J a v a

JAVA-INTRODUCTION

- Java is a high-level programming language
- developed by Sun Microsystems and initiated by James Gosling and released in 1995
- Java runs onWindows, Mac OS, and the various versions of UNIX
- Java is used to develop mobile apps, web apps, desktop apps, games and much more
- WRITE ONCE RUN ANYWHERE

What is Java

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

Java is a high level, robust, secured and object-oriented programming language.

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

Types of Java Applications

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

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.

History of Java

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

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

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

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

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

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

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

Meaning of JAVA

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

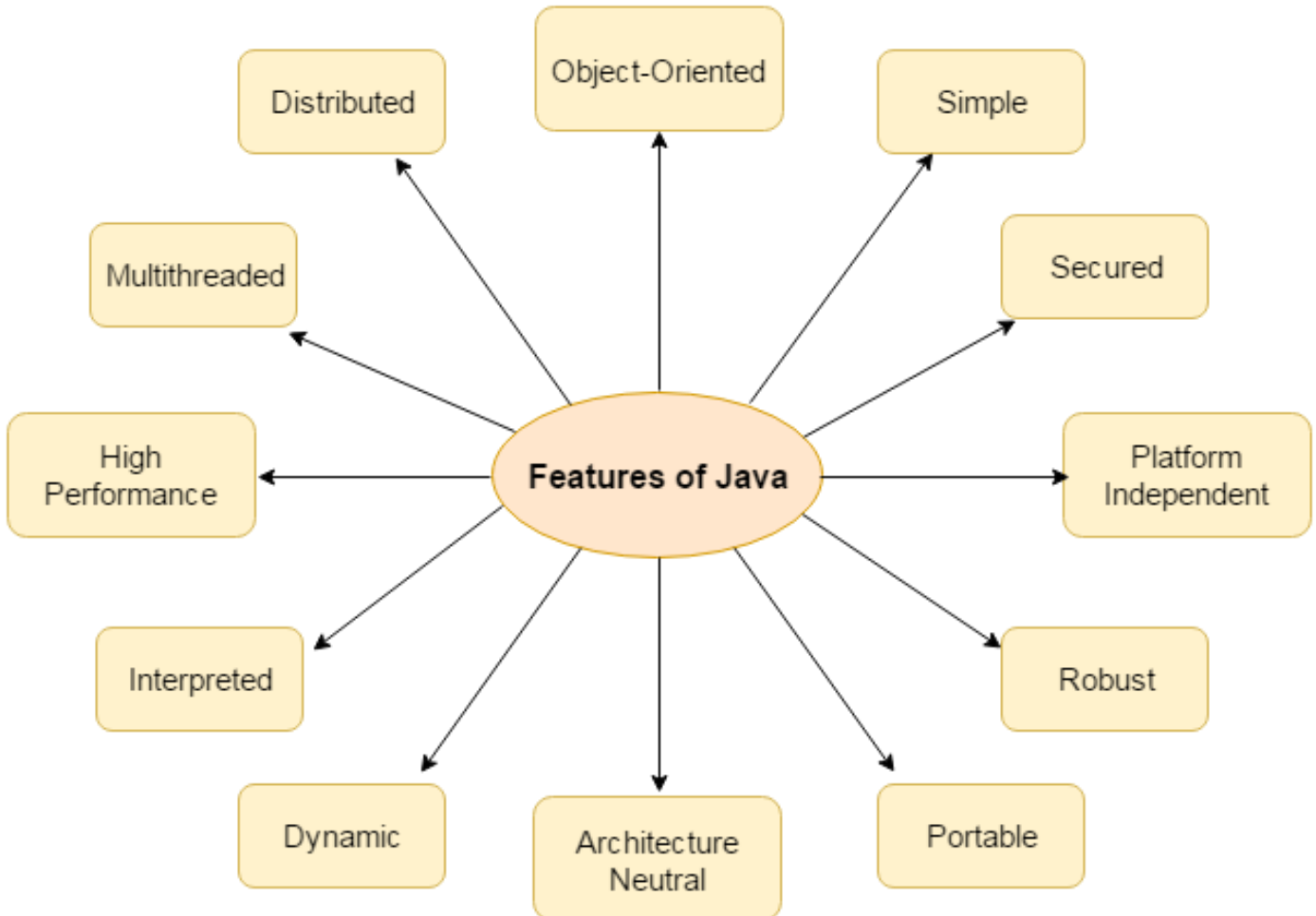
Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

JDK 1.0 released in(January 23, 1996).

here are many java versions that has been released.

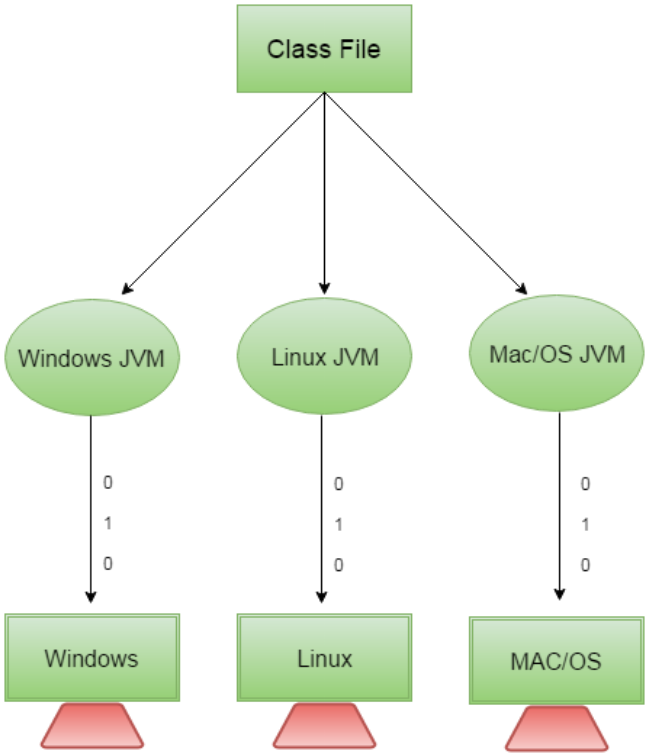
1.JDK Alpha and Beta (1995)
2.JDK 1.0 (23rd Jan, 1996)
3.JDK 1.1 (19th Feb, 1997)
4.J2SE 1.2 (8th Dec, 1998)
5.J2SE 1.3 (8th May, 2000)
6.J2SE 1.4 (6th Feb, 2002)
7.J2SE 5.0 (30th Sep, 2004)
8.Java SE 6 (11th Dec, 2006)
9.Java SE 7 (28th July, 2011)
10.Java SE 8 (18th March, 2014)

You can download java for free from oracle.com


Features of Java
Object-Oriented
Simple
Secured
Platform Independent
Robust
Portable
Architecture Neutral
Dynamic
Interpreted
High Performance
Multithreaded
Distributed

Platform Independent

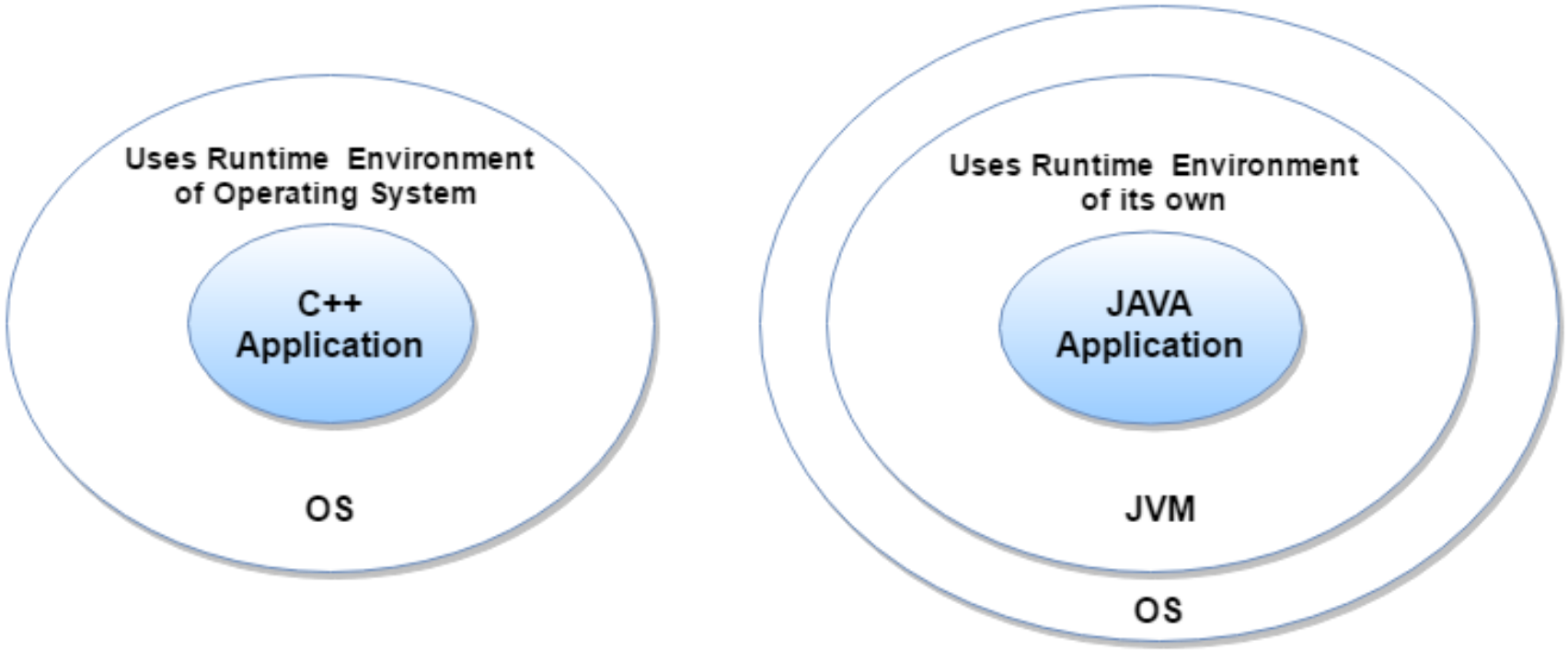
A platform is the hardware or software environment in which a program runs.
There are two types of platforms software-based and hardware-based. Java provides software-based platform.
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has tw 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**
- **Java Programs run inside virtual machine sandbox**



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 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. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

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

Simple Program of Java

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

Requirement for Hello Java Example

- For executing any java program, you need to install the JDK .
- set path of the jdk/bin directory.
- create the java program
- compile and run the java program

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

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

Understanding first java program

•**class** keyword is used to declare a class in java.

•**public** keyword is an access modifier which represents visibility, it means it is visible to all.

•**static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

•**void** is the return type of the method, it means it doesn't return any value.
•**main** represents startup of the program.

•**String[] args** is used for command line argument. We will learn it later.

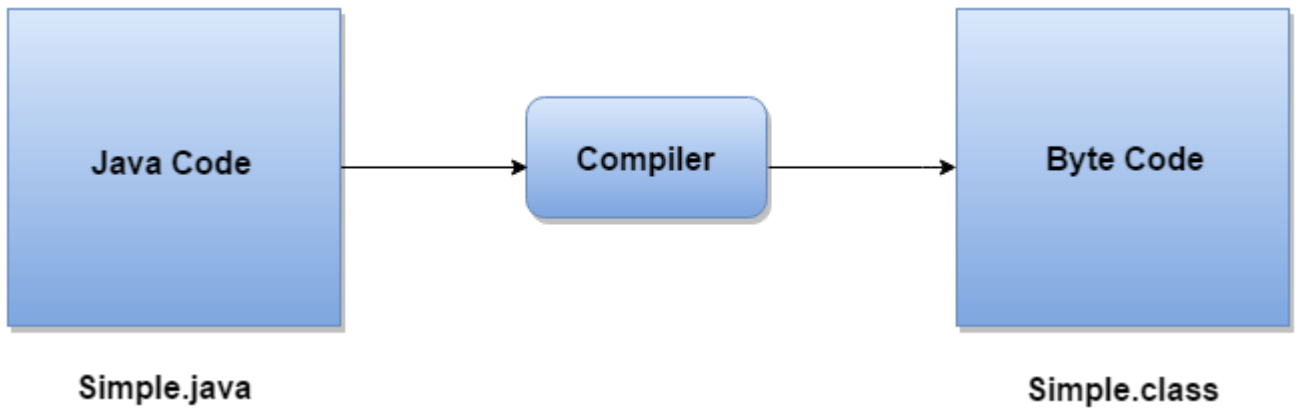
•**System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

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

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

What happens at compile time?

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



JVM

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

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

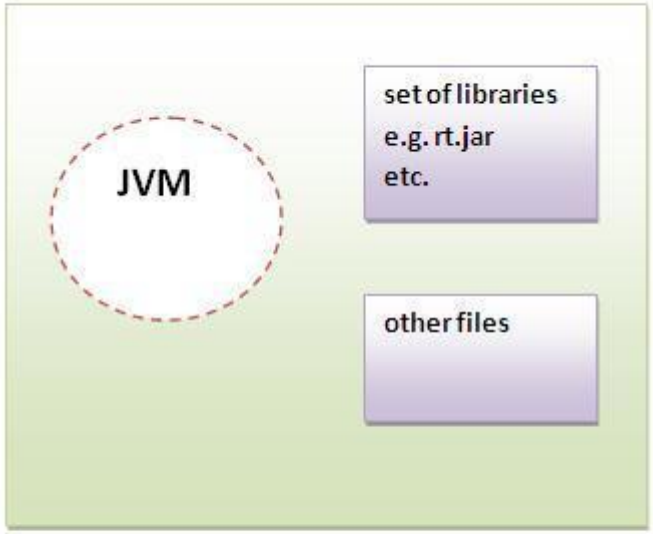
The JVM performs following main tasks:

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

JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

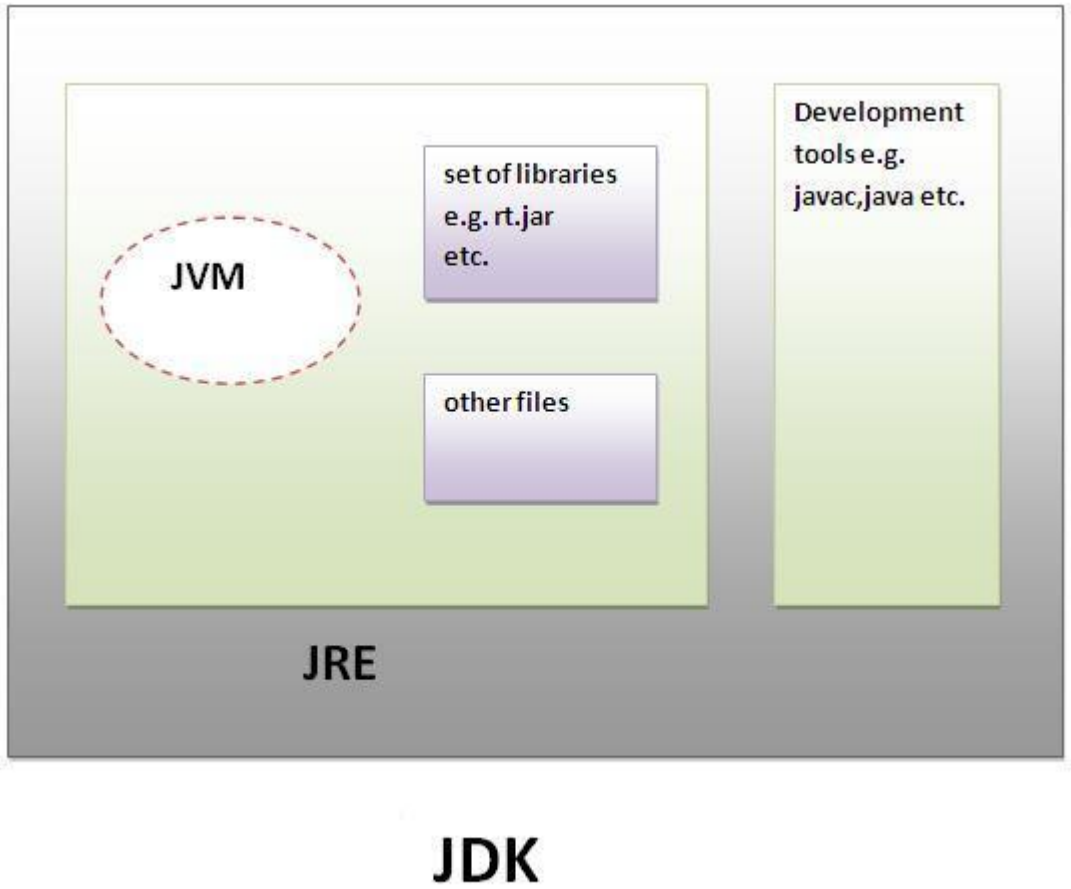
Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



JRE

JDK

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

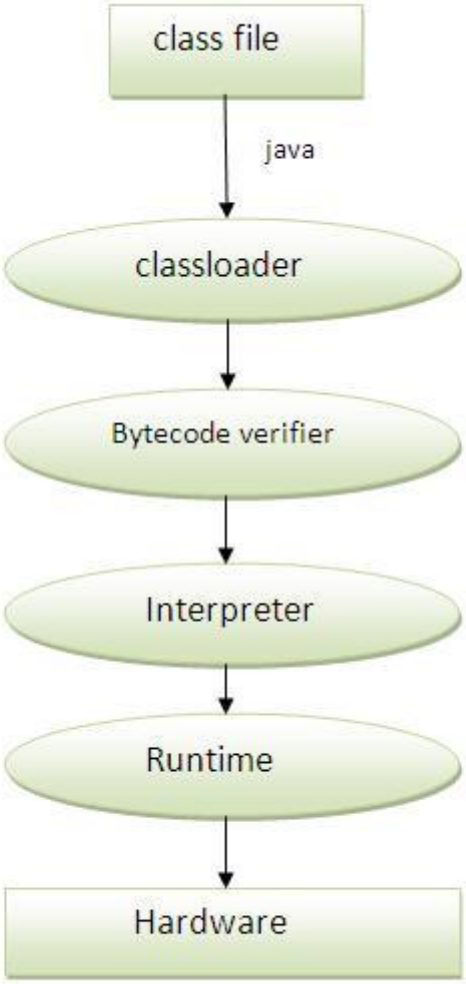


What happens at runtime?

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.



Unicode System

The Unicode Standard provides a unique number for every character, no matter what platform, device, application or language. It has been adopted by all modern software providers and now allows data to be transported through many different platforms, devices and applications without corruption.

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for chinese, and so on.

What is a token

A token is the smallest individual unit of a program that is meaningful to the compiler.

The tokens in Java are:

Identifiers

Literals

Operators

Separators

Keywords

What are keywords

Keywords are reserved words that convey a special meaning to the compiler. They cannot be used as identifiers.

Example:

class, void, int, private, static, etc.

What are identifiers?

Identifiers are names of variables, classes, packages, methods, and packages in a java program.

Example:

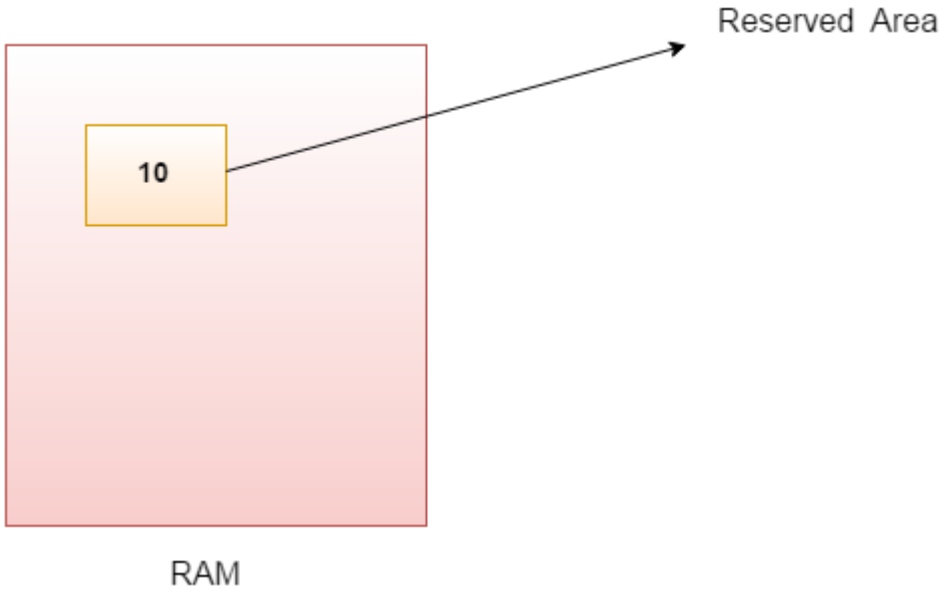
Sample,Simple,Hello_world,x,first_name etc;

Variables and Data Types in Java

Variable is a name of memory location

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

```
int data=10;
```

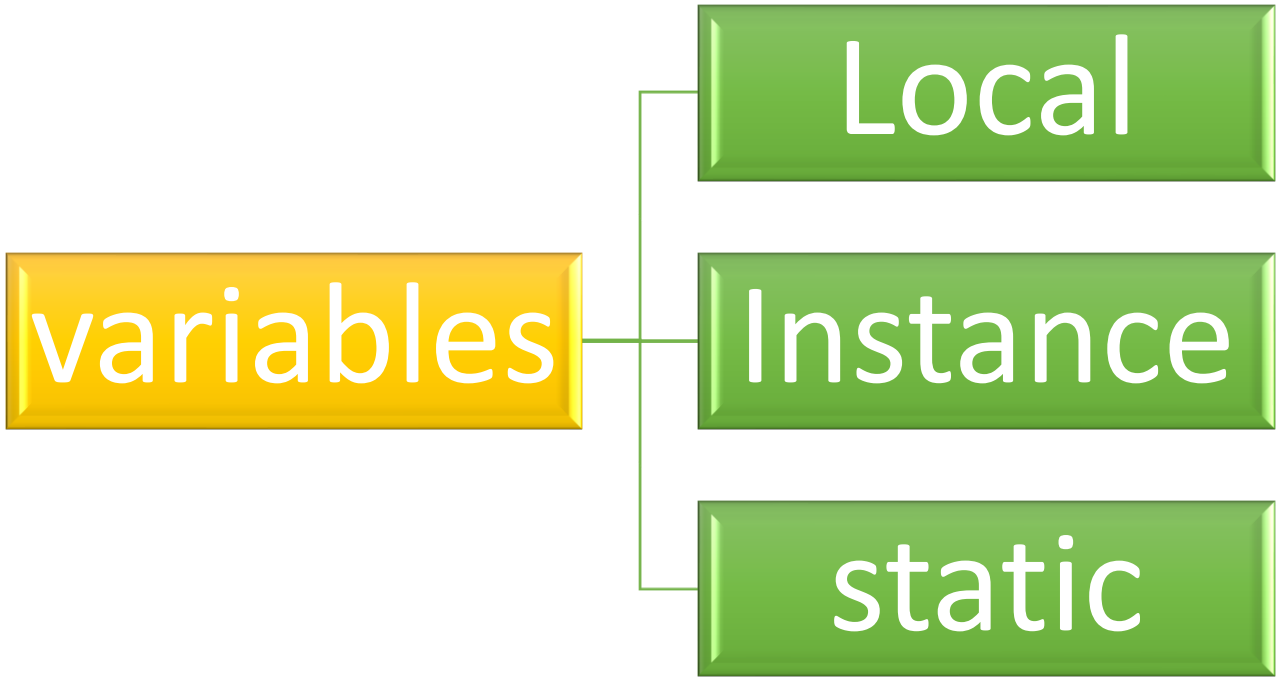


Types of Variable

1) Local Variable : A variable which is declared inside the method is called local variable.

2) Instance Variable : A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) Static variable : A variable that is declared as static is called static variable. It cannot be local.
We will have detailed learning of these variables in next chapters.



Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

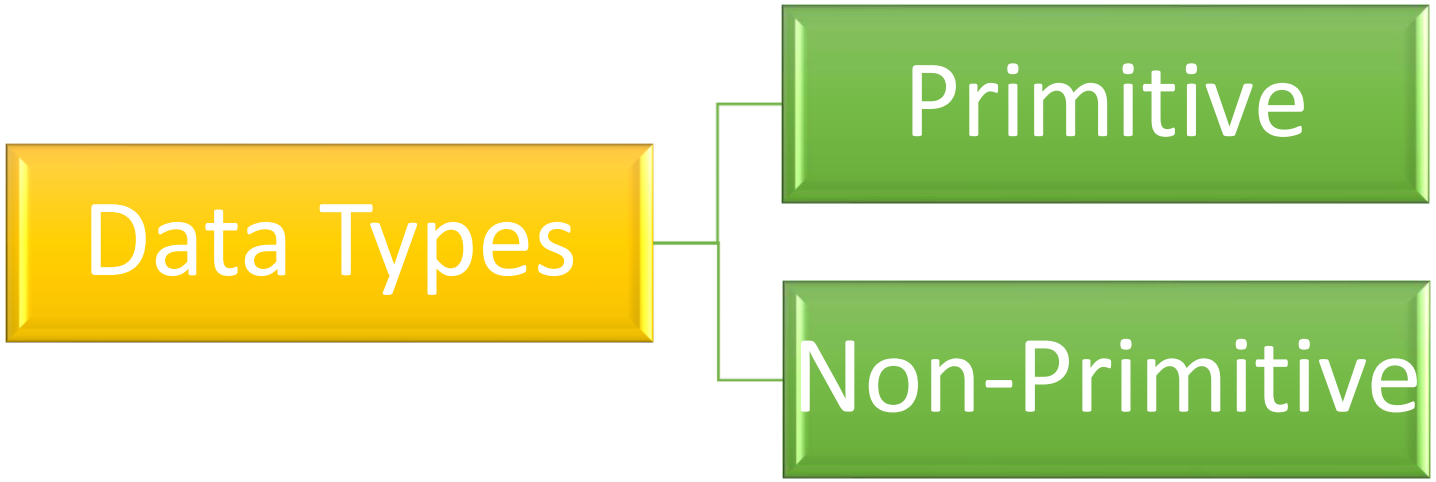
All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

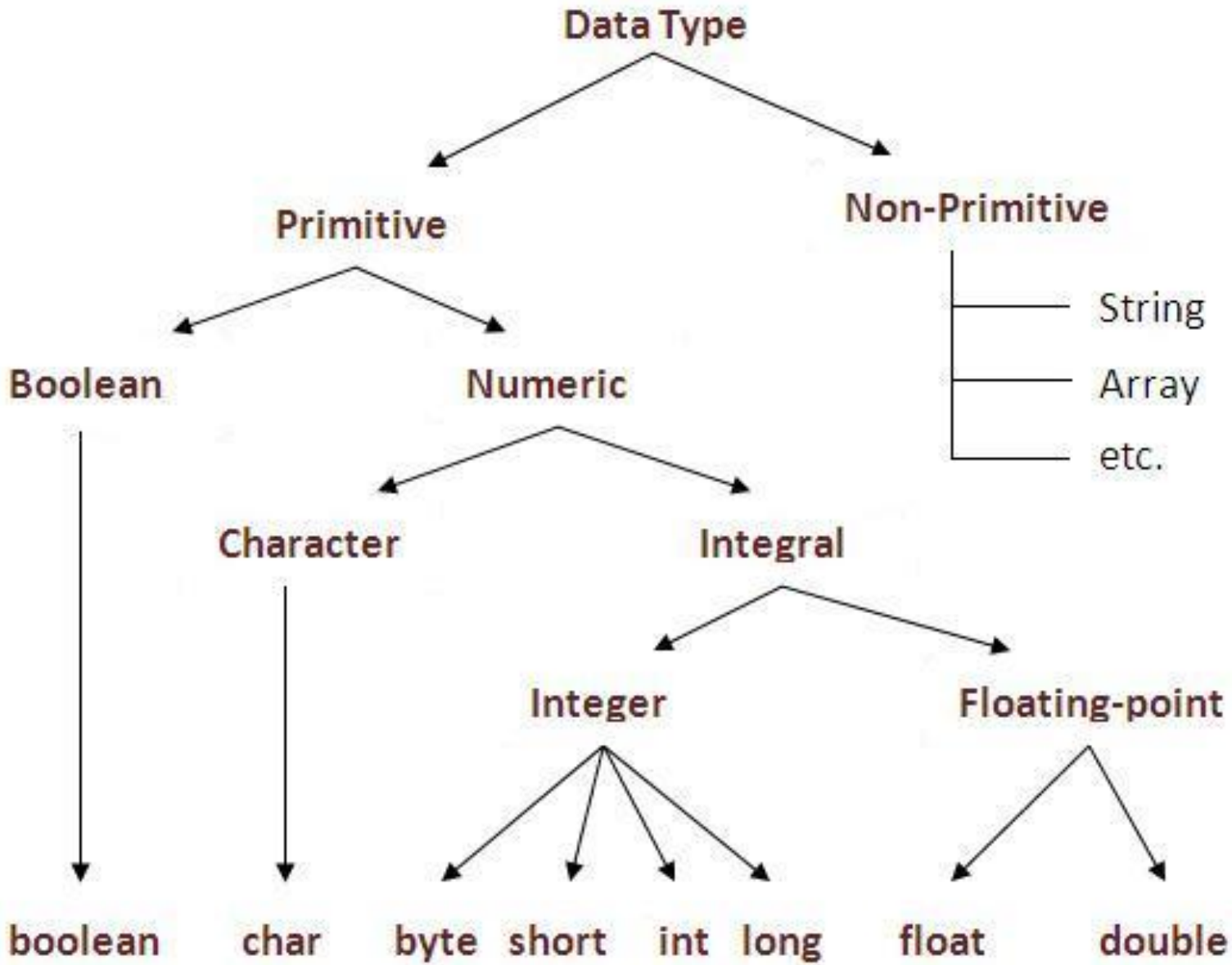
Data Types in Java

Data types represent the different values to be stored in the variable.

In java, there are two types of data types:



Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

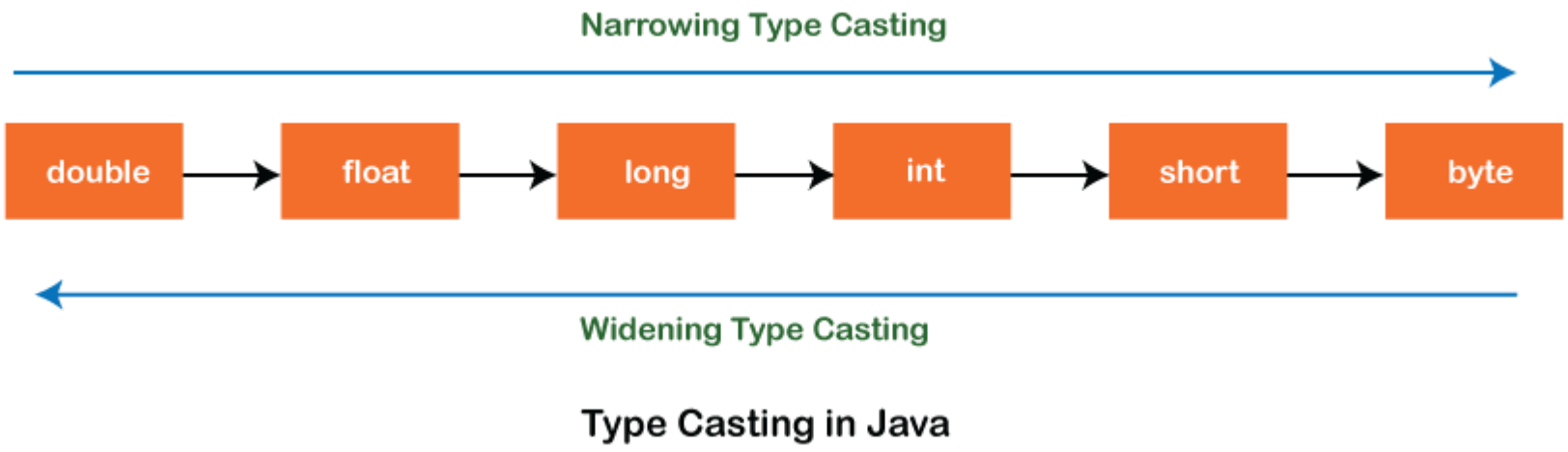
Data Type	Default Value	Default size	Minimum value	Maximum value
boolean	false	1 bit		
char	'\u0000'	2 byte		
byte	0	1 byte		
short	0	2 byte	-32,768 (-2^15)	32,767
int	0	4 byte	- 2,147,483,648 (-2^31)	2,147,483,647
long	0	8 byte	9,223,372,036,854 ,775,808(-2^63)	9,223,372,036,854,775,80 7
float	0.0	4 byte	1.4E-45	3.4028235E38
double	0.0	8 byte	4.9E-324	1.79769313486231 57E30 8

Java Variable Example: Add Two Numbers

```
class Simple
{
  public static void main(String[] args)
 {
    int a=10;
    int b=10;
    int c=a+b;
    System.out.println(c);
 }
}
```

Type Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.



Type Casting in Java

Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

```
class Cast1
{
	public static void main(String a[])
	{
		byte b=10;
		System.out.println("b="+b);
		short s=b;
		System.out.println("s="+s);
		int i=s;
		System.out.println("i="+i);
		long l=i;
		System.out.println("l="+l);
		float f=l;
		System.out.println("f="+f);
		double d=f;
		System.out.println("d="+d);
	}
}
```

Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

```
class Cast2{
public static void main(String a[])
{
double d=145.88;
System.out.println("d="+d);
long l=(long)d;   //Type casting
System.out.println("l="+l);
int i=(int)d;
System.out.println("i="+i);
}
}
```

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc. There are many types of operators in java which are given below:

- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Arithmetic Operator A=10 B=20

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

Java Unary Operator Example: ++ and –

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);// 10
System.out.println(++x);//  12
System.out.println(x--);//  12
System.out.println(--x);//  10
}}
```

Output:

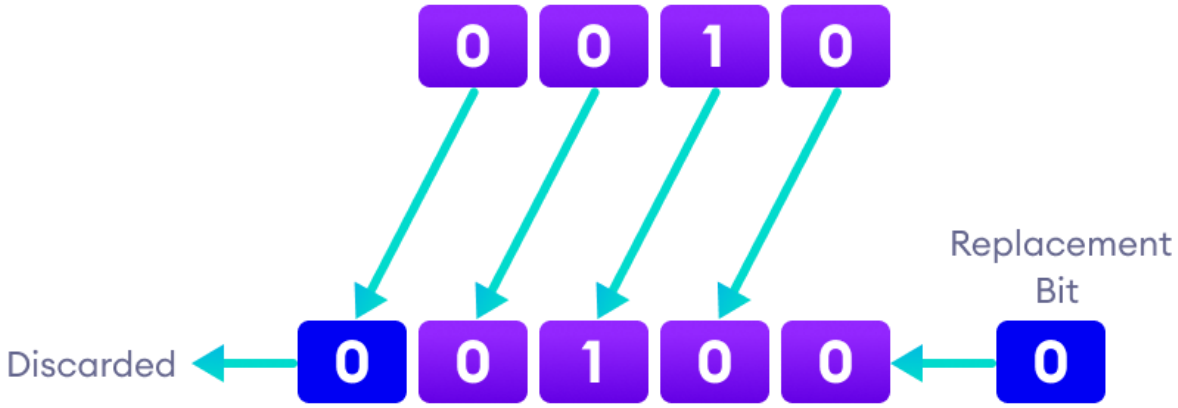
java Arithmetic Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Java Shift Operators

Java Left Shift Operator (<<)

The left shift operator shifts all bits towards the left by a certain number of specified bits. It is denoted by



Java Shift Operator Example: Left Shift

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

40 80 80 240

Logical Operators A=true B=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

Bitwise Operator

a = 0011 1100
b = 0000 1101
0011 0001

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

0011 1100
0000 1101

0011 0001

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is f

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//
System.out.println(a>b|a<c);//
System.out.println(a>b||a++<c);//
System.out.println(a);//
System.out.println(a>b|a++<c);
System.out.println(a);//
}}
```

Output:

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample
{
public static void main(String args[])
{
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

Output:

false false

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

Java Assignment Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a*=4;//a=a*4 (a=10*4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

14 16

Java Ternary Operator Example (?:) conditional oprtr

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b; 2<5  true then a else b
System.out.println(min);
}
}
```

Output:2

RELATIONAL OPERATOR

Assume variable A holds 10 and variable B holds 10,

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.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

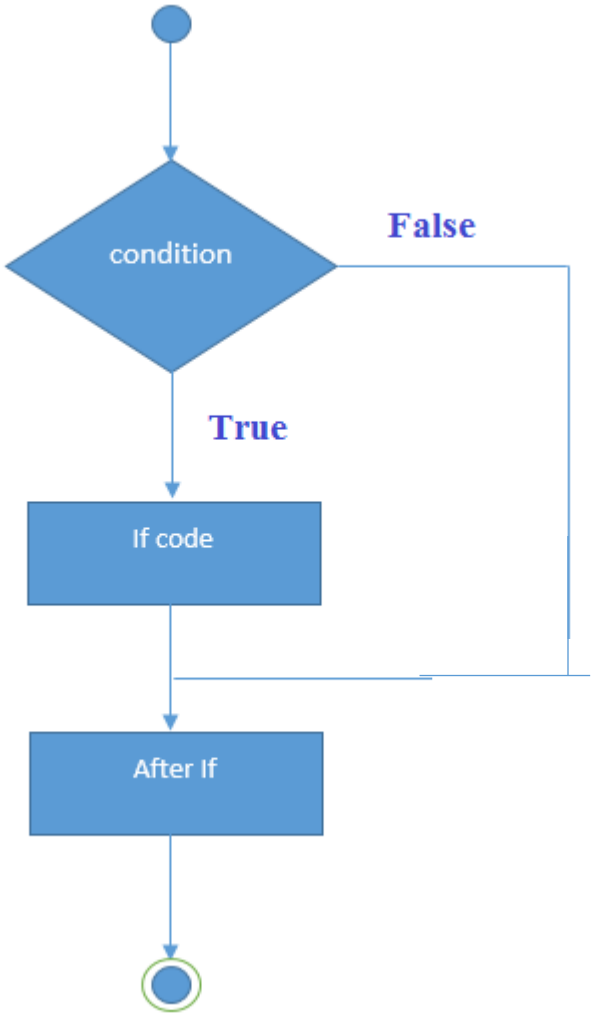
- if statement
- if-else statement
- nested if statement
- if-else-if ladder

Java IF Statement

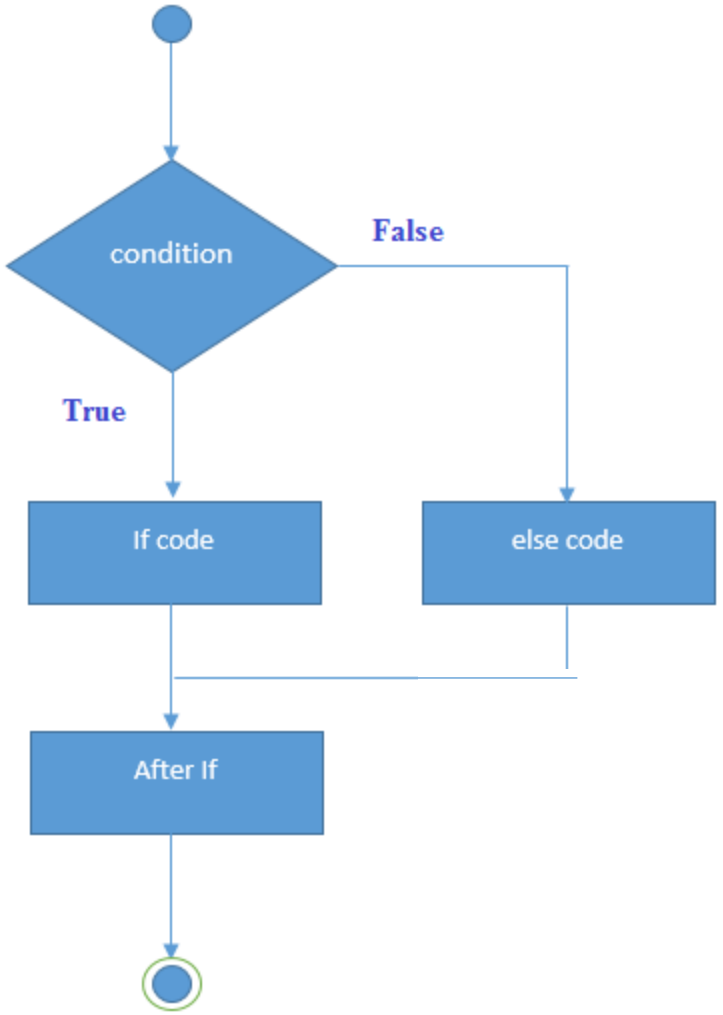
The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

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




```
public class IfExample {
public static void main(String[] args) {
    int age=20;
    if(age>18){
        System.out.print("Age is greater than 18");
    }
}
}
```

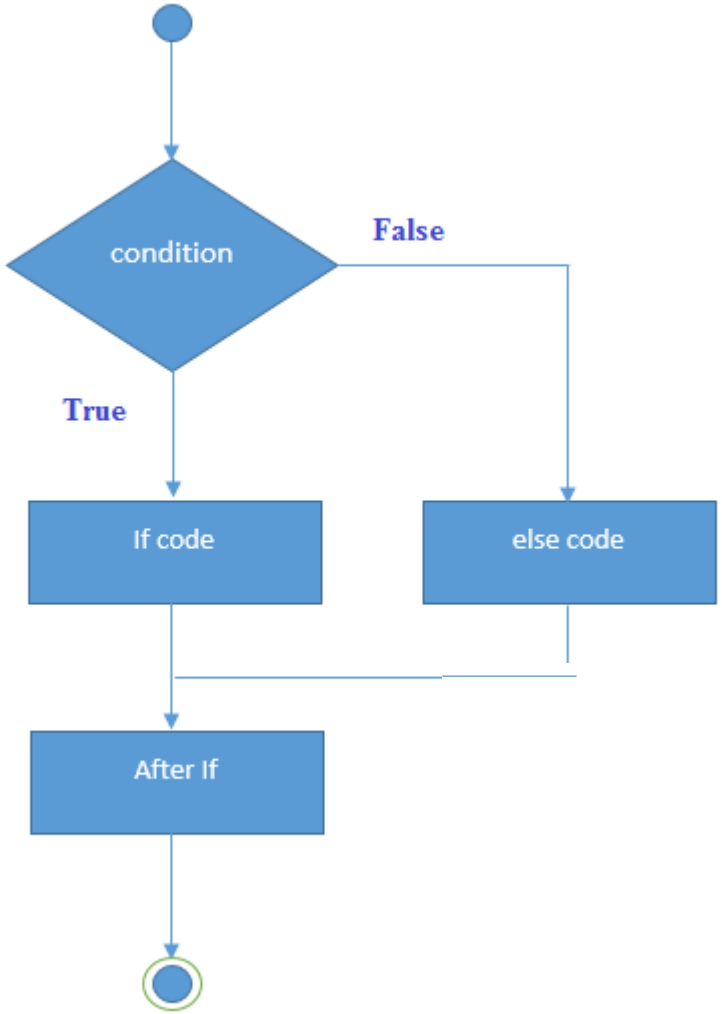


Java IF-else Statement

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
}
```




```
public class IfElseExample {
public static void main(String[] args) {
    int number=13;
    if(number%2==0){
        System.out.println("even number");
    }else{
        System.out.println("odd number");
    }
}
}
```

Java IF-else-if ladder Statement

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

IF ELSE Assignments

Write a Java program to check a number and print whether it is positive or negative.

Take three numbers from the user and print the greatest number.

Write a Java program that keeps a number from the user (integer between 1 and 7) and displays the name of the weekday.

Test Data
Input number: 3
Expected Output :
Wednesday

Write a Java program that takes a year from user and print whether that year is a leap year or not.
1.Year divisible by 4 its leap
2.If century, divisible by 400

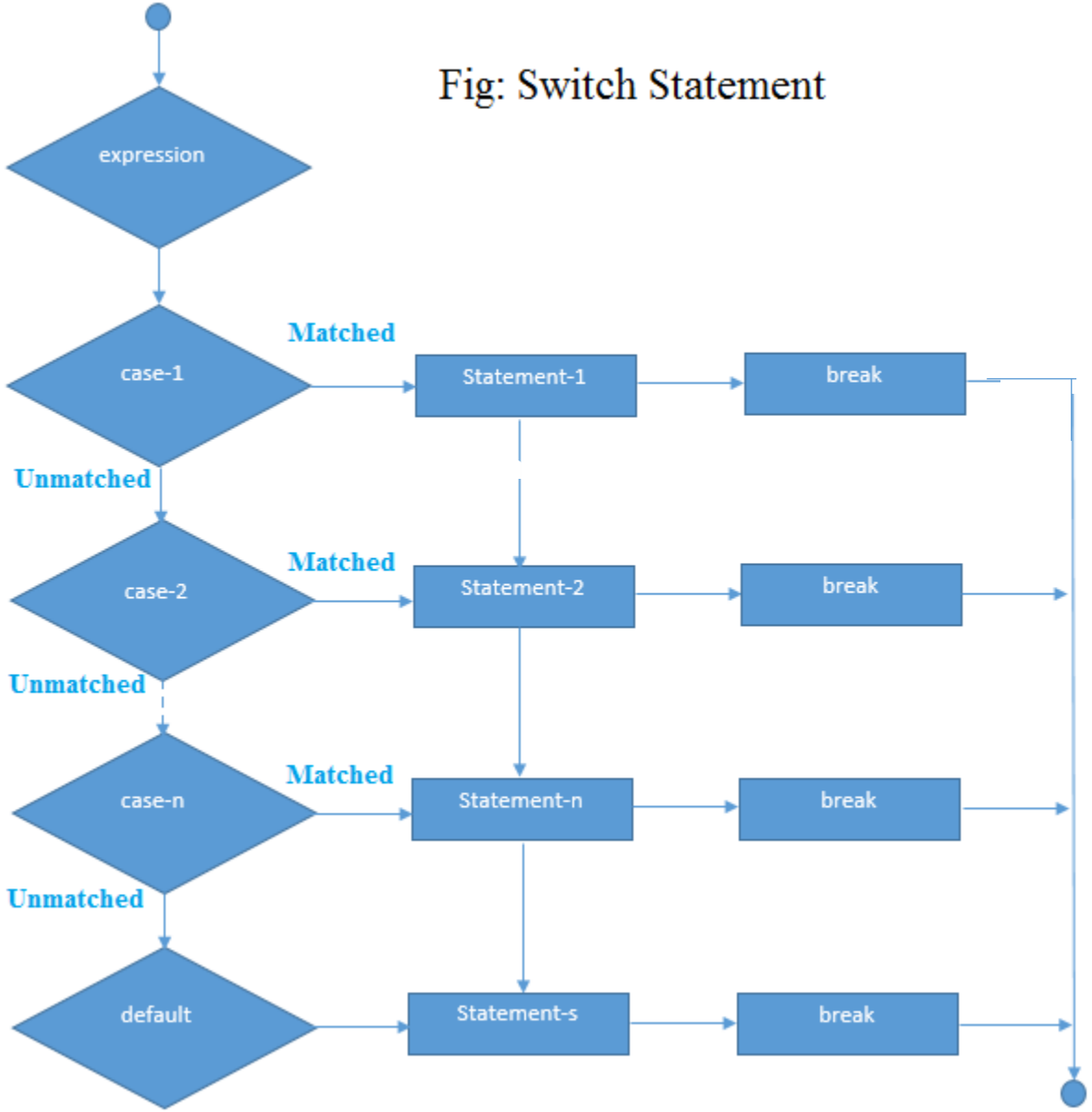
The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
switch(expression){
case value1: //code to be executed;
                    break;  //optional
case value2: //code to be executed;
                  break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

Fig: Switch Statement



```
public class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");break;
    case 20: System.out.println("20");break;
    case 30: System.out.println("30");break;
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Java Switch Statement is fall-through

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

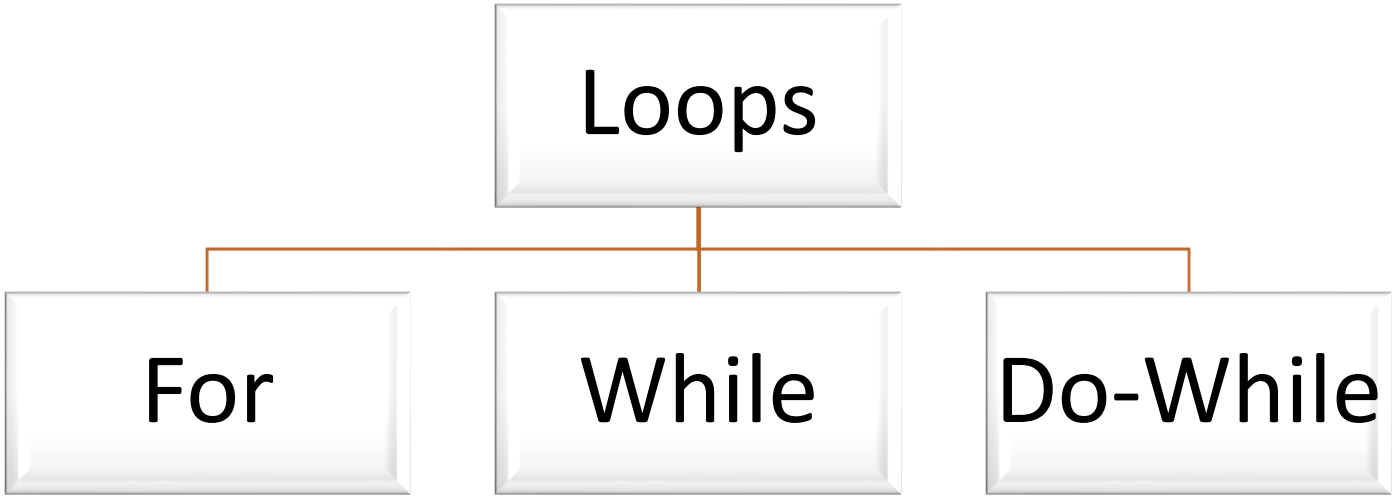
Example:

```
public class SwitchExample2 {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");
    case 20: System.out.println("20");
    case 30: System.out.println("30");
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

LOOPS

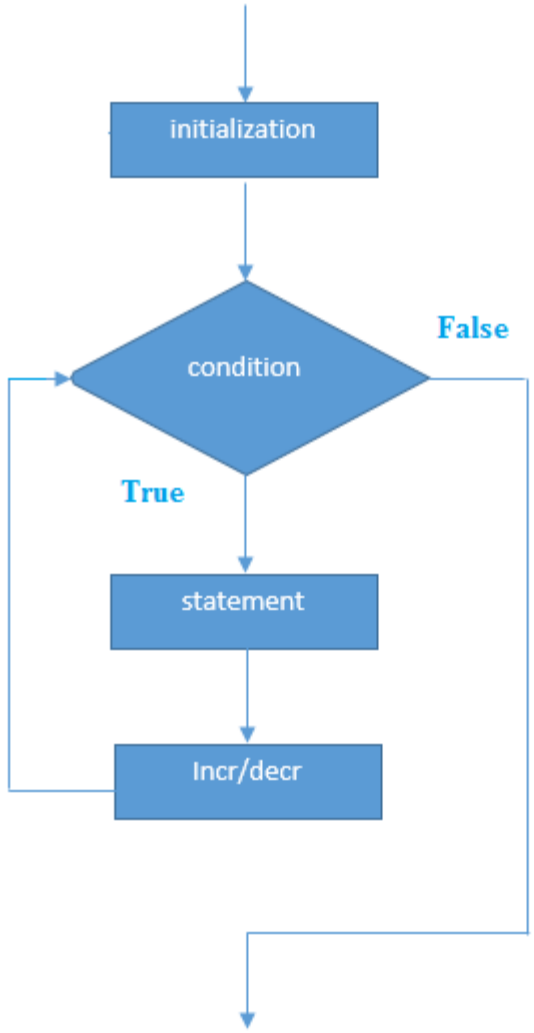
Looping in programming languages is a feature which facilitates the execution of a set of instructions repeatedly while some condition evaluates to true.

Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java we have three types of basic loops:



Java 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.



Syntax:

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

Example:

```
public class ForExample {
public static void main(String[] args) {
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

Java Infinitive For Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Syntax:

```
for(;;){
//code to be executed
}
```

Example:

```
public class ForExample {
public static void main(String[] args) {
    for(;;){
        System.out.println("infinitive loop");
    } }
}
```

Output:

infinitive loop infinitive loop infinitive loop infinitive loop infinitive loop ctrl+c

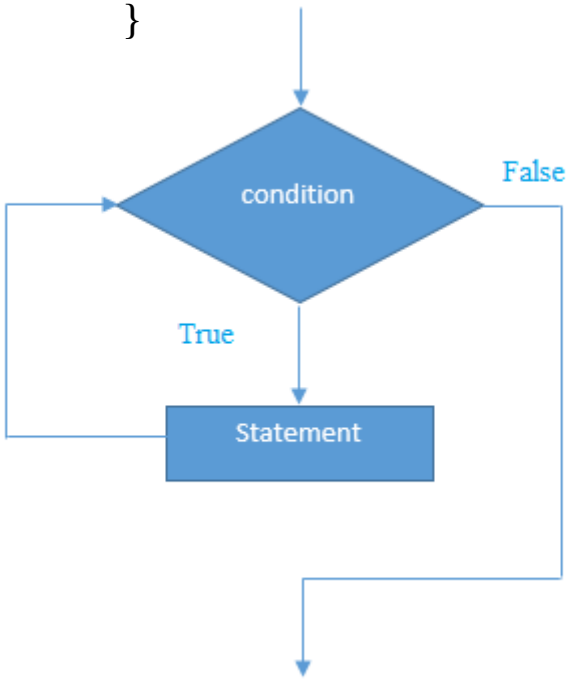
Now, you need to press ctrl+c to exit from the program.

Java While Loop

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

Syntax:

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



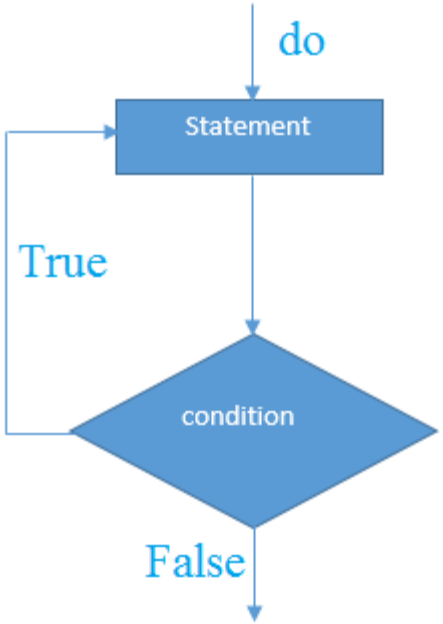

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10)
{
        System.out.println(i);
   i++;
    }
}
}
```

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed
}while(condition);
```




```
public class DoWhileExample {
public static void main(String[] args) {
    int i=11;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

Java Break Statement

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

1.jump-statement;
2.break;

Java Continue Statement

The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop

Syntax:

1.jump-statement;
2.continue;

Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

Types of Java Comments

1) Java Single Line Comment //

2) Java Multi Line Comment /* ……*/

The single line comment is used to comment only one line.

Syntax:

```
//This is single line comment
```

The multi line comment is used to comment multiple lines of code.

Syntax:

```
/*
This is
multi line  comment
*/
```

ASSIGNMENT 1

1) Fibonacci series

Write a java program to print fibonacci series
Input: 10
Output: 0 1 1 2 3 5 8 13 21 34

2) Prime number

Write a java program to check prime number.
Input: 44
Output: not prime number
Input: 7
Output: prime number

3 Palindrome number

Write a java program to check palindrome number.

Input: 329

Output: not palindrome number

Input: 12321

Output: palindrome number

4) Factorial number

Write a java program to print factorial of a number.

Input: 5

Output: 120

Input: 6

Output: 720

6 Armstrong number

Write a java program to check armstrong number.

153=(1*1*1)+(5*5*5)+(3*3*3)=1+125+27 =153

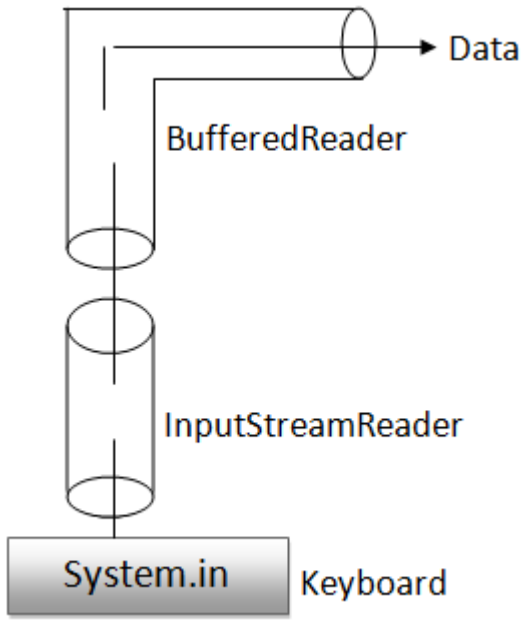
Input: 123

Output: not armstrong number

Input: 153

Output: armstrong number

Accepting Data From Keyboard



Reading data from keyboard

There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Scanner

InputStreamReader class

InputStreamReader class can be used to read data from keyboard.It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

BufferedReader class

BufferedReader class can be used to read data line by line by readLine() method.

```
import java.io.*;
class G5{
public static void main(String args[])throws Exception{


InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);
  System.out.println("Enter your name");
String name=br.readLine();
System.out.println("Enter your class");
String clas=br.readLine();
System.out.println("Welcome "+name);
 }
}
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
class Test {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Enter a id:");
        int id = Integer.parseInt(br.readLine());

        System.out.println("Enter Gender(M/F):");
        char gender = (char) br.read();

        System.out.println("Enter name:");
        String name = br.readLine();

        System.out.println("Id:" + id);
        System.out.println("Gen:" + gender);
        System.out.println("Enter name:" + name);
    }
}
```

Java User Input (Scanner)

Java User Input

The `Scanner` class is used to get user input, and it is found in the `java.util` package.
To use the `Scanner` class, create an object of the class and use any of the available methods found in the `Scanner` class documentation.

```
import java.util.Scanner; // Import the Scanner
class class MyClass
{
public static void main(String[] args)
{
 Scanner myObj = new Scanner(System.in); // Create a Scanner object
System.out.println("Enter username");
 String userName = myObj.nextLine(); // Read user input
System.out.println("Username is: " + userName); // Output user input
}
 }
```