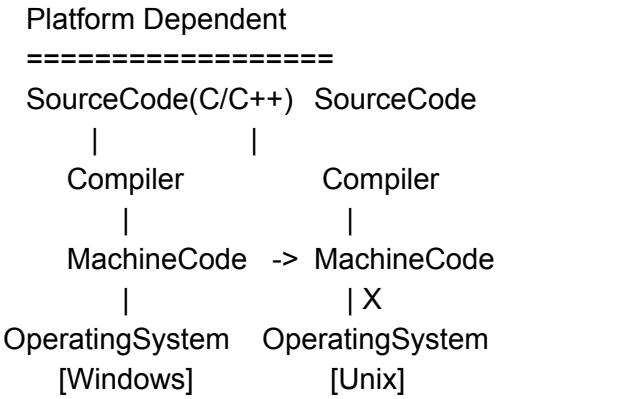
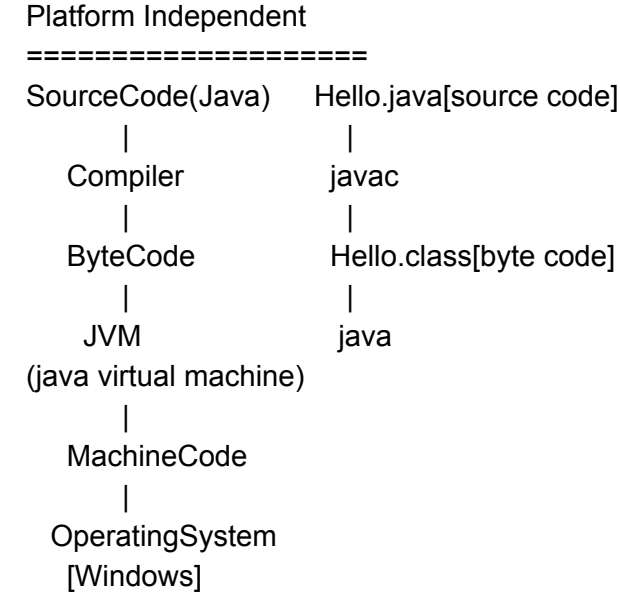
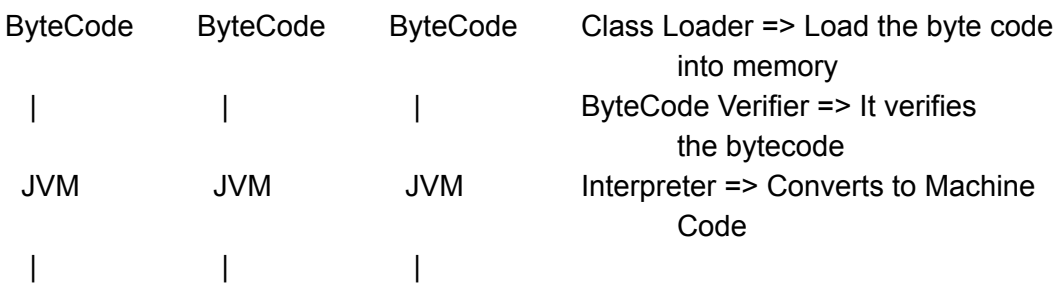
Platform: It is a set of software/hardware which facilitates to run the application.
It provides a runtime environment to execute the application.

Platform Dependent: When the application depends on its development environment for its execution, then it is called as Platform Dependent.

Platform Independent: When the application runs on any environment/platform irrespective of its development environemnt/configuration are called platform independent.







MachineCode MachineCode MachineCode
  | | |
 Unix Solaris Linux

JVM is platform dependent
ByteCode can be downloaded into any operating system for its execution.
ByteCode is Architecture Neutral.
Java is Platform Independent because of ByteCode.

So using java we can develop Platform independent applications.

Java is a simple, highlevel, secured, object oriented and platform independent programming language.

Java is WORA, WODA.
Write Once Run Anywhere
Write Once Deploy Anywhere
==========
==========

James Gosling From Sun Microsystems => 1991
1995 -> Beta version of Java
1996 -> JDK 1.0

Developed by Green Team of Sun microsystems => Green Talk -> OAK -> JAVA

Different Editions of java
==========
Java SE -> Java Standard Edition[Core Libraries of Java Prog Lang]
	[Standalone]
	[Console based application[CUI], windows/desktop applications[GUI]

Java EE -> Java Enterprise Edition
	[Web application, Enterprise/Distributed applications]
	Static Web Application => HTML
	Dynamic Web Application => Servlet, JSP[Web Component]
	EJB : Enterprise Java Bean [Business Component] => Spring Framework
==========
Java ME -> Java Micro Edition
	[Mobile applications]

Different Versions of Java
==========
jdk => java development kit

jdk alpha & beta => 1995
jdk 1.0 => 1996 [LTS]
jdk 1.1 => 1997
j2se 1.2 => 1998 [LTS]
j2se 1.3 => 2000
j2se 1.4 => 2002
j2se 5.0 => 2004 [LTS]
JavaSE 6 => 2006
JavaSE 7 => 2011
JavaSE 8 => 2014 [LTS]
JavaSE 9 => 2017

JavaSE 10 => 2018 -> March
JavaSE 11 => 2018 -> September [LTS]

JavaSE 12 => 2019 -> March
JavaSE 13 => 2019 -> September

JavaSE 14 => 2020 - March
JavaSE 15 => 2020 -> September

JavaSE 16 => 2021 - March
JavaSE 17 => 2021 - September [LTS]

JavaSE 18 => 2022 - March
JavaSE 19 => 2022 - September

JavaSE 20 => 2023 - March
JavaSE 21 => 2023 - September

JavaSE 22 => 2024 - March
JavaSE 23 => 2024 - September

==========
JamesGosling developed a new language which was initially named as OAK.

OAK was already registered by another company, so then renamed it to JAVA.

The Green Team of Sun Microsystems who developed Java, they consumed a lot of coffee

during the development of Java, so they thought of Coffee has a great contibution towards the development of JAVA, so they gave the Hot Coffee Cup with a Saucer as the symbol of java.

jdk -> Java Development Kit
JDK is required to design, develop, test and run a java application.
It is required for a Java Developer.

jre -> Java Runtime Environment
JRE is used to run a java application on the client system.

How to set the PATH
==========
There are two ways to set the PATH.
i. Temporarily
ii. Permanently

Temporarily setting the PATH:
Open the command prompt and type the following command:
SET PATH="C:\Program Files\Java\jdk1.8.0_221\bin"

It will work only for the active window.
Once the window gets closed, it needs to reset again.

Permanently setting the PATH:
i. Right click on MyComputer/ThisPC.
ii. Click on Properties.
iii.Click on Advanced System Settings.
iv. Click on Environment Variables.
vi. Click on new if there is no such variable as PATH,otherwise click on Edit.
variable name = PATH
variable value = C:\Program Files\Java\jdk1.8.0_221\bin
vii. Click on OK.
==========
First open any editor(Notepad, EditPlus etc).

Write the source code and save the file with .java as extension.

To compile: javac <FileName.java>
To execute: java <ClassName>

Bytecode will be generated with the name of Class not with the name of file.

If the class is a non-public class then the file can be saved with any name.Filename and

Classname need not to be same.

If the classname and filename are different:
To compile : javac <FileName.java>
To execute : java <ClassName>

If the class is a public class, then the name of the file must be same as public class name.

In a single source file, there can be only one public class.
==========
Token
=====
A smallest unit of an instruction is called as a Token.

int x = 10 ;

int => datatype/keyword
x => variable
= => operator
10 => constant
; => spl. char

Variable
========
Variable is a data item whose value keeps on changing from time to time during program execution.

It is a placeholder/container which can hold some content.

It is a named memory location where we can store and manipulate the values.

Rules to name a variable
==========
i. Allowed characters are a-z,A-Z,0-9,_(underscore) and $(dollar).
ii. First character should not be a digit. It can start with alphabet or special character.
iii. It should not be a keyword.
  [keyword: it is a reserved word. It's meaning can't be changed, and it can't be used as a variable.Every language have a set of keyword. Java has 53 keywords.]
iv. Spaces are not allowed during naming a variable. loanAmount
v. It is case-sensitive. loan, Loan, LOAN

Recommendation
==========
i. It should be a meaningful name.

x=10	rollNumber=10
y=20	strength=20
z=30	marks=30
	numberOfStudents

ii. It should follow camelCase.

ClassName => class
noOfStudent => variable
getNoOfStudents() => method
ems => package
MAXVALUE => constant

mark$1
_mark1
1stMark[invalid]
case[invalid]
While
while[invalid]
loan, Loan, LOAN [treated as 3 diff. variables]
loan amount[invalid]
loan_amount, loanAmount,loan$amount
x, y, z
rollNumber
noOfStudents
no_of_students
_1stMark

Declaration of a variable
==========
Syntax: datatype variableName;

Datatype: It describes the type of data/content which can be stored into a variable.

variable => container

datatype => typeofcontent

int intValue;

There are two types of datatypes:
i. Primitive datatype [Value Type]
ii. Non-primitive datatype [Reference Type]

Primitive datatype
==========
There are 8 primitive data types.
i. byte
ii. short
iii.int
iv. long

v. float
vi. double

vii. char

viii.boolean

Signed [Both +ve and -ve values]
Unsigned [Only +ve values]

byte byteValue; 1 byte[8 bits]
Min value: -128 => -2^7
Max value: +127 => 2^7-1

short shortValue; 2 bytes[16 bits]
Min value: -32768 => -2^15
Max value: +32767 => 2^15-1

int intValue; 4 bytes[32 bits]
Min value: -2147483648 => -2^31
Max value: +2147483647 => 2^31-1

long longValue; 8 bytes[64 bits]
Min value: -2^63
Max value: 2^63-1

MSB
Max value : [0] [1][1][1][1][1][1][1] => 127
64 32 16 8 4 2 1 => +127

Min value: [1] [0][0][0][0][0][0][0] => -128
1's comp => [1][1][1][1][1][1][1]
+[0][0][0][0][0][0][1]
==========
2's comp =>[1][0][0][0][0][0][0][0]=> -128

==========

float floatValue; 4 bytes[32 bits]
Accuracy upto 6 decimal places
-3.4e38 to 3.4e38
-3.4*10^38 to +3.4*10^38

double doubleValue; 8 bytes[64 bits]
Accuracy upto 15 decimal places
-1.7e308 to +1.7e308

IEEE Floating Point representaion

==========
-32768 to +32767

char charValue; 2 bytes[16 bits][unsigned]
Min value: [00000000 00000000] => 0
Max value: [11111111 11111111] => 65535 [2^16 -1]

Min value : [00000000 00000000]=0
Max value : [11111111 11111111] [2^16]-1=>65536-1=65535
==========

-128 to +127

[00000000] => 0
[11111111]
128+64+32+16+8+4+2+1=255 => [2^8]-1 => 255

char ch='z'; [0 - 255] [c/c++] -> 1 byte
char ch; [0 - 65535] [java] => 2 byte

Characters are internally treated as unsigned integers.
Every character is associated with a integer value, which is called as UNICODE.

Char	ASCII	UNICODE
'\0'	0	0
-	-	-
'0'	48	48
'9'	57	57
'A'	65	65
'Z'	90	90
'a'	97	97
'z'	122	122
	-	-
	255	255
		-
		65535

boolean booleanValue; NA[JVM Dependent]
1 bit[which represents true or false]
Possible values are true and false

There are 53 keywords in Java
==========
Among 53, 3 are Literals(constants): true, false, null

50 keywords
^

2(unused)	48(used)				
goto	byte	if	public	try	class
const	short	else	private	catch	interface
	int	switch	protected	throw	enum
	long	case	static	throws	extends
	float	default	abstract	finally	implements
	double	break	final	assert	import
	char	while	transient		package
	boolean		do	native	
	void	for	synchronized		this
		continue	strictfp		super
		return	volatile	new	

instanceof

Constants:
=========
It is a data item whose value does not change during program execution. It is always fixed.

Integral Constants
==================
Binary Constant
Octal Constant
Decimal Constant
Hexadecimal Constant

Binary Constant:
===============
It starts with 0b or 0B.
It consists of only 0's and 1's.
ex: 0b101
0B100
0b12[invalid]

Octal Constant:
===============
It starts with 0(Zero).
It consists of digits in the range of 0 to 7.
ex: 0123
0128[invalid]

Decimal constant:
================
It consists of digits in the range of 0 to 9.
But it should not starts with 0(zero).
ex: 123
210
013[invalid] not a decimal constant
018[invalid] neither it is a decimal constant nor it is octal constant

Hexadecimal constant:
====================
It starts with 0x or 0X.
It consists of digits in the range of 0 to 9 and a-f or A-F.
ex: 0x12FD
0XFACE

0xBEEF
0xBEAR[invalid]

int binValue = 0b101; => (2^2*1)+(2^1*0)+(2^0*1)
int octValue = 012; => (8^1*1)+(8^0*2)
int decValue = 10;
int hexValue = 0x1A; => (16^1*1)+(16^0*A)
int result = binValue+octValue+decValue+hexValue;
System.out.println(result); 51

Real constants
==========
By default all the real numbers are treated as double constant.

So inorder to make a real number as float, it needs to be suffix with f or F.

Inorder to store real number either we use float or double datatype.

float floatValue = 1.0d; [invalid]
float floatValue = 1.0f/1.0F; [valid]

double doubleValue = 1.0; [valid]
double doubleValue = 1.0d; [valid]
[d/D is optional for double constant]

Character constant
==========
Any single character enclosed in single quote can be considered as a character constant.

char charValue = 'x';
char value = '9';

char val = 65;
char val = 'A';
char val = '\u0041';

char ch = 65536;[invalid]

char val='\u0000'; or char val = 0;
char val='\uffff'; or char val = 65535;

char val = 'xy'; [invalid]

Escape sequence characters:
char val = '\n';
char val = '\b';
char val = '\t';
char val = '\'';
char val = '\"';
char val = '\\';
"Hello"
System.out.println("\"Hello\"");

boolean constants
==========
It is a constant which represnts true or false only.

boolean flag = true;
boolean flag = false;

boolean flag = True;[invalid]
boolean flag = "true"; [invalid]
boolean flag = 10; [invalid]

Default values of datatypes
==========

Datatype	DefaultValue	Memory
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long	0L	8 bytes
float	0.0f/F	4 bytes
double	0.0d/D	8 bytes
char	'\u0000'	2 bytes
boolean	false	NA/1 bit

==========

Operators
==========
Operators are used to perform some operations on the operands.

Types of Operators
==========
Mainly it is classified into 3 categories.
i. Unary
ii. Binary
iii. Ternary

Unary: Operator which works on a single operand.
Binary: Operator which works on two operands.
Ternary: Operator which works on three operands/expressions.

Increment/Decrement Operator
==========

PostIncrement	PreIncrement
int x = 10;	int x = 10;
x++;	++x;
System.out.println(x);	System.out.println(x);
Output:11	Output:11

Increment operator increments the value of the operand by one.

int x = 10;	int x = 10;
int y = x++;	int y = ++x;
System.out.println(x);	System.out.println(x);
System.out.println(y);	System.out.println(y);
y = x++;	y = ++x;
[or]	[or]
y = x;	x = x+1;
x = x+1;	y = x;
x=11 y=10	x=11 y=11

```
int x = 10;
System.out.println(x++);  10
System.out.println(x);    11

int x = 10;
System.out.println(++x);  11
```

```
int x = 5;
```

```
int y = 3;
int z = x++ + ++x + x++ + --x + y-- + --y;
System.out.println(x + " " + y + " " + z);

==================
Arithmetic Operator[ +, -, *, /, % ]
==================
int x = 10;
int y = 20;
int z = x+y;
System.out.println(z);

[ / operator returns quotient,
whereas % operator returns remainder.]

System.out.println(10/3);  3
System.out.println(10%3);  1
System.out.println((double)(10/3)); 3.000
System.out.println(10/3.0); 3.33333
System.out.println(10.0/3); 3.33333
System.out.println( (double)10/3); 3.3333
System.out.println(10/(double)3); 3.3333

byte + byte = int => max(int, byte, byte)
byte + short = int
byte + int = int
byte + long = long => max(int, byte, long)

short + short = int
short + int = int
short + long = long

int + int = int
int + long = long

long + long = long

int + float = float
int + double = double => max(int, int, double)

x (ArithmeticOperator) y => max(int, type(x), type(y) )
```

```
byte b = 10;
//b = b + 1; =>max(int,byte,int) //CE
b = (byte)(b+1);
System.out.println(b);

byte b = 10;
b++; => b=(byte)(b+1);
System.out.println(b); 11

byte b = 10;
b += 1; => b=(byte)(b+1);
System.out.println(b);
---------------------------------

b += 10 =>[b = b + 10;]
b -= 10 => b = b - 10;
b *= 10 => b = b * 10;
b /= 10 => b = b / 10;
b %= 10 => b = b % 10;
```

Write a program to find the sum of the digits of a given two digit number.

```
Input : 45      98
Output : 9      17
```

Write a program to reverse any two digit number.

```
Input : 46      93
Output : 64     39

int x =  4   => 00000000 00000000 00000000 00000100
int x = -4   => 11111111 11111111 11111111 11111011
                +                                 1
                ==================================
2's complement          11111111 11111111 11111111 11111100
```

Relational Operators

==========

Relational operators are used to compare the values of two operands and returns a boolean value either true or false.

< [less than]

> [greater than]

<= [less than or equal to]
>= [greater than or equal to]
== [equals to]
!= [not equals to]

```
int x = 10;
int y = 20;
System.out.println("x < y?" + (x<y));
System.out.println("x > y?" + (x>y));
System.out.println("x <= y?" + (x<=y));
System.out.println("x >= y?" + (x>=y));
System.out.println("x == y?" + (x==y));
System.out.println("x != y?" + (x!=y));
```

Assignment Operator
==========
=, +=, -=, *=, /=, %=

= [equals operator assigns a value to an operand]

```
int x = 10;
x=x+1; => x++ => x+=1

x=x-5;  => x-=5;
x=x*10; => x*=10;
x=x/5;  => x/=5;
x=x%5;  => x%=5;
```

Write a program to check the given 3 digit number is a pallindrome or not.
==========

```
(expression)?statement1 : statement2;

  if(expression==true)
     statement1;
  else
     statement2;

int x = 10;
System.out.println(x=-10?"Hello":"Hi");
System.out.println(-10?"Hello":"Hi");CE
```

```
int x = 10;
System.out.println(x==10?"hello":"hi");

boolean flag = true;
System.out.println(flag=false?"Hello":"Hi");
System.out.println(false?"Hello":"Hi");

boolean flag = true/false;
System.out.println(flag==true?"Hello":"Hi");
System.out.println(flag?"Hello":"Hi");

boolean flag = true/false;
System.out.println(flag == false?"Hello":"Hi");
System.out.println(!flag?"Hello":"Hi");
```

Logical Operator(Short-Circuit Operator)
==========
&& [AND]
|| [OR]
! [NOT]

Logical operators are used to make complex expression by combining more than one simple expression.

Logical operators are applicable only on boolean expressions.

They can't be used on any other type other than boolean values.

Logical AND (&&)
==========
true && true => true true && false => false
false && false => false false && true => false

Logical OR (||)
==========
true || true => true true || false => true
false || false => false false || true => true

Logical NOT (!)
==========
!true => false
!false => true

5 && 7 [invalid]
5 || 7 [invalid]
!5 [invalid]

```
int x=5,y=10,z=15;

if(x++ == 5 && y++ == 10 && z++ == 15)
  System.out.println("Hello");
else
  System.out.println("Hi");

System.out.println(x + "" + y + "" +z);
```

==========

```
if(++x == 5 && y++ == 10 && z++ == 15)
  System.out.println("Hello");
else
  System.out.println("Hi");

System.out.println(x + "" + y + "" +z);
```

==========

```
 if(x++ == 5 || y++ == 10 || z++ == 15)
  System.out.println("Hello");
else
  System.out.println("Hi");

System.out.println(x + "" + y + "" +z);
```

==========

```
if(++x == 5 || y++ == 10 || z++ == 15)
  System.out.println("Hello");
else
  System.out.println("Hi");

System.out.println(x + "" + y + "" +z);
```

==========

Write a program to check the given number is a three digit number or not.

```
int num = 12;
if(!(num >= 100 && num <= 999))
  System.out.println("Not a  three digit number");
else
 System.out.println("It is a three digit number");
```

==========

Input : 035 124 2345 23756 23467854
Output : 100 200 2400 23800 23467900

Input : 15 10 9 17
Output : false true true false
==========
If the given number is one or two less than the multiples of 10 then display "true" otherwise display "false".

Input: 18 29 8 58 27 36
Output: true true true true false false
==========

Conditional Operator(?:)[Ternary Operator]
==========
Conditional operator replaces if else statement.
  [expression]?statement1:statement2;
If the expression returns true, statement1 will execute otherwise statement2 will execute.

Bitwise Operator
==========
AND - &
OR - |
XOR - ^

Complement - ~

Left Shift - <<

Right Shift- >>

Unsigned
Right Shift
/ - >>>
Zero Filled
Right Shift

Bitwise AND(&) and OR(|) is applicable on both boolean and integral expressions.

In a complex expresion, if Bitwise AND(&) or OR(|) is used then all the expressions will get an opportunity to be evaluated irrespective of true or false.

```
true & true => true                5 & 4 => 4
true & false => false
false & false => false
false & true => false

true | true => true                5 | 4 => 5
true | false => true
false | true => true
false | false => false

  1 0 1
& 1 0 0
=======
  1 0 0[4]

   1 0 1
|  1 0 0
========
   1 0 1[5]

int x = 5;
int y = 10;
int z = 15;

//if(++x == 5 && y++ == 10 && z++ == 15)
if(++x == 5 & y++ == 10 & z++ == 15)
//if(x++ == 5 || y++ == 10 || z++ == 15)
//if(x++ == 5 | y++ == 10 | z++ == 15)

  System.out.println("Hello");
else
  System.out.println("Hi");
System.out.println(x + "" + y + "" +z);

true ^ true => false               5 ^ 4 => 1
true ^ false => true
false ^ true => true
false ^ false => false

   1 0 1
^  1 0 0
========
   0 0 1
```

Complement (~) : It returns the complement of the given number.
It is applicable only on integral values not on boolean values.

~5 [valid] ~true [invalid]
!5 [invalid] !true [valid]

5 => 00000000 00000000 00000000 00000101
~5 => 11111111 11111111 11111111 11111010
1's com => 0000000 00000000 00000000 00000101
  + 1
  ==========
2's com => 0000000 00000000 00000000 00000110 => -6

0 1 0 1
1 0 1 0
 1 0 1 + 1 = 110 =-6

Left Shift(<<)
==========
int x = 4;
System.out.println(x<<2); //16

4 00000000 00000000 00000000 00000100
4<<2 00000000 00000000 00000000 00010000

x<<y => x * 2^y
4<<2 => 4 * 2^2 => 4*4 = 16

Right Shift(>>)[Signed Right Shift]
==========
int x = 8;
System.out.println(x>>2);

8 00000000 00000000 00000000 00001000
8>>2 00000000 00000000 00000000 00000010

x>>y => x / 2^y
8>>2 => 8 / 2^2 => 8/4 = 2

Unsigned Right Shift(>>>)
Zero filled Right Shift
==========

```
int x = 8;
System.out.println(x>>>2);

8      00000000 00000000 00000000 00001000
8>>>2  00000000 00000000 00000000 00000010

 8>>2 ==  8>>>2
-8>>2 != -8>>>2
==========================================

int x=-8;
System.out.println(x<<2);

8          00000000 00000000 00000000 00001000
           11111111 11111111 11111111 11110111
      +                                 1
           ==================================
-8         11111111 11111111 11111111 11111000

-8<<2   11111111 11111111 11111111 11100000
         0000000 00000000 00000000 00011111
        +                          1
            ================================
            0000000 00000000 00000000 00100000=>32
           [-32]
==============================================
-8         11111111 11111111 11111111 11111000
-8>>2   11111111 11111111 11111111 11111110
System.out.println(-8>>2);
==============================================

-8         11111111 11111111 11111111 11111000
-8>>>2  00111111 11111111 11111111 11111110
System.out.println(-8>>>2);
==============================================

Implicit TypeCast
=================
i. Converting from Lower type to Higher type is called Implicit typecast.
ii. This conversion is done by compiler automatically.
iii. This is also called as widening.
iv. There is no loss/overflow of data during implicit typecast.
           byte byteValue = 127;
```

```
int intValue = byteValue; [Implicit typecast]

byte=>short=>int=>long=>float=>double
char=>int=>long=>float=>double
```

Explicit TypeCast
==========
i. Converting from Higher type to Lower type is called Explicit typecast.
ii. This conversion has to be done by programmer explicitly.
iii. This is also called as Narrowing.
iv. There might be loss and/or overflow of data during explicit typecast.

```
int intValue= 127;
byte byteValue = intValue;=>CE

int intValue= 127;
byte byteValue = (byte)intValue;=>127

int intValue =128;
byte byteValue = (byte)intValue;
//Data overflow          [-128]

double doubleValue = 125.46;
byte byteValue = (byte)doubleValue;
//Loss of data  [125]

double doubleValue = 129.345;
byte byteValue = (byte)doubleValue;[-127]
//Overflow and loss of data.

double=>float=>long=>int=>short=>byte
double=>float=>long=>int=>char

long longValue=10L;
int intValue=(int)longValue;
short shortValue=(short)longValue;

int intValue=(int)0L;
```

Conditional Statement
==========
Whenever a statement or a group of statements executes based on a given condition, those statements are called as Conditional Statements.

Conditional statements can be written using,
i. if
ii.switch-case

if
==
"if" can be written in 4 different ways.
i. simple if
ii. if else
iii.else if ladder
iv. nested if

simple if
=========
syntax:

```
if<condition> {
    statement1;
    statement2;
    statement3;
}
```

When the given condition evaluates to true, then the statements 1,2,and3 will execute, otherwise nothing will be there to execute.

if else
=======
syntax:

```
if<condition>{
  statement1;
  statement2;
}else{
  statement3;
  statement4;
}
```

When the given condition evalulates to true, then the first block will execute otherwise the else block will execute.

```
if<condition>
  statement1;
//statement2;
else
```

```
  statement3;
statement4;
```

Find biggest among 3 given numbers.

==========

```
int n1=10, n2=30, n3=20;

if(n1 > n2 && n1 > n3 && n1 > n4)
  System.out.println("n1 is big");
else if(n2 > n3 && n2 > n4)
  System.out.println("n2 is big");
else if(n3 > n4)
  System.out.println("n3 is big");
else
  System.out.println("n4 is big");
```

==========

```
if(n1>n2)
  if(n1>n3)
    if(n1>n4)
      System.out.println("n1 is big");
    else
      System.out.println("n4 is big");
  else
    if(n3 > n4)
      System.out.println("n3 is big");
    else
      System.out.println("n4 is big");
else
  if(n2>n3)
    if(n2>n4)
      System.out.println("n2 is big");
    else
      System.out.println("n4 is big");
  else
    if(n3>n4)
      System.out.println("n3 is big");
    else
      System.out.println("n4 is big");
```

==========

```
int n1=10, n2=20, n3=25, n4= 22;
int big = n1;
if(n2 > big)
  big = n2;
```

```
if(n3 > big)
  big = n3;
if(n4 > big)
  big = n4;
System.out.println("Big="+big);
```

else if ladder
==========
syntax:

```
if<condition1>
  statement1;
else if<condition2>
  statement2;
else if<condition3>
  statement3;
else
  statementn;
```

nested if
=========
syntax:

```
if<condition1>
  if<condition2>
    statement1;
  else
    statement2;
else
  if<condition3>
    statement3;
  else
    statement4;
```

==========
Write a program to find biggest among 3 given numbers.
==========
switch-case
==========
It is a multibranching statement like else-if ladder.
Every case is associated with a statement, which ever case becomes true, the corresponding statement will be the output.If none of the cases are true, then default statement will be the output.

```
switch(expression)
{
```

```
        case constant1:                 if(expression==constant1)
                statement1;             statement1;
                break;
        case constant2:                 else if(expression==constant2)
                statement2;             statement2;
                break;
        case constant3:                 else if(expression==constant3)
                statement3;             statement3;
                break;
        default:                else
                statementn;             statementn;
                break;
}
```

==========

i. Curly braces are mandatory in switch case.
ii.case,default, and break are not mandatory.
   switch(expression) { }
iii.default can be anywhere within switch-case.
iv. Allowed datatypes in expression are:
   byte,short,int,char[Since JDK 1.0]
   Byte,Short,Integer,Character,Enum [Since jdk1.5]
   String [Since jdk1.7]
v. Constants in the cases should not be duplicate.
vi. Constants should be in the range of expression type.
vii. Constants can't be a variable, it must be a compile time constant.
viii. All the statements in switch-case must be inside case or default.

==========

```
switch(expression){ } //valid switch-case

switch(expression) //invalid switch-case
{
  System.out.println("switch-case demo");
}

long value=10L;
switch(value) //invalid
{
}

byte value=127;
switch(value+1)
```

```
{
          case 65:
            System.out.println("A");
            break;
          case 97:
            System.out.println("a");
            break;
          case 90:
            System.out.println("Z");
            break;
          case 128:
            System.out.println("128");
            break;
       default:
         System.out.println("Some character");
         break;
           case 'A':
            System.out.println("A");
            break;
}
==========================================
int value=30;
//int c1=10,c2=20,c3=30;
final int c1=10,c2=20,c3=30;
switch(value)
{
          case c1:
            System.out.println("TEN");
            break;
          case c2:
            System.out.println("TWENTY");
            break;
          case c3:
            System.out.println("THIRTY");
            break;
          default:
            System.out.println("NOT KNOWN");
            break;
}

==========================================

nextByte()
```

nextShort()
nextInt()
nextLong()
nextFloat()
nextDouble()
nextBoolean()
next()
nextLine()

==========
Iterative Statement[Loop]
==========
Executing one or more statements repeatedly for a given period of time or until certain condition fulfilled, is called as Loop.

Loop can be constructed using:
i. while
ii. do-while
iii. for

while
=====
syntax: variable initialization;
```
        while<condition>
        {
                statement1;
                statement2;
                increment/decrement statement;
        }

        int i=1;
        while(i <= 10)
        {
                System.out.println(i);
                i++;
        }
```

==========
```
        int i=1;
        while(i <= 10)
          System.out.println(i);
          i++;
```

```
=====================================
        int i=1;
        while(i <= 10)
          i++;
        System.out.println(i);
=====================================
        int i=1;
        while(++i <= 10);
        System.out.println(i);
=====================================
        int i=1;
        while(i++ <= 10);
        System.out.println(i);
=====================================
        int x=10;
        int y=20;
        while(x<y)
          System.out.println("Hello");
        System.out.println("Hi");
=====================================
        while(10<20)
          System.out.println("Hello");
        System.out.println("Hi"); //unreachable code
=====================================
        final int x=10;
        final int y=20;
        while(x<y)
          System.out.println("Hello");
        System.out.println("Hi");
=====================================
        while(10>20)
           System.out.println("Hello");//unreachable code
        System.out.println("Hi");
=====================================
        int x=10;
        int y=20;
        int z=0;
        if(x<y)
          z=x;
        System.out.println(z);
=============================================================
Write a program to find the sum of first 10 natural numbers.
=============================================================
```

```
class Sum
{
  public static void main(String[] args)
  {
          int i=1;
          int sum=0;

          while(i <= 10)
          {
            sum=sum+i;
            i++;
          }

          System.out.println("Sum = " + sum);
  }
}
```

==========

```
class Factorial
{
  public static void main(String[] args)
  {
          int i=2,num=5;
          int fact=1;
          while(i <= num)
          {
            fact=fact*i;
            i++;
          }
          System.out.println("Factorial of " +num +" = " + fact);
  }
}
```

==========

Sum of the digits of any given number.

==========

```
int sum=0;
int num=1234;

while(num!=0)
{
  int rem=num%10;
  sum=sum+rem;
```

```
  num=num/10;
}
System.out.println("Sum of the digits = " + sum);
```

==========

Reverse of any given number.

==========

```
int rev=0;
int num=1234;

while(num!=0)
{
  int rem=num%10;
  rev= rev*10+rem;
  num=num/10;
}
System.out.println("Reverse = " + rev);
```

==========

Write a program to check the given number is a pallindrome or not

==========

```
int rev=0;
int num=1234;
int temp=num;

while(num!=0)
{
  int rem=num%10;
  rev= rev*10+rem;
  num=num/10;
}

if(temp==rev)
  System.out.println("It is pallindrome");
else
  System.out.println("It is not pallindrome");
```

==========

```
int num=6;
int sum=0;       int sum=1;
int i=1; int i=2;

while(i <= num/2)
{
```

```
        if(num%i == 0)
          sum=sum+i;
        i++;
}
if(sum==num)
  System.out.println("It is a perfect number");
else
  System.out.println("It is not a perfect number");
```

==========

```
int num=7;
int i=1,count=0;

while(i<=num)
{
   if(num%i == 0)
        count++;
   i++;
}
if(count==2)
  System.out.println("It is a prime number");
else
  System.out.println("It is not a prime number");
```

==========

```
int num=37;
int i=2;
boolean isPrime=true;          //n iterations
while(i<num)                   //n-2 iterations
//while(i<=num/2)              //n/2-1 iterations
//while(i<=Math.sqrt(num))     //sqrt(n)-1 iterations
{
        if(num%i == 0)
        {
          System.out.println("Not a prime number");
          isPrime = false;
          break;
        }
        i++;
}
//if(i==num)
if(isPrime)
  System.out.println("It is a prime number");
```

==========

Write a program to check the given number is a strong number or not.

145=>1!+4!+5!=>1+24+120=>145

```
==========================================================
int num=145;
int sum=0;
int temp=num;

while(num!=0)
{
   int rem=num%10; 1
   int i=2, fact=1;
   while(i<=rem)
   {
         fact=fact*i;
         i++;
   }
   sum=sum+fact;
   num=num/10;
}

if(sum==temp)
  System.out.println("It is a Strong number");
else
  System.out.println("It is not a Strong number");
```

```
=========================================================
```

Find the multiplication of two given number using Russian Multiplication alogorithm.

sum=0+12+24+96=132

```
11   12
5    24
2    48-discard
1    96
0
```

```
=========================================================
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

class Pattern
{
  public static void main(String[] args)
  {
```

```
   int i=1;
   while(i <= 5)
   {
          int j=1;
          while(j <= i)
          {
            System.out.print(j+ " ");
            j++;
          }
          i++;
          System.out.println();
   }
  }
}
```

==========

```
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
```

```
class Pattern
{
  public static void main(String[] args)
  {
   int i=1;
   while(i <= 5)
   {
          int j=i;
          while(j <= 5)
          {
            System.out.print(j+ " ");
            j++;
          }
          i++;
          System.out.println();
   }
  }
}
```

==========

```
5 4 3 2 1
4 3 2 1
3 2 1
```

2 1
1

```
class Pattern
{
  public static void main(String[] args)
  {
   int i=5;
   while(i >= 1)
   {
          int j=i;
          while(j >= 1)
          {
             System.out.print(j+ " ");
             j--;
          }
          i--;
          System.out.println();
   }
  }
}
```

==========

```
         1
        1 2
       1 2 3
      1 2 3 4
     1 2 3 4 5

        *
       * *
      *   *
     *     *
    * * * * *
```

0 1 1 2 3 5 8 13 21 34

Accept a number, and check whether the given number is a part of fibonacci series or not.

Display the non-fibonacci series

4 6 7 9 10 11 12 14 .....

==========

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

========

do-while

========

variable initialization;

do

{

  statement1;

  statement2;

  increment/decrement statement;

}while<condition>;

int i=11;

do

{

 System.out.println(i);

 i++;

}while(i<=10);

==========

int i=1;

do

 System.out.println(i);

 i++;

while(i<=10); CE

==========

int i=1;

do

 System.out.println(i);

```
while(i<=10);
======================
int i=1;
do
  ++i;
while(i<=10);
System.out.println(i);
======================
int i=1;
do
while(++i<=10);
System.out.println(i); CE
Atleast there should be a  single statement in do-while loop.
======================
int i=1;
do
     ;//null statement
while(++i<=10);
System.out.println(i);
======================
int i=1;
do
   while(++i<=10);
while(i<=10);
System.out.println(i);
======================
for loop
========
for(initialization; condition; incr/decr expr)
{
   statement1;
   statement2;
   statement3;
}

for(int i=1; i<=10; i++)
{
    System.out.println(i);
}

variable initialization;
for( ;condition; incr/decr expr)
{
```

```
        statement1;
        statement2;
        statement3;
}

int i=1;
for( ;i<=10; i++)
{
  System.out.println(i);
}

variable initialization;
for( ;condition; )
{
  statement1;
  statement2;
  incr/decr statement;
  statement3;
  statement4;
}

int i=1;
for( ;i<=10; )
{
  System.out.println(i);
  i++;
}
for(int i=1; i<=10; i++)
  System.out.println(i); 1 2 3 4 5 6 7 8 9 10
=========================
for(int i=1; i<=10; i++){


}
  System.out.println(i); CE
=========================
int i;
for(i=1; i<=10; i++);
  System.out.println(i); 11
=========================
```

```
class Pattern
{
  public static void main(String[] args)
  {
   for(int i=1; i<=5; i++)
   {
          for(int j=i; j<=5; j++)
            System.out.print(j+ " ");
          System.out.println();
   }
  }
}
```

==========

```
class Pattern
{
  public static void main(String[] args)
  {
   int i=1;
   do
   {
          int j=i;
          do
          {
            System.out.print(j+ " ");
            j++;
          }while(j <= 5);
          i++;
          System.out.println();
   }while(i <= 5);
  }
}
```

=====
Array
=====

It is an indexed collection of fixed number of homogenous data elements.

Array is a reference type variable which holds the address of memory where we can store a finite number of homogenous data elements.

```
int mark1 = 40;
int mark2 = 30;
int mark3 = 70;
```

int mark4 = 80;
int mark5 = 60;

i. Declaration of an Array
==========
datatype[] arrayName; //Recommended
datatype []arrayName;
datatype arrayName[];
ex:
int[] marks = null;

ii. Allocating memory for an Array
==========
arrayName = new datatype[size];
ex:
marks = new int[5];

"new" operator will allocate memory for 5 elements, each element occupy 4 bytes of memory, so 20 bytes of memory will be allocated and its base(initial) address will be stored back to the array name(marks).

Allocating memory at the time of array declaration
==========
datatype[] arrayName = new datatype[size];
int[] marks = new int[5];

int[] marks = new int[0]; //valid
int[] marks = new int[-10]; //invalid(Runtime Error)

int SIZE=10;
int[] marks = new int[SIZE]; //valid

int marks[10]; //invalid

iii. Assigning values to an Array
==========
arrayName[index] = value;
ex:
marks[0] = 40;
marks[1] = 50;
marks[2] = 30;
marks[3] = 80;
marks[4] = 70;

Array index always starts with 0(zero).
If the size of array is n, then the last index would be n-1.
Valid indexes are 0 to n-1.

iv. Retrieving values from an Array
==========
System.out.println(arrayName[index]);
ex:
System.out.println(marks[0]);
System.out.println(marks[1]);
System.out.println(marks[2]);
System.out.println(marks[3]);
System.out.println(marks[4]);

v. Initialization of an Array

```
datatype[] arrayName=new datatype[]{val1,val2,val3,...};
datatype[] arrayName = {val1,val2,val3, ...};

int[] marks = new int[]{30,40,20,80,75};
int[] marks = {30,40,20,80,75};

for(int mark:marks)
   System.out.println(mark);

for(int index=0; index<marks.length; index++)
     System.out.println(marks[index]);
```

==========
Two Dimension Array
==========
One Dimension Array
==========
int n1=10;
int n2=20; int[] n={10,20,30};
int n3=30;

n[0] => 10 n[1] => 20 n[2] => 30

Two Dimension Array

==========
int[] n1={1,2,3};
int[] n2={4,5,6}; int[][] n={{1,2,3},{4,5,6},{7,8,9}};
int[] n3={7,8,9};

n[0] => {1,2,3}   n[1] => {4,5,6} n[2] => {7,8,9}
n[0][0] => 1   n[1][1] => 5   n[2][0] => 7

Three Dimension Array
==========
int[][] n1={{1,2,3},{4,5,6},{7,8,9}};
int[][] n2={{1,2,3},{4,5,6},{7,8,9}};
int[][] n3={{1,2,3},{4,5,6},{7,8,9}};

```
int[][][] n = {
                    {{1,2,3},{4,5,6},{7,8,9}},
                    {{1,2,3},{4,5,6},{7,8,9}},
                    {{1,2,3},{4,5,6},{7,8,9}}
               };
```

n[0] => {{1,2,3},{4,5,6},{7,8,9}}
n[0][1] => {4,5,6}
n[0][1][2] => 6
==========
i. Declaration of two dimension array
syntax: datatype[][] arrayName;
eg. int[][] matrix;

int[][][] threeDArray;

ii. Allocate memory for two dimesion array
syntax: arrayName = new datatype[NoOf1-dArray][Sizeof1-dArray]
eg. matrix = new int[3][3];
It is an array of 3 elements, where each element is a single dimension array of size 3.

threeDArray = new int[3][3][2]
It is an array of 3 elements, where each element is a two dimension array, and each two dimension array have 3 elements and each element is a single dimension array of size 2.

{{{1,2},{3,4},{5,6}}, {{},{},{}}, {{},{},{}} }

iii.Assigning values to Two Dimension Array

```
              oth             1st                  2nd
 oth-> matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2]=3;
 1st-> matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2]=6;
 2nd-> matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2]=9;

        for(int i=0;i<3;i++)
        {
          for(int j=0;j<3;j++)
          {
             System.out.println("Accept a value");
             matrix[i][j] = sc.nextInt();
          }
        }
```

iv. Retrieve the values from the above matrix

```
        for(int i=0;i<3;i++)
        {
          for(int j=0;j<3;j++)
          {
             System.out.print(matrix[i][j]+" ");
          }
          System.out.println();
        }
```

==========

```
int[] arr1 = {1,2,3};
int[] arr2 = {1};
int[] arr3 = {1,2,3,4,5};
int[] arr4 = {1,2};
int[] arr5 = {1,2,3,4};

JaggedArray
===========
int[][] jaggedArray = null;

jaggedArray = new int[5][];

jaggedArray[0] = new int[3];
jaggedArray[1] = new int[1];
jaggedArray[2] = new int[5];
jaggedArray[3] = new int[2];
```

```
jaggedArray[4] = new int[4];

for(int i=0;i<jaggedArray.length;i++)
{
   for(int j=0;j<jaggedArray[i].length;j++)
   {
         System.out.println("jaggedArray["+i+"]["+j+"]");
         jaggedArray[i][j] = sc.nextInt();
   }
}

for(int i=0;i<jaggedArray.length;i++)
{
   for(int j=0;j<jaggedArray[i].length;j++)
   {
         System.out.print(jaggedArray[i][j]+" ");
   }
   System.out.println();
}
```

==========

Command line arguments

==========

Supplying arguments to a program during execution through the command line are called as command line arguments.

Source file: CommandLineArgs.java
To compile : javac CommandLineArgs.java
To execute : java CommandLineArgs 10 20 30

Command line arguments are received by the main method of given program.

Arguments are stored internally in a form of String as follows.

```
String args={"10","20","30"};
```

==========

User-defined Methods
==========

Find out NCR. Where n and r is given.
NCR = n!/r!(n-r)!

```
class NCR
{
   public static void main()
   {
         int n=5,r=2,ncr;

         int i=1,nfact=1;
         while(i<=n)
         {
           nfact=nfact*i;
           i++;
         }

         i=1;
         int rfact=1;
         while(i<=r)
         {
           rfact=rfact*i;
           i++;
         }

         i=1;
         int nrfact=1;
         while(i<=n-r)
         {
           nrfact=nrfact*i;
           i++;
         }

         ncr=nfact/(rfact*nrfact);
         Sytem.out.println("NCR= " + ncr);

   }
```

}
Userdefined Method
==========
A method is a set of code/statements referred to by a name, which does a particular operation whenever it is being called from some other part of a program.

Which can be called from anywhere just by the name by supplying suitable number of arguments/inputs.

Whenever a method is called, the control will be transferred to the called method for its execution.

Every method will have a name, inputs as well as return type.

syntax:

```
<modifier> <returntype> methodName(arg1,arg2, ...)
  {
      //code goes here
  }
```

A method is a reusable component, which can be used from anywhere.

It prevents the repetitive code in the program.
==========

Function Oriented Programming
==========

```
int num=10;

int fun1(){
}

int fun2(){
}

int fun3(){
}

int main(){
  fun1();
  fun2();
  fun3();
}
```

Function Oriented programming is an approach to solve a given problem. It has a set of rules to be followed.
The language which follows FOP approach, those languages are called as Function Oriented Programming language.

In function oriented programming, function is the main component.
A group of functions makes a program.
More importance is given to a function rather than data.
Data is global which can be accessible by any function in the program.
It does not provide data security/data hiding at all.

Object Oriented Programming
==========
Object Oriented Programming is a methodology which organizes the data and functions/methods together in a particular structure.So that data hiding/data security can be implemented.

Main component of Obect Oriented Programming is Class and Object.

```
class <ClassName>
{
        int data1,data2;

        void method1(){ }
        void method2(){ }
}
```

```
class <ClassName>
{
        int data;
        void methodONe(){ }
}
```

Features of Object Oriented Programming
==========
i. Abstraction
ii. Encapsulation
iii. Inheritance
iv. Polymorphism

Class
=====
i. Class is a blueprint/template on which objects can be constructed.

ii. Class is a collection of datamembers(properties/attributes) and methods(behaviour)

iii. Class defines the structure of an object.

iv. Class creates an user defined data type.

v. Class is an imagination of something, which is really not existing.

```
class Dog {
        String dogName;
        String colour;
        String breed;

        void eat(){}
        void sleep(){}
        void bark(){}
}
```

Syntax:

```
<modifier> class <ClassName>
{
        dataMember1;
        dataMember2;

        method1(); //are the behaviours which operate on datamember
      method2();
}
```

```
public class Account {
        //Instance Data Member
        int accNo;
        String accName;
        double balance;

        //Instance Methods
        void deposit(double amount)
        {
        }
        double withdraw(double amount)
        {
        }
        double getBalance()
        {
        }
}
class TestAccount {
```

```
	public static void main(String[] args){
		Account account = new Account();
		System.out.println(account.accNo);
		System.out.println(account.accName);
		System.out.println(account.balance);
	}
}
```

Object:
=======
Object is an instance of a class.

Instance: The memory which is allocated for all the instance members of a class during runtime.

Object is a real world entity, which has got 3 properties.
i. State
ii. Identity
iii. Behaviour

```
Account account = new Account();
```

State: Values of data members of an object represent the state of an object.

```
	account.accNo=101;
	account.accName="Sammer";
	account.balance=5000;
```

Identity: Each object has its unique identification. Each object is identified by a unique name.

```
	Account account1 = new Account();
	Account account2 = new Account();
```
	i.e account1, account2

Behaviour: It is an method which does some operation on the data members. It modifies the state of an object.

```
	account1.deposit(5000);
	System.out.println(account1.balance);
```

getter and setter methods
==========
setter methods are used to set the value for the datamembers of the object.
settter methods always takes input from the user but never returns any result.
syntax: <modifier> void setterMethod(arg1, arg2, arg3, ...)

```
	{
	}
```

getter methods are used to get the value of the data members from an object.
getter methods always returns a value but never takes any input from the user.
syntax: <modifier> returntype getterMethod()

```
    {
      }

      void setAccountDetails(int no,String name,double bal)
      {
            accNo=no;
            accName=name;
            balance=bal
      }

      String getAccountDetails()
      {
            return accNo+":"+accName+":"+balance;
      }
```

==========

```
class ClassName
{
      DataMember
      ----------
      Static Data Member/Class Data member
      Non-Static Data Member/Instance Data member

      Method
      ------
      Static Method/Class Methods
      Non-Static Method/Instance Method

      Block
      -----
      Static Block
      Non-Static Block

      Constructor(Non-static)
}
```

Constructor
==========

It is a special kind of method whose name is same as class name.

It does not have any return type not even void.

It gets executed automatically at the time of object creation.

It's main purpose is to initialize the object. Initializing the object is nothing but initializing the data members of the class.

It can be overloaded.(i.e. A class can have more than one constructor with different signature.)
Signature:
- No.of argument
- Type of argument
- Sequence/Order of arguemnts

It can be a parameterless or parameterized construcotr.

When a class does not contain any constructor, compiler supplies a default(no-arg) constructor to the class automatically.

Method Overloading
==========
More than one method with same name and different signatue within a class is called as Method overloading.

Method signature:[No. of arguments
- Type of arguments
- Sequence/order of arguments]

Whenever more than one method does the similar kind of operations on different data types, diff. number of inputs then rather giving different names to those methods, we can have the same name for all the methods which does the similar kind of operations.

```
class Calculator{
   int add(int n1,int n2){ }
   int add(int n1,int n2,int n3){ }
   double add(double n1,double n2){ }
   double add(int n1, double n2){ }
   double add(double n1, int n2){ }
}
```

Method Signature: No. of arguments

Type of arguments
Sequence/order of arguments

==========
How to call a method?
==========
i. First identify the method is a static or non-static method.
ii.If the method is a static method:
a. If the calling method and called method both are in same class then just call the method by its name with suitable number of arguments.
b. If they are in different classes, then call the method with classname.methodname along with suitable number of arguments.
iii. If the method is a non-static method:
a. Create an object of the container class.
b. Call the method by object reference with suitable no. of arguements.

```
class Calculator {
	static int add(int n1, int n2) {
		return n1+n2;
	}
	int sub(int n1, int n2) {
		return n1-n2;
	}
	public static void main(String[] args) {
		System.out.println(add(10,20));
		Calculator calc = new Calculator();
		System.out.println(calc.sub(20,10));
	}
}
class TestCalculator {
	public static void main(String[] args) {
	  System.out.println(Calculator.add(100,200));
	  Calculator calc = new Calculator();
	  System.out.println(calc.sub(20,10));
	}
}
```

==========

```
class Circle {
	double radius;
	static double pieValue=3.14;
```

```
	Circle(double rad){
		radius = rad;
	}

	double getArea(){
		return pieValue*radius*radius;
	}

	double getCircumference(){
		return 2*pieValue*radius;
	}
}

class TestCircle {
	public static void main(String[] args) {
		System.out.println(Circle.pieValue);
		Circle c1 = new Circle(5.0);
		System.out.println(c1.getArea());
		System.out.println(c1.getCircumference());
	}
}
```

==========

Non-static Datamember

==========

Nonstatic data members are also called as instance data members because they belong to an instance of a class(i.e object).

Nonstatic data members come into existence only when there is an object exist.

Memory gets allocated for nonstatic members only when we create an object of a class.

Every object will have a separate copy of nonstatic data members.

Nonstatic members can be accesed only by an object reference.

==========

Static Datamember

==========

Static data members are also called as class members/variables because they belong to the class.

Memory gets allocated for static members at the time of loading the class.

Memory allocation for static data members does not depend on the object creation.

Only one copy of memory for static data members will be allocated and it can be shared by all the objects of the class.

Static members can be accessed by its name with in class. Outside the class it can be accesed by its classname.(i.e.ClassName.datamemberName)

When to make the method as static method?
==========
When the class does not contain any data members but there are certain methods to perform some operations, then those methods should be declared as static methods.

When a method does not consume non-static data members of a class, then those methods are also recommended to declare as static.

When to make the method as non-static method?
==========
When a method consume the non-static members of a class, then those methods are recommended to declare as nonstatic methods.

During class load:
==========
i. Identify all the static elements from top to bottom, during identification, if it comes across any static data members then it will be initialized with a default value.

ii. Assignment of static data members and execution of static blocks in which order they have been declared within the class.

iii. Execution of main method.

During object creation:
==========
i. Identify all the non-static elements from top to bottom, during identification, if it comes across any nonstatic data members, then it will be initialized with a default value.

ii. Assignment of nonstatic data memers and execution of nonstatic blocks in which order they have been declared with in the class.

iii. execution of constructor takes place.

this
====
"this" is a reference which holds the address of invoking/calling object.

"this" is a reference which holds the address of current class object.

"this" is a reference which is available implicitly inside every non-static method.

Whenever datamember and local variables have got same name, inorder to differentiate them datamembers can be referred by "this" reference.

```
this.hours = hours;
```

Whenever a method has to return the invoking object itself then we can return "this" reference.

```
p1.compareAge(p2);
```

==========

```
class Student
{
	int studentId;
	String studentName;
	String plotNo,houseNo,street,city,pinCode;
	String c_plotNo,c_houseNo,c_street,c_city,c_pinCode;
}

class Employee
{
	int employeeId;
	String employeeName;
	String plotNo,houseNo,street,city,pinCode;
	String c_plotNo,c_houseNo,c_street,c_city,c_pinCode;
}

class Address
{
	String plotNo;
	String houseNo;
	String street;
	String city;
	String pinCode;
}
```

```
class Student
{
	int studentId;
	String studentName;
	Address permanentAddress;
	Address communicationAddress;
}
=====================================
class Address
{
	String plotNo;
	String houseNo;
	String street;
	String city;
	String pinCode;
}

class Student
{
	int studentId;
	String studentName;
	Address permanentAddress;
	Address commAddress;
}


class StudentTest
{
	public static void main(String[] args)
	{
		Address pAddress=new Address();
		pAddress.plotNo="71";
		pAddress.houseNo="4-6-17";
		pAddress.street="RamNagar";
		pAddress.city="Hyderabad";
		pAddress.pinCode="500089";

		Address cAddress=new Address();
		cAddress.plotNo="72";
		cAddress.houseNo="5-6-17";
		cAddress.street="ShyamNagar";
		cAddress.city="Hyderabad";
		cAddress.pinCode="500079";
```

```
                Student stud=new Student();
                stud.studentId=101;
                stud.studentName="Sameer";
                stud.permanentAddress= pAddress;
                stud.commAddress=cAddress;

                System.out.println("Id: "+stud.studentId);
                System.out.println("Name: "+stud.studentName);
                System.out.println("Permanent Address");
System.out.println(stud.permanentAddress.houseNo);
                System.out.println(stud.permanentAddress.plotNo);
                System.out.println(stud.permanentAddress.street);
                System.out.println(stud.permanentAddress.city);
                System.out.println(stud.permanentAddress.pinCode);

                System.out.println("Communication Address");
                System.out.println(stud.commAddress.houseNo);
                System.out.println(stud.commAddress.plotNo);
                System.out.println(stud.commAddress.street);
                System.out.println(stud.commAddress.city);
                System.out.println(stud.commAddress.pinCode);
        }
}
```

==========

Packages in Java

==========

A Package is a collection of related classes. It helps to organize your classes into a folder structure and make it easy to locate and use them.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.

Packages are used for:
->Preventing naming conflicts
->Searching/locating and usage of classes, interfaces, enumerations and annotations easier
->Providing controlled access

java.util.Date
java.sql.Date

java.util.Scanner
java.lang.System
java.lang.String
java.io.FileInputStream

Oracle Corporation => www.oracle.com => com.oracle

Sun Microsystems => www.sun.com => com.sun

Apache Foundation => www.apache.org => org.apache

Spring Foundation => www.springframework.org => org.springframework

SSSIT => www.sssit.info => info.sssit

```
info.sssit.p1                 info.sssit.p2          Access Modifiers:
|                             |                      1.private
|-ClassOne [public]           |-ClassI [public]      2.default
| 1 2 3  4                    | 1 2 3  4             3.protected[default+child]
|-ClassTwo [default]          |-ClassII [default]    4.public
| 1 2 3  4                    | 1 2 3 4

|-ChildOne-> ClassOne                |-ChildI -> ClassOne
  1 2 3 4                            1 2 3 4
```

	private	default	protected	public
With in Class	Y	Y	Y	Y
Class with in same package	N	Y	Y	Y
Child with in same package	N	Y	Y	Y
Class in diff package	N	N	N	Y
Child in diff package	N	N	Y	Y

[Using Child Object]

N N N Y

[Using Parent Object]

```
package info.sssit.p1;

public class ClassOne{
        private int data1;
        int data2;
        protected int data3;
        public int data4;

        private int getData1(){ }
        int getData2(){ }
        protected int getData3(){ }
        public int getData4(){ }
}
```

protected => default+child
package+inheritance

==========

Access Modifiers

==========

Java access modifiers are used to provide access control in java. Java provides access control through three keywords **private**, **protected** and **public**. We are not required to use these access modifiers always, so we have another one namely **default access**, **package-friendly**.

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

There are 4 access modifiers.

i. private
ii. [default]
iii.protected
iv. public

We are allowed to use only "public" or "default" access modifiers with java classes.

class MyClass { } public class MyClass {}

[note: Inner classes can be private or protected]

=>If a class is "public" then we can access it from anywhere, i.e from any other class located in any other packages etc.
=>We can have only one "public" class in a source file and file name should be same as the public class name.
=>If the class has "default access" then it can be accessed only from other classes in the same package.

Java Access Modifiers with Class Member
==========
We can have all the four access modifiers for class member variables and methods. However, member access modifier rules get applied after the class level access rules. For example, if a class is having default access then it will not be visible in other packages and hence methods and variables of the class will also be not visible.

private:
========
If a class member is "private" then it will be accessible only inside the same class. This is the most restricted access and the class member will not be visible to the outer world. Usually, we keep class variables as private and methods that are intended to be used only inside the class as private.

default:
========
If a class member doesn"t have any access modifier specified, then it"s treated with default access. Class member with default access will be accessible to the classes in the same package only.

protected:[default+child class]
==========
If class member is "protected" then it will be accessible only to the classes in the same package and to the subclasses. we use this keyword to make sure the class variables are accessible only to the subclasses.

public:
=======
If a class member is "public" then it can be accessed from anywhere.These member variable

or method is accessed globally.

(Least Accessible) private < default < protected < public (Most Accessible)

==========

Abstraction

==========

Abstraction is a process of collecting important information which will create a base for building up a complex system.
It is a mechanism of extracting the essential elements for the creation of a system, without its implementation details. In abstraction, we have to focus only on what is to be done instead of how it should be done. Abstraction is a thought process; it solves the problem at the design level.

Encapsulation

==========

Encapsulation is a process of making a complex system easier to handle for an end-user, without worrying about its internal complexities.
It is a mechanism that binds data and code together and also keeps them safe from the outside interference. In simple words, encapsulation hides the complexity of a system.

The basic difference between "abstraction" and "encapsulation" is that abstraction focuses on "identifying the necessary components for building a system" whereas, encapsulation focuses on "hiding the internal complexities of a system"

Abstraction	Encapsulation
Shows, what elements are necessary to build a system.	Hides the complexity of a system.
Focus is on "what" should be done.	Focus is on "how" it should be done.

Data hiding

==========

Data hiding is a concept in object-oriented programming which confirms the security of members of a class from unauthorized access. Data hiding is a technique of protecting the data members from being manipulated or hacked from any other source.

Data hiding focuses on the security of data, the encapsulation focuses on reducing the complexity of the system in order to make the application more user-friendly.

Inheritance
==========
Inheritance is a mechanism in which one class acquires the properties and methods of another class.

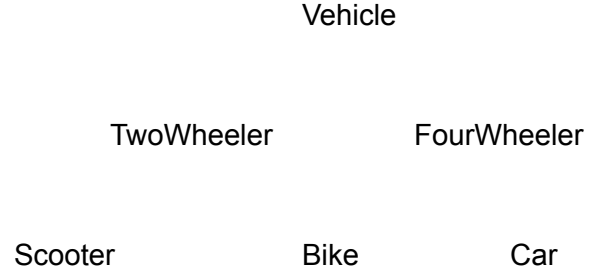
With inheritance, we can reuse the fields and methods of the existing class. Hence,Inheritance is one of the Re-usability feature of an Object Oriented Programming concept.

Inheritance is also called as IS-A relationship.

Inheritance helps to create a component by using an existing component, hence it saves development as well as testing time during the development.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Existing class is called as Base/Super/Parent class.
New class is called as Derived/Sub/Child class.



Syntax:
======
```
class <ChildClass> extends <ParentClass> {
        //Child class properties
        //Child class methods
}
```
ChildClass acquires the properties/methods of the Parent class.

Example:
========
```
class Employee {
        int empId;
        String firstName;
```

```
	String lastName;

	String getFullName(){
		return firstName+" "+lastName;
	}
}

class PartTimeEmployee extends Employee {
	int noOfHours;
	int payPerHour;

	int getMonthlySalary(){
		return noOfHours*payPerHour;
	}
}
```

Types of Inheritance
==========
Single Inheritance: In Single Inheritance one class extends another class (one class only).

Multiple Inheritance: In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance.

Hierarchical Inheritance: In Hierarchical Inheritance, one class is inherited by many sub classes.

Multilevel Inheritance: In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.

Hybrid/Virtual Inheritance: Hybrid inheritance is a combination of Single and Multiple inheritance.
==========

```
public class Employee
{
	int empId;
	String firstName;
	String lastName;

	String getFullName()
	{
		return firstName+" "+lastName;
	}
```

```
}
================================================
public class PartTimeEmployee extends Employee
{
	int noOfHours;
	int payPerHour;

	int getMonthlySalary()
	{
		return noOfHours*payPerHour;
	}
}
================================================
public class FullTimeEmployee extends Employee
{
	int annualSalary;

	int getMonthlySalary()
	{
		return annualSalary/12;
	}
}
================================================

public class EmployeeDemo {
	public static void main(String[] args) {
		PartTimeEmployee pte=new PartTimeEmployee();
		pte.empId=101;
		pte.firstName="Ajay";
		pte.lastName="Mishra";
		pte.noOfHours=120;
		pte.payPerHour=75;
		System.out.println("Id: "+pte.empId);
		System.out.println("Full Name: "+
					pte.getFullName());
		System.out.println("Monthly Salary: "+
			pte.getMonthlySalary());

		FullTimeEmployee fte=new FullTimeEmployee();
		fte.empId=102;
		fte.firstName="Akshay";
		fte.lastName="Choudhury";
```

```
                fte.annualSalary=600000;
                System.out.println("Id: "+fte.empId);
                System.out.println("Full Name: "+
                                fte.getFullName());
                System.out.println("Monthly Salary: "+
                        fte.getMonthlySalary());
        }
}
=======================================
Invocation of Constructor in Inheritance
=======================================
class Base extends Object {
        /*Base(){
          super();
        }*/
}
class Derived extends Base {
        /*Derived(){
          super(); //call the parent class no-arg constructor
        }*/
}
class BaseDerived {
        public static void main(String[] args) {
                Derived d = new Derived();
                System.out.println("Oject created");
        }
}
=================================================
class Base extends Object
{
        Base(){
          super();
          System.out.println("From Base");
        }
}
class Derived extends Base
{
        Derived(){
          //super();
          System.out.println("From Derived");
        }
}
class BaseDerived
```

```
{
	public static void main(String[] args)
	{
		Derived d = new Derived();
		System.out.println("Oject created");
	}
}
==================================================
class Base extends Object
{
	/*Base(){
	  super();
	}*/
}
class Derived extends Base
{
	Derived(){
	  super();
	  System.out.println("From Derived");
	}
}
class BaseDerived
{
	public static void main(String[] args)
	{
		Derived d = new Derived();
		System.out.println("Oject created");
	}
}
==================================================
class Base extends Object
{
	Base(){
	  super();
	  System.out.println("From Base");
	}
}
class Derived extends Base
{
	/*Derived(){
	  super();
	}*/
}
```

```
class BaseDerived
{
	public static void main(String[] args)
	{
		Derived d = new Derived();
		System.out.println("Oject created");
	}
}
==================================================
class Base {
	int x;
	Base() { }
	Base(int x) {
		this.x=x;
	}
}
class Derived extends Base {
	int y;
	Derived(int x,int y) {
		//super()
		super(x);
		this.y=y;
	}
}
class BaseDerived {
	public static void main(String[] args) {
		Derived d = new Derived(10,20);
		System.out.println(d.x +" "+d.y);
	}
}
==================================================

public class Employee
{
	int empId;
	String firstName;
	String lastName;

	Employee(){}

	Employee(int empId,String firstName,String lastName)
	{
		this.empId = empId;
```

```
                this.firstName = firstName;
                this.lastName = lastName;
        }

        String getFullName()
        {
                return firstName+" "+lastName;
        }
}
=========================================

public class PartTimeEmployee extends Employee
{
        int noOfHours;
        int payPerHour;

        PartTimeEmployee(int empId,String firstName,
         String lastName, int noOfHOurs, int payPerHour)
        {
                super(empId,firstName,lastName);
                this.noOfHours=noOfHOurs;
                this.payPerHour=payPerHour;
        }

        int getMonthlySalary()
        {
                return noOfHours*payPerHour;
        }
}

=============================================
public class FullTimeEmployee extends Employee
{
        int annualSalary;

        FullTimeEmployee(int empId,String firstName,
                        String lastName, int annualSalary)
        {
                super(empId,firstName,lastName);
                this.annualSalary=annualSalary;
        }

        int getMonthlySalary()
```

```
	{
		return annualSalary/12;
	}
}
================================================

public class EmployeeDemo
{
	public static void main(String[] args)
	{
		PartTimeEmployee pte=
		new PartTimeEmployee(101,"Ajay","Mishra",120,75);
		System.out.println("Id: "+pte.empId);
		System.out.println("Full Name: "+
				pte.getFullName());
		System.out.println("Monthly Salary: "+
			pte.getMonthlySalary());

	FullTimeEmployee fte=
	new FullTimeEmployee(102,"Akshay","Choudhury",600000);
	System.out.println("Id: "+fte.empId);
	System.out.println("Full Name: "+
					fte.getFullName());
	System.out.println("Monthly Salary: "+
					fte.getMonthlySalary());
	}

}
=========================================

class Base {
	int x;
	//Base(){}
	Base(int x) {
		this.x=x;
	}
}
class Derived extends Base {
	int y;
	Derived(int x,int y) {
		super(x);
		this.y=y;
	}
```

```
}
class BaseDerived {
	public static void main(String[] args) {
		Derived d = new Derived(10,20);
		System.out.println(d.x +" "+d.y);
	}
}
```

==========

Method Overloading

==========

More than one method have same name with different signature with in a class is called as Method Overloading.

It happens within class.

Singature must be differnt.

Method Overriding

==========

When both Parent and child have a method with same name, signature and return type then it is called as method overriding.

It happens between Parent and child class.

Signature must be same.

The process of redefining/reimplementing the parent method in child class is called as Method Overriding.

Why overloading?

==========

When more than one method does the same functionality with different set of arguments, different types of argument then we need to write all the method with same name and different signature.

Why overriding?

==========

Whenever parent class method is not suitable for child class, then child class redefines the parent class method in a different way according to its requirement.

Method present in Parent class is called overridden method.
Method present in Child class is called overriding method.

Parent Class	Child Class
public	public
protected	protected, public
default	default, protected, public
private	NA

==========

Parent p = new Parent();
Using reference "p", we can call all the properties and methods of Parent class only.

Child c = new Child();
Using reference "c", we can call all the properties and methods of Child class as well as Parent class.

Parent p = new Child();
Using reference "p", we can call all the properties and methods of parent class and overriding methods in child class only. Child class properties and methods are not accessible.

Abstract Method: Method which has no implementation/body are called as abstract methods.

Concrete Method: Method which has implementation/body are called as Concrete Methods.

If there is an abstract method in a class, then class must be declared as abstract.

But if the class is abstract, it doesn't mean that it must have abstract methods in it. It may or may not have abstract methods.

An abstract class can have both abstract and concrete methods in it.

An abstract class can't be instantiated(can't create an object).
Because it is an incomplete class.

Though we can't create an object of an abstract class, but it can have constructors.

The constructors of an abstract class is used by its child class to initialize the parent class members.

If a parent class contains an abstract method, then child class has to override all the abstract methods of its parent class, otherwise the child class should be declared as abstract.

Hence, abstract method in parent class forces its child classes to implement it without fail.

final
=====
It is a modifier, which can be applied on
=> data member
=> method
=> class

"final" data member
==========
=> It must be initialized. Once it is initialized its value can't be modified.

=> If the data member is non-static final. Then there are 3 ways to initialize it.
* At the time of declaration
* Using constructor
* Using non-static blocks
=> If the data member is static final, Then there are 2 ways to initialize it.
* At the time of declaration
* Using static blocks.

"final" method
==========
When the method is declared as final in parent class, then it can be redefined by its child class.
It does not support method overriding.

final methods can't be overridden.

"final" class
==========
When the class is declared as final, it can't be subclassed.
Final classes can't be extended by other classes.

```
final class MyClass{
    //data
```

```
        //methods
    }

    class YourClass extends MyClass{  //CE
    }
```

A method can't be declared as both final and abstract.

A class can't be declared as both final and abstract.

When the class is abstract, it is not mandatory that it must have got abstract methods, It may or may not have.

But if there is an abstract method, the class must be declared as an abstract method.

==========
==========