Java

Computer understands binary language(1s and 0s)

LowLevel Language - Assembly language (Close to machine understandable)

HighLevel ProgrammingLanguage Languages - C, C++, Java, Python, Ruby...

C, C++ are considered low level

Compilers /interpreters are softwares used for programs compilations/executions.

Compiler - This takes entire code as input at once and Intermediate

object code is generated by compiler.

example for compiled languages: c,c++,scala, smalltalk

byte code(intermediate code) is object code which is actually processes by virtual machine, it is different

from machine code which processor can understand

Interpreter - It takes single line or instruction as an input and executes it

, no intermediate code generated , it is faster, memory requirement is less

example for interpreted languages: Ruby, Python

int add(int a, int b){

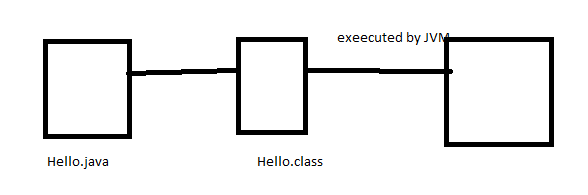
int sum = a+b;

return sum;

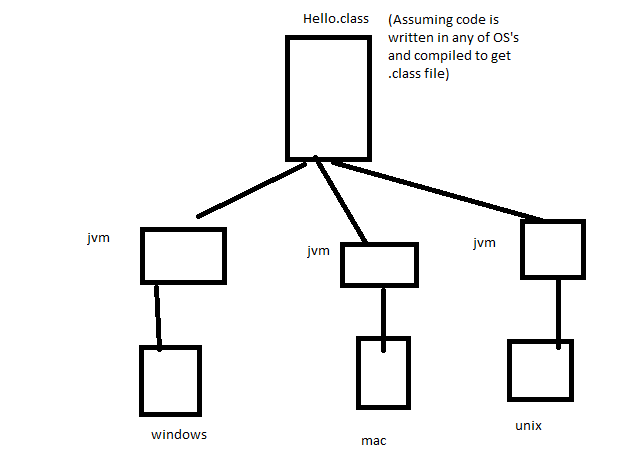
}

Hello.java --> Hello.class(byte code)-> understood by virtual machine

and it should be converted to binary code



Java follows WORA principle(write once and run anywhere) - as it is system independent



System Independent Language - can be executed on any machine irrespective of which OS

you have used to write and compile the code.

System dependent Language - need to compiled again if you want to execute in

some other machine with different OS.

JVM(java virtual machine) is software system dependent but it makes Java language system independent.

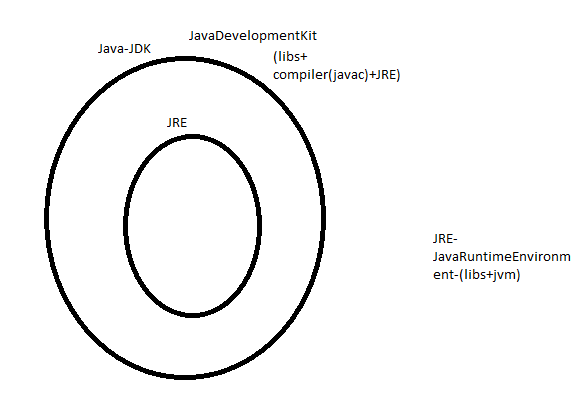
Java Features:

* Simple
* ObjectOriented - Everything in Java is inside a class and they are accessed using objects for class
* SystemIndependent
* Distributed- can communicate over network applications and with protocols tcp/ip , udp

It became popular for internet based appications which desire system independent feature

* Robust- supports good exception handling mechanisms and it handles memory and deallocation internally(by jvm)
* Secure
* MultiThreaded

**Java- jdk - jre :**

****

Download:

http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

click jdk download, accept agreement

Java SE Development Kit 8u121 - under this download your OS specfic executable

Go to downloads-> double click the executable

Set Env variables- JAVA\_HOME -C:\Program Files (x86)\Java\jdk1.8.0\_101

edit path variables and add C:\Program Files (x86)\Java\jdk1.8.0\_101\binary

After you download Java- observer the download folders-we see

Java is case sensitive:

single line comments: //

multi line comments:/\* \*/

Class: Class is a blueprint that defines variables(properties) and methods(actions).

Object: Object is instance of class which allows to access properties and methods of class.

Object is the things which you see in real world - Animal, Vehicle, Printer, Mobile, Loan, Account,Customer etc

Class Animal{

//variables- properties

String name;

String breed;

String color;

//methods

public void eatFood(){

} }

Animal a1 = new Animal();

a1.name="punto";

a1.color="white";

a1.eatFood();

Animal a2 = new Animal();

a2.name="princess";

a2.color="brown";

a2.eatFood();

public class Calculator{

public int add(int a, int b){

return a+b;

}

}

public static void main(String[] args){

A a = new A();

a.logic();

}

Explain what main method syntax is meant for:

public - accessable from anywhere

static- Can be accessed ithout creating object

- to call statci method or variable we do not need to create object

void- method is not returning anything back to whoever is calling the method

main- this is the main method that jvm looks for to start java program execution

String[] args- method input parameters

[]- means its list/array of string elements

Eclipse Tool

New Java Project - Package - Class

.project, .classpath

package, import statements, class definitions, methods

Class – model or blueprint which has properties and methods

Variable – Variable is a container which hold the values.

Method – are actions which define logic of object

Object – instance of class. this has the properties and it can call class methods

Mobile – class

color, size, buttons – variables

make a call, send a email, click pic, post, delete – operations/ methods

iphone, Samsung, google - object

Furniture – class

chair, table, bed, sofa – objects

color, legs, size, material – variables

sit, sleep – methods

Syntax for Variables :

accessModifier dataType variableName;

String cloor;

**int** size;

Syntax for Methods :

accessModifiers returnType methodName(){ code }

**public** **void** makeACall(String inputColor){

System.***out***.println("I can make a call now..");

}

public int makeACall(){

return 1;

}

**variable syntax :**

accesModifier dataType nameOfVariable;

eg:

public int id=10;

public String name=”Alan”;

**method synatax:**

access modifier returnType/output methodName(input parameters){

}

return type and input parameters are not mandatory to be present in every method.

access modifier returnType/output methodName(){}

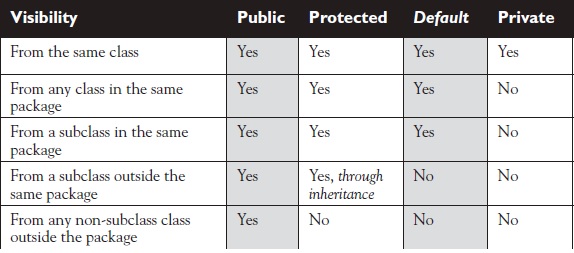
access modifier void(not returning anything) methodName(){

}

access modifier void(not returning anything) methodName(input parameters){

}

AccessModifiers- to define scope of class, method and variable



public, protected, default, private

**Rules for Names/Identifiers:**

Do not start with a number, it can only start with numbers, $, \_, it can include numbers.

Correct : mobile, mobile123, m12obil$\_e, $mobile, \_mobile, m12548

Wrong : 1mobile, 123455,

**Idetifier Rules & Naming Convetions**

Java is Case Sensitive

Class names, Interfaces – initCaps – HelloWorld, Hello, MobileTest

Package Names – in lower case – com.wbl.oops

variables, methods – camelCase - noOfButtons, size

constants – All Caps, **double** PI = 3.14;

IDE- Integrated development Environment

Java- Eclipse, IntellijIdea, NetBeans

Javascript- webstorm, phpstorm, cloud9

Python- Pydev

Eclipse download: <https://www.eclipse.org/downloads/>?

Maven – build tool – this does not replace eclipse

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class – is a blue print

Object – instance of class

package – folder structure

Access Modifiers:

public – accessible from anywhere

private- accessible only within class

**protected**- It is accessible within the package + subclasses even if they are in other package

Protected variables can be accessed in other package only through inheritance.

package/default- It is accessible within the package

Inheritance :

A{

}

B extends A{

}

main🡪 displaySUbjects()🡪subjects()

**Constructor in Java:**

Constructor is called before creating an object.

It is similar to a method but it will have same name as class name,

It do not have return type like method, but it can take input arguments.

We can have overloaded constructors – means having more than one constructor in the same class

with difference in parameters(either no of parameters or datatype should be different)

Constructors are useful to initialize object data at the time of creation of object.

**this** - key word that refers to current object of the class

We use it mostly in constructors/ getter setter methods to refer and initialiaze class variables

**public** String name;

**public** **int** id;

**public** Employee(String name,**int** id){

System.***out***.println("constructor is called first...");

**this**.name = name;

**this**.id= id;

}

**public** **void** setName(String name) {

**this**.name = name;

}

this can also be used to call overloaded constructor within same class

**public** Car(){

//calling parmeterised constructor from default constructor by using this

**this**(5678);

System.***out***.println("parent class constructor");

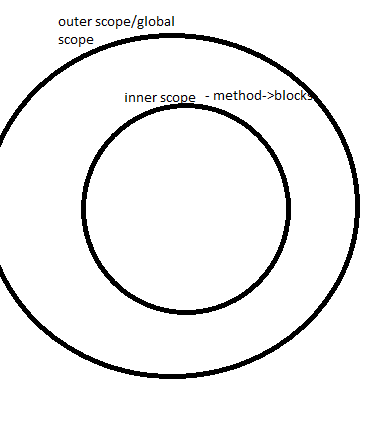
}

**public** Car(**int** regNo){

System.***out***.println("parent class constructor"+ regNo);

}

**variable scopes:**



**static** -

We can use static for variables/methods and blocks.

* Static means class level not object level. Static variables data is shared between the objects.
* we do not need objects to be created for accessing static variable or methods – we can access by class name.
* In a class - we cannot access non static members inside static method/block.
* But we can access static members inside a non static method/block/constructor
* static blocks are called at class loading time- useful for initialization.

example to understand static data is shared between objects

: count no of objects created for a class

**public** **class** Employee {

//global variables or class level variables

**public** String name;

**public** **int** id;

// static variable

**public** **static** **int** *count*=0;

//constructor

**public** Employee(String name,**int** id){

System.***out***.println("constructor is called first...");

**this**.name = name;

**this**.id= id;

++*count*;

}

**public** **void** displayDetails(){

System.***out***.println("name : " + name);

System.***out***.println("id : " + id);

// u can access static variable/method inside non static methods

System.***out***.println(*count*);

}

**public** **static** **void** timeSheet(){

// we cannot access non static variables inside static method/block

System.***out***.println("time sheet");

}

}

**public** **class** EmpTest {

**public** **static** **void** main(String[] args) {

Employee e1 = **new** Employee("Alan", 1);

// e1.name="Alan";

// e1.id=1;

e1.id = e1.id + 1;

// we can access non static variables/methods only with an object

e1.displayDetails();

// we can access static variables/methods without an object

System.***out***.println("count of objects::" + Employee.*count*);

Employee.*timeSheet*();

}

}

**final** : final is keyword which means it cannot be modified

final variables means values cannot be reassigned

final class – we cannot created subclass/child for final class

* cannot be inherited

final methods cannot be overridden.

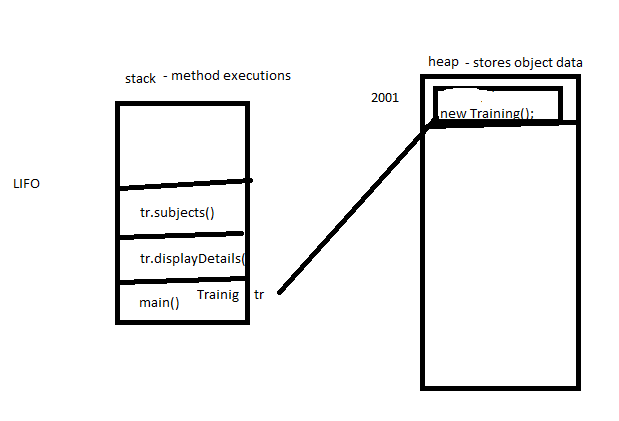
int i=10;

i=11;//valid

final int j=10;

j=11;//invalid

Stack and heap:



**OOPS:**

**Encapsulation** – hiding the implementation behind an interface/ binding the variables and methods together by making variables as private and giving public getters and setters

**Inheritance** : A class can inherit/extend other class and make use of common logic, the main advantage if inheritance is to avoid code redundancy(duplicate code).

All public and protected variables and methods are inherited to child class from parent class.

A child class can override the method in parent class to have any specific logic for child.

Parent class reference can hold parent objects as well as child class objects.

Overriding – It is possible when we have inheritance only.

Super/Parent Class method is again overridden in child class.

Rules for overriding:

method name, return type in child class should be same as parent class.

access modifier should also be same or less restrictive

public->protected->package->private(left to right 🡪 less restrictive to more restrictive)

Object type decides which method to be called at run time.

when a method is overridden(it is present in both parent and child)- If we create child class object it calls child class method, if we create parent class object it calls parent class method.

Car car = **new** SportsCar();

//child class method is called

car.speed();

Car car1 = **new** Car();

//Parent class method is called

car1.speed();

super is keyword used w.r.to parent and child classes

super can be used to call parent class constructors and parent class methods/variables from child class

**public** SportsCar(**int** regNo){

//if in parent class - there is only parameterised constructor child class constructor's

//first statement should be super(parameters reqd for parent)

**super**(regNo);

System.***out***.println("child class constructor");

}

**public** **void** test(){

//super is used in child class to spefically get parent class methods or variables

//which are present in both parent and child classes

**super**.speed();

}

**Polymorphism:** Existing in multiple forms

Same methods can be present in same class with diff arguments.- overloading

Static polymorphism- At Compile time it decides which method to be called.

Calculator calc = **new** Calculator();

**float** a=10f;

**float** b=11f;

//At compile time-Java knows which method to be called based on no of arguments or

//data type of arguments - virtual method invocation- static polymorphism

calc.add(5f, 6f);

calc.add(a, b);

calc.add(5, 6);

Same methods can be present in both parent and child.- overriding

Dynamic polymorphism- At run time it decides which method to be called.

Training tr = **new** QaTraining();

System.***out***.println(tr.displayCourseContent());

//At run time - which method to be called - whether method in parent or child-

//jvm knows based on object passed - virtual method invocation- dynamic polymorphism

tr.getRecordings();

Same reference can hold different objects

eg : Car can hold Car and SportsCar objects

Car car = **new** Car();

car.color="white";

car.price=10000;

car.model="nissan";

System.***out***.println(car.getCarDetails());

//same reference can hold different objects

car= **new** Car();

car.color="black";

car.price=50000;

car.model="ford";

System.***out***.println(car.getCarDetails());

Same reference can hold different type of objects(through inheritance)

Car car = **new** SportsCar();

Car car = **new** Car();

TypeCasting w.r.to Objects:

Implicit/Upcasting :

Explicit/downCasting :

Car car = **new** Car();

SportsCar sportCar = **new** SportsCar();

//child class reference to parent class reference

//upcasting or implicit casting

car = sportCar;

car.driveIt();

// when we are downcasting make sure parent reference still hold child object

Car car1 = **new** SportsCar();

SportsCar sportCar1 = **new** SportsCar();

//downcasting or explcit casting

sportCar1 = (SportsCar)car1;

sportCar1.driveIt();

// this is wrong-- class cast exception is thrown at run time at line-50

Car car2 = **new** Car();

SportsCar sportCar2 = **new** SportsCar();

//downcasting or explcit casting

sportCar2 = (SportsCar)car2;

sportCar2.driveIt();

Overloading:

Having more than method with same name in same class with different arguments.

Rule:

method should differ in no of arguments or datatype of arguments

Overloaded methods may or may not have same return type and access modifiers.

Overloading can be for both constructors and methods.

|  |  |
| --- | --- |
| Overloading | Overriding |
| Method arguments **must** change-either by number of args or by datatype | Method arguments **must not** change |
| Return type can be changed | Cannot change the return type except the covariant return types |
| Exceptions declared in method signature can change | Eg hierarchy for exceptions: Exception->IOException->FileNotFoundException  Cannot change exceptions of super class in subclass method overriding-u can still add any unchecked (runtime)exceptions,narrower checked exxcpetions, but u cannot throw new or broader checked exceptions |
| Access modifiers can change | Cannot make **more restrictive** access modifier or we cannot reduce the visibility… |
| It decides which method to call at compile time- it exhibits virtual method invocation at run time. | At run time based on which object is getting(subclass or superclass) created it calls that particular method, but at compile time it just decides on object reference – so we need to make sure that the reference objet has the method u r calling at compile time. But remember that at runtime, Java uses virtual method invocation to dynamically select the actual version of the method that will run, based on the actual instance |
| Overloading is Compile Time polymorphism | Overriding is RunTime polymorphism |

**Abstraction-** hiding certain features. Achieved through interface and

abstract class.

**Abstract class and Interface:**

**Abstract class** – has both abstract methods and concrete methods

We cannot create object for abstract class.

Abstract class can have a constructor which is called when creating child class objects.

In abstract class – concrete methods can use any access modifier- public, protected, private or package.

In abstract class – abstract methods can use any access modifier- public, protected, or package.

but prefer public and protected.=> abstract methods should not be final or private

abstract class cannot be declared final.

**General variables are allowed in abstract class**

**public** String name;

WHEN WE NEED abstract class:

If a parent class need to provide some common implementations to child class as well as

declare some method to be implemented by child class we use abstract class.

**Interface**:

It is a contract and it is 100% abstract class.

By default interface variables are public, static and final

By default interface methods are public and abstract

They do not have constructors and we cannot create object for interface.

WHEN WE NEED:

Interface is used to set rules/ contract to all child classes and allow them to provide specific

implementations.

interface supports multiple inheritance.

**MultiLevel inheritance** for classes- supported in java:

class A{

}

Class B extends A{

}

class C extends B{

}

**Multiple inheritance** – not supported in Java using classes but we can achieve it by interfaces

class A{

}

Class B {

}

Not supported:

class C extends A,B{

}

**Multiple inheritance –interfaces**

interface A{

}

interface B {

}

class C implements A,B{

}

**abstract class can extend another abstract class.**

**public** **abstract** **class** A {

}

**public** **abstract** **class** B **extends** A {

}

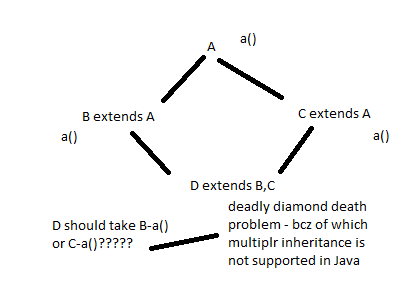
**interface can extends one or more interfaces.**

**public** **interface** IC **extends** IB,IA {

}

**Java do not support multiple inheritance but we can achieve it through interface, Java classes**

**support multilevel inheritance.**



|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Abstract class can have both concrete and abstract methods | Interface is 100% abstract class, purely a contract. |
| We can have constructors in abstract class.  We cannot create object for abstract class. Constructor is called when we are creating object for no abstract child class. | We cannot have constructors in interface.  We cannot create object for interface too. |
| Multiple inheritance is not possible through abstract classes | Multiple inheritance is possible through interface |
| We need to use abstract keyword for methods explicitly. | Methods are public, abstract by default.  Variables are public, static, final by default. |
| Abstract class allows both final and general (non final)variables. | interface do not allow general variables, only final variables are allowed. |
| \*\*We use abstract class when we need common logic to be shared to child classes through concrete methods and also we need some abstract methods. | \*\*We use it when there is no common logic to be shared to child classes, all methods are abstract and as per design if we need multiple inheritance to be supported in future. |
| We use abstract keyword for methods and class declaration. Child classes use extends | We use interface keyword class declaration .Child classes use implements |

**is a** – all child classes pass Is A relation with parent(class, abstract class, interface)

**has a –** one class dependent on other class

A{

//A is dependent on B – A has B

B b= new B();

int a(){

b.b();

}

}

B{

int b(){

}

}

**DataTypes :**

Java is **statically** typed language.

**Primitive data types:**

8 different primitive data types. Other than primitives everything is a class and we access it by creating object.

byte – 8 bit

ranges : - 128 to 127(-2^8 - 1 to 2^8)

short – 16 bit

ranges : - 128 to 127(-2^16 - 1 to 2^16)

int - 32 bit

ranges : - 128 to 127(-2^32 - 1 to 2^32)

long – 64 bit

ranges : - 128 to 127(-2^64 - 1 to 2^64)

float – 32 bit

double – 64 bit

char – 16 bit Unicode character(any keyboard character)

boolean – true or false

byte->short->int->long->float->double

left-right : lower range->higher range (upcasting/implicit)

right-left : higher range->lower range (downcasting/explicit)

Primitive TypeCasting:

//implicit cast or upcasting

b=32;

i = b;

System.***out***.println("upcast:"+i);

//explicit cast or downcasting

i=56;

b= (**byte**)i;

System.***out***.println("downcast:"+b);

Be careful with ranges when you are downcasting – make sure that the value being assigned to low range data type is within its range.

**Assignment 1:**

**Write a program to define Employee and Department class with some required variables**

**Class that calculates bonus for employee based on department.**

**Payroll class that displays salary for employees with different departments**

**Operators and loops:**

**https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html**

Operands – on which we apply operator

eg: a+b-> a and b are operands, + is operator

if operator is used between 2 operands – binary operators

if operator is used for single operand – unary operator

**Summary of Operators**

The following quick reference summarizes the operators supported by the Java programming language.

**Simple Assignment Operator**

= Simple assignment operator

**Arithmetic Operators**

+ Additive operator (also used

for String concatenation)

- Subtraction operator

\* Multiplication operator

/ Division operator

% Remainder operator

**Unary Operators**

+ Unary plus operator; indicates

positive value (numbers are

positive without this, however)

- Unary minus operator; negates

an expression

++ Increment operator; increments

a value by 1

-- Decrement operator; decrements

a value by 1

! Logical complement operator;

inverts the value of a boolean

**Equality and Relational Operators**

== Equal to

!= Not equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

**Conditional Operators**

&& Conditional-AND

|| Conditional-OR

?: Ternary (shorthand for

if-then-else statement)

**Type Comparison Operator**

instanceof Compares an object to

a specified type

**Bitwise and Bit Shift Operators**

~ Unary bitwise complement

<< Signed left shift

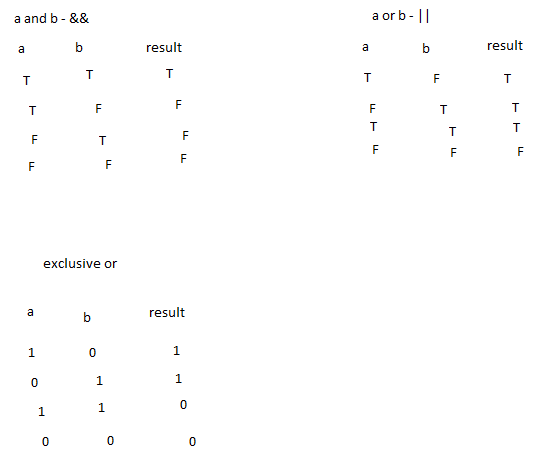
>> Signed right shift

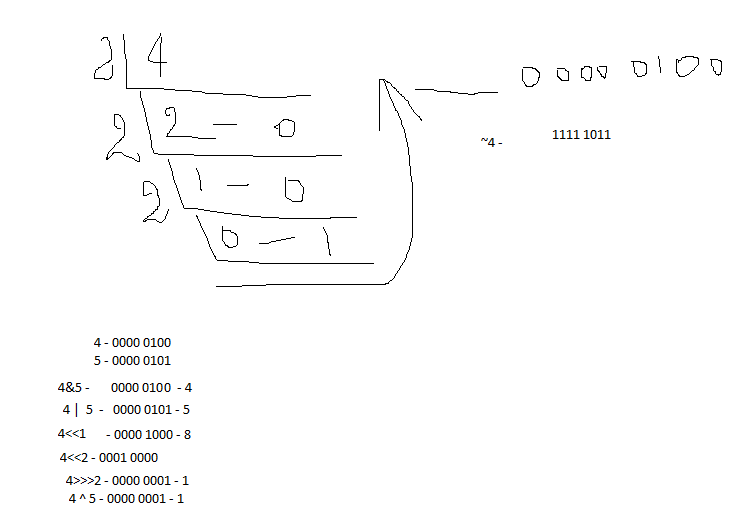
>>> Unsigned right shift

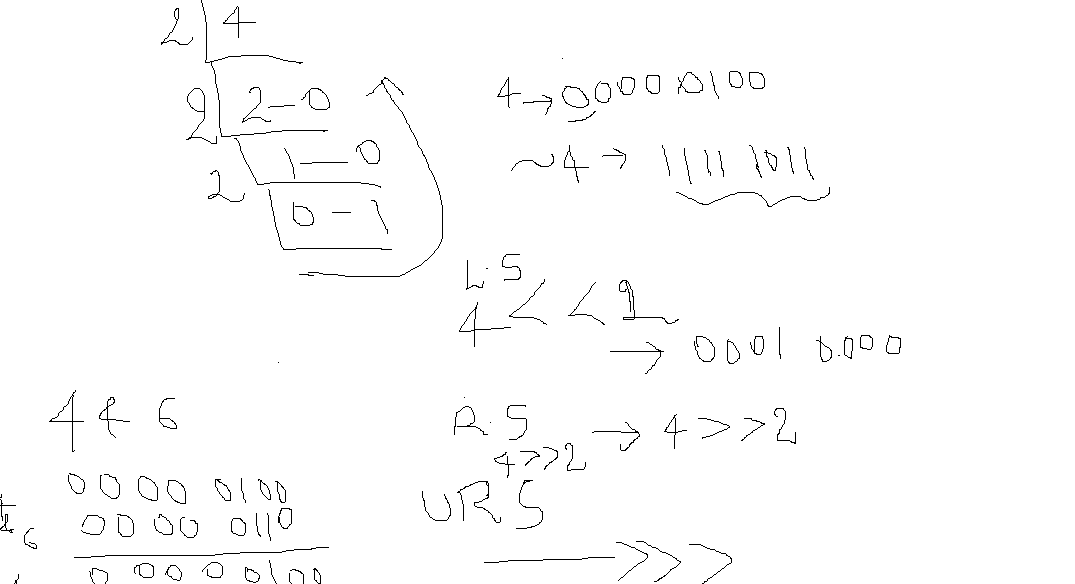
& Bitwise AND

^ Bitwise exclusive OR

| Bitwise inclusive OR





****

**use of bitwise:**

http://www.leepoint.net/data/expressions/bitops.html

conditionalOperators/short circuit:

&& - if first expression is false then it wil not execute second expression.

||- if first expression is true then it wil not execute second expression.

**postfix increment/decrement:**

**value is incremented after executing the current statement**

**prefix increment/decrement:**

**value is incremented before executing the current statement**

//++c or c++ => c=c+1

**int** c=3;

System.***out***.println(c++);//3

System.***out***.println(c);//4

System.***out***.println(++c);//5

System.***out***.println(c);//5

**Object** class – is the **super** class for all classes in Java by default.

method of Object class:

toString

hashCode

equals

wait

notify

notifyAll

finalize

clone

**ConditionalStatements/Flow Control:**

if else:

can be single or more than one by using else if

switch case: data passed in side switch can be either int, byte, short,char, String, Enum

Byte, Character, Short,Character

switch(""){

case "":

case "":

...

default:

}

branching statements- break, continue

loops- for, while ,do while