3B07BCA

Java Programming

fppt.com

Introduction

- Java is a high-level Object Oriented programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX

History of Java

- **The history of Java** is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time.
- The history of Java starts with the Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc.

- However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.
- The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic".

- Java was developed by **James Gosling**, who is known as the **father of Java**, in 1995. James Gosling and his team members started the project in the early '90s.
- Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc.

Significant points that describe the history of Java.

- **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Initially designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.
- After that, it was called Oak and was developed as a part of the Green project.

Why Java named "Oak"?

; a symbol of strength and chosen as a national tree
ıy countries like the U.S.A., France, Germany,
nia, etc.

)5, Oak was renamed as **"Java"** because it was
ly a trademark by Oak Technologies.

Why Java Programming named "Java"?

- 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 than other names.

- Java is an island of Indonesia where the first coffee was produced (called java coffee). Java name was chosen by James Gosling while having coffee near his office.
- Notice that Java is just a name, not an acronym.

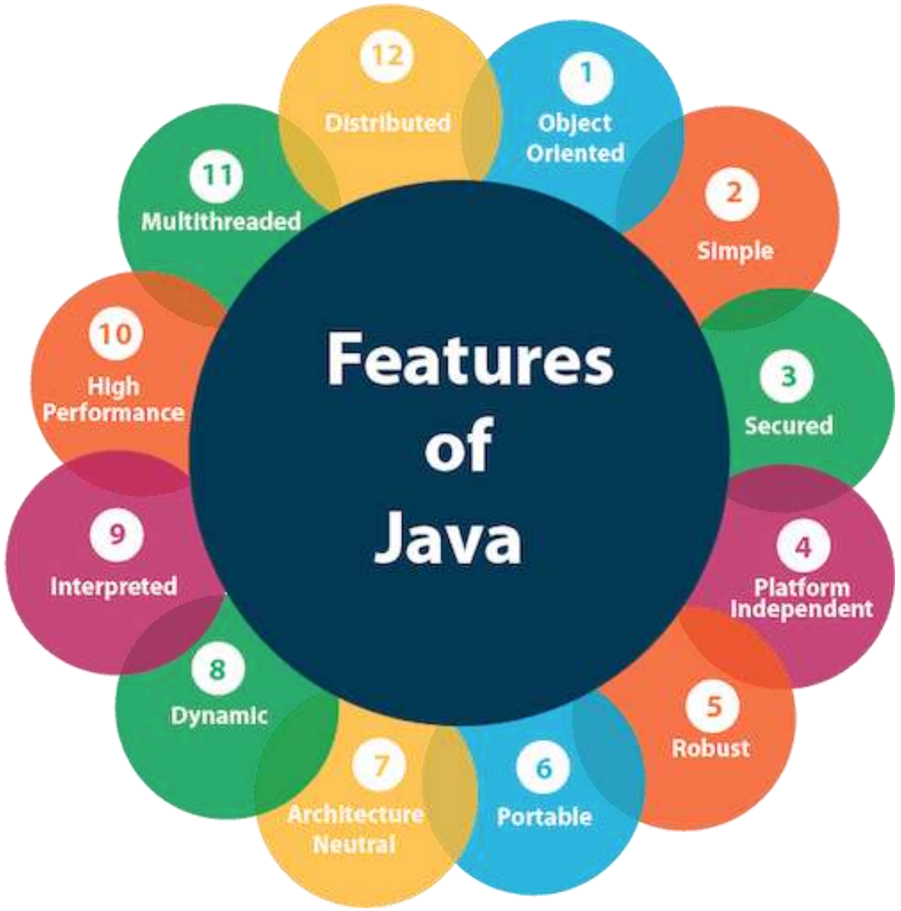
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
- JDK 1.0 released in(January 23, 1996). After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds the new features in Java.

Java Version History

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)

Features of Java

- The primary objective of Java programming language creation was to make it portable, simple and secure programming language.
- Apart from this, there are also some excellent features which play an important role in the popularity of this language.
- The features of Java are also known as java *buzzwords*.


Features of Java
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded
12 Distributed

Simple

- Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:
    - Java syntax is based on C++ (so easier for programmers to learn it after C++).
    - Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
    - There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

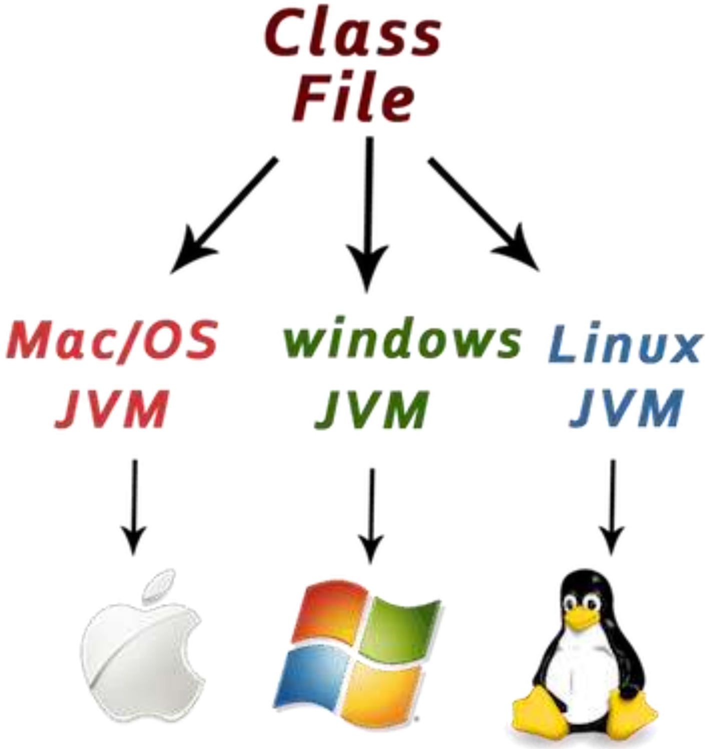
Object-oriented

- Java is an object-oriented programming language. Everything in Java is an object.
- Basic concepts of OOPs are:
    - Object
    - Class
    - Inheritance
    - Polymorphism
    - Abstraction
    - Encapsulation

Platform Independent

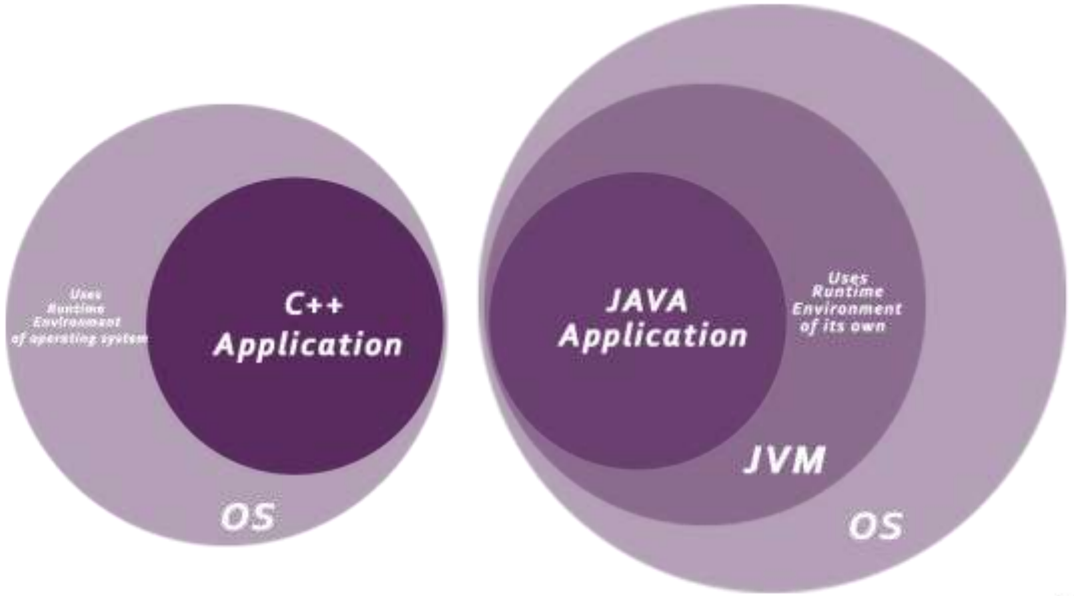
- Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a **Write Once Run Anywhere (WORA)** language.
- A platform is the hardware or software environment in which a program runs.

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


Class File
Mac/OS JVM
windows JVM
Linux JVM

Secured

- Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
    - **No explicit pointer**
    - **Java Programs run inside a virtual machine sandbox**
    - **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
    - **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
    - **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.
- Java language provides these securities by default. Some security can also be provided by an application developer explicitly


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

Robust

- Robust simply means strong. Java is robust because:
    - It uses strong memory management.
    - There is a lack of pointers that avoids security problems.
    - There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
    - There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java

Portable

- Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High Performance

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
- It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

- Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications.
- This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi Threading

- 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 doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

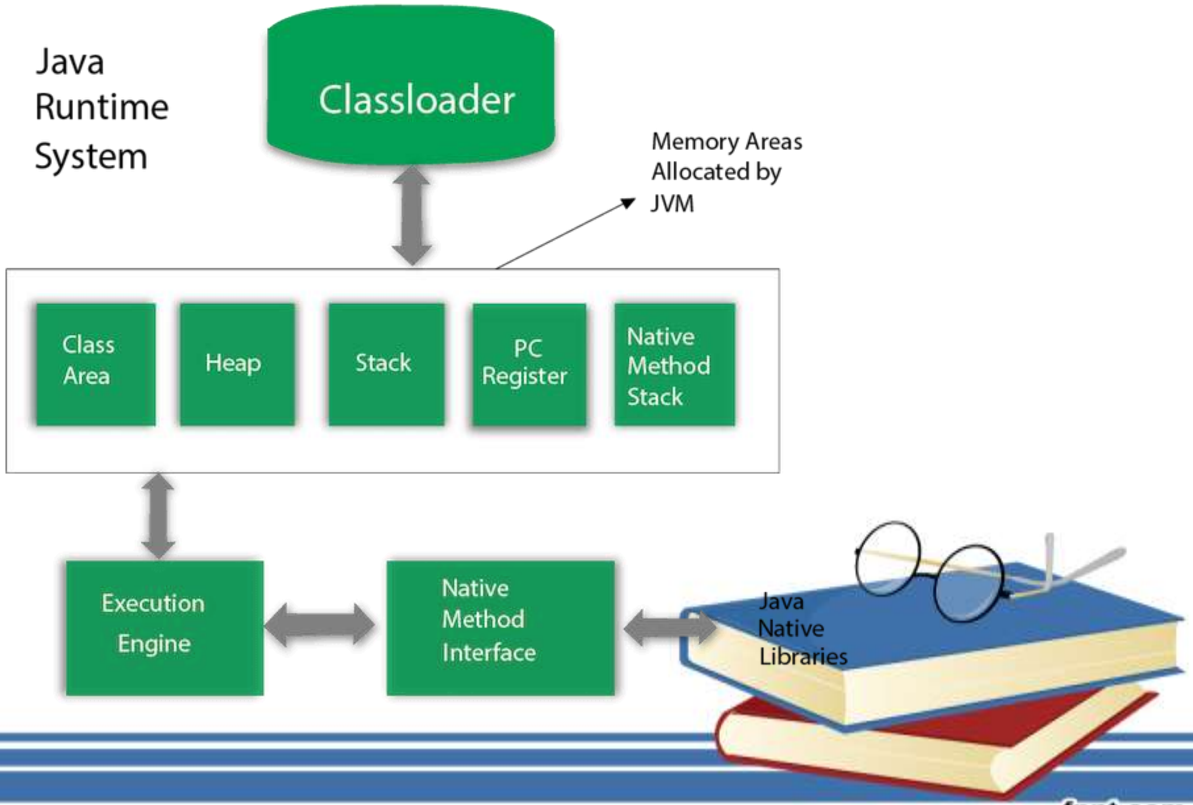
- Java is a dynamic language.
- It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
- Java supports dynamic compilation and automatic memory management (garbage collection).

JVM (Java Virtual Machine)

- JVM is an abstract machine. It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java bytecode can be executed.
- It can also run those programs which are written in other languages and compiled to Java bytecode.

- 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.

JVM Architecture



- 1) Classloader
    - Classloader is a subsystem of JVM which is used to load class files.
    - Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.
- 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 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:
  - **A virtual processor**
  - **Interpreter:** Read bytecode stream then execute the instructions.
  - **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.

- 8) Java Native Interface
    - Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Java Runtime Environment (JRE)

- It is also written as Java RTE.
- The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment. It is the implementation of JVM.
- It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK(Java Development Kit)

- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
    - Standard Edition Java Platform
    - Enterprise Edition Java Platform
    - Micro Edition Java Platform

- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

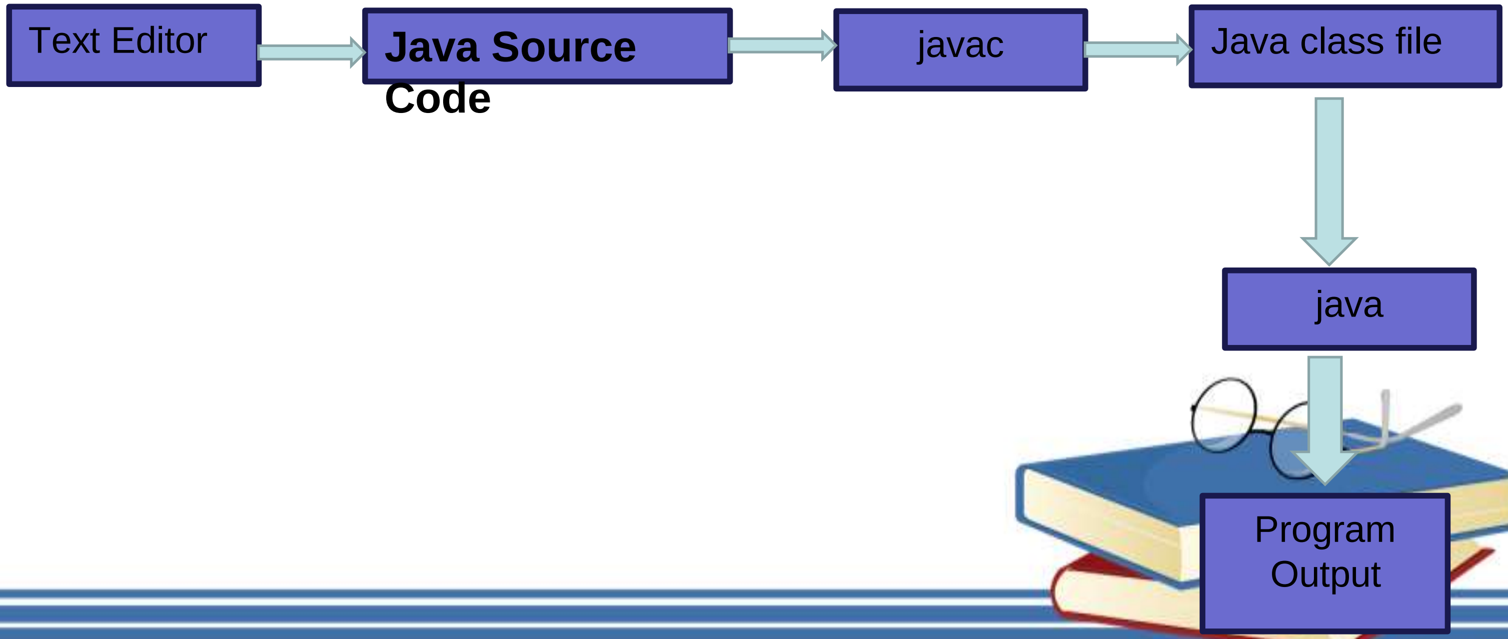
JDK Tools

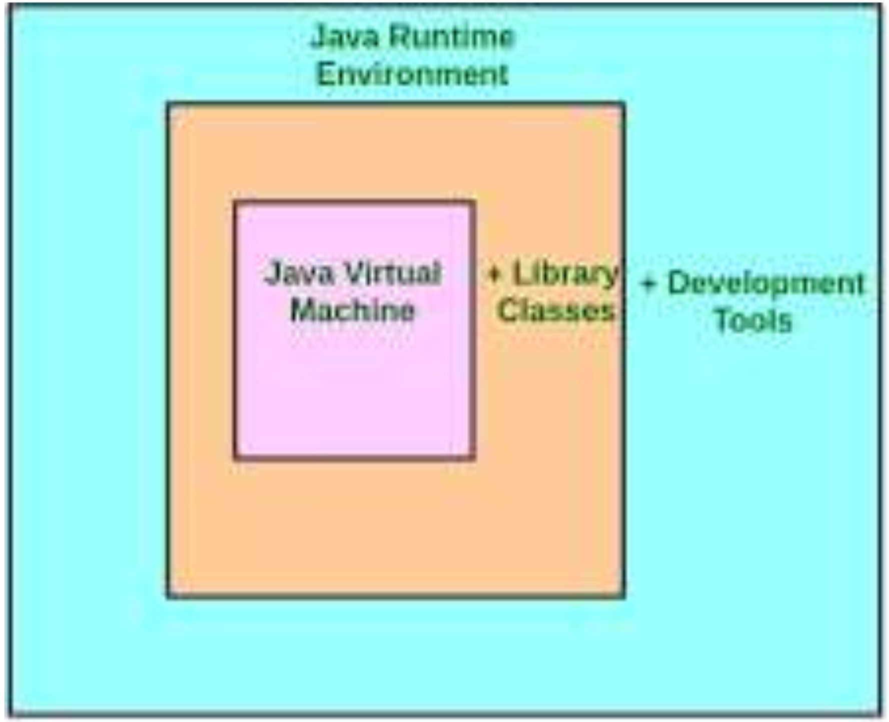
Tools	Description
javac	Java compiler,which translates java source code to byte code.
java	Java interpreter,which runs applet & application program by reading and interpreting bytecode files.
javad	Java debugger ,which helps to find errors in our program.
javadoc	Creates HTML format documentation from java source code files.
appletviewer	Enables to run java applets.

Application Program Interface (API)

- The java standard library (API) includes hundreds of classes and methods grouped into several functional packages.

Process of building and running java applicaton program





JDK = JRE + Development Tool
JRE = JVM + Library Classes

Setting Path

- 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, the path is not required to be set because all the tools will be available in the current directory.
- However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

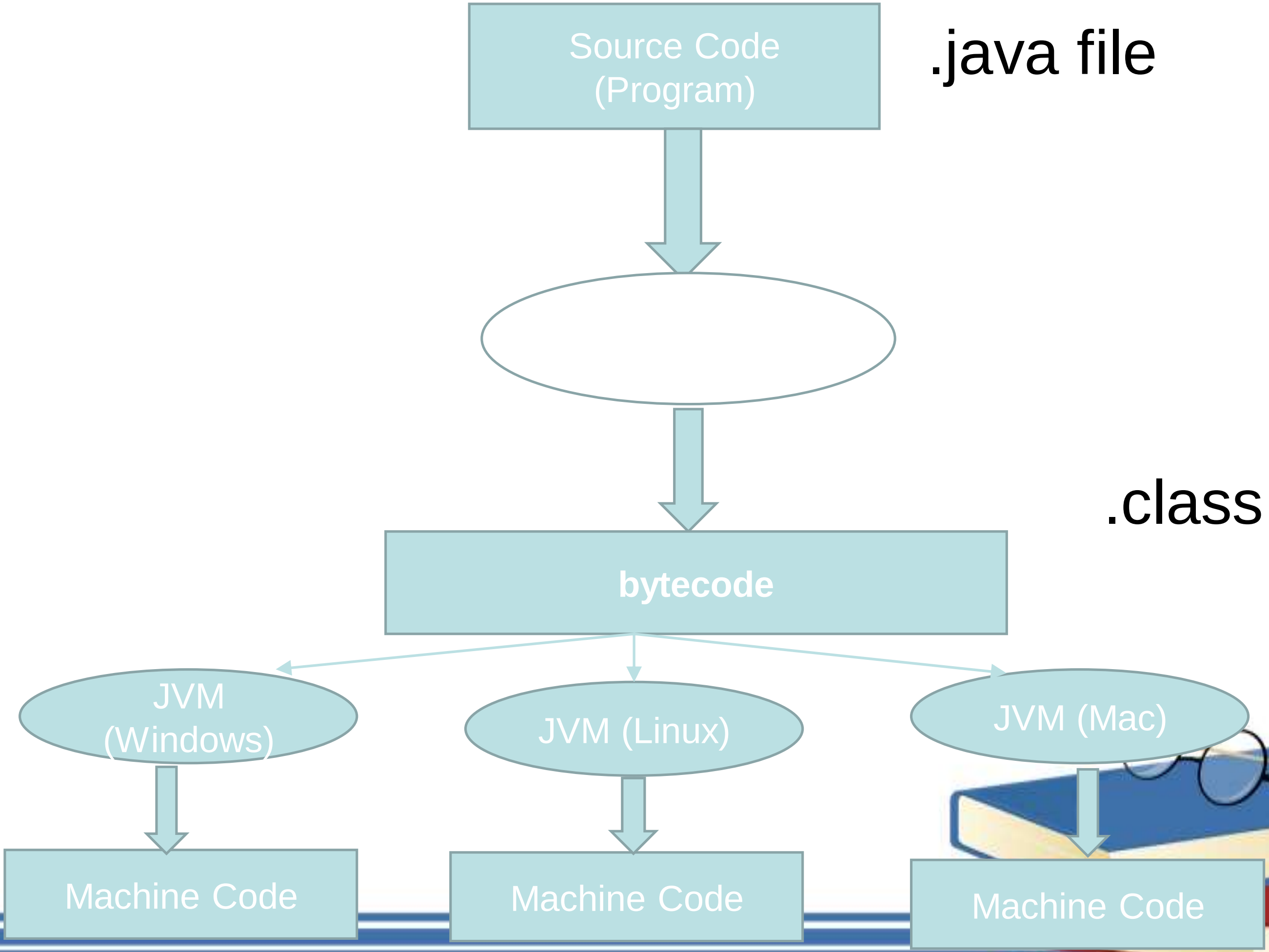
- There are two ways to set the path in Java:
  - Temporary
  - Permanent

- **1) Setting the Temporary Path of JDK in Windows**
- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path
- Eg: **set path=C:\Program Files\Java\jdk1.6.0_23\bin**

- **2) Setting Permanent Path of JDK in Windows**
- 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

ByteCode

- Java bytecode is the instruction set for the Java Virtual Machine.
- It acts similar to an assembler which is an alias representation of a C++ code.
- As soon as a java program is compiled, java bytecode is generated.
- With the help of java bytecode we achieve platform independence in java.



.java file

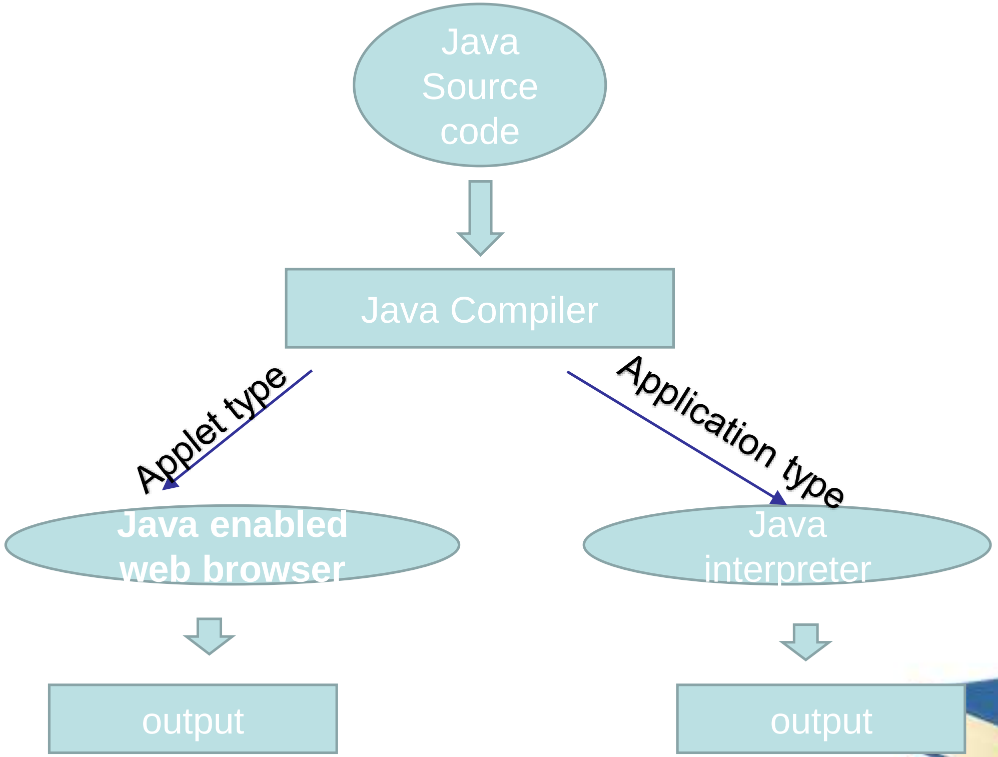
.class file

Java programming fundamentals

Overview of Java Program

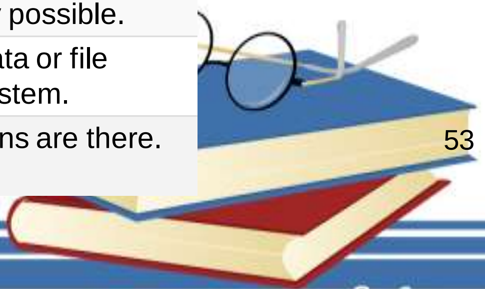
- Java is a general purpose ,object oriented programming language.
- We can develop two types of Java Programs.
  - Stand alone applications
  - Web applets

- ***Stand alone applications*** are programs written in java to carry out certain tasks on stand alone local computer.
- ***Applets*** are small java programs developed for internet applications.


Java Source code
Java Compiler
Applet type
Application type
Java enabled web browser
Java interpreter
output
output

Comparison of Applet & Application Program

BASIS FOR COMPARISON	APPLET	APPLICATION
Basic	It is small program uses another application program for its execution.	An application is the programs executed on the computer independently.
main() method	Do not use the main method	Uses the main method for execution
Execution	Cannot run independently require API's (Ex. Web API).	Can run alone but require JRE.
Installation	Prior installation is not needed	Requires prior explicit installation on the local computer.
Read and write operation	The files cannot be read and write on the local computer through applet.	Applications are capable of performing those operations to the files on the local computer.
Communication with other servers	Cannot communicate with other servers.	Communication with other servers is probably possible.
Restrictions	Applets cannot access files residing on the local computer.	Can access any data or file available on the system.
Security	Requires security for the system as they are untrusted.	No security concerns are there.



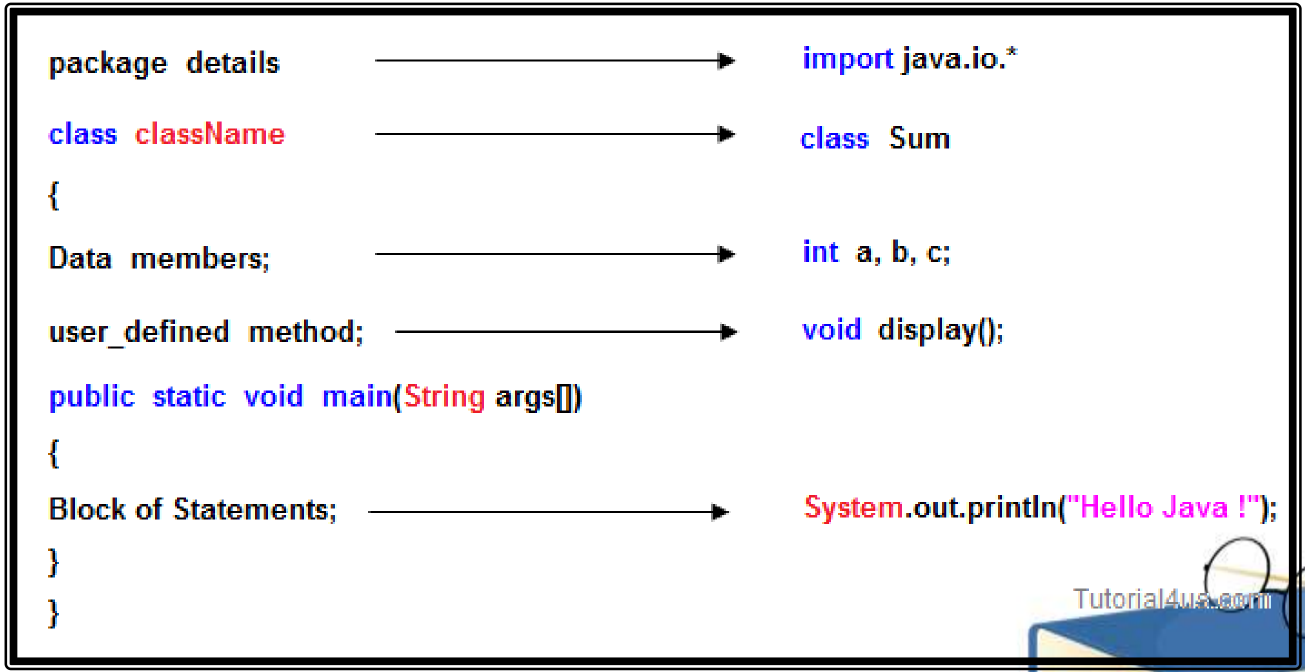
Java Program Structure

Documentation Section	suggested
Package Section	optional
Import Section	optional
Interface Section	optional
Class Definitions	optional

Main Method Class
{
Main method Definition
}
	essential

- **Documentation Section**
  - It comprises a set of comment lines giving the name of program,the author and the other details.
- **Package Section**
  - It is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages
- **Import Statements**
  - This statement instructs the interpreter to load the packages required for our program.

- **Interface Statements**
  - It is similar to class ,but contains only abstract methods.
- **Class Definitions**
  - A java program may contain multiple class definitions.Classes are the primary and essential elements of a java program.
- **Main Method Class**
  - Every java program requires a main() as it is the starting point of a program.



```
package  details          ────────────►   import java.io.*

class  className          ────────────►   class  Sum

{

Data  members;            ────────────►   int  a, b, c;

user_defined  method;     ────────────►   void  display();

public  static  void  main(String args[])

{

Block of Statements;      ────────────►   System.out.println("Hello Java !");

}

}
```

- A **package** is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.
- **Class** is keyword used for developing user defined data type and every java program must start with a concept of class.
- **"ClassName"** represent a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.
- **Data member** represents either instance or static they will be selected based on the name of the class.

- **User-defined** methods represents either instance or static they are meant for performing the operations either once or each and every time.
- Each and every java program starts execution from the main() method. And hence main() method is known as **program driver.**
- Since main() method of java is not returning any value and hence its return type must be **void.**
- Since main() method of java executes only once throughout the java program execution and hence its nature must be **static.**
- Since main() method must be accessed by every java programmer and hence whose access specifier must be **public.**
- Each and every main() method of java must take **array of objects of String**.
- **Block of statements** represents set of executable statements which are in term calling user-defined methods are containing business-logic.

Java Tokens

- The smallest individual unit of a program is called tokens.The compiler recogonises them for building up expressions and statements.
- Java language includes 5 tokens
  - Reserved words(keywords)
  - Identifiers
  - Literals
  - Operators
  - Seperators

Character set

- The smallest units of a java language are characters used to write java tokens.
- These characters are defined by a UNICODE character set.

- **Keywords**
- Keywords in Java are reserved words that represent predefined actions, internal processes etc. Because of this, keywords cannot be used as names of variables, functions, objects etc.
- **Identifiers**
- Identifiers are the name given to variables, classes, methods, etc.

Literals

- Any constant value which can be assigned to the variable is called as literal/constant.
- Java supports 5 main literals
  - Integer
  - Floating point
  - Character
  - String
  - Boolean

- **Operators**
  - operators are the Special Symbols those have specific functions associated with them for Performing the operations it needs some Operands
- **Separators**
  - These are Special Symbols used to Indicate the group of code that is either be divided or arrange into the Block The Separators includes Parentheses, Open Curly braces Comma, Semicolon, or either it will be period or dot Operator

CamelCase in java naming conventions

- Java follows camelcase syntax for naming the class, interface, method and variable.
  According to CamelCase if name is combined with two words, second word will start with uppercase letter always.
- Eg : actionPerformed()

Constants

- Constants in Java refers to fixed values that do not change during the execution of a program.
- Java supports several types of constants.

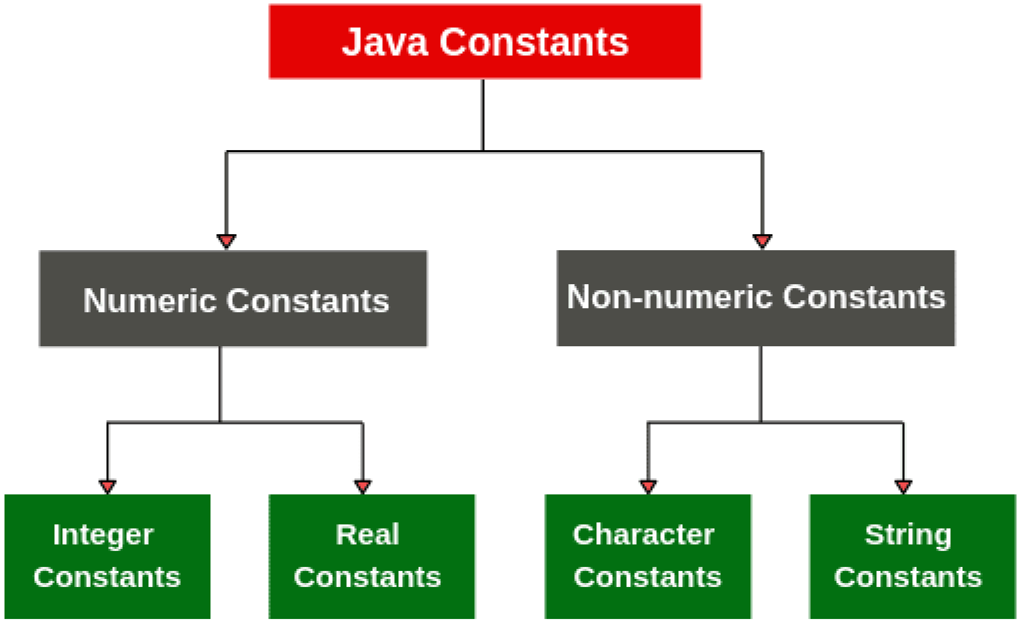
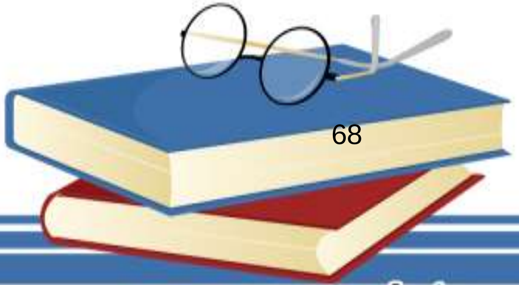


Fig: Classification of Java Constants

variables

- **Variable** is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program.

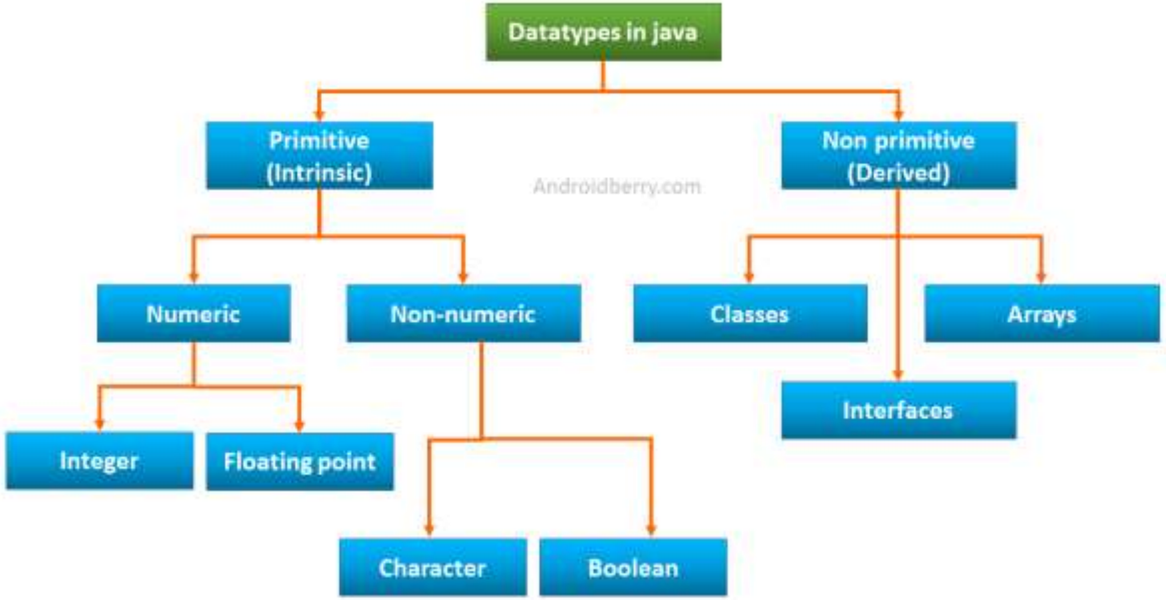


Rules to declare a Variable

- Every variable name should start with either alphabets or underscore (_) or dollar ($) symbol.
- No space are allowed in the variable declarations.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Variable name always should exist in the left hand side of assignment operators.
- Maximum length of variable is 64 characters.
- No keywords should access variable name.

Data Types

- Data types specify the different sizes and values that can be stored in the variable.



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

Declaration of variables

- It tells the compiler ,what the variable name is.
- It specifies what type of data the variable will hold.
- The place declaration decides the scope of variable.
  - Syntax : data_type variable1,variable2,…….,variablen;
  - Example : int a,b,c;

                float x;

Initialisation of variables

- Variable can be initialised in two ways.
  1. Using assignment operator (Compile time)
  2. Using read statement (Runtime)

Scope of variables

There are three types of variables in Java.

- local variable
- instance variable
- static variable

Local Variable

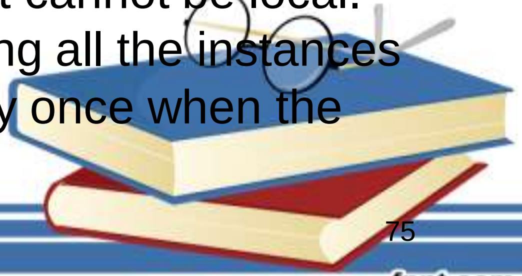
- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

Instance Variable

- A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
- It is called instance variable because its value is instance specific and is not shared among instances.

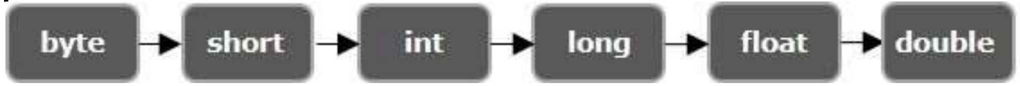
Static (class) variable

- A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

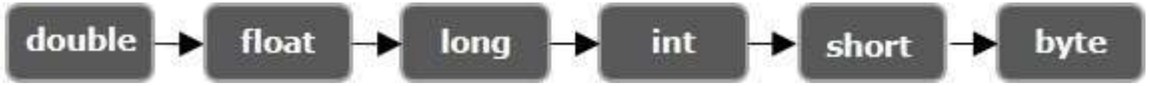


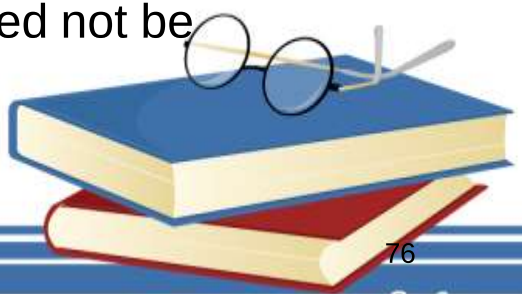
Type Casting/type conversion

- Converting one primitive datatype into another is known as type casting (type conversion) in Java. You can cast the primitive datatypes in two ways namely,
- **Widening** − Converting a lower data type to a higher datatype is known as widening. In this case the casting/conversion is done automatically therefore, it is known as implicit type casting. In this case both datatypes should be compatible with each other.



- **Narrowing** − Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator “( )” explicitly. Therefore, it is known as explicit type casting. In this case both datatypes need not be compatible with each other.





Standard default values

- In java , every variable has a default value.
- If we don’t initialise a variable when it is first created ,java provides ,default value to that variable type automatically.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	‘u0000’
String (or any object)	null
boolean	false

```
class Demo
{
        static boolean val1;
        static double val2;
        static float val3;
        static int val4;
        static long val5;
        static String val6;
                public static void main(String[] args)
                        {
                                System.out.println("Default values.....");
                                System.out.println("Val1 = " + val1);
                                System.out.println("Val2 = " + val2);
                                System.out.println("Val3 = " + val3);
                                 System.out.println("Val4 = " + val4);
                                System.out.println("Val5 = " + val5);
                                System.out.println("Val6 = " + val6);
                        }
}
```

Output

```
Default values.....
Val1 = false
Val2 = 0.0
Val3 = 0.0
Val4 = 0
Val5 = 0
Val6 = null
```

Operators

Decision Making & Branching Statements

- **Decision making statement** statements is also called selection statement.
- That is depending on the condition block need to be executed or not which is decided by condition.
- If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

- In java there are three types of decision making statement.

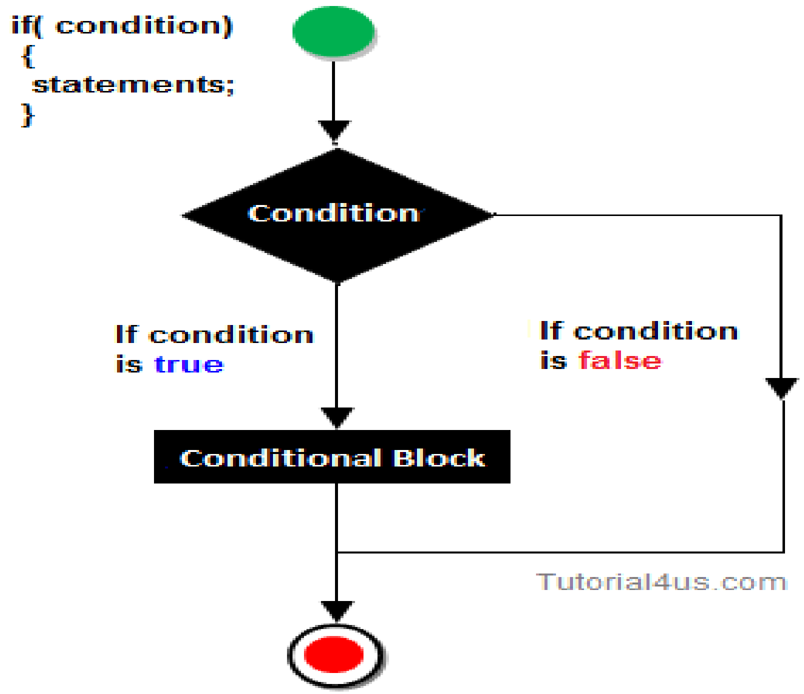
- ➢ if
- ➢ if-else
- ➢ nested if statement
- ➢ switch

if Statement

- It is one of the simplest decision-making statement which is used to decide whether a block of JavaScript code will execute if a certain condition is true.
- **Syntax**

```
if (condition) {
  // block of code will execute if the condition is true
}
```

Flowchart

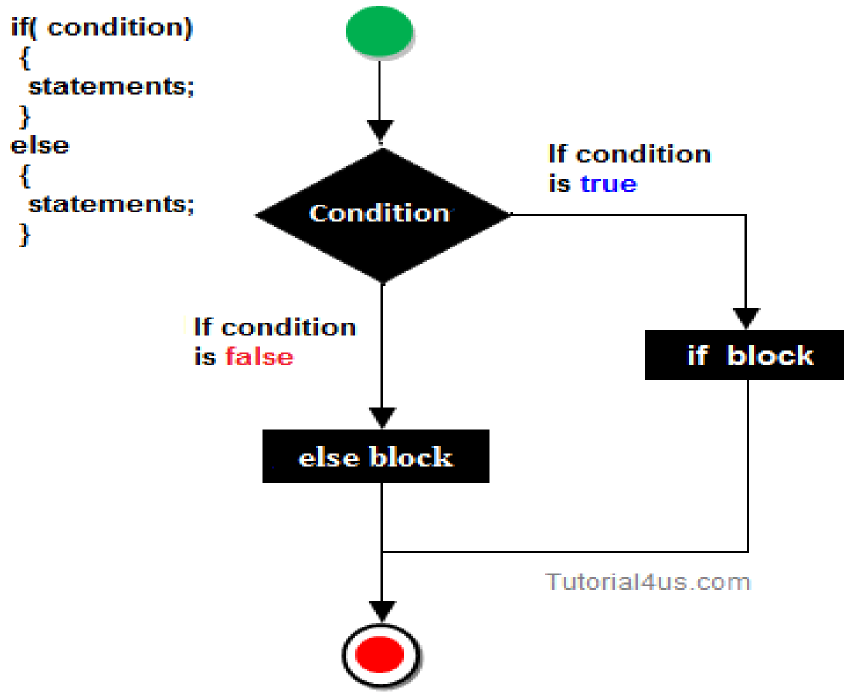

if(condition)
{
statements;
}
Condition
If condition is true
If condition is false
Conditional Block
Tutorial4us.com

if….else statement

- The Java if-else statement also tests the condition.
- It executes the *if block* if condition is true otherwise *else block* is executed.
- **Syntax**

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Flowchart



```
if( condition)
 {
  statements;
 }
else
 {
  statements;
 }
```

Flowchart

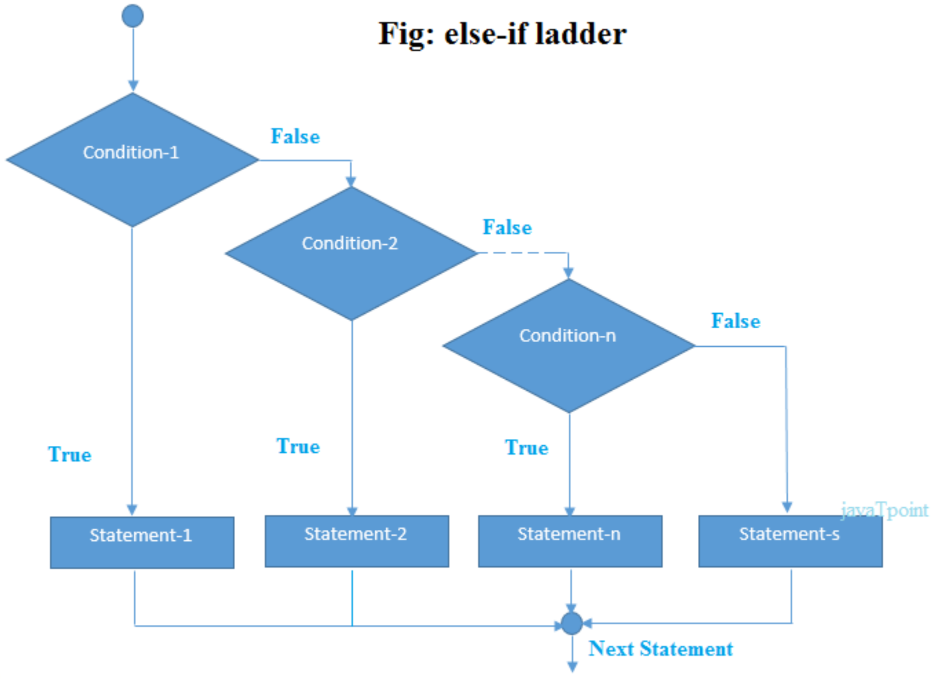


Fig: else-if ladder
Condition-1
False
Condition-2
False
Condition-n
False
True
True
True
Statement-1
Statement-2
Statement-n
Statement-s
javaTpoint
Next Statement

if-else-if ladder Statement

- The if-else-if ladder statement executes one condition from multiple statements.
- **Syntax:**

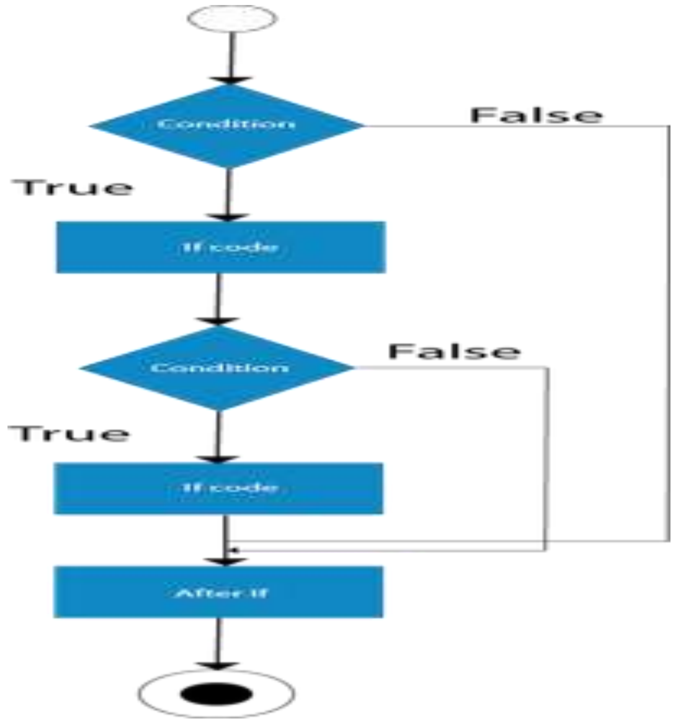
```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Nested if statement

- The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.
- **Syntax:**

```
if(condition){
     //code to be executed
          if(condition){
             //code to be executed
    }
}
```

Flowchart


Condition
False
True
If code
Condition
False
True
If code
After If

Switch Statement

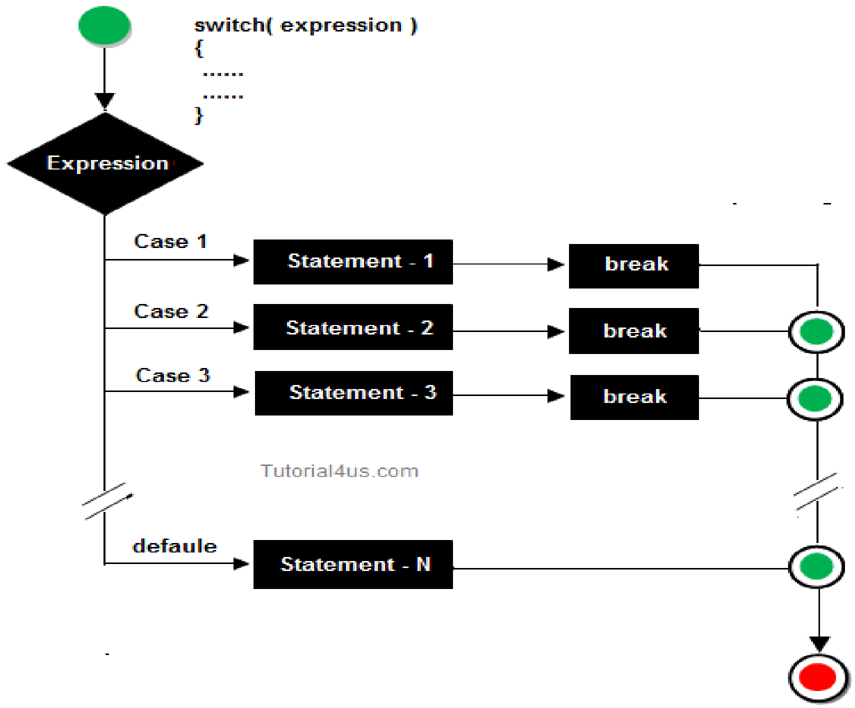
- The **switch** statement in java language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement.
- A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.

- **Rules for apply switch statement**
- With switch statement use only byte, short, int, char data type (float data type is not allowed). You can use any number of case statements within a switch. Value for a case must be same as the variable in switch.
- **Limitations of switch statement**
- Logical operators cannot be used with switch statement. For instance

- Syntax

```
switch(expression/variable)
 {
 case value: //statements // any number of
case statements
break;
//optional
default: //optional //statements
}
```

Flowchart


switch(expression)
{
……
……
}
Expression
Case 1
Statement - 1
break
Case 2
Statement - 2
break
Case 3
Statement - 3
break
Tutorial4us.com
defaule
Statement - N

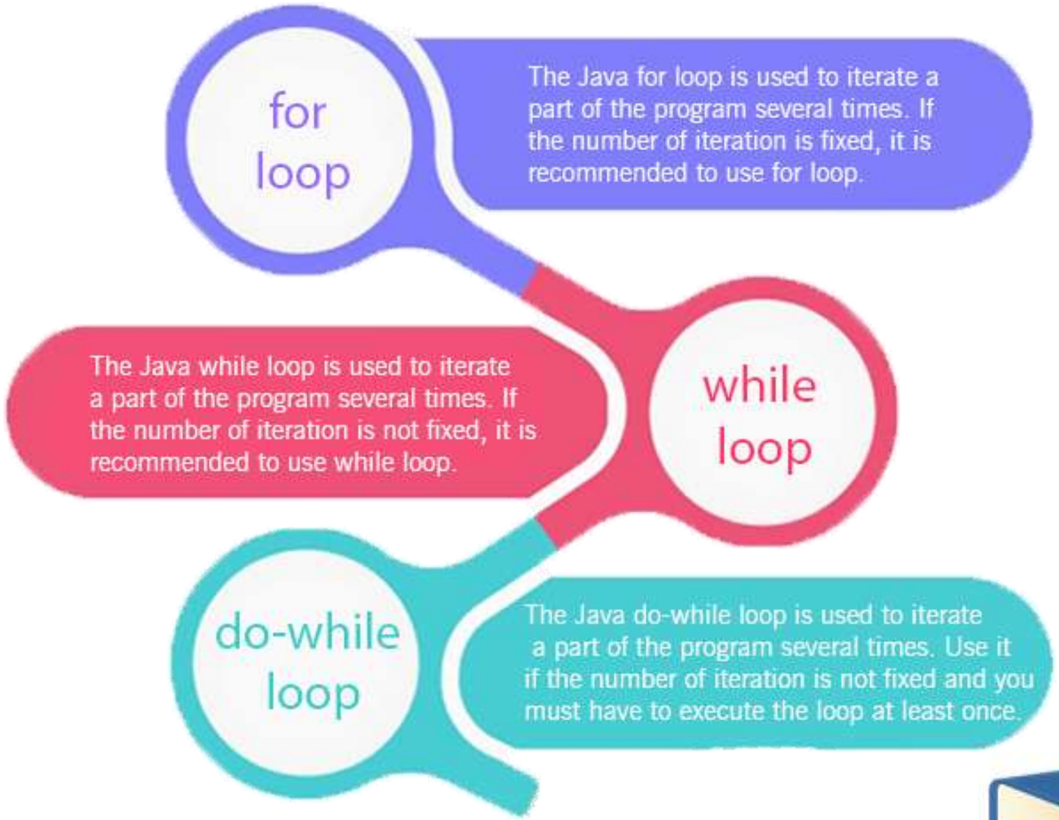
Loops

- loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
- There are three types of loops in Java.

❑for loop

❑while loop

❑do-while loop



for loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

while loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

do-while loop

The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

For Loop

- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
- There are three types of for loops in java.

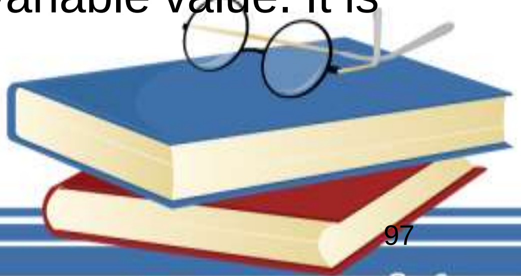
❑Simple For Loop

❑For-each or Enhanced For Loop

❑Labeled For Loop

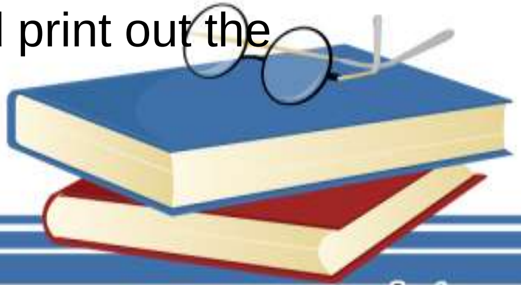
Simple For Loop

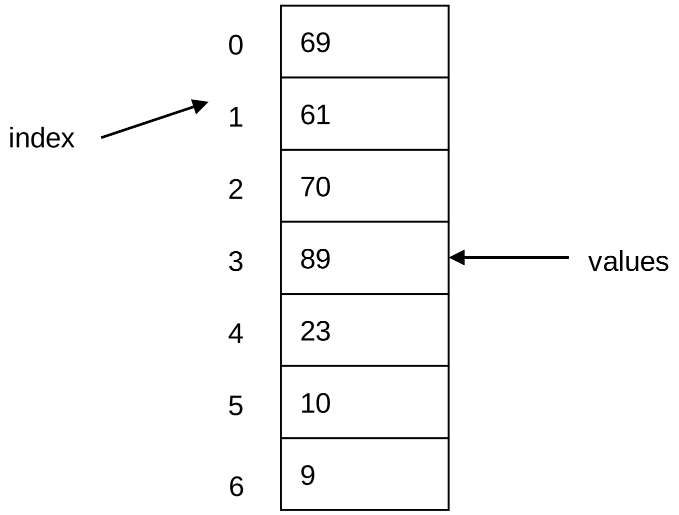
- It consists of four parts:
- **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
- **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
- **Statement**: The statement of the loop is executed each time until the second condition is false.
- **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.



Arrays

- An array is a group of contiguous or related data items that share a common name.
- Used when programs have to handle large amount of data
- Each value is stored at a specific position
- Position is called a index or superscript. Base index = 0
- The ability to use a single name to represent a collection of items and refer to an item by specifying the item number enables us to develop concise and efficient programs. For example, a loop with index as the control variable can be used to read the entire array, perform calculations, and print out the results.




index
0
1
2
3
4
5
6
69
61
70
89
23
10
9
values

Declaration of Arrays

- Like any other variables, arrays must declared and created before they can be used. Creation of arrays involve three steps:
  - Declare the array
  - Create storage area in primary memory.
  - Put values into the array (i.e., Memory location)
- Declaration of Arrays:
  - Form 1:

    Type arrayname[]
  - Form 2:
    - Type [] arrayname;

  - Examples:

    ```
    int[] students;
    int students[];
    ```
  - Note: we don't specify the size of arrays in the declaration.

Creation of Arrays

- After declaring arrays, we need to allocate memory for storage array items.
- In Java, this is carried out by using “new” operator, as follows:
  - Arrayname = **new** type[size];
- Examples:
  - students = new int[7];

Initialisation of Arrays

- Once arrays are created, they need to be initialised with some values before access their content. A general form of initialisation is:
  – Arrayname [index/subscript] = value;
- Example:
    - students[0] = 50;
    - students[1] = 40;
- Like C, Java creates arrays starting with subscript 0 and ends with value one less than the size specified.
- Unlike C, Java protects arrays from overruns and under runs. Trying to access an array beyond its boundaries will generate an error message.

Arrays – Length

- Arrays are fixed length
- Length is specified at create time
- In java, all arrays store the allocated size in a variable named “length”.
- We can access the length of arrays as arrayName.length:

```
e.g.  int x = students.length;     //  x = 7
```

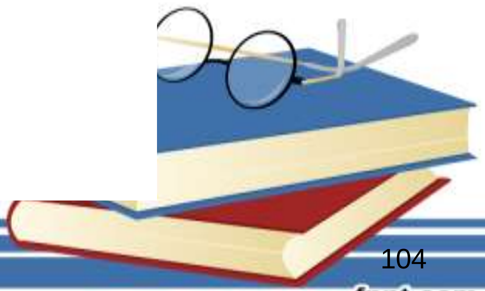
- Accessed using the index

```
e.g.   int x = students [1];            //  x = 40
```

Arrays – Example

```
public class StudentArray{
        public static void main(String[] args) {
                int[] students;
                students = new int[7];
                System.out.println("Array Length = " + students.length);

                for ( int  i=0;  i < students.length;  i++)
                        students[i] = 2*i;
                System.out.println("Values Stored in Array:");
                for ( int  i=0;  i < students.length;  i++)
                        System.out.println(students[i]);
        }
}
```

:w



Arrays – Initializing at Declaration

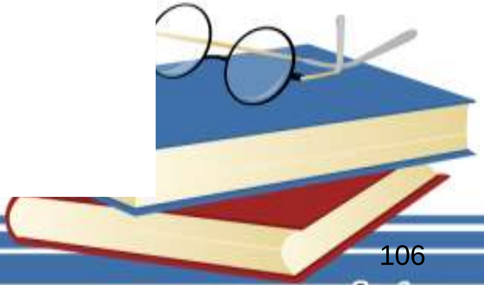
- Arrays can also be initialised like standard variables at the time of their declaration.
  - – Type arrayname[] = {list of values};
- Example:

```
        int[] students = {55, 69, 70, 30, 80};
```

- Creates and initializes the array of integers of length 5.
- In this case it is not necessary to use the *new* operator.

Arrays – Example

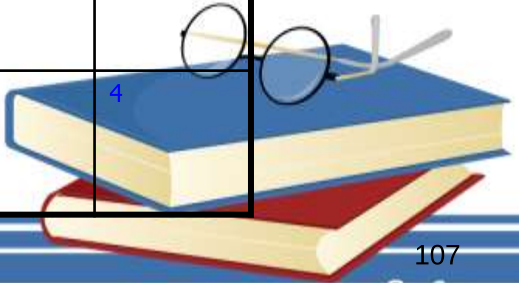
```
public class StudentArray{
        public static void main(String[] args) {
        int[] students = {55, 69, 70, 30, 80};

        System.out.println("Array Length = " + students.length);
        System.out.println("Values Stored in Array:");
         for ( int  i=0;  i < students.length;  i++)
                System.out.println(students[i]);
        }
}
```



Two Dimensional Arrays

- Two dimensional arrays allows us to store data that are recorded in table. For example:
- Table contains 12 items, we can think of this as a matrix consisting of 4 rows and 3 columns.

Sold / Person	Item1	Item2	Item3
Salesgirl #1	10	15	30
Salesgirl #2	14	30	33
Salesgirl #3	200	32	1
Salesgirl #4	10	200	4

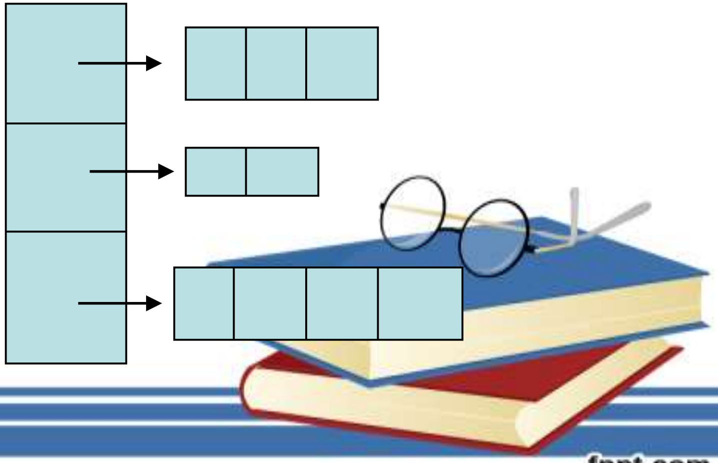


2D arrays manipulations

- Declaration:
  - int myArray [][];
- Creation:
  - myArray = new int[4][3]; // OR
  - int myArray [][] = new int[4][3];
- Initialisation:
  - Single Value;
    - myArray[0][0] = 10;
  - Multiple values:
    - int tableA[2][3] = {{10, 15, 30}, {14, 30, 33}};
    - int tableA[][] = {{10, 15, 30}, {14, 30, 33}};

Variable Size Arrays

- Java treats multidimensional arrays as “arrays of arrays”. It is possible to declare a 2D arrays as follows:
  - int a[][] = new int [3][];
  - a[0]= new int [3];
  - a[1]= new int [2];
  - a[2]= new int [4];



Arrays of Objects

- Arrays can be used to store objects

```
Circle[] circleArray;
circleArray = new Circle[25];
```

- The above statement creates an array that can store references to 25 Circle objects.
- Circle objects are not created.

Arrays of Objects

- Create the Circle objects and stores them in the array.
  - //declare an array for Circle

```
Circle circleArray[] = new Circle[25];
int r = 0;
// create circle objects and store in array
for (r=0;  r <25; r++)
    circleArray[r] = new Circle(r);
```

Strings

- String manipulation is the most common operation performed in Java programs. The easiest way to represent a String (a sequence of characters) is by using an array of characters.
  - Example:
  - char place[] = new char[4];
  - place[0] = ‘J’;
  - place[1] = ‘a’;
  - place[2] = ‘v’;
  - place[3] = ‘a’;
- Although character arrays have the advantage of being able to query their length, they themselves are too primitive and don’t support a range of common string operations. For example, copying a string, searching for specific pattern etc.
- Recognising the importance and common usage of String manipulation in large software projects, Java supports String as one of the fundamental data type at the language level. Strings related book keeping operations (e.g., end of string) are handled automatically.

String Operations in Java

- Following are some useful classes that Java provides for String operations.
  - String Class
  - StringBuffer Class
  - StringTokenizer Class

String Class

- String class provides many operations for manipulating strings.
  - Constructors
  - Utility
  - Comparisons
  - Conversions
- String objects are read-only (immutable)

Strings Basics

- Declaration and Creation:
  - **String** stringName;
  - stringName = **new String** (“string value”);
  - Example:
    - String city;
    - city = new String (“Bangalore”);
  - Length of string can be accessed by invoking length() method defined in String class:
    - int len = city.length();

String operations and Arrays

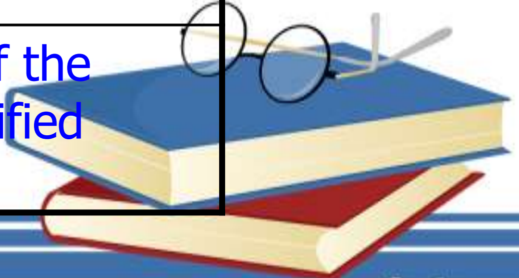
- Java Strings can be concatenated using the + operator.
  - String city = “New” + “York”;
  - String city1 = “Delhi”;
  - String city2 = “New “+city1;
- Strings Arrays
  - String city[] = new String[5];
  - city[0] = new String(“Melbourne”);
  - city[1] = new String(“Sydney”);
  - …
  - String megacities[] = {“Brisbane”, “Sydney”, “Melbourne”, “Adelaide”, “Perth”};

String class - Constructors

public String()	Constructs an empty String.
Public String(String value)	Constructs a new string copying the specified string.

String – Some useful operations

public int length()	Returns the length of the string.
public charAt(int index)	Returns the character at the specified location (*index)*
public int compareTo(String anotherString) public int compareToIgnoreCase(String anotherString)	Compare the Strings.
reigonMatch(int start, String other, int ostart, int count)	Compares a region of the Strings with the specified start.

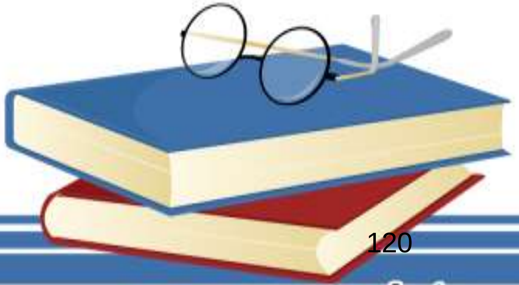


String – Some useful operations

public String replace(char oldChar, char newChar)	Returns a new string with all instances of the *oldChar* replaced with *newChar.*
public trim()	Trims leading and trailing white spaces.
public String toLowerCase() public String toUpperCase()	Changes as specified.

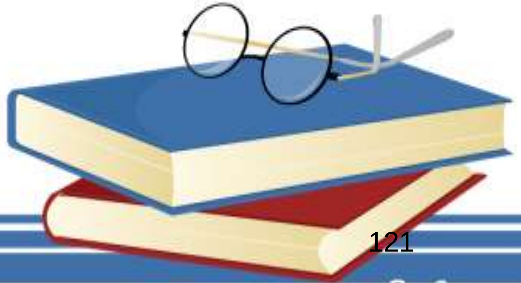
StringBuffer class

- StringBuffer class is used to create a **mutable** string object. It means, it can be changed after it is created.
- It represents growable and writable character sequence.
- It is similar to String class in Java both are used to create string, but StringBuffer object can be changed.



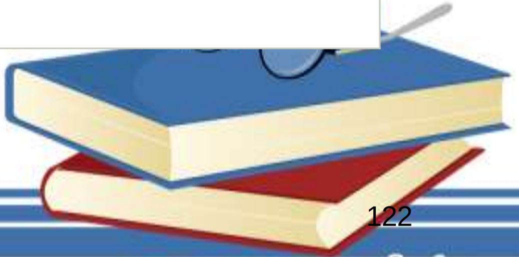
fppt.com

- StringBuffer defines 4 constructors.
- **StringBuffer**(): It creates an empty string buffer and reserves space for 16 characters.
- **StringBuffer**(int size): It creates an empty string and takes an integer argument to set capacity of the buffer.
- **StringBuffer**(String str): It creates a stringbuffer object from the specified string.
- **StringBuffer**(charSequence []ch): It creates a stringbuffer object from the charsequence array.



Difference between String and StringBuffer

String	**StringBuffer**
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

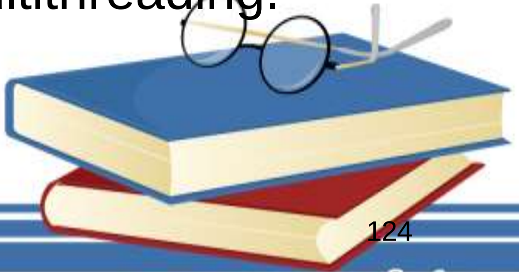


Wrapper classes

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
- The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.
- A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
- The classes in java.util package handles only objects and hence wrapper classes help in this case also.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
- An object is needed to support synchronization in multithreading.

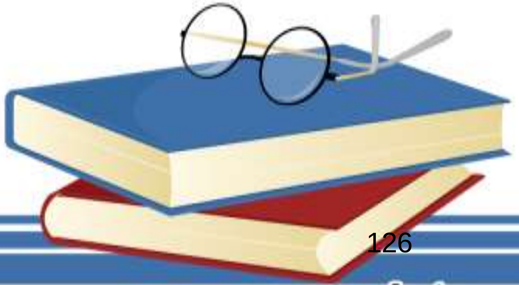


- **Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.
- Eg:

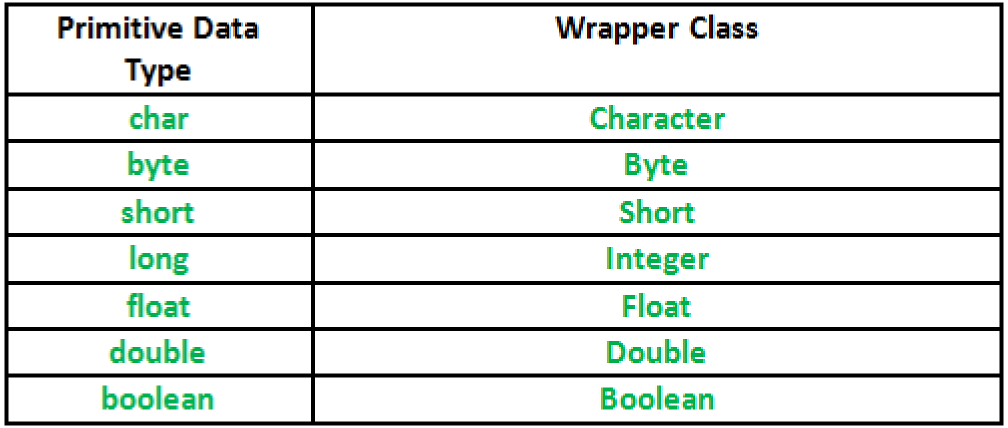
```
import java.util.ArrayList;
class Autoboxing
{
        public static void main(String[] args)
        {
                char ch = 'a';
                Character a = ch;
                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                arrayList.add(25);
                System.out.println(arrayList.get(0));
        }
}
```

- **Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.
- Eg :

```
import java.util.ArrayList;
class Unboxing
{
        public static void main(String[] args)
        {
                Character ch = 'a';
                char a = ch;
                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                arrayList.add(24);
                int num = arrayList.get(0);
                System.out.println(num);
        }
}
```



Primitive Data types and their Corresponding Wrapper class



Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean