# OOPS concepts

Object

Class

Inheritance

Polymorphism

Abstraction

Encapsulation

# NAMING CONVENTIONS

### Class

It should start with uppercase letter

It should be a noun

public class Employee

{

//code snippet

}

### Interface

It should start with uppercase letter

It should be an adjective

interface Printable

{

//code snippet

}

### Method

It should start with lowercase letter.

It should be a verb like main(), print(), run().

If the name contains multiple words then start with lowercase letter followed by an uppercase letter

actionPerformed().

### Variable

It should start with lowercase letter.

It should not start with special characters like like & (ampersand), $ (dollar), \_ (underscore).

If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.

Avoid using one-character variables such as x, y, z.

class Employee

{

//variable

int id;

//code snippet

}

### Package

It should be a lowercase letter such as java, lang.

If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

package com.javatpoint; //package

class Employee

{

//code snippet

}

### Constant

It should be in uppercase letters such as RED, YELLOW.

If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY.

It may contain digits but not as the first letter.

class Employee

{

//constant

static final int MIN\_AGE = 18;

//code snippet

}

# Polymorphism

## Dynamic method dispatch:

If child and parent class are having same method name ,return type and parameters then

If we create reference variable to the parent class and it refers to child class object, the call to the method that exist in parent and child class will pick the child class method at run time.

**class** parent{

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

{

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

}

}

**public** **class** check **extends** parent{

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

{

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

}

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

// **TODO** Auto-generated method stub

parent c1 = **new** check();

c1.test();

}

}

Output:

child class

This is applicable only to methods and not variables

**class** parent{

**int** x=50;

}

**public** **class** check **extends** parent{

**int** X =100;

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

// **TODO** Auto-generated method stub

parent c1 = **new** check();

System.***out***.println("value is "+c1.x);

}

}

Output: 50

You can only call parent class methods/variables/common variables and methods in parent and child classes.

If a method does not exist in parent class and you try to call it with parent class reference variable then it will throw an error/it wont recognize the method or variable.

**class** parent{

**int** x=50;

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

{

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

}

**public** **void** test3()

{

System.***out***.println("Method only in parent class");

}

}

**public** **class** check **extends** parent{

**int** y =100;

**public** **void** test2()

{

System.***out***.println("Method only in child class");

}

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

{

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

}

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

// **TODO** Auto-generated method stub

parent c1 = **new** check();

System.***out***.println("value is "+c1.y);//error

C1.test2(); //error

C1.test3();// executes

System.***out***.println("value is "+c1.x) //executes

}

}

## Dynamic parameter count

<https://www.geeksforgeeks.org/variable-arguments-varargs-in-java/>

# Constructor:

It has same name as method but no return type. Whenever an object is created using new or another way, then it creates an object and invokes constructor to initialize the newly created object.

### Default constructor

which the user don’t mention. It is created by default.

### No-argument constructor

when ever the user declares the constructor.

Myclass

{  
Myclass()

{

Syso(“print”);  
}

}

### Parameterized constructor

where the user sends parameters to the constructor.

public class Employee {

int empId;

String empName;

//parameterized constructor with two parameters

Employee(int id, String name){

this.empId = id;

this.empName = name;

}

void info(){

System.out.println("Id: "+empId+" Name: "+empName);

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

Employee obj1 = new Employee(10245,"Chaitanya");

Employee obj2 = new Employee(92232,"Negan");

obj1.info();

obj2.info();

} }

#### Note:

if you parameterize a constructor and tries to invoke without parameter or mismatched parameters then it throws compilation error, as default constructor wont be created once you parameterize a constructor.

class Example3

{

private int var;

public Example3(int num)

{

var=num;

}

public int getValue()

{

return var;

}

public static void main(String args[])

{

Example3 myobj = new Example3();// **this throws error**

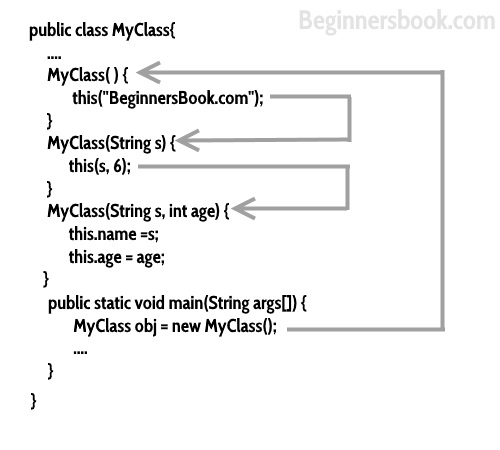
System.out.println("value of var is: "+myobj.getValue());

}

}

### Constructor Chaining with example

Calling a constructor from the another constructor of same class is known as Constructor chaining. The real purpose of Constructor Chaining is that you can pass parameters through a bunch of different constructors, but only have the initialization done in a single place. This allows you to maintain your initializations from a single location, while providing multiple constructors to the user. If we don’t chain, and two different constructors require a specific parameter, you will have to initialize that parameter twice, and when the initialization changes, you’ll have to change it in every constructor, instead of just the one.



#### Note:

this() should always be the first statement in **constructor** otherwise you will get this error message: **Exception in thread “main” java.lang.Error: Unresolved compilation problem: Constructor call must be the first statement in a constructor.**

**A constructor can have either this or super keyword but not both.**

class Employee

{

public String empName;

public int empSalary;

public String address;

//default constructor of the class

public Employee()

{

//this will call the constructor with String param

this("Chaitanya");

}

public Employee(String name)

{

//call the constructor with (String, int) param

this(name, 120035);

}

public Employee(String name, int sal)

{

//call the constructor with (String, int, String) param

this(name, sal, "Gurgaon");

}

public Employee(String name, int sal, String addr)

{

this.empName=name;

this.empSalary=sal;

this.address=addr;

}

void disp() {

System.out.println("Employee Name: "+empName);

System.out.println("Employee Salary: "+empSalary);

System.out.println("Employee Address: "+address);

}

public static void main(String[] args)

{

Employee obj = new Employee();

obj.disp();

}

}

### Super()

When ever child class constructor is invoked it first invokes parent class constructor, complier implicitly inserts super () at the beginning of the child class constructor.

class MyParentClass {

MyParentClass(){

System.out.println("MyParentClass Constructor");

}

}

class MyChildClass extends MyParentClass{

MyChildClass() {

System.out.println("MyChildClass Constructor");

}

public static void main(String args[]) {

new MyChildClass();

}

}

Output:

MyParentClass Constructor

MyChildClass Constructor

# Static

## Static variable

When a variable is declared as static then the variable is shared by all the instances. Only one memory allocation happens to the variable at the time the class is loaded in the memory. The value of static variable can be changed by the methods as per requirement, but different instances cannot have their own value.

**package** javatests;

**public** **class** Staticex1 {

**static** **int** *var*=5;

**public** **void** print()

{

System.***out***.println("value in printmethod is "+*var*);

}

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

// **TODO** Auto-generated method stub

Staticex1 s1 = **new** Staticex1();

*var* =10;

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

s1.print();

}}

Output: 10

value in printmethod is 10

#### Note:

Static variables are also known as Class Variables.

Unlike non-static variables, such variables can be accessed directly in static and non-static methods.

Static variables are initialized when class is loaded.

Static variables are initialized before any object of that class is created.

Static variables are initialized before any static method of the class executes.

Default values for static and non-static variables are same.

primitive integers(long, short etc): 0

primitive floating points(float, double): 0.0

boolean: false

object references: null

### Static Final variables:

The static final variables are constants.

public class MyClass{

public static final int MY\_VAR=27;

}

#### Note:

Constant variable name should be in Caps! you can use underscore(\_) between.

1) The above code will execute as soon as the class MyClass is loaded, before static method is called and even before any static variable can be accessed.

2) The variable MY\_VAR is public which means any class can use it. It is a static variable so you won’t need any object of class in order to access it. It’s final so the value of this variable can never be changed in the current or in any class.

final variable always needs initialization, if you don’t initialize it would throw a compilation error. have a look at below example-

public class MyClass{

public static final int MY\_VAR;

}

Error: variable MY\_VAR might not have been initialized

## Static method

A static method belongs to the class rather than the object of a class.

A static method can be invoked without the need for creating an instance of a class.

A static method can access static data member and can change the value of it.

Restrictions for the static method

#### There are two main restrictions for the static method.

* The static method can not use non static data member or call non-static method directly.
* this and super cannot be used in static context.

class A{

int a=40;//non static

public static void main(String args[]){

System.out.println(a);

}

Here in main method which is static we cannot use non-static member directly.

## Static Class

In java we cannot directly create static class. We can only use it for inner class in a nested class.

The static inner class can access only static data members of the outer class.