**Java**

Programming Language

int a, b

a=10, b=20

int c = a+b

print(c)

C, C++, Java, Python, Ruby..

Low level language – Assembly language(Machine level language)

add r0,r1 – 1000 00 01

High Level languagues – Human understandable languages

C, C++, Java, Python, ruby, perl..

Compiler n Interpreters

**Compiler** – this takes entire code as input at once and generates Intermidiate object code

c, c++, scala, small talk

byte code/intermediate code is object code which is actually processes by virtual machine,

**Interpreter** – It takes single line or instructions as input and execute it

java, python

fast, consumes less memory, no intermediate code is generated

int add(int a, int b){

int sum = a+b;

return sum;

}

Java – as both compiled and interpreted language

**System Independent Language** – can be executed on any machine irrespective of which OS you have used to write and compile the code/program. Ex – Java, Python,

**System dependent Language** – need to be complied again if you want t o execute in some other machine with different OS. C, C++, smalltalk

JVM(Java Virtual machine) is a software system dependent but it makes java system independent language

Java Features:

Simple

ObjectOriented – Everything is written under 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 applications which desire system Independent feature

Robust – strong, stable, supports good exception handling mechanisms and it handles memory allocation n deallocation internally(by JVM)

Secure

Multithreaded

public class Test(){

public int add(){

int c = int a+ int b;

return c;

}

public int sub(){

int c = int a- int b;

return c;

}

}

Test t = new Test();

int x = t.add();

int y = t.sub();

**Java - JDK, JRE, JVM**

**Download, run, set environment path**

java follows the principle WORA - write once run anywhere

Java is case sensitive

Comments

Single line comment - //

Multi Line comment - /\* lines \*/

Class – Is a blueprint that defines variables(properties) and methods(actions)

Object – is a instance of a class which allows to access properties and methods of class

Object is the thing which you see in real world

Animal – class

cat, dog, rat, tiger - objects

Furniture(Class) – chairs, bed, sofa

Mobile(Class) – Iphone, Samsung, blackberry(Objects)

Class Animal(){

//variables – properties

String name = “pintu”;

String color;

String breed;

//methods

public void eatFood(){

}

public void makeNoise(){

}

Animal dog = new Animal();

dog.color =”White”;

dog.breed = “pug”;

dog.eatFood();

Animal cat = new Animal();

cat.name = “Kitty”;

Mobile – iphone, Samsung, motog

Properties/variables - color, size, nofofbuttons, …

Function/Methods – makeCall(), doMessage(), writeEmail()..

Class Mobile(){

String color;

int size;

int noofButtons;

public void makeaCall(){

logic;

}

public void writeEmail(){

logic;

}

Mobile iphone = new Moblie();

iphone.color = “White”;

iphone.size = 10;

iphone.noofButton = 1;

public class MainMobile(){

public void static main(String[] args){

Mobile iphone = new Moblie();

iphone.color = “White”;

iphone.size = 10;

iphone.noofButton = 1;

}

}

public – accessable from anywhere

static – can be accessedwithout creating object

to call static method or variable we do not need to create object

void – method is not returning anything to whoever is calling

main – is a method that jvm looks for to start java program execution

String[] agrs – String array, methods input parameters

int a = 10;

int[] a = {10, 20, 30}

Using Notepad writing a java program n executing

open notepad – write a java program

save as – helloworld.java

go to command prompt – javac helloworld.java(this will compile n create a .class file which is machine understandable code)

for executing – java helloworld

**Naming Conventions:**

Java is case sensitive

package names – lower case- com.wbl.oops

variable, methods – camelCase – makeNoise(), eats(), nameAnimal, name

Classname, InterfaceNames – init caps – Student, StudentDetails

Constants – PI =3.14;

PI\_VALUE = 3.145

**Rules for Idenfiers/Names:**

Names should not start with numbers, it can only start with alphabets,$ , \_

It can include Numbers in between

Identifiers can have alphabets, numbers, $, \_

Each identifier should have atleast one alphabet

Name of the class, method, variable should be any java keywords

**Access Modifiers** (class, method, variable)

Public –Can be accessed from anywhere in the project, if it is accessed outside the package we need to add import

Private – can be accessed only with in the class where it is declared

Protected – can be accessed within the same package **& also the subclass of different packages**

Default/Package – accessed within the package

A{

int i=10;

string name=”Ali”

}

B extends A{

system.out.println(i);

system.out.println(name);

}

Wider 🡪 Narrow Scope

public, protected, default, private

**DataTypes:**

**Primitive**: it is not declared / created using new keyword, it is stored in heap memory.

int i= 10;

float i = 23.898;

String str = “Hello”

**Object**: Other than primitive all remaining are objects in java.

String is a class. It is not primitive and we create object for that class

**Primitive DataTypes:**

byte – int with 8bits (8bit representation) : -128 to 128

8(decimal number) – 00001000(binary represtation)

short – int with 16 bits

int - int with 32 bits

long - int with 64 bits

float – decimal with 32 bit

double - decimal with 64 bit

char – any keyborab releated characters, alphabets, special characters, numbers

char c = ‘f’;

Boolean – true/ false

60 students

byte countOfStu;

signed - 128 to 127 – based on MSB I can say whther the value is –ve or +ve

MSB -0 = +ve

MSB -1 = -ve

unsigned - 0-255

byte🡪short🡪int🡪long🡪float🡪double

Type Casting:

Explicit or downcasting: - giving lower range value to higher range which is done by default

Implicit or upcasting : giving higher range to lower range

**Syntax for declaring a variable:**

accessmodifier type refvariable= value;

public int i = 10;

int i = 10;

type refvariable = value;

**Syntax for declaring a method:**

accessmodifier returntype methodname(input parameter);

public void test(String input){

}

**Constructor:**

Constructor is similar to method in java but it will not return anything, it will have same name as calss name.

it is called by default whenever we create object.

it can take parameters which is a parameterized constructor.

Purpose of constructor – it will create an empty frame in memory to reserve certain memory

which is needed for object.

Uses of Constructor-

Initialize nay data req for the object because it will be called initially when we create object.

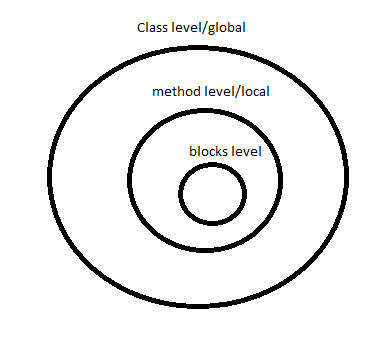
A class can have more than one constructor with same class name, but different inout parameters which is called constructor overloading.

**Scopes:**

Outer scope/global scope/ class level

Method level scope/Inner

Block level scope



**OOPS Concepts:**

**Encapsulation** : Hiding the implementation behind the interface, binding the variables & methods together by making the variables as private & giving the public access by getters or setters methods.

Inheritance, overriding, super

Polymorphism

Abstraction

abstraction class, interface

static, final

Method Overloading, Overriding

Inheritance: to avoid code redundancy.

parent class A{

Public int i=10;

public String str;

public void test(){

}

}

Child class B extends A{

public int iChild;

public int i=30;

@override

public void test(){

print(i); // child class

print(super.i);//thevalue wt is there in the parent class

}

public static void main(String[] args){

B b = new B();

b.iChild =10;

b.i = 20;

}

public A{

int i=10;

}

public B extends A{

int i=20;

}

public C extends B{

super.i

}

**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

**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

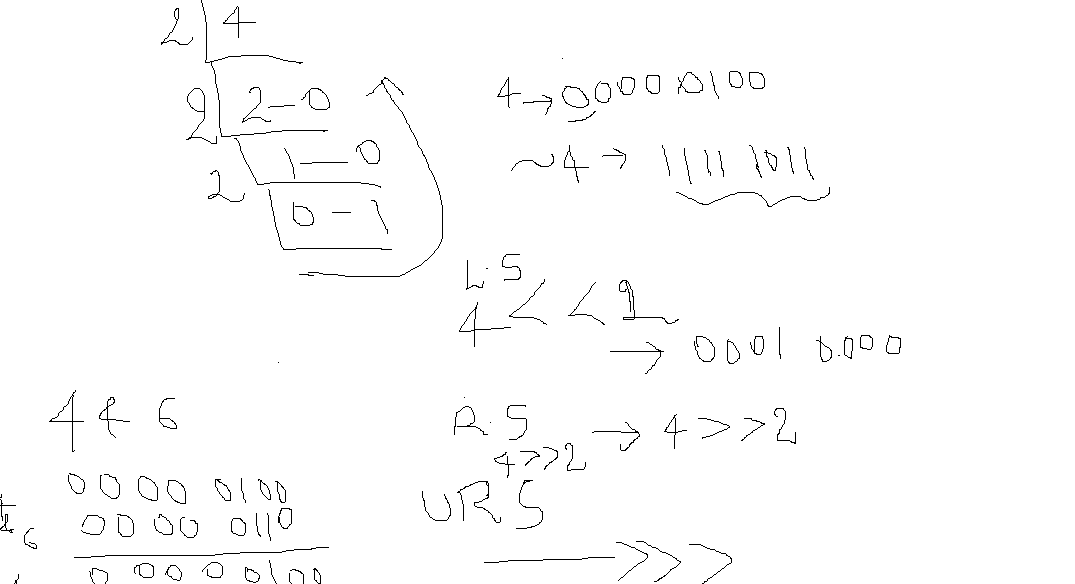
>> Signed right shift

>>> Unsigned right shift

& Bitwise AND

^ Bitwise exclusive OR

| Bitwise inclusive OR

****

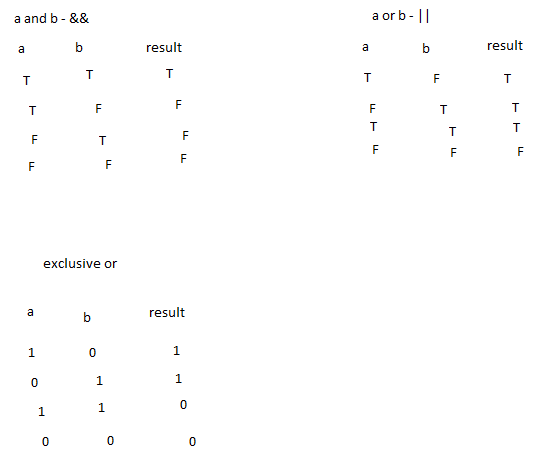
**use of bitwise:**

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.



4 – 00000100

~4 = 11111011

>> = 00000010

<< = 00001000

**Conditional Statements:**

if else, switch case

if(condition){ logic}

elseif(condition){}

else{logic}

SwitchCase:

data ispassed in side switch can be either int, byte, short, char, long, string, enum

switch(“input”)

case “input” : logic

break;

case “apple” : logic

break;

**Loops:**

while – it checks for the condition first and then it will go inside the loop

initialization;

While(condition){

logic;

increment;

}

do..while – it will execute the loop once and then checks for the condition, if it satisfies

it will go inside the loop

for

for(initialization ; condition ; increment)

for(int i=1 ; i<=10 ; i++){

logic;

}

Assignments:

1. given number check it is even or odd
2. print all the multiples 5 between 1 to 100

**Polymorphism**:

Ability to exist in many forms

Same class reference can hold multiple objects

//Animal object 1

Animal **animal** = **new** Animal();

animal.setName("Punto");

animal.makeNoise();

//Animal object 2 to same animal reference

**animal** = **new** Animal();

animal.setName("Prince");

animal.makeNoise();

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

//parent class reference and child class object

**animal** = **new** Dog();

animal.makeNoise();

Whenever we have is-a relationship (inheritance) object type determines which method to be called at run time(dynamic polymorphism)

At compile time reference type decides which method to be called(compile time polymorphism).

Dog dog = **new** Dog();

dog.catchThieves();

dog.catchThieves("Robert");

dog.catchThieves(6);

|  |  |
| --- | --- |
| 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 excpetions, 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…  Public->protected->default->private |
| 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 the implementation behind the interface.

In Java, we achieve abstraction using abstract class and interface.

**Abstract Class:**

* Abstract class is a class with abstract keyword; it can have both abstract and concrete methods.
* abstract methods are methods without implementation/method body.
* concrete methods are methods with implementation/method body.
* Using abstract class:

1. Whenever we want to provide common logic to all child classes we can use concrete methods and provide common logic.

2. Whenever we want to declare some methods (as rules) to be definitely implemented by all child classes we can use abstract methods.

* We cannot create an object for abstract class (there is no meaning to use abstract class without complete implementation)
* We can have variables/concrete methods inside abstract class with any scope(public , private, protected or default)
* abstract methods cannot be private
* abstract class can extend another abstract class
* abstract classes can have constructors.
* We cannot achieve multiple inheritance through abstract classes.

**Interface:**

**Before Java8- interface is 100% abstract class, which is like a contract**

**But from Java8- interface allows default and static methods- which are like concrete methods.**

Interface allows abstract methods by default without any abstract keyword(by default it adds abstract keyword)

Interface cannot have a constructor

We cannot create objects for an Interface

Interface cannot have any general variables other than constants(ALL VARIABLES IN INTERFACE ARE public, static and final by default)

**We can use interfaces when we have to declare some common logic to all child classes(using default methods) and also declare abstract methods(rules) which should be implemented by all child classes,**

**and when we want to achieve multiple inheritance.**

**Java supports only multilevel inheritance when it comes to classes.**

**MultiLevel Inheritance:**

**class A{**

**void a(){**

**}**

**}**

**Class B extends A{**

**void b(){**

**}**

**}**

**class C extends A{**

**//inherits both a and b**

**}**

**Java supports only multiple inheritance through interfaces.**

**Multiple Inheritance:**

**interface A{**

**a();**

**}**

**interface B{**

**b();**

**}**

**class C implements A,B{**

**a(){}**

**b(){}**

**}**

Training

QaTraining

UITraining

Car

SportsCar

NormalCar

|  |  |
| --- | --- |
| **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 |

**deadly diamond death problem**

**final**

**wrapper classes**