheritance (IS-A relationship)
- object composition (HAS-A relationship).

IS-A Relationship:

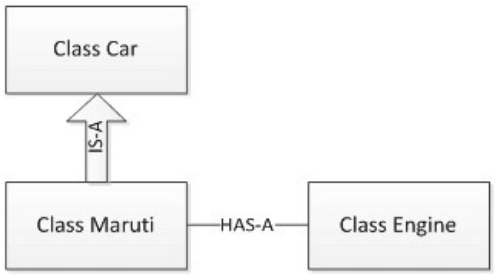
In object-oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional. For example, House is a Building. But Building is not a House.

It is a key point to note that you can easily identify the IS-A relationship. Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

HAS-A Relationship:

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Aggregation or Composition(HAS-A) simply mean the use of instance variables that are references to other objects. For example Maruti has Engine, or House has Bathroom.



Inheritance (IS-A Relationship)

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

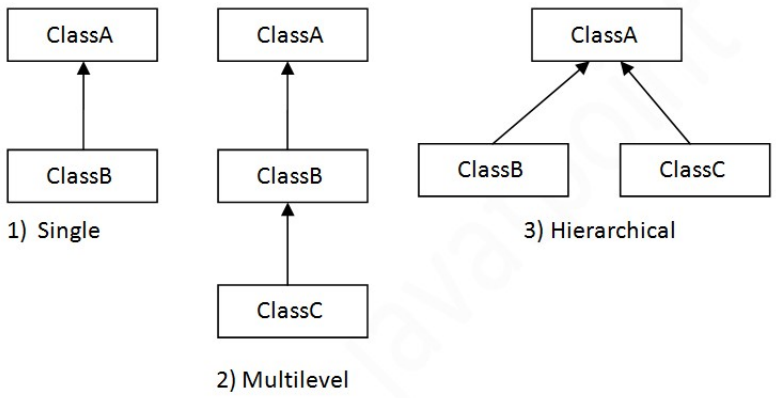
The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Java developed based on Single Inheritance Model.

Types of Inheritance

On the basis of class, there can be three types of inheritance: single, multilevel and hierarchical.



Method Overriding in Java

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as Method Overriding.

Rules for Method Overriding

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.

- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Note

static method cannot be overridden because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Differences between the method overloading and method overriding

Method Overloading	Method Overriding
1) Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2) method overloading is performed within a class.	Method overriding occurs in two classes that have IS-A relationship.
3) In case of method overloading parameter must be different.	In case of method overriding parameter must be same.

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type

super keyword

The super is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of super Keyword

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

Final Keyword In Java

The final keyword in java is used to restrict the user. The final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

final method

If you make any method as final, you cannot override it.

final class

If you make any class as final, you cannot extend it.

Runtime Polymorphism in Java

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting.

Downcasting

When reference variable of Child class refers to the object of Parent class, it is known as downcasting.

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

A class that is declared with abstract keyword, is known as abstract class.

abstraction is the quality of dealing with ideas rather than events.

For example, when we consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol our e-mail server uses are hidden from the user. Therefore, to send an e-mail we just need to type the content, mention the address of the receiver, and click send.

Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In Java, abstraction is achieved using **Abstract classes** and **interfaces**.

Abstract Class

A class which contains the abstract keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract Methods

If we want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, we can declare the method in the parent class as an abstract.

- abstract keyword is used to declare the method as abstract.
- You have to place the abstract keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semicolon (;) at the end.

Declaring a method as abstract has two consequences −

The class containing it must be declared as abstract.

Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Note − Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

abstract - rules

- Abstract classes can have concrete(non-abstract) methods in it.
- Abstract methods cannot be marked as final.
- Abstract classes cannot be marked as final.
- Abstract methods cannot be static.
- Abstract methods cannot be private

Interfaces

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- We cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.
- Declaring Interfaces
- The interface keyword is used to declare an interface.

Interfaces have the following properties −

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Implementing Interfaces

- When a class implements an interface, we can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.
- A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration

When implementation interfaces, there are several rules −

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

Tagging Interfaces

An interface with no methods in it is referred to as a tagging interface or Marker Interface.

A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.

Ex:

Remote ()

Clonnable()

Serilizable()

	Interfaces
An abstract class can extend another class and it can implement one or more interfaces.	An interface can extend one or more interfaces but cannot extend a class. It cannot implement an interface.
An abstract class can have constructors defined within it.	You cannot define constructors within an interface.
An abstract class cannot be instantiated using “new” Keyword	An interface cannot be instantiated.
You can execute(invoke) an abstract class, provided it has public static void main(String[] args) method declared within it.	You cannot execute an interface

Access Modifiers	Non-Access Modifiers
private	static
default or No Modifier	final
protected	abstract
public	synchronized
	transient
	volatile
	strictfp

Modifiers	Applicable to	And What is it ?	In simple terms
Static Modifier	Variables	The static key word is used to create variables that will exist independently of any instances created for the class. Only one copy of the static variable exists regardless of the number of instances of the class.	**Only one** always
	Methods	One rule-of-thumb: ask yourself "does it make sense to call this meth	
Final Modifier	Variables	A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to an different object.	Strictly **disables** the **Inheritance** concept
	Methods	A final method cannot be overridden by any subclasses. As mentioned previously the final modifier prevents a method from being modified in a subclass.	
	Classes	It prevent the class from being sub classed, if a class is marked as final then no class can inherit any feature from the final class.	
Abstract Modifier	Methods	An abstract method is a method declared without any implementation. The methods body(implementation) is provided by the subclass. Abstract methods can never be final or strict.	Strictly **enables** the **Inheritance** concept
	Classes	An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended.	
Synchronized Modifier	Methods	Synchronized methods enable a simple strategy for preventing thread interference and memory consistency errors: if an object is visible to more than one thread, all reads or writes to that object's variables are done through synchronized methods	**One thread** at a time
Volatile Modifier	Variables	The volatile modifier tells the JVM that writes to the field should always be synchronously flushed to memory, and that reads of the field should always read from memory. This means that fields marked as volatile can be safely accessed and updated in a multi-thread application without using native or standard library-based synchronization	**Do not cache** value of this variable and **always read/write it** from/to main memory
Transient Modifier	Variables	An instance variable is marked transient to indicate the JVM to skip the particular variable when serializing the object containing it. The transient keyword in Java is used to indicate that a field should not be serialized.	**Disable serialization**

Modifiers-Elements Matrix in Java

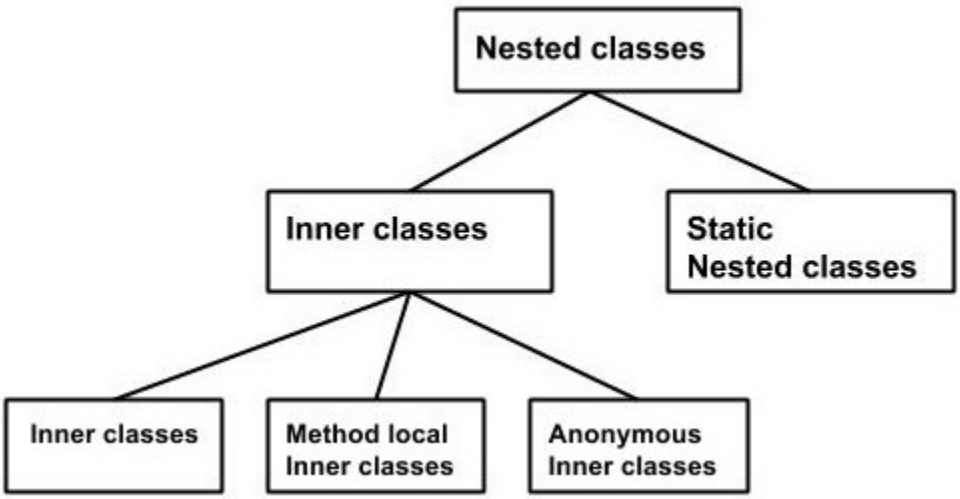
element / modifier	Data field	Method	Constructor	Class		Interface	
				top level (outer)	nested (inner)	top level (outer)	nested (inner)
abstract	no	yes	no	yes	yes	yes	yes
final	yes	yes	no	yes	yes	no	no
native	no	yes	no	no	no	no	no
private	yes	yes	yes	no	yes	no	yes
protected	yes	yes	yes	no	yes	no	yes
public	yes	yes	yes	yes	yes	yes	yes
static	yes	yes	no	no	yes	no	yes
synchronized	no	yes	no	no	no	no	no
transient	yes	no	no	no	no	no	no
volatile	yes	no	no	no	no	no	no
strictfp	no	yes	no	yes	yes	yes	yes

Nested Classes

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.

Nested classes are divided into two types −

- Non-static nested classes − These are the non-static members of a class.
- Static nested classes − These are the static members of a class.



Inner Classes (Non-static Nested Classes)

Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier private, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.

Inner classes are of three types depending on how and where you define them. They are −

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

Inner Class

Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

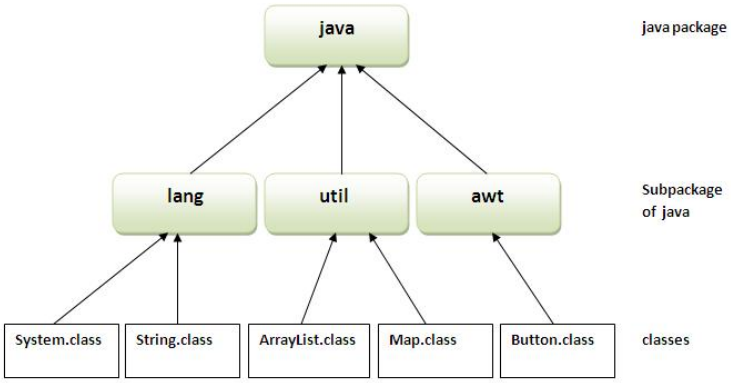
Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management.

Some of the existing packages in Java are −

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



Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

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

Creating a Package

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

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 then the class, interfaces, enumerations, and annotation types will be placed in the current default package.

To compile the Java programs with package statements, you have to use -d option as shown below.

```
javac -d Destination_folder file_name.java
```

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

The import Keyword

If a class wants to use another class in the same package, the package name need not be used.

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

Wrapper class in Java

Java is an object-oriented language and can view everything as an object. A simple file can be treated as an object and a simple data type can be converted into an object (with **wrapper classes**). This tutorial discusses wrapper classes. **Wrapper classes** are used to convert any data type into an object.

What are Wrapper classes?

As the name says, a wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type.

Wrapper class in java provides the mechanism *to convert primitive into object and object into primitive*.

The eight classes of *java.lang* package are known as wrapper classes in java. The list of eight wrapper classes are given below:

Primitive	Wrapper Class	Constructor Argument
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
int	Integer	int or String
float	Float	float, double or String
double	Double	double or String
long	Long	long or String
short	Short	short or String

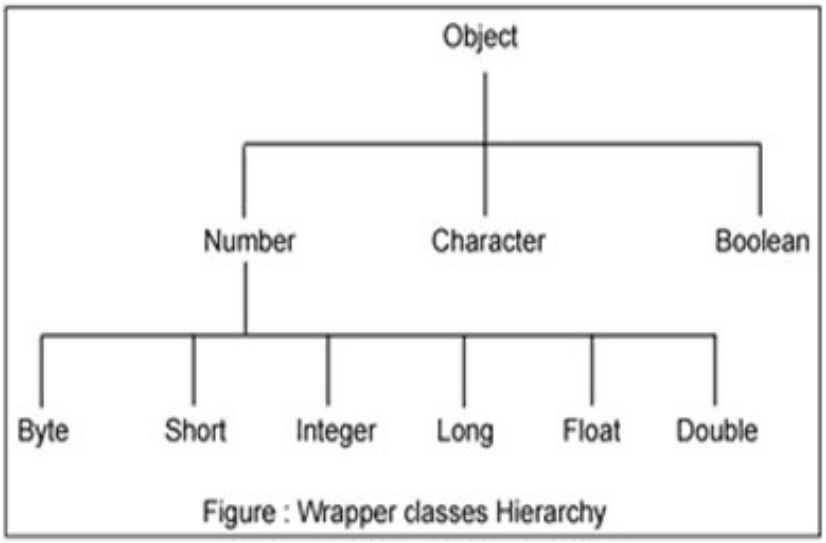


Figure : Wrapper classes Hierarchy

All the 8 wrapper classes are placed in java.lang package so that they are implicitly imported and made available to the programmer. As you can observe in the above hierarchy, the super class of all numeric wrapper classes is Number and the super class for Character and Boolean is Object. All the wrapper classes are defined as final and thus designers prevented them from inheritance.
Importance of Wrapper classes

There are mainly two uses with wrapper classes.

1. To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.

2. To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

Wrapping and Unwrapping

```
int a = 50;
Integer m1 = new Integer(a);
```

In the above statement, an **int** data type **a** is given an object form **m1** just by passing the variable to the constructor of **Integer** class. Wherever, **a** is required as an object, **m1** can be used.
After using the Integer object in programming, now the programmer may require back the data type, as objects cannot be used in arithmetic operations. Now the object **m1** should be unwrapped.

```
int iv = m1.intValue();
```

For unwrapping, the method **intValue()** of Integer class used. The **int** value **iv** can be used in arithmetic operations.
int value with **intValue()** method of Integer class.

Similarly, there exists methods like **floatValue()** and **doubleValue()** of classes Float and Double that return a float value and double value.

finally, a string value can be converted to a data type in two ways –

- using **parseInt()** method and
- using **intValue()** method; both belonging to Integer class.

The most common methods of the Integer wrapper class are summarized in below table.

Method	Purpose

parseInt(s)	returns a signed decimal integer value equivalent to string s
toString(i)	returns a new String object representing the integer i
byteValue()	returns the value of this Integer as a byte
doubleValue()	returns the value of this Integer as a double
floatValue()	returns the value of this Integer as a float
intValue()	returns the value of this Integer as an int
shortValue()	returns the value of this Integer as a short
longValue()	returns the value of this Integer as a long
int compareTo(int i)	Compares the numerical value of the invoking object with that of i. Returns 0 if the value lower value. Returns a positive value if the invoking object has a greater value.
static int compare(int num1, int num2)	Compares the values of num1 and num2. Returns 0 if the values are equal. Returns a neg num1 is greater than num2.
boolean equals(Object intObj)	Returns true if the invoking Integer object is equivalent to intObj. Otherwise, it returns f

valueOf (), toHexString(), toOctalString() and toBinaryString() Methods:

This is another approach to creating wrapper objects. We can convert from binary or octal or hexadecimal before assigning a value to wrapper object using two argument constructor

J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.