CMT 211 Object Oriented Programming II

- Welcome to CMT 211, object oriented programming II using Java
- This is the second object oriented course. You have already taken oop I.
- Note that if you have not OOP I (In C++) and passed, then you should not be undertaking this course.

Course References and Teaching materials

1. You can refer from Java T Point (https://www.javatpoint.com/)
2. Some video from Java T point
3. Recorded Video from the Lecturer
4. Handout Notes from the Lecturer
5. Live Zoom Classes At times

Course Expectations

Since the mode of teaching in this semester is **online**, you are expected to be a student who :

- Knows what you want from the course and therefore you should not be pushed to attend to sessions, do assignment, and online cats.
- This calls for you to be a self disciplined student who is focused and know what is at stake.
- You should display a high level of commitment to your studies. The Online platform present to us a new of accomplishing same goals but with different strategy. It will be great tool to you if raise up to the occasion.
- Those of us who only do things when followed up will find it hard to fit in the new way of doing things. **Can you be counted among those who will thrive.**
- Ensure you are on top and are aware of any communication either through the platform and emails from the Lecturer and also regular communications from School

Class Participation

- Along the course, we will be using available technology to see that we get the best of training possible.
- There will be Forums, chats and live streams that will be set and you are expected to participate without failure.

Delivery Approach

To ensure that student get the most out of this course,we will have two sections

1. Java Programming Basics and
2. Object Oriented Programming In Java

Section A: Java Programming Basics

Java is an object-oriented, class-based, concurrent, secured and general-purpose computer-programming language which is a widely used and a robust technology.

What is Java

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

- It is a high level, robust, secured and object-oriented programming language.
- **It is also a Platform**: A platform is any hardware or software environment in which a program runs. Remember Operating Systems are softwares that runs on hardware platforms, But Os also acts as platforms on which applications runs.
    - Since Java has its own runtime environment (**JRE**) and **API**, it is called platform.

History of Java

- **Java history** is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as **set-top boxes**, **televisions** etc.
- The team includes **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991.
- For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology was incorporated by Netscape.
- Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.
- After that, it was called **Oak** and was developed as a part of the Green project.
- In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies
- Currently, Java is used in **internet programming**, **mobile devices**, **games**, **e-business** solutions etc.

Why sun choosed "Java" name?

1. **Why they chose java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
2. Java is an island of Indonesia where first coffee was produced (called java coffee).
3. Notice that Java is just a name not an acronym.
4. In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
5. The first version JDK 1.0 was released in(January 23, 1996).

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 (18th March, 2014)

Java SE 10 (March, 20, 2018)

Java SE 10 was released to remove primitive data types and move towards 64-bit addressable arrays to support large data sets.It was released on 20 March 2018, with twelve new features confirmed. These features are:

- Local-Variable Type Inference
- Experimental Java-Based JIT Compiler This is the integration of the Graal dynamic compiler for the Linux x64 platform
- Application Class-Data Sharing This allows application classes to be placed in the shared archive to reduce startup and footprint for Java applications
- Time-Based Release Versioning
- Parallel Full GC for G1
- Garbage-Collector Interface
- Additional Unicode Language-Tag Extensions
- Root Certificates
- Thread-Local Handshakes
- Heap Allocation on Alternative Memory Devices
- Remove the Native-Header Generation Tool - javah
- Consolidate the JDK Forest into a Single Repository

Java is 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.

Java is Object-oriented

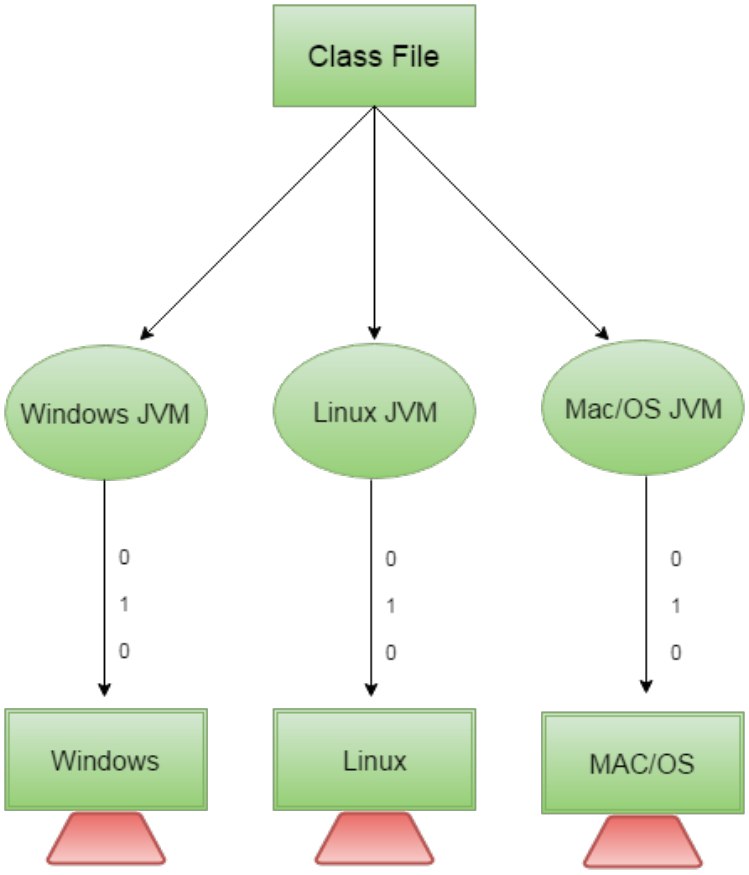
1. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.
2. Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
3. Basic concepts of all Object Oriented Programming Languages are:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

You should be able to define, explain & implement all these concepts

Java is 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 a **software-based platform** that runs on the top of other hardware-based platforms
- It has two components:
    1. Runtime Environment
    2. API(Application Programming Interface)
- By platform independent we mean that **the same Java code** can be run on multiple OS platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.
- In essence **Java source 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)**.

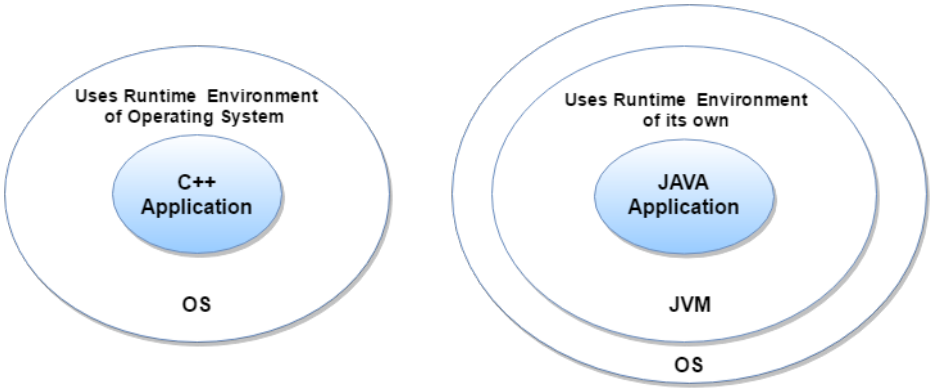


- The only requirement for each OS is a run-time environment called Java Virtual Machine (JVM which can be downloaded from the internet.
- Note **JVM** is platform dependent. This means that you download a JVM specifically for Linux, Windows etc.

Java Program is Secured

A program written in Java is said to be more secure because of the following:

- No explicit use of pointers (Explicitly ,pointers are used in c, and c++.)
- Java 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.

Some security can also be provided by application developer through SSL, Cryptography etc.

Java is 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.

Java is 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.

Java is 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++) .

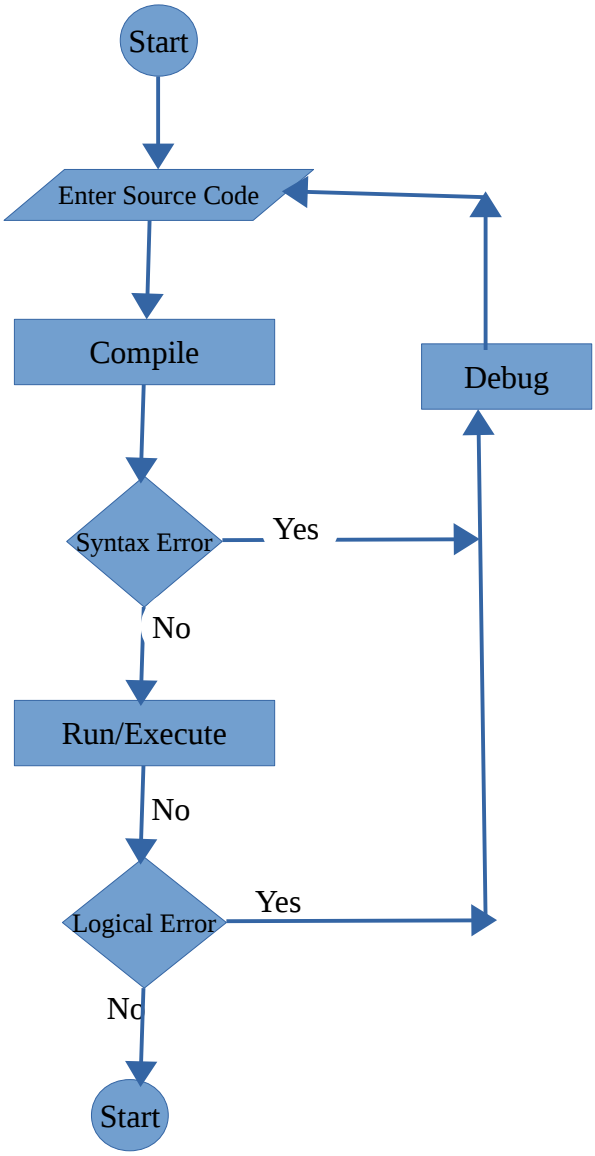
Java is Distributed

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.

Java is Multi-threaded

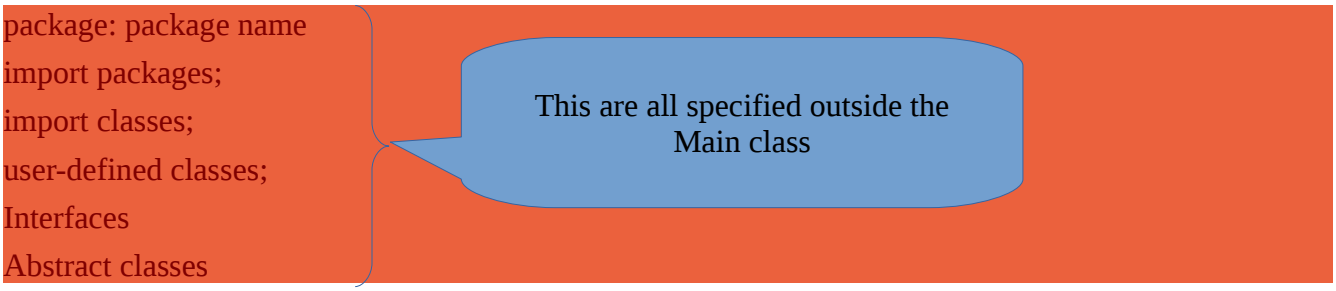
- A thread is like a separate program, executing concurrently.
- We can write Java programs that deal with many tasks at once by defining multiple threads.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread.
- It shares a common memory area. Threads are important for multi-media, Web applications etc.

Java Program Development Flow Chart

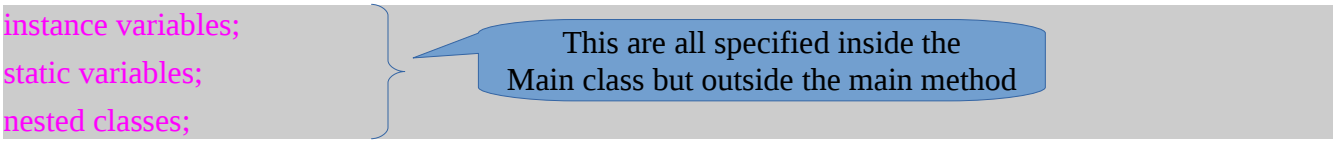


Structure of A Java Program

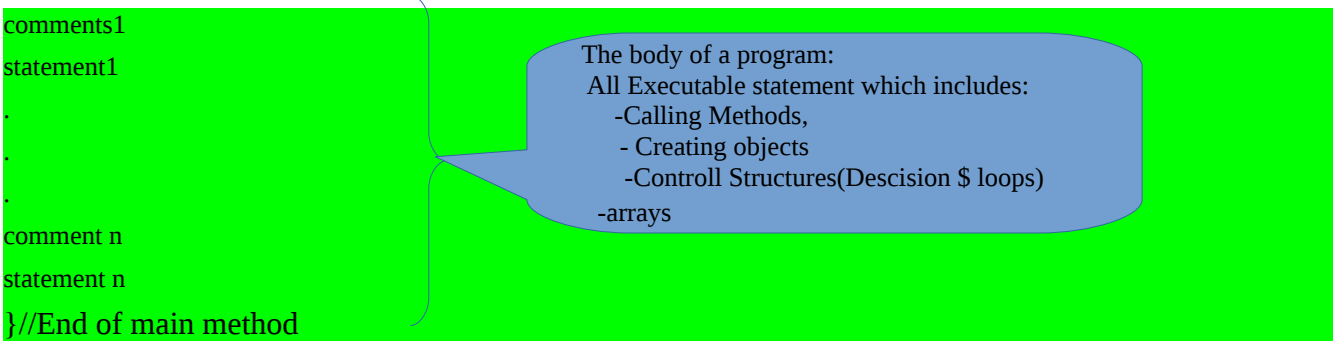
All java programs may assume the form below

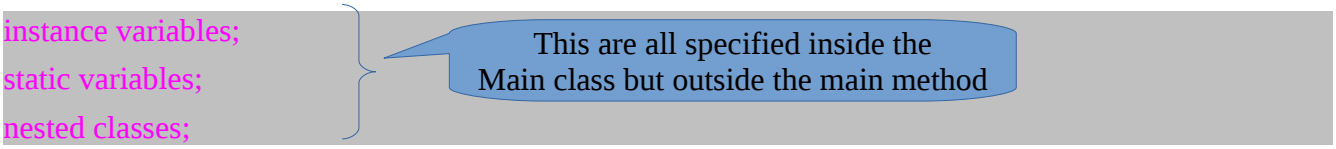


public class MainClass{//Beginning of current class



public static void main(String [] args){ //Main method





}// End of current class

Learning how to use java in writing a program is running how to write simple programs to complicated programs that makes use of all variable types, abstract classes, nested classes, interfaces, and packages.

We will start right from the basics as we build up our knowledge and experiences to a level where we can confidently develop a system in java.

Writing A Program in Java: What You Need

1. A choice of the development Operating System Platform; For our work, we will be using Linux **OS**
2. A text Editor. This may depend on operating system. In our case we will be using geditor /kwriter/emacs/vi/vim
3. JSDK installed and configured. Linux OS come shipped with a JSDK although it can be upgraded

Writing A Program in Java: Setting Up Your Development Environment

1. Since we will be working in Linux it is important you work from you own account:
2. Login as the root user or use su to switch to root user
3. Create you own account(Make sure you master very well the password)
4. Login using your account.
5. Create a directory (**Java**) on the desktop to store you work. (use **mkdir Desktop/Java**)

Writing A Program in Java: The Steps

1. Enter your source code in you text editor.
2. Save the file as **classname.java** in the directory you created above
3. Open the terminal widow
4. From the terminal move to the directory you created.
5. To compile and check for syntax errors run the command javac **classname.java.** This command if successful compiles the source to a bytecode **classname.class.** Open the directory to view the list. (**ls -l**)
6. To execute the bytecode run the command java **classname**.

Example 1: Welcome

```
public class Welcome{// class name Welcome.java should be the name of the file
public static void main(String [] args){// Ensure you muster this method
System.out.println("Welcome to Java Programming. “\n” A pure Object Oriented Programming Language");
}
}
```

Understanding first java program

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

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the

main method. So it saves memory.

- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement.

More on **System.out.println()**

We can use System.out.println() to :

1. Prompt the use for an input or an advice on how to use a particular aspect of the program.
2. To print out the output of an execution (part of a program)

In the above example it outputs the following meaning “\n” is used just is its is supposed to. The new line escape character. Other include the “\t” which represent the tab character.

Welcome to Java Programming.

A pure Object Oriented Programming Language

Exercise

Using the System.out.println() and “\n” and “\t” write a Java program that will explain the step by step process of java program development cycle. Let us work with the class name **Jpdc**

C++ vs Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
1. Platform-independent	C++ is platform-dependent.	Java is platform-independent.
2. Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
3. Goto	C++ supports goto statement.	Java doesn't support goto statement.
4. Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
5. Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
6. Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
7. Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
8. Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
9. Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
10. Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
11. Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
12. Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
13. unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
14. Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

An Example to Illustrate Pointers in C++

```
#include<iostream>
using namespace std;
int main(){
int x=  10;
int y=  20;
int *xptr=  &x;//Creating a pointer xptr and assigning it memory addres of x
int *yptr=  &y;;//Creating a pointer xptr and assigning it memory addres of y
cout<<"xptr held value="<<  xptr<<endl;// Memory addr of x
cout<<"yptr held value="<<  xptr<<endl;// Memory addr of y
cout<<"Value of x using xptr="<<*xptr<<  endl;
cout<<"Value of y using yptr="<<*yptr<<  endl;
cout<<"Doing Mathematics Using Pointers"<<  endl;
cout<<"x+y="<<*xptr+*yptr<<  endl;
//Extend this lie to print out product, division, and modulus of x and y
return 0;
}
```

Understanding JDK, JRE and JVM

Understanding the difference between JDK, JRE and JVM is important.

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

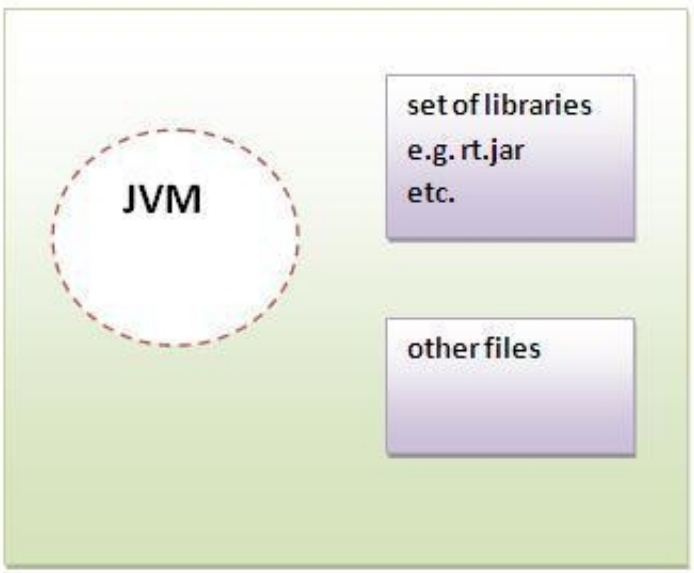
JVM provides definitions for the:

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

JRE

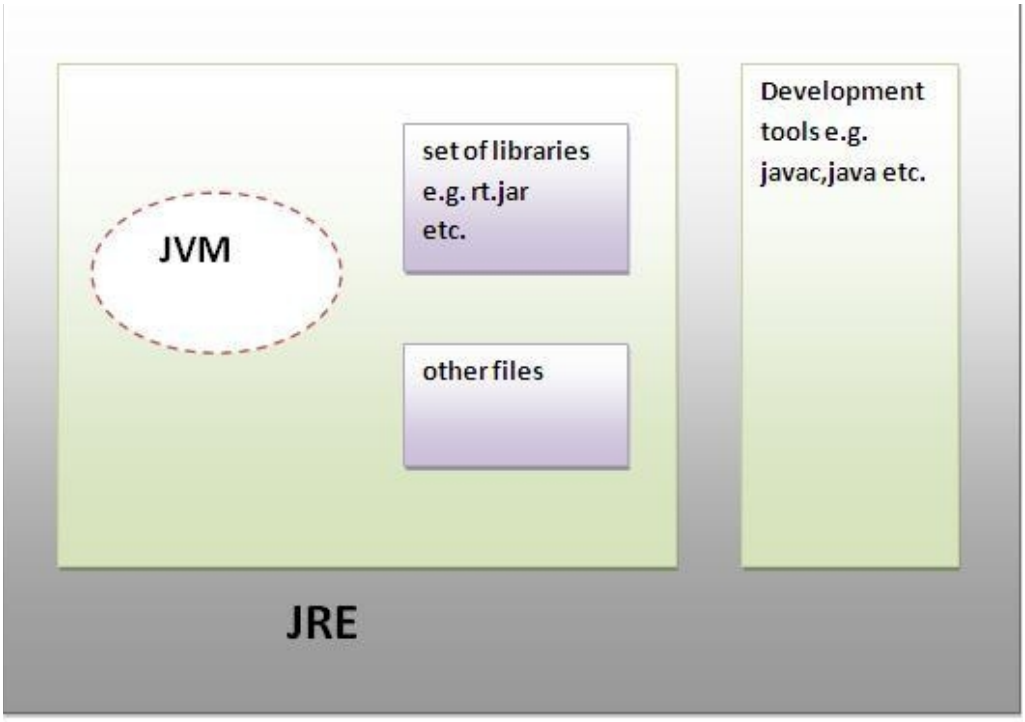
JRE is an acronym for **Java Runtime Environment**.It is used to provide runtime environment.It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

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

Variable and Datatype in Java

Variable

Variable is name of reserved area allocated in memory.

Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable

Local Variable: A variable that is declared inside the method is called local variable.

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

Static variable : A variable that is declared as static is called static variable. It cannot be local.

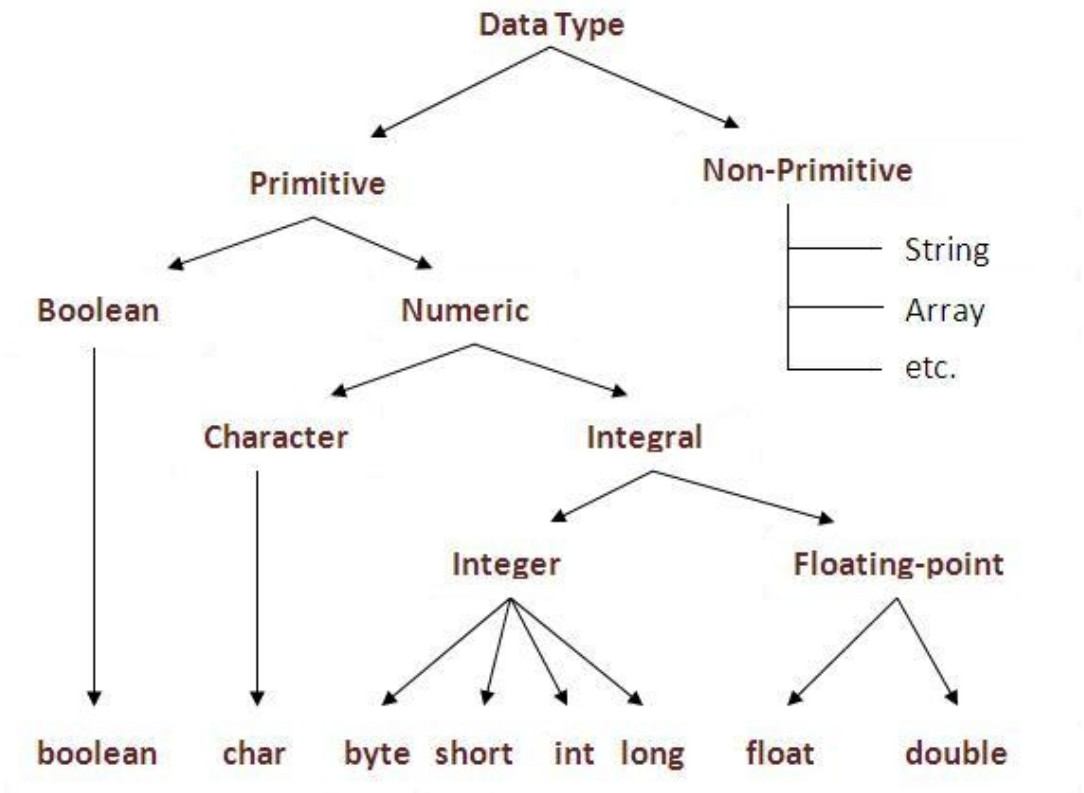
Example: Illustrating Instance and static types of variables

```
public class VariableExample{
int data=50;//instance variable
static int m=100;//static variable
public static void main(String [] args){
System.out.println("Working with variables");
 //System.out.Println(data+"");//Introduces an error
  int a=50; //Local variable
  System.out.println(m+"");
   System.out.println(a);
}
}
```

Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



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

N.B

1. The size of each data type indicates how much memory will be allocated.
2. char uses 2 byte in java and what is \u0000.

This is because java uses **unicode system** rather than **ASCII code** system.

\u0000 is the lowest range of unicode system

Why Use Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

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.

Problem

There are problems with these earliear system:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length. Some common characters are encoded as single bytes, other require two or more byte.

Solution

To solve these problems, a new language standard was developed i.e. Unicode System.
In unicode, character holds 2 byte, so java also uses 2 byte for characters. A total of 2^{16} Characters.

lowest value:\u0000

highest value:\uFFFF

Operators in java

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

Operators	Precedence	
postfix	*Expr*++ , *expr*--	
unary	++*expr* --*expr* +*expr* -*expr* ~ !	
Arithmetic	+ - * / %	
Shift Operators	>> (op1 >> op2)	shift bits of op1 right by distance op2 e.g 13 >>1= 110, or 6 in decimal 13=1101 in binary.
Shift Operators	<<(op1 << op2)	shift bits of op1 left by distance op2
Shift Operators	(op1 >>> op2)	shift bits of op1 right by distance op2 (unsigned)
relational	< > <= >= instanceof (op1 instanceof op2)	The `instanceof` operator tests whether its first operand is an instance of its second. `op1` must be the name of an object and `op2` must be the name of a class. An object is considered to be an instance of a class if that object directly or indirectly descends from that class
equality	== !=	
bitwise AND	& (op1 & op2)	`op1` and `op2` are both `true`, always evaluates `op1`

		and `op2`
bitwise exclusive OR	`^ ( op1 ^ op2)`	if op1 and op2 are different--that is if one or the other of the operands is true but not both
bitwise inclusive OR	`| (op1 op2)`	either `op1` or `op2` is `true`, always evaluates `op1` and `op2`
logical AND	&&	
logical OR	`|| (op1 || op2)`	either `op1` or `op2` is `true`, conditionally evaluates `op2`
ternary	`? : ( e.g` **op1 ? op2 : op3**)	Returns op2 if op1 is true or returns op3 if op1 is false.
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`	

Difference between ++ and -- operator as postfix and prefix
When i++ is used as prefix(like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

int c=2,d=2;
System.out.println(c++);//this statement displays 2 then, only c incremented by 1 to 3.
System.out.println(++c);//this statement increments 1 to c then, only c is displayed

Assignment Operator

The following table lists the shortcut assignment operators and their equivalents:

Operator	Use	Equivalent to
+=	`op1 += op2`	`op1 = op1 + op2`
-=	`op1 -= op2`	`op1 = op1 - op2`
*=	`op1 *= op2`	`op1 = op1 * op2`
/=	`op1 /= op2`	`op1 = op1 / op2`
%=	`op1 %= op2`	`op1 = op1 % op2`
&=	`op1 &= op2`	`op1 = op1 & op2`
|=	`op1 |= op2`	`op1 = op1 | op2`
^=	`op1 ^= op2`	`op1 = op1 ^ op2`
<<=	`op1 <<= op2`	`op1 = op1 << op2`
>>=	`op1 >>= op2`	`op1 = op1 >> op2`
>>>=	`op1 >>>= op2`	`op1 = op1 >>> op2`

Java Scanner class(java.util.Scanner)

- Provides many methods for reading input from the keyboard. There are other methods.
- The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault.
- Java Scanner class is widely used to parse text for string and primitive types using regular expression.
- Java Scanner class **extends** Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method	Description
public String next()	Returns the next token from the scanner.
public String nextLine()	Moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	Scans the next token as a byte.
public short nextShort()	Scans the next token as a short value.
public int nextInt()	Scans the next token as an int value.
public long nextLong()	Scans the next token as a long value.
public float nextFloat()	Scans the next token as a float value.
public double nextDouble()	Scans the next token as a double value.

Example to get input from console

An example which reads the int, string and double value as an input:

```
1. import java.util.Scanner;
2. class ScannerTest{
3.  public static void main(String args[]){
4.    Scanner sc=new Scanner(System.in);
5.    System.out.println("Enter your stnum");
6.    int stnum=sc.nextInt();
7.    System.out.println("Enter your First name");
8.    String name=sc.next();
9.    System.out.println("Enter your school fees");
10.   double fee=sc.nextDouble();
11.   System.out.println("Student Number:"+stnum+" name:"+name+" Shool Fees:"+fee);
12.   sc.close();
13. }
    }
```

Example2 with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

```
1. import java.util.*;
2. public class ScannerTest2{
3. public static void main(String args[]){
4.     String input = "10 tea 20 coffee 30 tea buiscuits";
5.     Scanner s = new Scanner(input).useDelimiter("\\s");
6.     System.out.println(s.nextInt());
7.     System.out.println(s.next());
8.     System.out.println(s.nextInt());
9.     System.out.println(s.next());
10.    s.close();
11.}}
```

Exercise

1. Write a java program that prompt you for the value of integer x and integer y and return the modulus x%y.

Input from a dialog window

An alternative way to read strings from input is to use the predefined method `showInputDialog()`, which is defined in the class `JOptionPane`, which in turn is part of the `swing` library. Using such a method, we can read input from the keyboard according to the following schema:

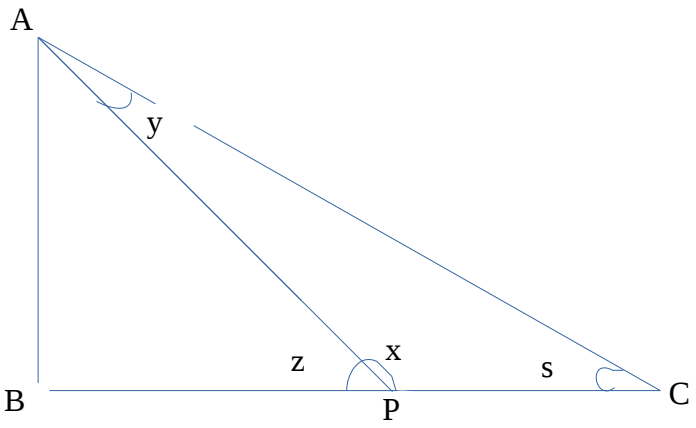
```
import javax.swing.JOptionPane;
public class KeyboardInput {
  public static void main (String[] args) {
    ...
    String inputString = JOptionPane.showInputDialog("Insert a string");
    ...
    System.out.println(inputString);
    ...
    System.exit(0);
  }
}
```

Revision Exercise 1: Arithmetic

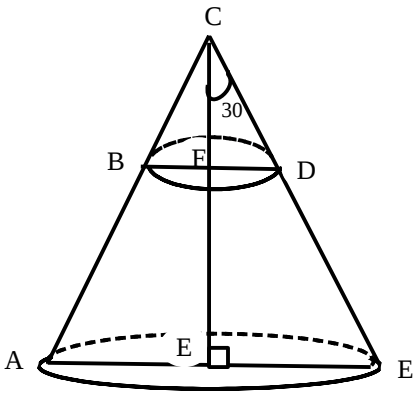
For each of the programs below determine first the variables required and their types. Workout first how to do the calculation in you book. Write the corresponding code in the book before going into the computer.

1. Write a program to calculate the area of rectangle.
2. Write a program to calculate the area of triangle.
3. Write a program to calculate the area of circle.

4. In a country with a population of 14,000,000 people, 55% are females, 45% of the males are empl0yed, and 25% 0f the females are employed. Write a program that will out put:
   i. The male population in the country.
   ii. The female population unemployed.
   iii. The total number of people employed.
5. A rectangular tank whose capacity is 200 liters has a length of 50 cm and width 320 cm. Write a program that will receive the above parameters and output the height of the tank.
6. What is the relationship between mas, density and volume. Write a program that prompts the user for the mass of an object, the density and the calculate the volume.
7. The exchange rate of a Kenyan shilling to Uganda shilling is 1Ksh=24Ush. And an American dollar to Uganda shillings is 1$=1,950Ush. Write a program that will prompts for any amount of Kenya shillings and output the corresponding American dollars.
8. Custom duty and purchase tax are levied on a certain imported commodities and are calculated as follows.
   i. Custom Duty 30% of the value of the commodity
   ii. Purchase Tax 20% of (the value + Custom duty). Write a program that prompts the user for the value of the imported item and then output the Custom duty, The purchase duty and the total percent charged in imported items.
9. Consider the triangle below. Write a program that prompts the user for the value of angle S and Z and then output the angles x and y. Comment on you program to explain the working.



10. A rectangular prism has a length of x cm, width of y cm and a height of zcm. Write a program that prompts the user for the prism parameters and the out put
    i. The total surface area
    ii. The Prism volume
    iii. The capacity of the prism
11. ABCDE is a right solid cone. The ratio CD:DE=2:3 The cone BCD was cut off. Assuming angle <FCD=30 degrees , Write a program that prompts the user for the length CE and then output:
    i. The total surface area of the remaining portion ABDE
    ii. The volume of the cone BCD.
    iii. parameters and the out put



The Math Class Methods

Java like any other programming and scripting languages provide a rich set of methods that can help a programmer carry out mathematical evaluations.
The Math class methods and their counter parts in other classes shipped with java help the programmer to focus more on their problems rather than waste time on none problem issues.

Modifier and Type	Method and Description
`static double`	**`abs(double a)`** :Returns the absolute value of a `double` value.
`static float`	**`abs(float a)`** :Returns the absolute value of a `float` value.
`static int`	**`abs(int a)`** :Returns the absolute value of an `int` value.
`static long`	**`abs(long a)`** :Returns the absolute value of a `long` value.
`static double`	**`acos(double a)`** :Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
`static int`	**`addExact(int x, int y)`** :Returns the sum of its arguments, throwing an exception if the result overflows an `int`.
`static long`	**`addExact(long x, long y)`** :Returns the sum of its arguments, throwing an exception if the result overflows a `long`.
`static double`	**`asin(double a)`** :Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
`static double`	**`atan(double a)`** :Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
`static double`	**`atan2(double y, double x)`** :Returns the angle *theta* from the conversion of rectangular coordinates (`x`, `y`) to polar coordinates (r, *theta*).
`static double`	**`cbrt(double a)`** :Returns the cube root of a `double` value.
`static double`	**`ceil(double a)`** :Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer.
`static double`	**`copySign(double magnitude, double sign)`** Returns the first floating-point argument with the sign of the second floating-point argument.
`static float`	**`copySign(float magnitude, float sign)`** :Returns the first floating-point argument with the sign of the second floating-point argument.
`static double`	**`cos(double a)`** :Returns the trigonometric cosine of an angle.
`static double`	**`cosh(double x)`** :Returns the hyperbolic cosine of a `double` value.
`static int`	**`decrementExact(int a)`** :Returns the argument decremented by one, throwing an exception if the result overflows an `int`.
`static long`	**`decrementExact(long a)`** :Returns the argument decremented by one, throwing an exception if the result overflows a `long`.
`static double`	**`exp(double a)`** :Returns Euler's number *e* raised to the power of a `double` value.
`static double`	**`expm1(double x)`** :Returns e^x -1.
`static double`	**`floor(double a)`** :Returns the largest (closest to positive infinity) `double` value that is less than or equal to the argument and is equal to a mathematical integer.
`static int`	**`floorDiv(int x, int y)`** :Returns the largest (closest to positive infinity) `int` value that is less than or equal to the algebraic quotient.

`static long`	**`floorDiv(long x, long y)`**: Returns the largest (closest to positive infinity) `long` value that is less than or equal to the algebraic quotient.
`static int`	**`floorMod(int x, int y)`** :Returns the floor modulus of the `int` arguments.
`static long`	**`floorMod(long x, long y)`**: Returns the floor modulus of the `long` arguments.
`static int`	**`getExponent(double d)`** Returns the unbiased exponent used in the representation of a `double`.
`static int`	**`getExponent(float f)`** Returns the unbiased exponent used in the representation of a `float`.
`static double`	**`hypot(double x, double y)`** Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.
`static double`	**`IEEEremainder(double f1, double f2)`** Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
`static int`	**`incrementExact(int a)`** :Returns the argument incremented by one, throwing an exception if the result overflows an `int`.
`static long`	**`incrementExact(long a)`** :Returns the argument incremented by one, throwing an exception if the result overflows a `long`.
`static double`	**`log(double a)`**: Returns the natural logarithm (base *e*) of a `double` value.
`static double`	**`log10(double a)`**:Returns the base 10 logarithm of a `double` value.
`static double`	**`log1p(double x)`** :Returns the natural logarithm of the sum of the argument and 1.
`static double`	**`max(double a, double b)`**:Returns the greater of two `double` values.
`static float`	**`max(float a, float b)`**:Returns the greater of two `float` values.
`static int`	**`max(int a, int b)`**: Returns the greater of two `int` values.
`static long`	**`max(long a, long b)`**: Returns the greater of two `long` values.
`static double`	**`min(double a, double b)`**:Returns the smaller of two `double` values.
`static float`	**`min(float a, float b)`**:Returns the smaller of two `float` values.
`static int`	**`min(int a, int b)`**:Returns the smaller of two `int` values.
`static long`	**`min(long a, long b)`** :Returns the smaller of two `long` values.
`static int`	**`multiplyExact(int x, int y)`** :Returns the product of the arguments, throwing an exception if the result overflows an `int`.
`static long`	**`multiplyExact(long x, long y)`** :Returns the product of the arguments, throwing an exception if the result overflows a `long`.
`static int`	`negateExact(int a)`: Returns the negation of the argument, throwing an exception if the result overflows an `int`.
`static long`	`negateExact(long a)` :Returns the negation of the argument, throwing an exception if the result overflows a `long`.
`static double`	`nextAfter(double start, double direction)` Returns the floating-point number adjacent to the first argument in the direction of the second argument.
`static float`	`nextAfter(float start, double direction)` Returns the floating-point number adjacent to the first argument in the direction of the second argument.
`static double`	`nextDown(double d)` :Returns the floating-point value adjacent to `d` in the direction of negative infinity.

`static float`	`nextDown(float f)` :Returns the floating-point value adjacent to `f` in the direction of negative infinity.
`static double`	`nextUp(double d)` :Returns the floating-point value adjacent to `d` in the direction of positive infinity.
`static float`	`nextUp(float f)`:Returns the floating-point value adjacent to `f` in the direction of positive infinity.
`static double`	`pow(double a, double b)` :Returns the value of the first argument raised to the power of the second argument.
`static double`	**`random()`**:Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static double`	`rint(double a)` :Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.
`static long`	**`round(double a)`** :Returns the closest `long` to the argument, with ties rounding to positive infinity.
`static int`	**`round(float a)`** :Returns the closest `int` to the argument, with ties rounding to positive infinity.
`static double`	`scalb(double d, int scaleFactor)` Returns `d` × $2^{\texttt{scaleFactor}}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
`static float`	`scalb(float f, int scaleFactor)` Returns `f` × $2^{\texttt{scaleFactor}}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
`static double`	`signum(double d)` Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
`static float`	`signum(float f)` Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
`static double`	`sin(double a)` Returns the trigonometric sine of an angle.
`static double`	`sinh(double x)` Returns the hyperbolic sine of a `double` value.
`static double`	`sqrt(double a)` Returns the correctly rounded positive square root of a `double` value.
`static int`	`subtractExact(int x, int y)` Returns the difference of the arguments, throwing an exception if the result overflows an `int`.
`static long`	`subtractExact(long x, long y)` Returns the difference of the arguments, throwing an exception if the result overflows a `long`.
`static double`	`tan(double a)` Returns the trigonometric tangent of an angle.
`static double`	`tanh(double x)` Returns the hyperbolic tangent of a `double` value.
`static double`	`toDegrees(double angrad)` Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
`static int`	`toIntExact(long value)` Returns the value of the `long` argument; throwing an exception if the value overflows an `int`.

`static double`	`toRadians(double angdeg)` Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
`static double`	`ulp(double d)` Returns the size of an ulp of the argument.
`static float`	`ulp(float f)` Returns the size of an ulp of the argument.

Example: Illustrating Math Class Methods

```
public class MathMethods{
public static void main(String [] args){
System.out.println("Working with Math Class Methods");
int x=4;
int m=3;//static variable
int a=2.5; //Local variable
System.out.println(Math.max(m,x));
System.out.println(Math.min(m,x));
System.out.println(Math.sqrt(Math.pow(m,x)));
System.out.println(Math.max(m,x));
}
}
```

Example2: Illustrating Generating Random numbers

```
public class MathRandom{
public static void main(String [] args){
System.out.println(Math.random());// Generating any random value between 0 and 1 ( 1 exclusive)
System.out.println(Math.random()*10);// Generating any random value between 0 and9.9999
System.out.println(Math.random()*100);// Generating any random value between 0 and99.9999
System.out.println(Math.ceil(Math.random()*10));// Generating any random value between 0 and 10
System.out.println(Math.ceil(Math.random()*100));// Any random value between 0 and 100
System.out.println(5+Math.ceil(Math.random()*20));// Any random value between 5 and 25

}
}
```

Example3: Generating Random numbers

```
public class MathRandom{
public static void main(String [] args){
System.out.println("CAT1\tCAT2\t TCAT\tEXAM\t AGG\t");// Any random value between 5 and 25
for ( int i=1;i<=10;i++){
int cat1= 2+Math.ceil(Math.random()*10)));
int cat2= 5+Math.ceil(Math.random()*15)));
int exam= Math.ceil(Math.random()*70));
int tcat= cat1+ cat2;
int agg= tcat+ exam;
System.out.println(cat1+"\t"+cat2+"\t"+tcat+"\t"+exam+"\t"+agg);
}
}
```

}

TASK 1

1. Implement nos 10 1nd 11 in revision exercise 1 using randomly generated values
2. Consider 10 rectangular prisms as described in no 10 in revision exercise 1. Write a program that will generate, the total surface area, volume and the capacity of the prism for each prism using randomly generated values and out put them in a tabular form as shown below.

Length	**Width**	**Height**	**Total Surface area**	**Volume**	**Capacity**
……	……	……	……	……	……
……	……	……	……	……	……
……	……	……	……	……	……
……	……	……	……	……	……

Control Structures

Definition

- Control structure are programming constructs that alter the way a program behave.
- There are two major types of control structures:

1. Decision making
2. Repetitions/Loops

Decision Making

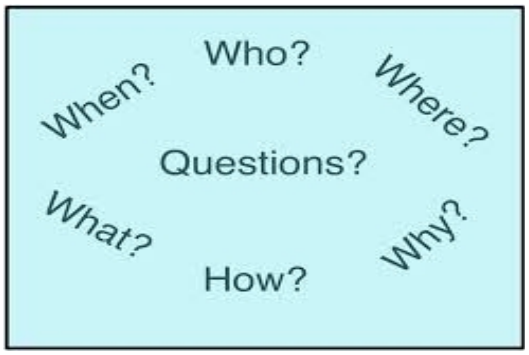
- Decision making structures makes the program control flow to change .
- It require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Programs are written to solve real life human problems. Many are the times when we make decisions and the actions we undertake are based on the decision we made.

• Life is such that we are always in a decision box. • The decisions we make are always based on a condition. • For example you want to eat Nyama Choma for every lunch but you have with you 100 shillings. What decision do you make? What do you do after the decision? • List down other situations that require you to make decisions.	

- And the decisions are about everything.
- Think about what should be answered by considering the keyword shown in the box.

Reflection

Programming help us to solve problems. It also help us to become better decision makers



1.

Similarly programming languages are equipped with structures designed to enable programs we write be able to make decisions.

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements 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.

- Most programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.
- Java programming language provides the following types of decision making statements.

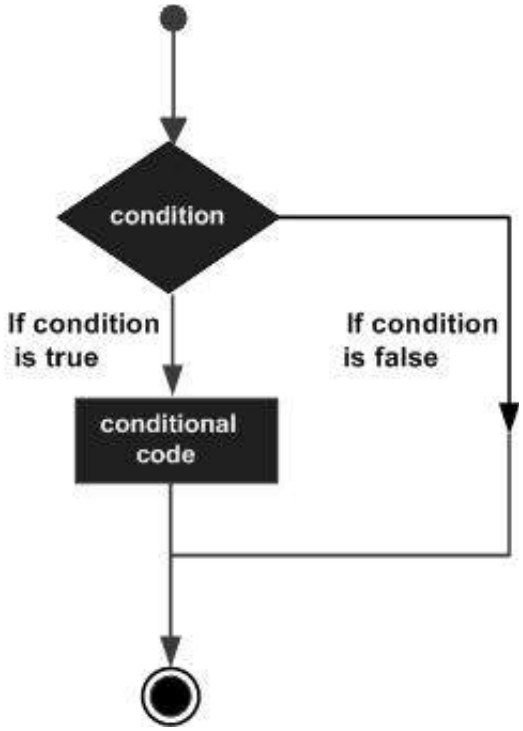
2. **if statement**
- An **if statement** consists of a boolean expression followed by one or more statements.

3. **if else statement**
- An **if else statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.

4. **Nested if else statement**
- You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

5. **switch statement**
- A **switch** statement allows a variable to be tested for equality against a list of values.

6. **Nested switch statement**
- You can use one **switch** statement inside another **switch** statement(s).



7.

Java If-else Statement

The Java *if statement* is used to test the condition. It returns *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 if statement tests the condition. It executes the if statement if condition is true.

Syntax:

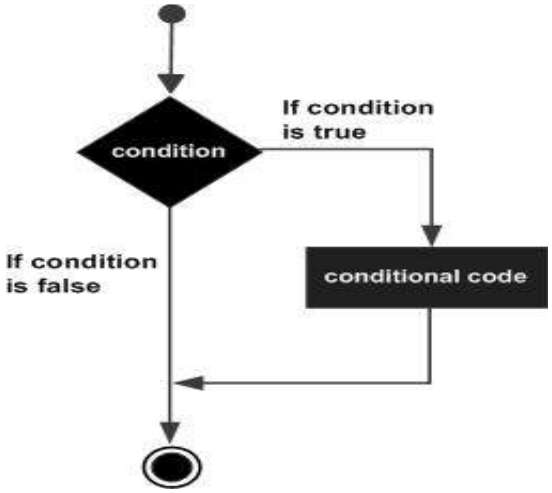
```
if(condition){
//code to be executed
}
```

Example:

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

Output:

```
Age is greater than 18
```



Java IF-else Statement

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

Syntax:

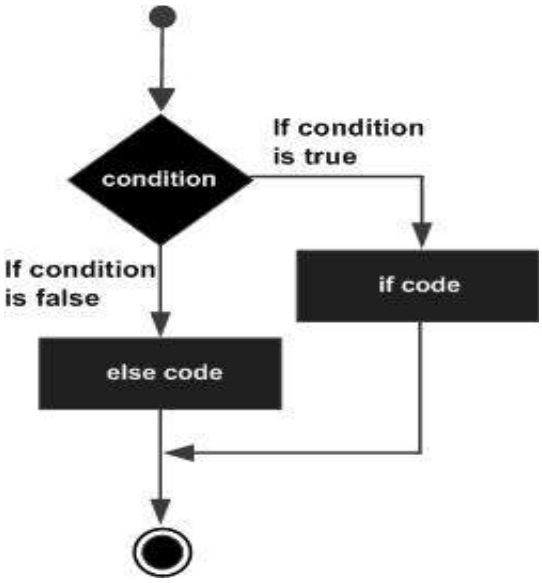
```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Example:

```
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");
    }
 }
}
```

Output:

```
odd number
```



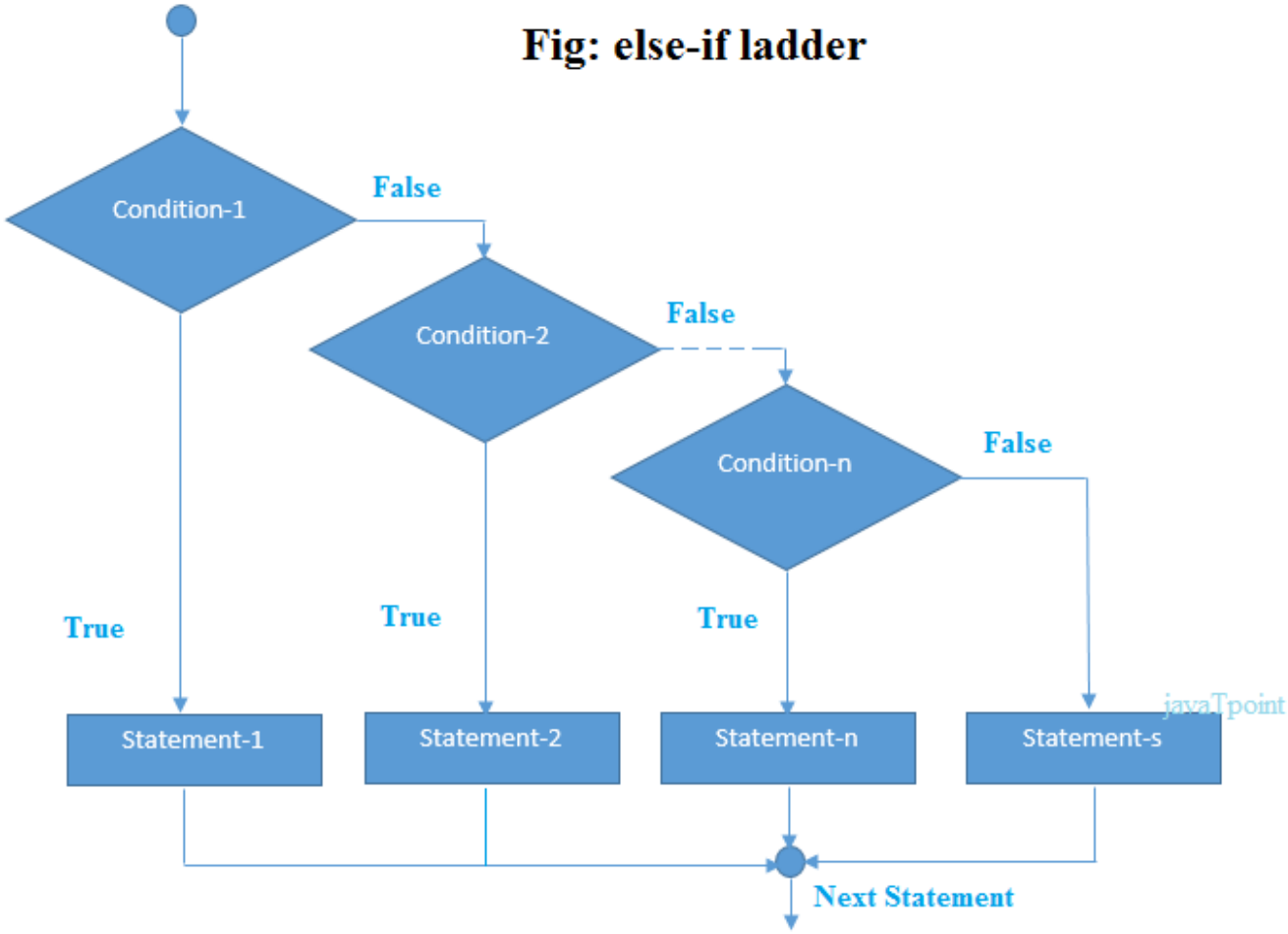
Java IF-else-if ladder Statement

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
}
```

Fig: else-if ladder




```
public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;
    if(marks<50){
System.out.println("fail");
    }
else if(marks>=50 && marks<60){
System.out.println("D grade");
 }
else if(marks>=60 && marks<70){
 System.out.println("C grade");
    }
else if(marks>=70 && marks<80){
System.out.println("B grade");
}
else if(marks>=80 && marks<90){
System.out.println("A grade");
}else if(marks>=90 && marks<100){
System.out.println("A+ grade");
}else{  System.out.println("Invalid!"); }
}
}
```

Java Switch Statement

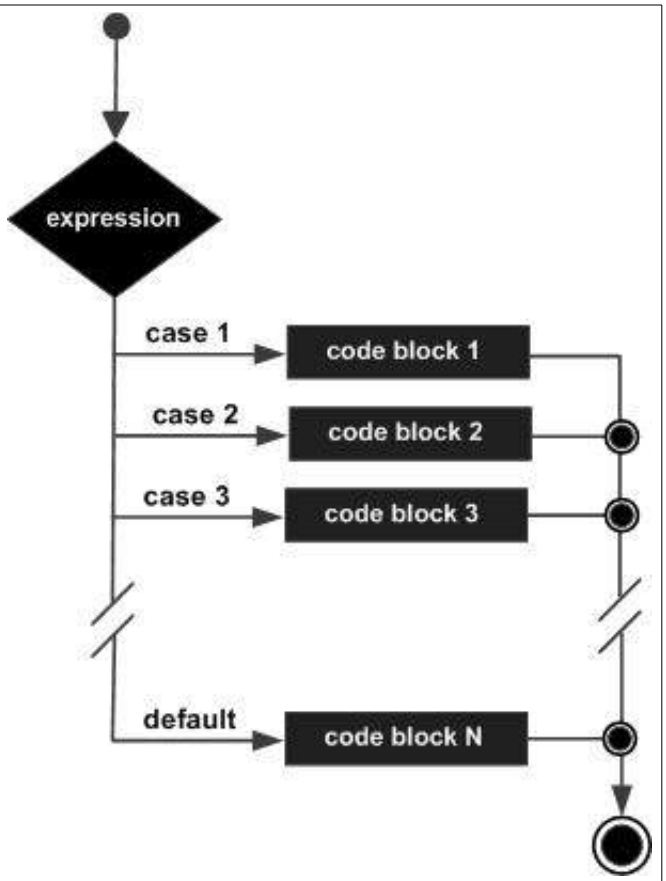
The Java *switch statement* is executes one statement from multiple conditions. It is like if-else-if ladder statement.

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;
}
```

Example:

```
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.

```
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");
 }
}
}
```

Exercise: Selection Structures

1. Draw the Flow chart that will accept an integer from the keyboard and print only if it is between 1 and 30.
   a) Write the Corresponding program of the above flowchart.
2. Draw the Flow chart that will accept an integer from the keyboard and print only if it is odd and is between 1 and 67 .
   a) Write the corresponding program of the above flowchart.
3. Draw the Flow chart that will accept an integer from the keyboard and check if it is the appropriate length of a triangle whose base length and hypotenuse length is given.(**Hint use the Pythagoras Theorem**)
   a) Write the corresponding program of the above flowchart.
4. Draw the Flow chart that will accept an integer from the keyboard and check if it is one of the six sides of a dice. (use the switch case selection)
   a) Write the corresponding program of the above flowchart.
5. Repeat question 4 above but this time use nested if else selection.
6. Every student in a University must sit for cat1 out of 10, cat2 out of 20 and exam out 70. The marks are summed together to obtain the aggregate marks. The aggregate is used to determine the student grade using a specified grading criteria. The cats , exam and aggregate marks and the grade are then printed in the format :
   CAT1 CAT2 EXAM AGGREGATE GRADE

   A tab is used as the white space.
   Draw the Flow chart of a program that can be used to receive cat1, cat2 and exam. It should ensure that they are within the required range. Once verified that they are within the range the flow chat then output result shown above.(Use the criteria used by Your College).
   a) Write the corresponding program of the above flowchart.
7. Modify the flow chart in quest 4, such that in addition, also prints a comment as shown below:
   CAT1 CAT2 EXAM AGGREGATE GRADE COMMENT
   The comment is evaluated using the criteria below:
   POOR: grade=F; FAIR: grade=D, GOOD: C<grade<A, EXECELLENT: grade=A,
   a) Write the corresponding program of the above flowchart.
8. Draw the Flow chart that will even print integer number between -20 and 51 and also their absolute sum.
   a) Write the corresponding program of the above flowchart.
9. Draw the Flow chart of a program that will even prompt you for two integer values. It should then use the two values to obtain the sum of integers between.
   a) Write the corresponding program of the above flowchart.
10. Draw the Flow chart of a program that will print the integers 1 to 148 in columns of 10.
    a) Write the corresponding program of the above flowchart.
11. Repeat 8 for columns of 5, 7 and 8 respectively.
12. Draw the Flow chart of a program that will prompt you for an integer. It should accept is only is it unsigned integer. It the print all the integers and their sum from 0 to the integer
    a) Write the corresponding program of the above flowchart.

Java For Loop

Many are the times when we repeat a certain set of actions until we have met the required condition to do otherwise. Certainly such repetition are based on evaluating the condition thus making decision and the actions we undertake are based on the decision we made.

Life is such that we are always in a decision box.

The decisions we make are always based on a condition.

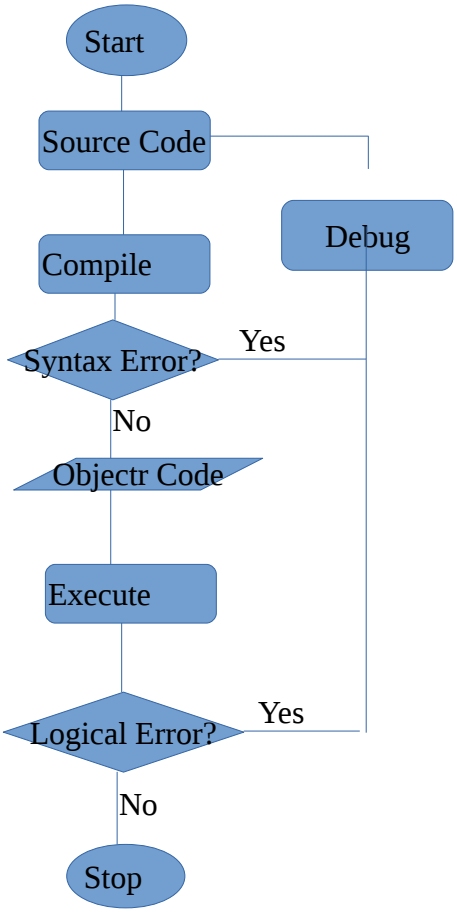
For example Consider the case of a soccer training for the prayer to acquire a certain skill.

You will be required to repeat a particular set of actions aimed at getting the skills until you get it.

Consider also the case of debugging a program you are writing. You will continue debugging until the program compile.

Think of other situations in real life that require you to repeat a set of actions. (***Consider what you every day from the time you start your day to the time you go to bed***)

Task
Suppose you wanted to know how many times you have repeated debugging syntax errors and also logical errors. What modifications can you make to the flow chart above?



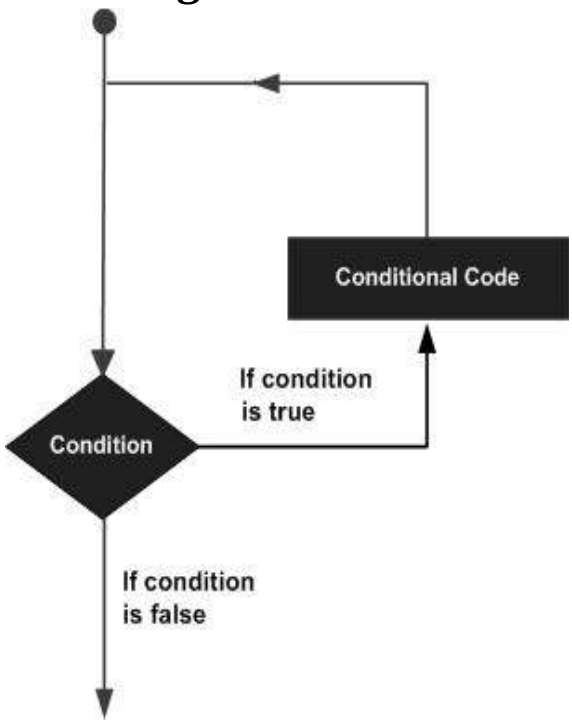
In programming there may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

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:

TASK 2
Redraw the Flow chat replacing it with actual statements.

Flow Diagram:



Most programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements:

Loops can be controlled by control statements. The change the execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

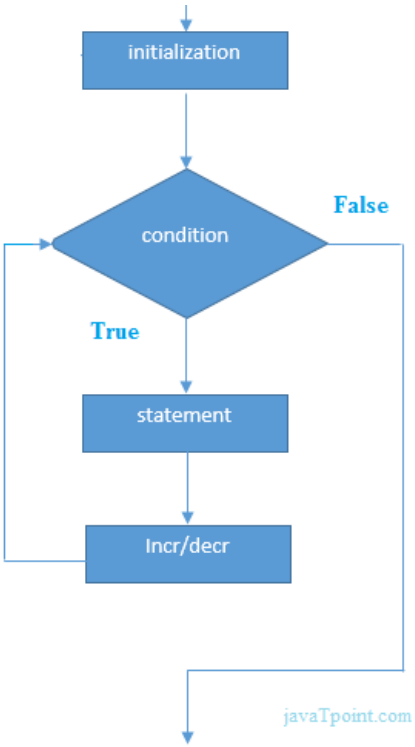
C supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

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 loop in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop



Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

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 For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array){
//code to be executed
}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
   int arr[]={12,23,44,56,78};
   for(int i:arr){
      System.out.println(i);
   }
}
}
```

Java Labeled For Loop

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

Syntax:

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

Example:

```
public class LabeledForExample {
public static void main(String[] args) {
 aa:
 for(int i=1;i<=3;i++){
 bb:
 for(int j=1;j<=3;j++){
      if(i==2&&j==2){
         break aa;
        }
      System.out.println(i+" "+j);
       }
   }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
```

If you use **break bb;**, it will break inner loop only which is the default behavior of any loop.

```
public class LabeledForExample    {
public static void main(String[]args)          {
                    aa:
          for(int   i=1;i<=3;i++){
                    bb:
                    for(int j=1;j<=3;j++){
                    if(i==2&&j==2){break     bb;}
                    System.out.println(i+" "+j);
               }
        }
    }
}
```

Output:
1 1
1 2
1 3
2 1
3 1
3 2
3 3

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.

Exercise: For Loop

1. Draw the Flow chart that will print all integers number between 1 and 30 and their sum.
   a) Write the Corresponding program of the above flowchart.
1. Draw the Flow chart that will generate 20 integers numbers between 1 and 30 and the print their sum.
   a) Write the Corresponding program of the above flowchart.
2. Draw the Flow chart that will will generate 20 integer number between 1 and 67 and print them only if they are odd and the corresponding sum.
   a) Write the corresponding program of the above flowchart.
3. Draw the Flow chart that will will print the integers 1-100 into two columns Sum and odd and the corresponding sums as shown below.

Evens	Odds
..	..
..	..
..	..

.. ..
SumEvens SumOdds

a) Write the corresponding program of the above flowchart.

4. Draw the Flow chart that will print all prime numbers between 1 and 100 and their count.
   a) Write the Corresponding program of the above flowchart.

5. Every student in a University must sit for cat1 out of 10, cat2 out of 20 and exam out 70. The marks are summed together to obtain the aggregate marks for each subject. The aggregate is used to determine the student grade for that subject using a specified grading criteria. The cats , exam and aggregate marks and the grade are then printed in the format :
   CAT1 CAT2 EXAM AGGREGATE GRADE COMMENT

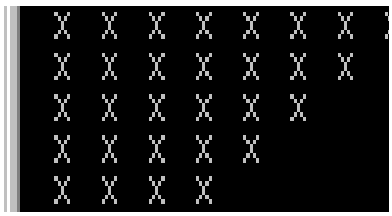
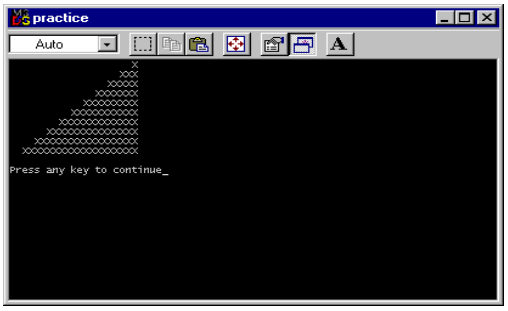
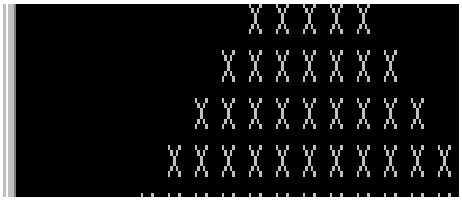
   A tab is used as the white space.
   The comment is evaluated using the criteria below:
   POOR: grade=F; FAIR: grade=D, GOOD: C<grade<A, EXECELLENT: grade=A,
   A student does a mandatory eight course units.
   Draw the Flow chart of a program that can be used to generate randomly, the number of the cat marks, exam marks for each subject and the processes them to display the output as shown .
   (Use the criteria used by Your College).
   a) Write the corresponding program of the above flowchart but the marks should be **randomly generated.**

   b) Modify the flow chart and hence the program to display the average aggregate marks and the average grade.
6. Draw the Flow chart that will print integer number between -20 and 51 and their absolute sum.
   a) Write the corresponding program of the above flowchart.
7. Draw the Flow chart of a program that will even prompt you for two integer values. It should then use the two values to obtain the sum of integers between.
   a) Write the corresponding program of the above flowchart.
8. Draw the Flow chart of a program that will print the integers 1 to 148 in columns of 10.
   a) Write the corresponding program of the above flowchart.
9. Repeat 8 for columns of 5, 7 and 8 respectively.
10. A program is required that can be able to keep count of the frequency of alphabet (Either A/a, B/b,C/c, D/d, E/e) as they are entered at the keyboard. Write a flow chart algorithm that can accomplish this task.
    Write a menu based program that can repeatedly prompt the user to enter an alphabet until the user enter q(to quit). It the display the alphabet frequencies.
11. Draw the Flow chart of a program that will prompt you for an integer. It should accept it only if it is unsigned integer. It then print all the integers and their sum from 0 to the integer
    a) Write the corresponding program of the above flowchart.
12. Write a program the can receive a long text and print
    a) The number of characters.
    b) The number of words
    c) the Longest word
    d) the Number of vowels
    e) The number spaces

13. Because of security needs every user who logs into a network system must have a a password. The password is constructed from the users two first names by concatenating the first three

characters of the first name and the last two characters of the second name. The result is concatenated to the current year. Writ a program the prompts you for the first two names of 20 users and the display the names and the corresponding password the is automatically generated.

1. Write flow chat for a program that can print the multiplication table of 10 by 10. Write the program.
2. Write flow chat for a program that can print the multiplication table of any order. Write the program.
3. Write a program that will draw the shapes below. You may not mind about the background.

<table>
<tr>
<td>  </td>
<td> 
</td>
</tr>
<tr>
<td>  </td>
<td></td>
</tr>
</table>

Java While Loop

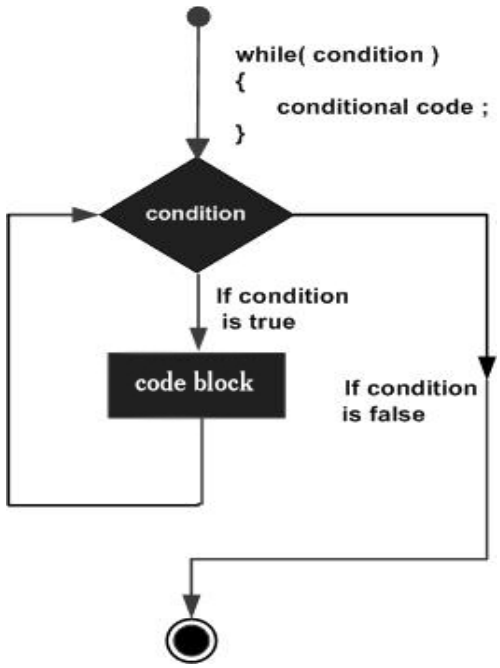
The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
1. while(condition){
2. //code to be executed
3. }
```

Example:

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



Java Infinitive While Loop

If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

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

Example:

```
public class WhileExample2 {
public static void main(String[] args) {
 while(true){ System.out.println("infinitive while loop");  }
}
}
```

Output:

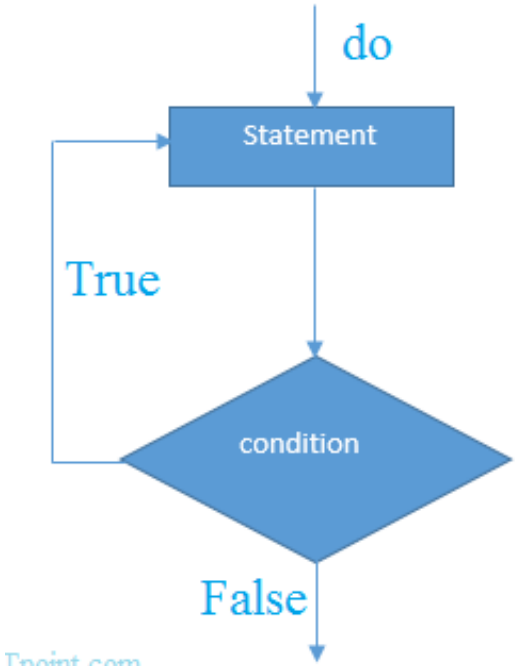
```
infinitive while loop
infinitive while loop
infinitive while loop
ctrl+c
```

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

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 while loop.

It is executed at least once because condition is checked after loop body.



Syntax:

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

Example:

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

Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

Syntax:	**Example:**

`while(true){`
`//code to be executed`
`}`
	`public class DoWhileExample2 {`
	`public static void main(String[] args) {`
	`do{ System.out.println("infinitive do while loop");`
	`}while(true);`
	`}`
	`}`

Output:

```
infinitive do while loop
infinitive do while loop
infinitive do while loop
ctrl+c
```

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

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;

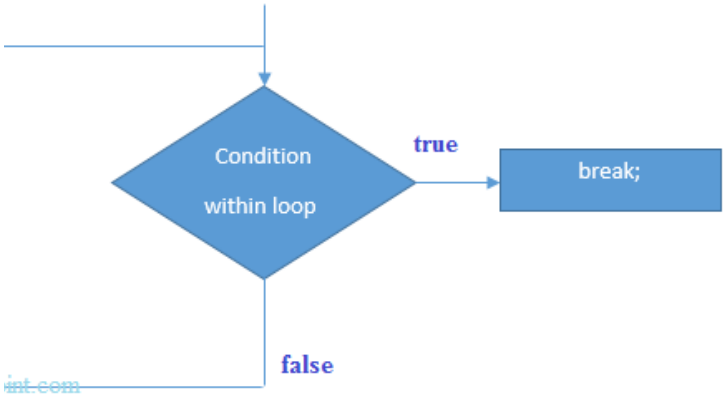


Figure: Flowchart of break statement

Break with Loop

Example:

```
public class BreakExample {
public static void main(String[] args){
 for(int i=1;i<=10;i++){
    if(i==5){ break;  }
   System.out.print(i);
 }
}
}
```

Output:1,2,3,4

Java Break Statement with Inner Loop

It breaks inner loop only if you use break statement inside the inner loop.

Example:	Output:

`public class BreakExample2 {`
`public static void main(String[] args) {`
`  for(int i=1;i<=3;i++){`
`  for(int j=1;j<=3;j++){`
`  if(i==2&&j==2){break; }`
`  System.out.println(i+" "+j);`
`  }`
`  }`
`}`
`}`
	`1 1`
	`1 2`
	`1 3`
	`2 1`
	`3 1`
	`3 2`
	`3 3`

Java Break Statement with Switch

See The Switch Statement

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:	**Example:**	Output:

1. jump-statement;
2. continue;
	public class ContinueExample {
	public static void main(String[] args) {
	for(int i=1;i<=10;i++){
	if(i==5){ continue; }
	System.out.println(i);
	}
	}
	}
		1
		2
		3
		4
		6
		7
		8
		9
		10

Continue Statement with Inner Loop

It continues inner loop only if you use continue statement inside the inner loop.

Example:	**Output:**

public class ContinueExample2 {
public static void main(String[] args) {
for(int i=1;i<=3;i++){
for(int j=1;j<=3;j++){
if(i==2&&j==2){continue;}
System.out.println(i+" "+j);
}
}
}
}
	1 1
	1 2
	1 3
	2 1
	2 3
	3 1
	3 2
	3 3

Repeat The **For loop** Exercise Using the

1. The While loop
2. The do -while loop