Modules:

* Java 8
* Hibernate
* Spring Core
* RESTful Webservices

Java: It is platform Independent & Object Oriented language

What is platform Independent?

* It can be run on any platforms (OS) without altering.
* It is write once and run anywhere.

What is object oriented programming language?

* It represents real world entities in the application, these real world entities are also called as Objects.
* An object will have properties and behaviours
* Properties: What object has.
* Behaviours: What object does.

Example of Objects: In a banking application - Customer, Account, Employee are all the objects it can have.

Customer: Properties are customerId, name, age, gender

Customer: Behaviours are withdraw(), deposit(), getDetails()

Employee: Properties are employeeId, name, salary

Employee: Behaviours are createCustomer(), deleteCustomer() and so on.

and etc.

Two basic building blocks of OOPs.

1. Classes: It is a blue print of an object or template of an object.
2. Objects: It is created from the class which is a real world entity.

How to create class?

Syntax:

class className {

object’s properties

object’s behaviours

}

Example:

class Employee {

int employeeId;

String employeeName;

.....

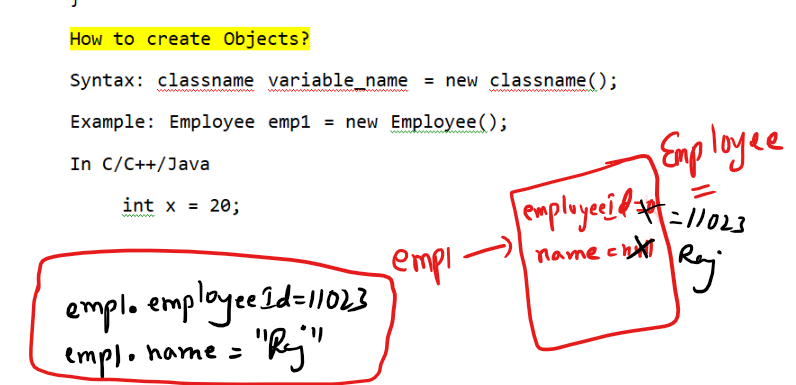
void takeComplaints() {

......

}

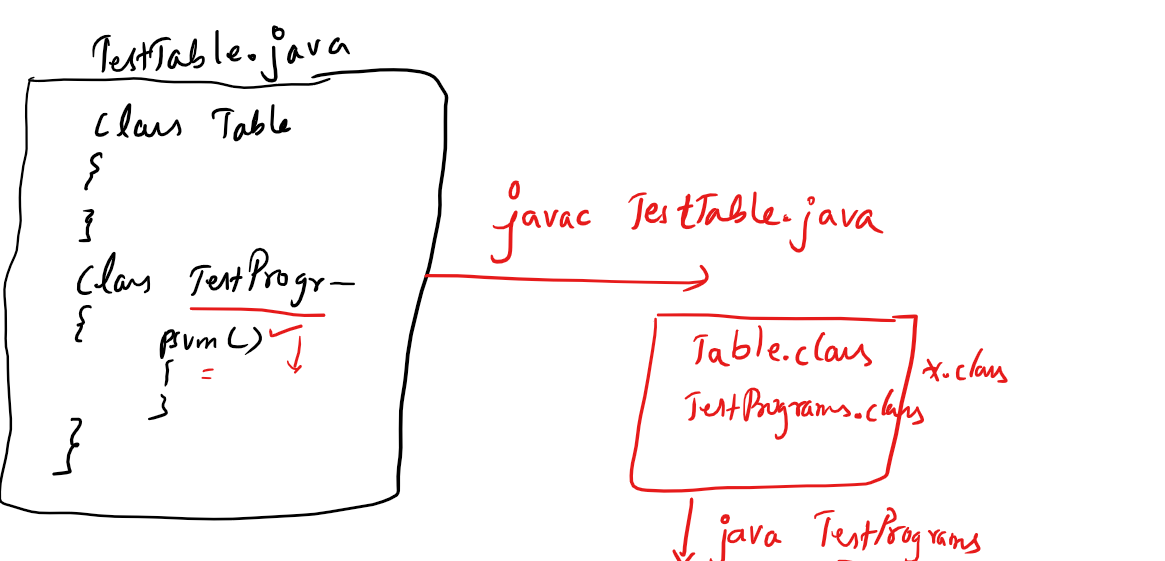
....

}



Setting Path:





Constructors: It is a kind of method whose name is same as class name but it will not have return type.

Note: if a class doesn’t have any constructor then compiler adds a default constructor.

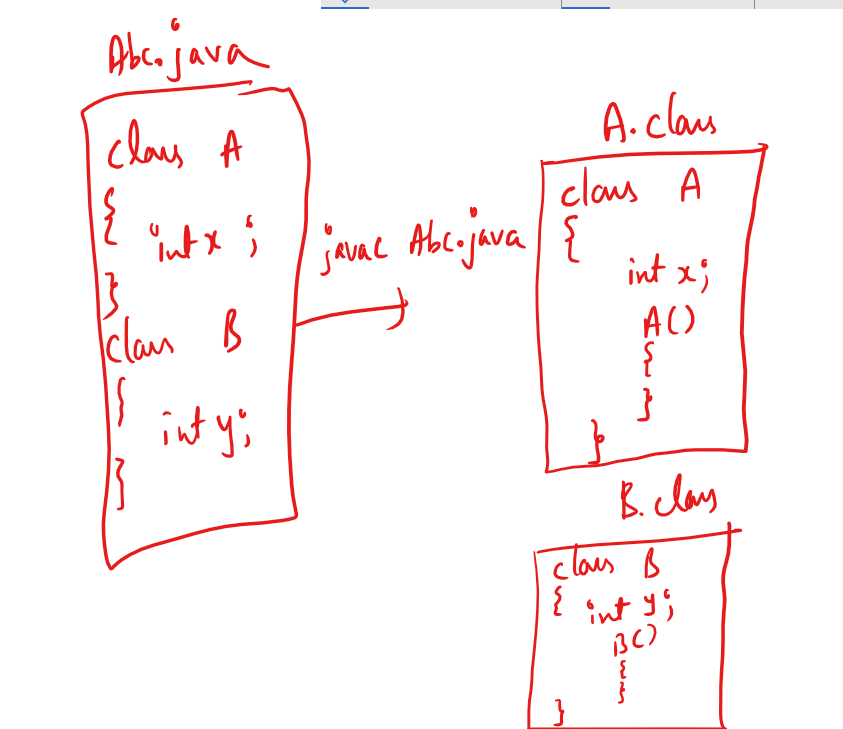
Note: If you add a constructor of your own then compiler doesn’t add a default constructor.

How the default constructor looks:

It looks same as class name followed by () i.e.,

Inside a class Table you will get a default constructor as Table(){

}



Note: A class can have any number of constructors i.e, you can overload the constructor.

Overloading: Same name but different set/type of parameters

i.e.,

Table(): Default constructor

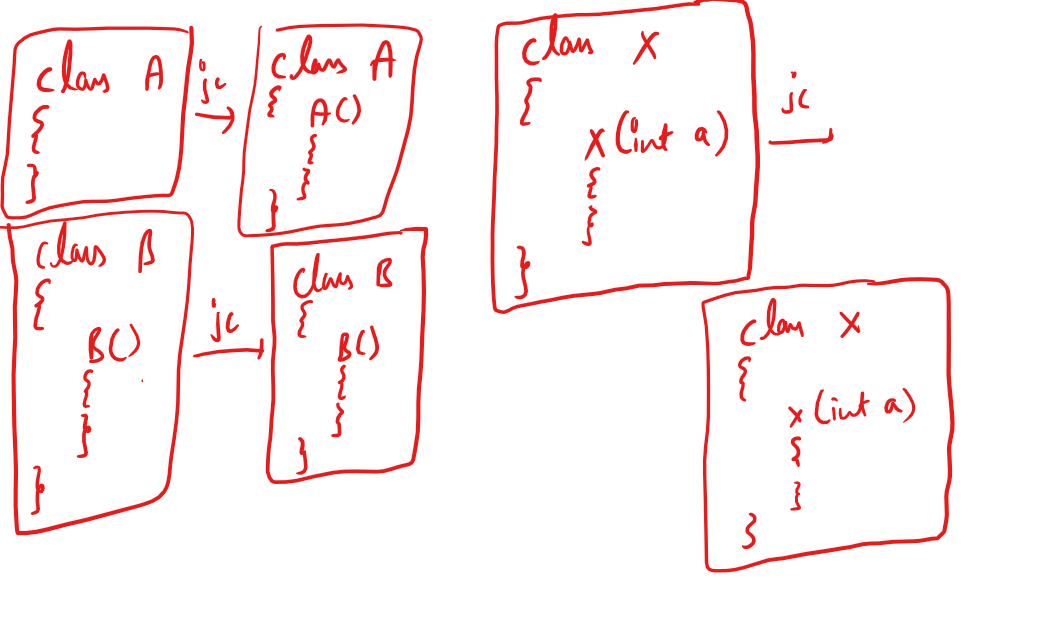
Table(int x, int y): 2 parameterized constructor

Table(double x, double y): 2 parameterized constructor

Table(int a, double b): 2 parameterized constructor

Table(double a, int b): 2 parameterized constructor

Table(int x): 1 parameterized constructor.



Constructor is called when you create object, so you can use it to initialize the variables while you create object.

You can find the informations present inside the class using javap command, stands for javaParse, which is used to disassemble the class,

Usage of javap command is same like java command i.e.,

* javap Table
* javap A
* javap TestPrograms

TestTable.java

// notepad TestTable.java

class Table {

int rows;

int cols;

Table() {

System.out.println("Table() created");

rows = 1;

cols = 1;

}

Table(int x, int y) {

System.out.println("Table(int, int) created");

rows = x;

cols = y;

}

void calculateCells() {

int r = rows \* cols;

System.out.println("Result = " + r);

}

}

class TestPrograms {

public static void main(String args[]) {

Table t1 = new Table(30, 40); // Table(int, int){}

Table t2 = new Table(10, 40);

Table t3 = new Table();

t3.rows = 100;

t3.cols = 200;

t1.calculateCells();

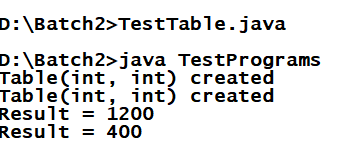
t2.calculateCells();

t3.calculateCells();

}

}

Output:



TestEmployee.java

class Employee {

int empId;

String empName;

double salary;

String gender;

Employee(int id, String name, double sal, String g) {

empId = id;

empName = name;

salary = sal;

gender = g;

}

void raiseSalary() {

salary = salary + (salary \* 0.05);

}

void updateSalary(double sal) {

salary = sal;

}

void display() {

System.out.println(empId+" "+empName+ " "+salary+ " "+gender);

}

}

class TestEmployee {

public static void main(String[] args)

{

Employee e1 = new Employee(101,"Rahul",20000,"Male");

Employee e2 = new Employee(102,"Jennifer",30000,"Female");

Employee e3 = new Employee(103,"Alex",10000,"Male");

e1.display(); e2.display(); e3.display();

System.out.println("------Raise Salary Request--------------");

e1.raiseSalary(); e2.raiseSalary(); e3.raiseSalary();

System.out.println("------Display After Raise--------------");

e1.display(); e2.display(); e3.display();

System.out.println("--------Update Salary Request------------");

e1.updateSalary(50000); e2.updateSalary(50000);

e3.updateSalary(50000);

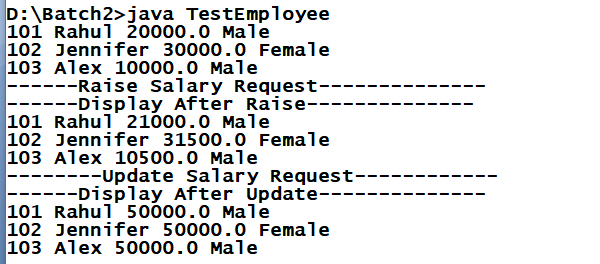
System.out.println("------Display After Update--------------");

e1.display(); e2.display(); e3.display();

}

}

Output:



The keyword this: This is a current object reference, it is used to avoid conflicts between local variable names and global variable names when their name’s are same

this keyword can be used inside Constructor and Non-Static methods.

Note: Using this keyword inside a static method is a compilation error, because static members are accessed without creating objects.

Like this keyword one more keyword called super also not allowed inside static methods.

TestEmployee.java

class Employee {

int empId;

String empName;

double salary;

String gender;

Employee(int empId, String empName, double salary, String gender) {

this.empId = empId;

this.empName = empName;

this.salary = salary;

this.gender = gender;

}

void raiseSalary() {

salary = salary + (salary \* 0.05);

}

void updateSalary(double salary) {

this.salary = salary;

}

void display() {

System.out.println(empId+" "+empName+ " "+salary+ " "+gender);

System.out.println(this.empId+" "+this.empName+ " "

+this.salary+ " "+this.gender);

}

}

class TestEmployee {

public static void main(String[] args)

{

Employee e1 = new Employee(101,"Rahul",20000,"Male");

Employee e2 = new Employee(102,"Jennifer",30000,"Female");

Employee e3 = new Employee(103,"Alex",10000,"Male");

e1.display(); e2.display(); e3.display();

System.out.println("------Raise Salary Request--------------");

e1.raiseSalary(); e2.raiseSalary(); e3.raiseSalary();

System.out.println("------Display After Raise--------------");

e1.display(); e2.display(); e3.display();

System.out.println("--------Update Salary Request------------");

e1.updateSalary(50000); e2.updateSalary(50000);

e3.updateSalary(50000);

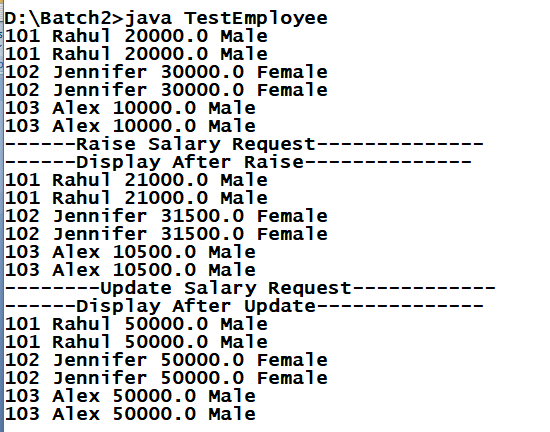
System.out.println("------Display After Update--------------");

e1.display(); e2.display(); e3.display();

}

}

Output:



Static keyword: static is used on variables & methods to access without creating objects.

When you want to access some members inside the class without instantiating it then you can have static variables & methods.

You can access static members using class names.

example:

Calculator.java

class Calculator {

static int add(int x, int y)

{

return (x + y);

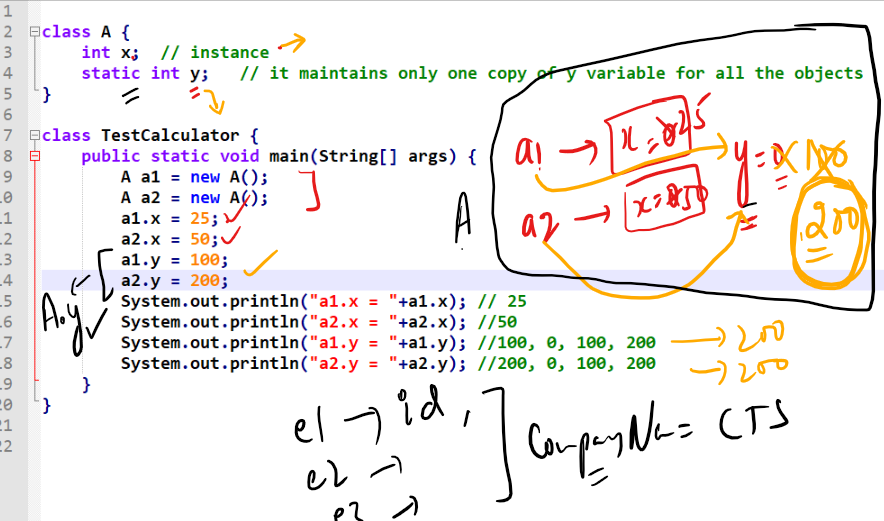
}

}

// to call add method you don’t need to create object of Calculator.

int result = Calculator.add(30, 40);

result value will be 70.

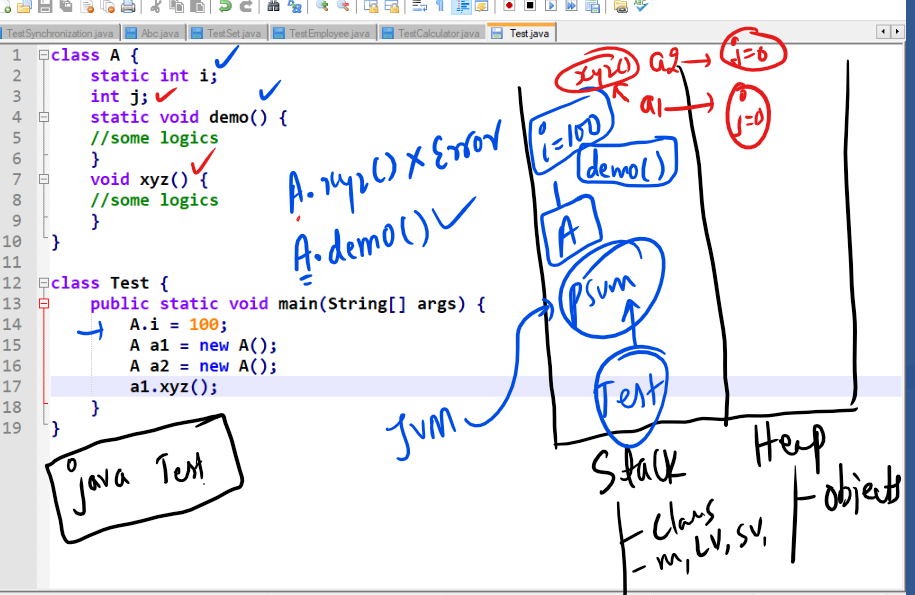


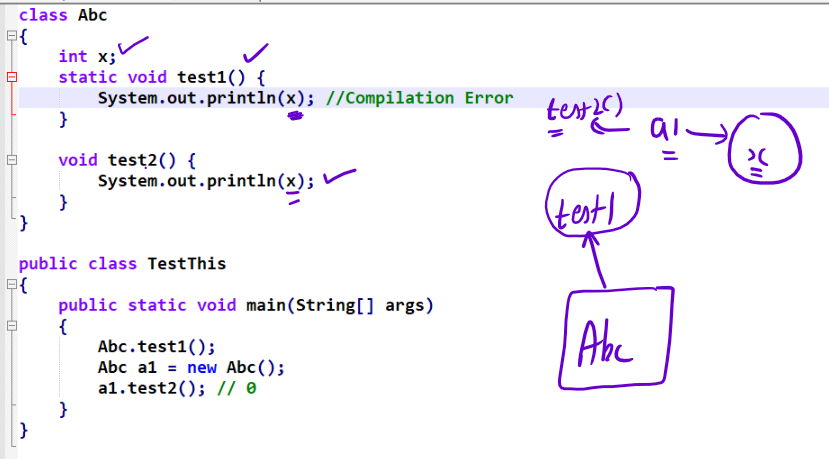
static members can be accessed either through class names or reference variables.

i.e.,

A.y or a1.y

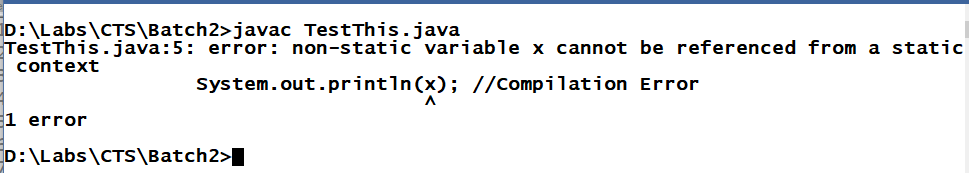
Memory management





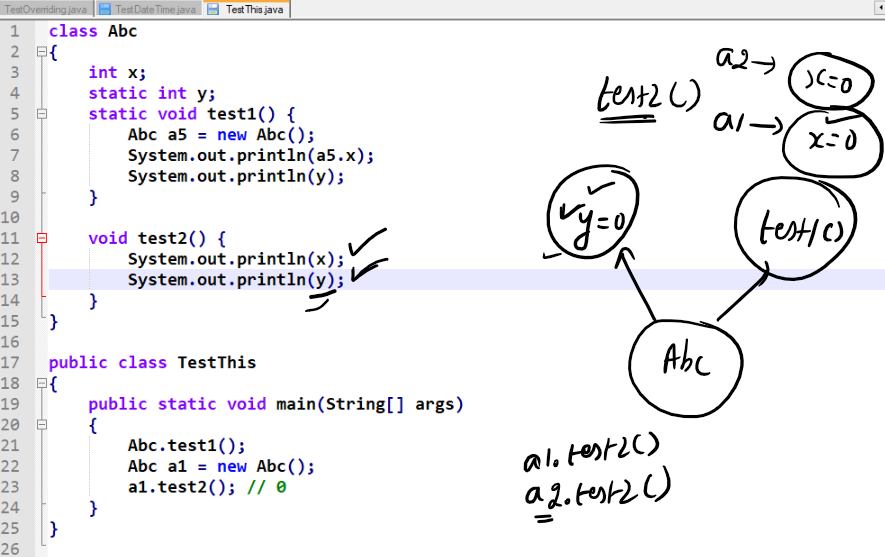
Note: static methods must not access non-static variables directly, but you can use object reference and access the non-static variable

Output:



Note: Static methods cannot access non-static members directly, whereas non-static methods can access static or non-static members directly.

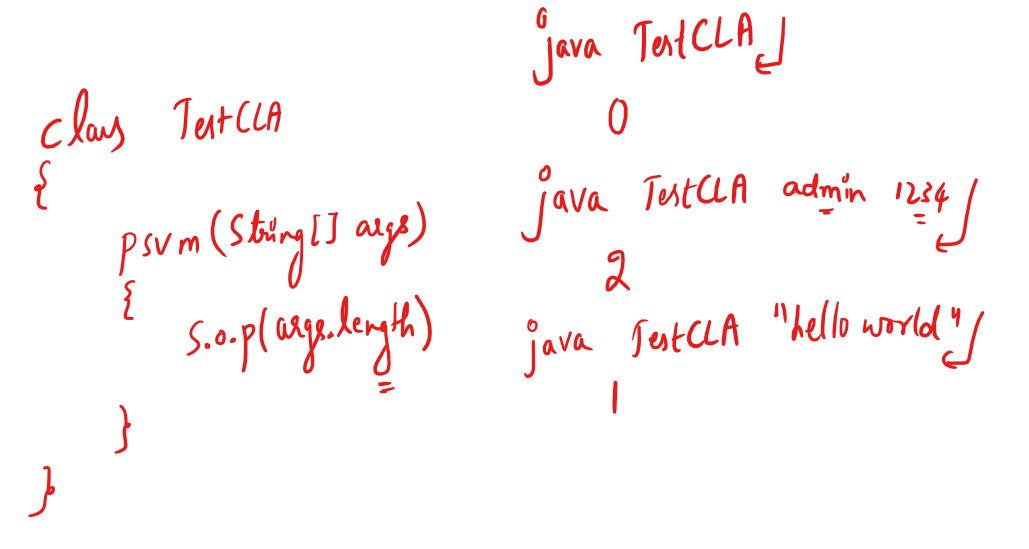
Example:

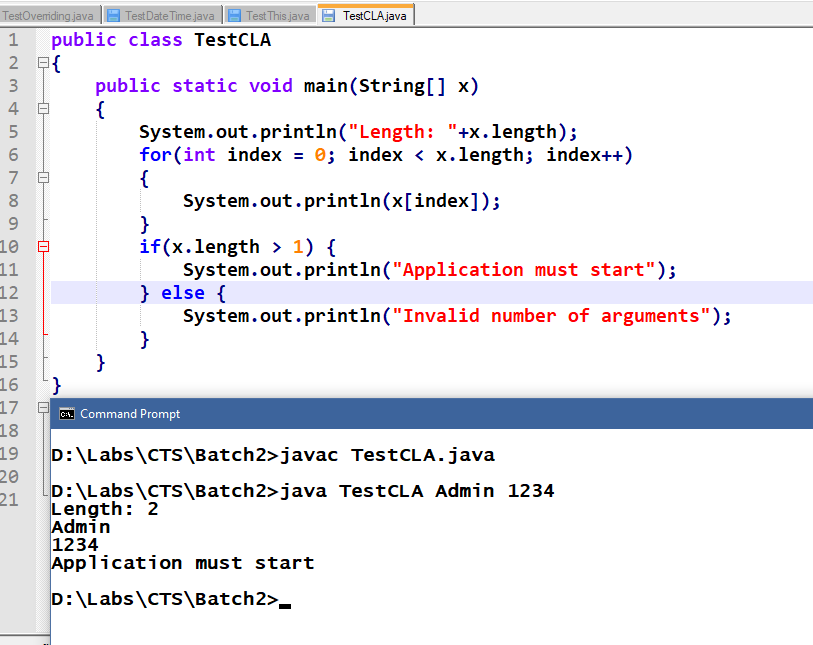


Command Line Argument:

It is an argument you can pass to the main method while running your code, String[] args in your main method is a command line argument.

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

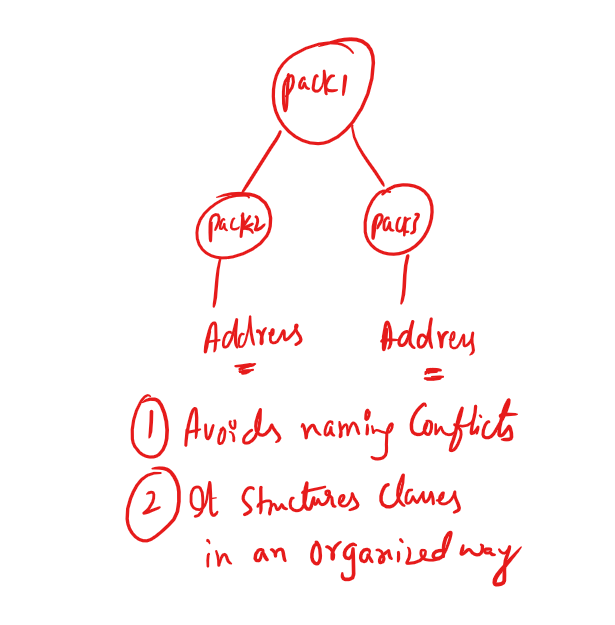
Note: [] bracket you can have after the data type or after variable name in an array.  




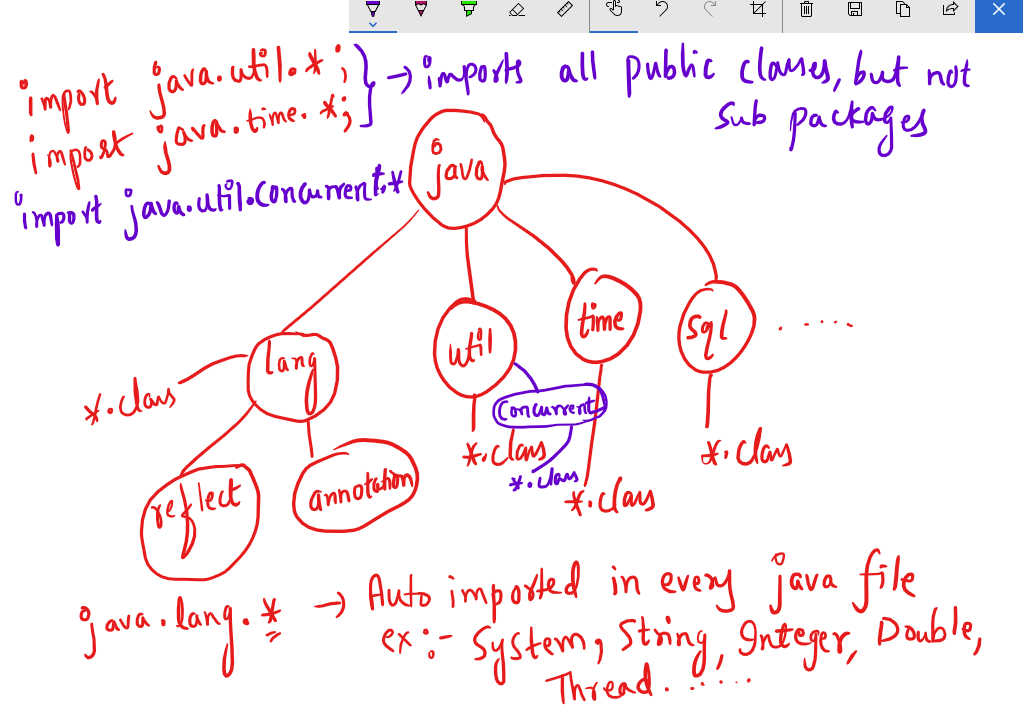
Packages in Java

What are packages?

They are like folders in Java, which will have collection of classes and sub packages.



In Java there are lot of predefined packages like



Note: import java.util.Scanner; will import only Scanner class whereas import java.util.\*; will import all the public classes.

Some of the predefined classes in java.lang are

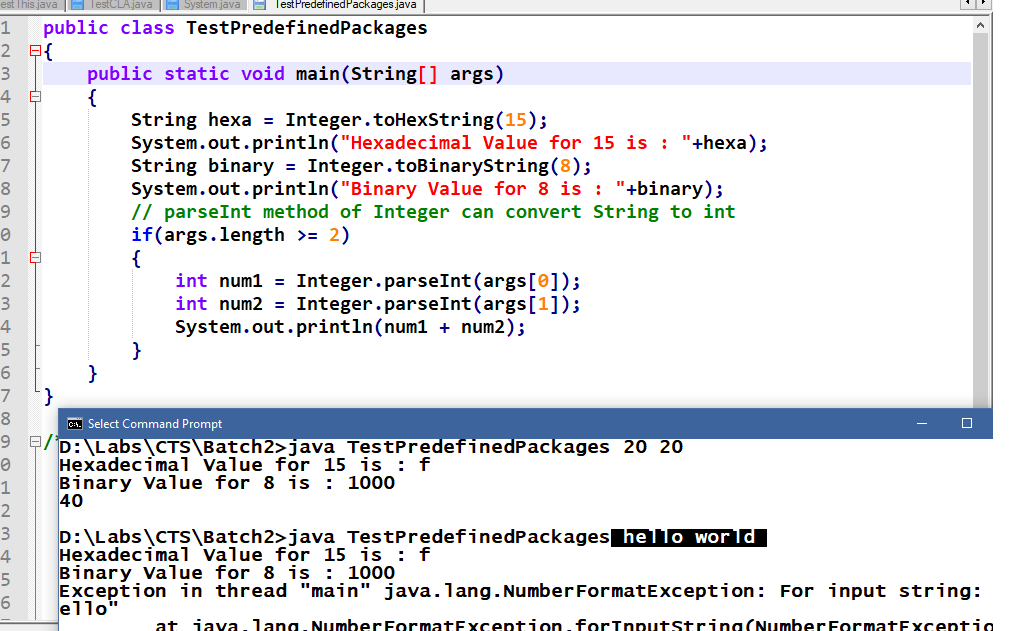
Integer, String, Float, Double, Exception and many more

javap java.lang.Integer will list

public static String toHexString(int)

public static String toBinaryString(int)

.....



Enhanced for loop or for each loop

It is same like traditional for loop but it simplifies the iteration, it can be used only if you are iterating in forward direction and all the elements.

Syntax:

for (type var : collections) {...}

Here collections are variables that holds array or collection of elements, type var must match to the collection type

int[] elements = {10, 50, 40, 30, 20};

for(int temp : elements) {

// now temp picks each item in the array till it iterates over all the elements.  
}

String[] args, a command line argument can also be iterated using for each loop i.e.,

for(String temp : args) {/\* some code \*/}

You can also declare an array using new operator as below:

int[] elements = new int[5];

Now elements will occupy 5 memory to store 5 integer values, by default each index elements will be having 0 (zero) you must initialize each index as below:

elements[0] = 20; elements[1] = 40 and so on.

You can also declare an array of Complex data types as below:

Employee[] employees = new Employee[4];

here you can have 4 employee objects in employees array, but initially all the index will have null value.

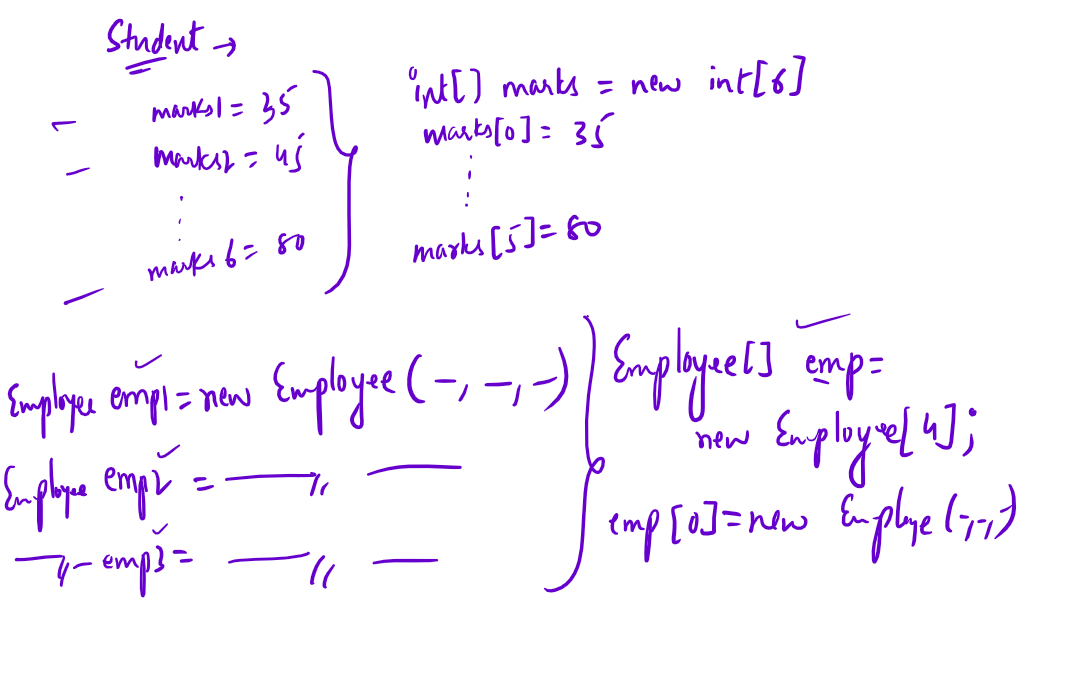
you can initialize each index by creating object.

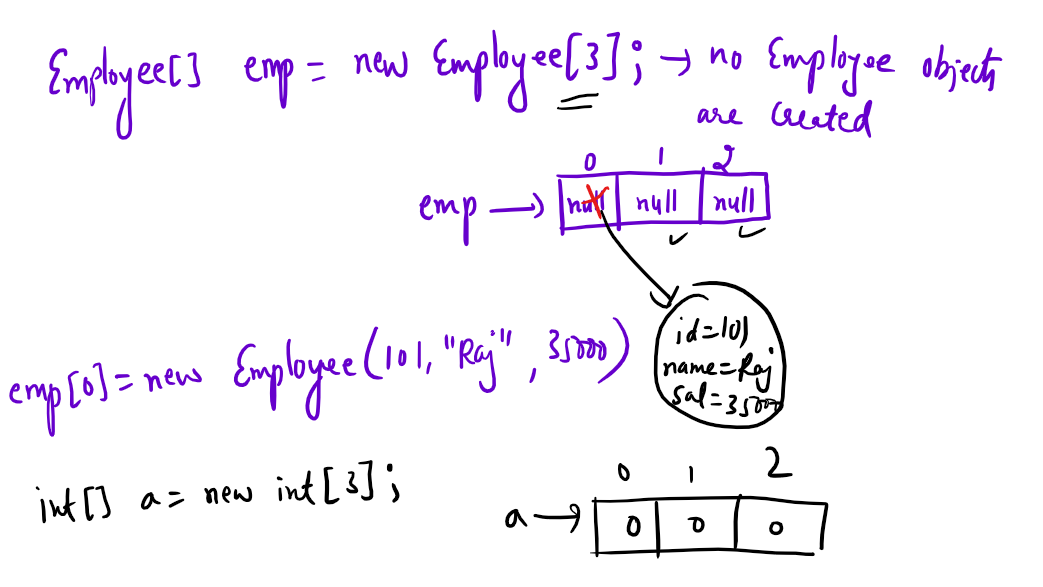
employees[0] = new Employee(101, “Rahul”, 35000);

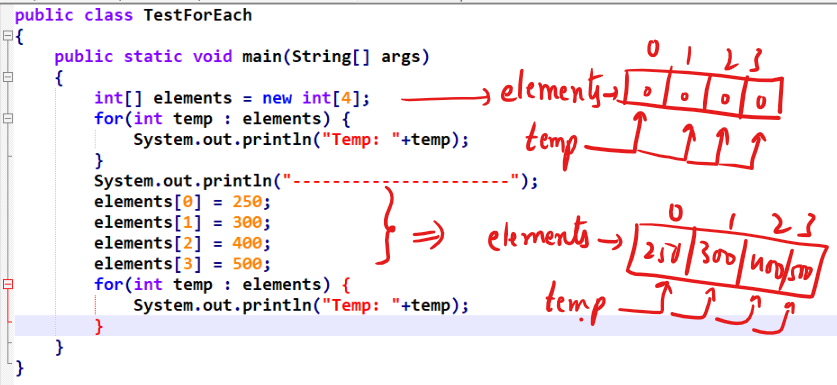
employees[1] = new Employees(102, “Vikram”, 40000);

and so on.

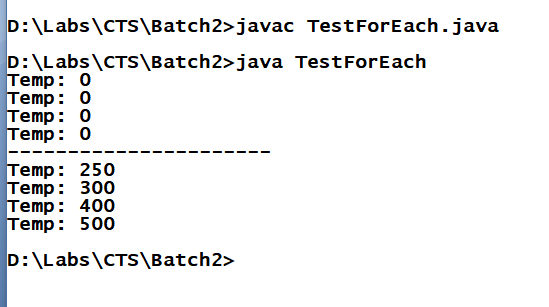
It is required if you want to maintain multiple Employee objects. Because creating 4 reference variables of same type is of no use instead you can have an array of a particular type and mention the size.



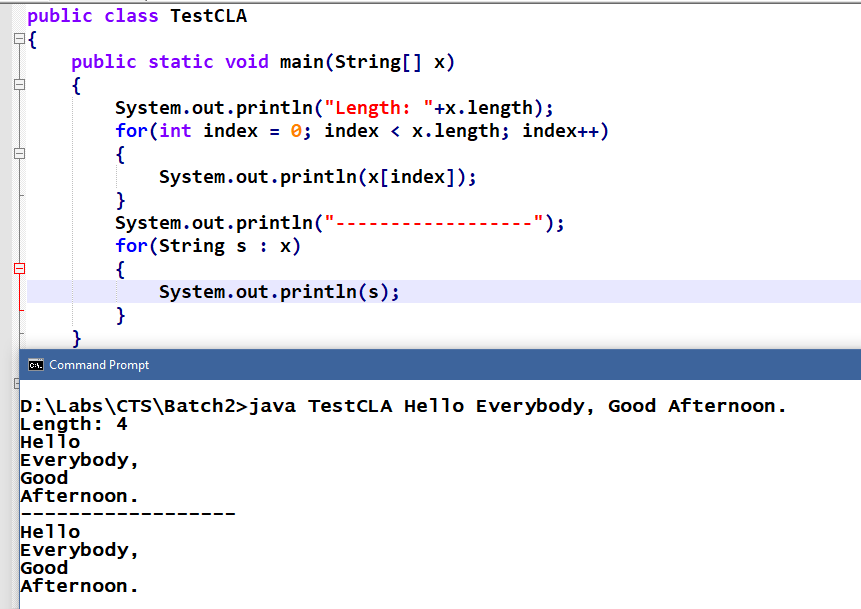




Output:



You can apply for-each loop on command line arguments as well as shown below:



OOPs concepts:

Encapsulation

Inheritance

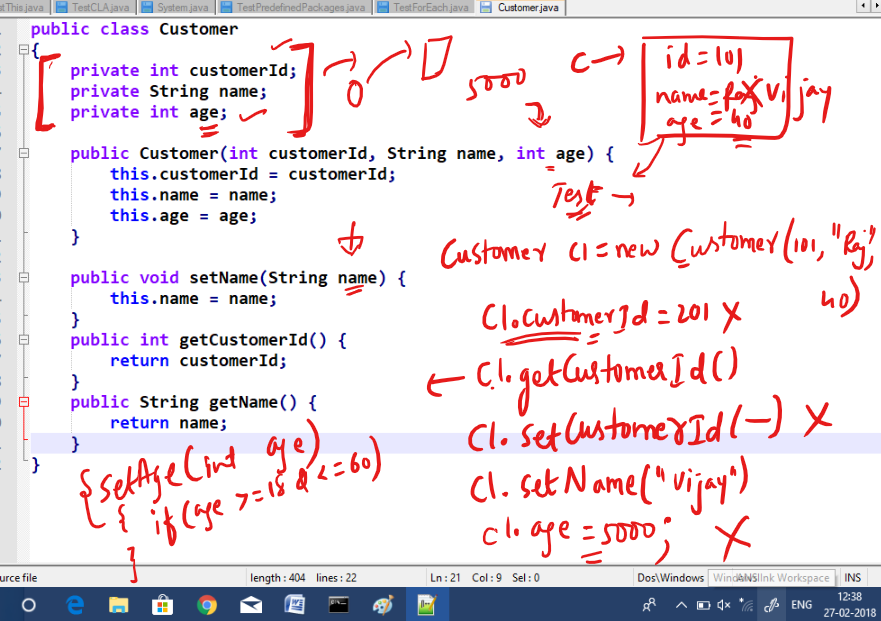
Polymorphism

Abstraction

What is Encapsulation?

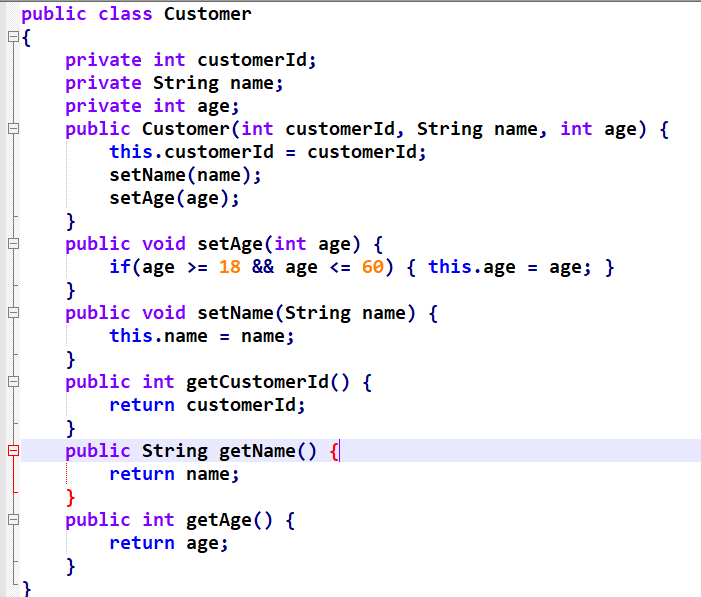
It is used to hide the variables using private so that you can access only through methods which are public.

An encapsulation will allow you to bind variables and methods in a single unit called class and a fully encapsulated class will have private variables & public methods which are also called as Java Beans.

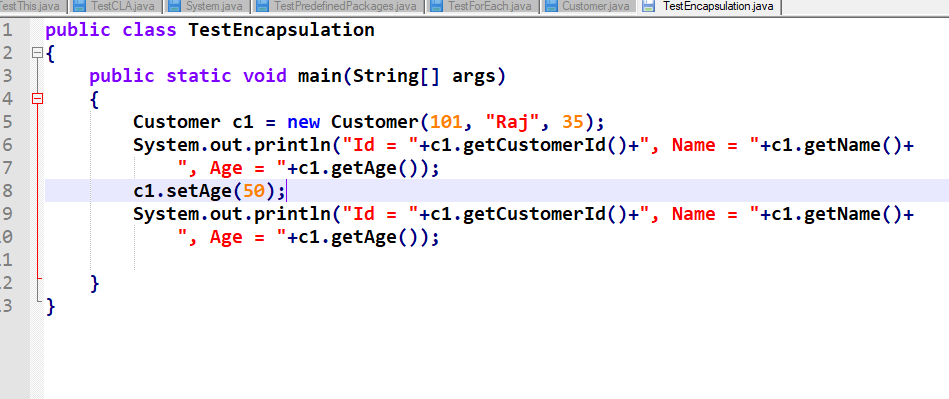


By making private you can have more control over the variables value.

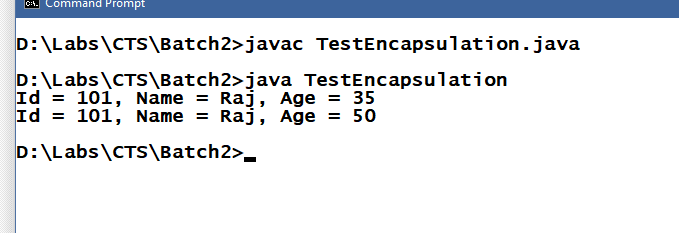
Customer.java



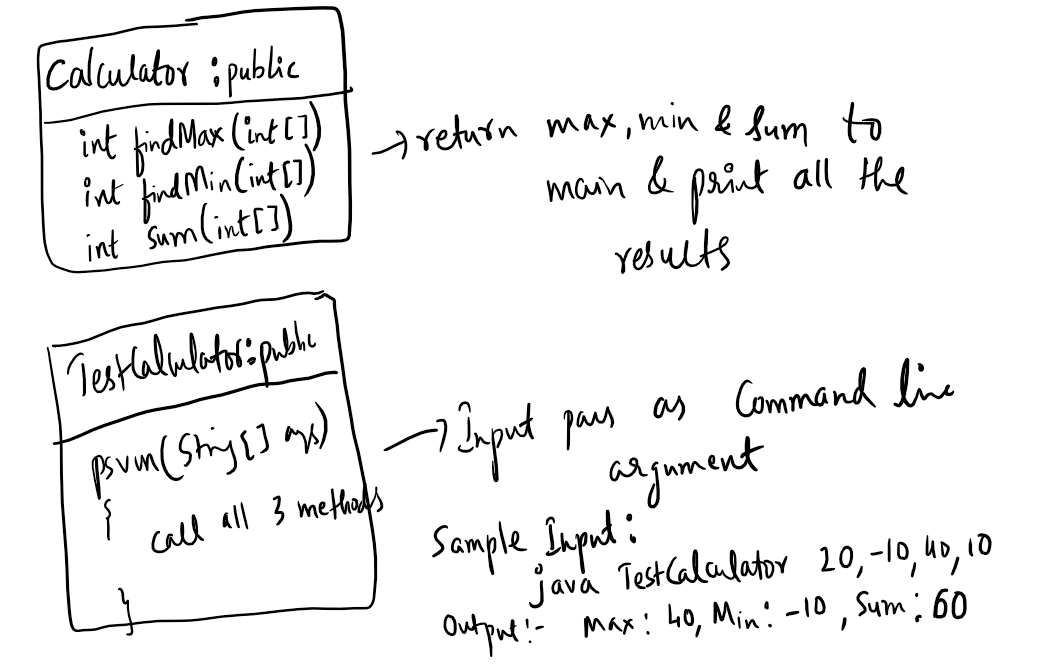
TestEncapsulation.java



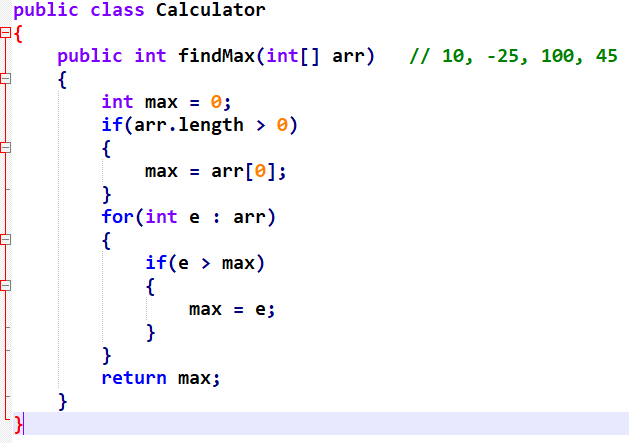
Output:



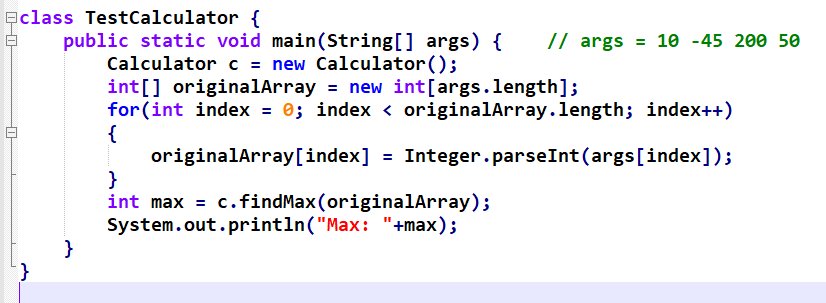
Assignment:



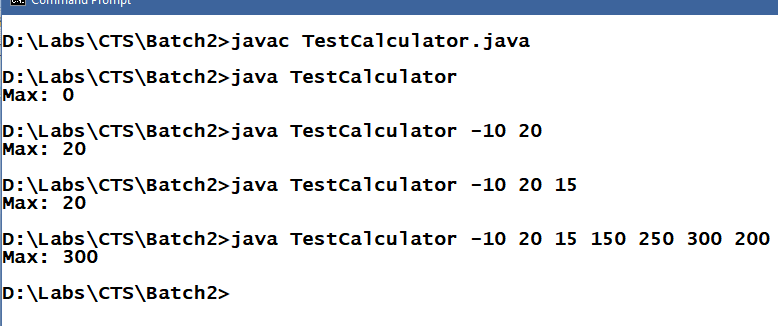
Solution for findMax(int[])



Calling findMax

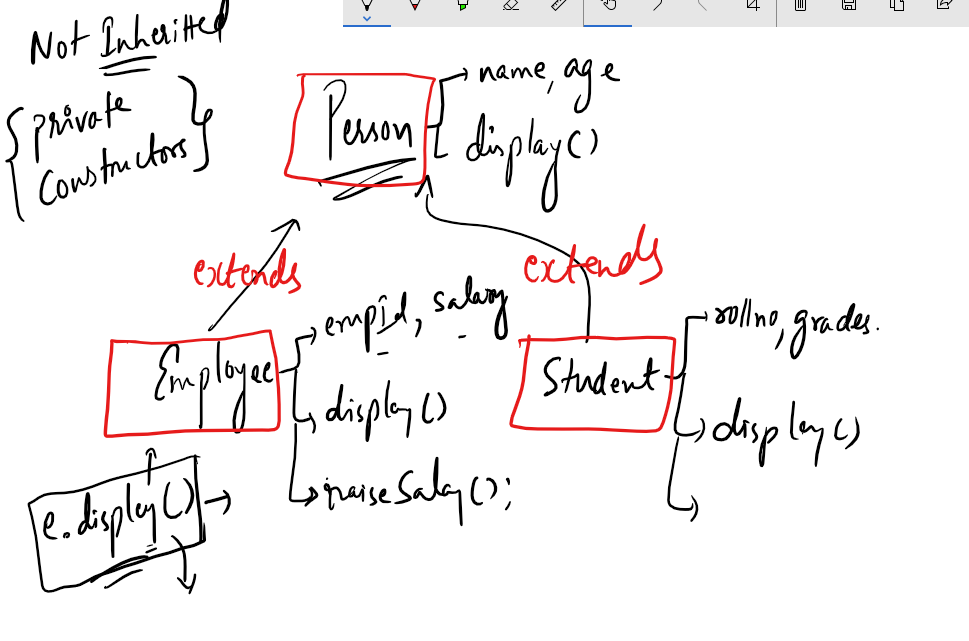


Output:



Inheritance

Process acquiring the properties and behaviours of an object from another object.



In subclass display() is overridden so that when you create an object of Employee or Student it calls overridden method, if you don’t override then it calls inherited method.

TestInheritance.java

class Person {

String name = "Rahul";

public void setName(String name) {

this.name = name;

}

public void display() {

System.out.println("Name = "+name);

}

}

class Employee extends Person {

int empId = 102;

public void display() {

System.out.println("EmpId = "+empId+", Name = "+name);

}

}

class Student extends Person {

String grades = "A";

public void display() {

System.out.println("Name = "+name+", Grade = "+grades);

}

}

class TestInheritance {

public static void main(String[] args) {

Person p1 = new Person(); p1.display();

Employee e1 = new Employee();

e1.display(); e1.setName("Vijay"); e1.display();

p1.display();

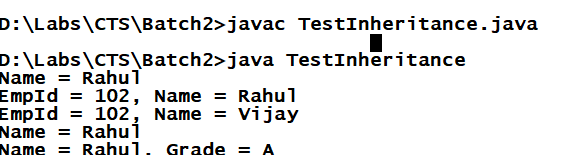
Student s1 = new Student();

s1.display();

}

}

Output:

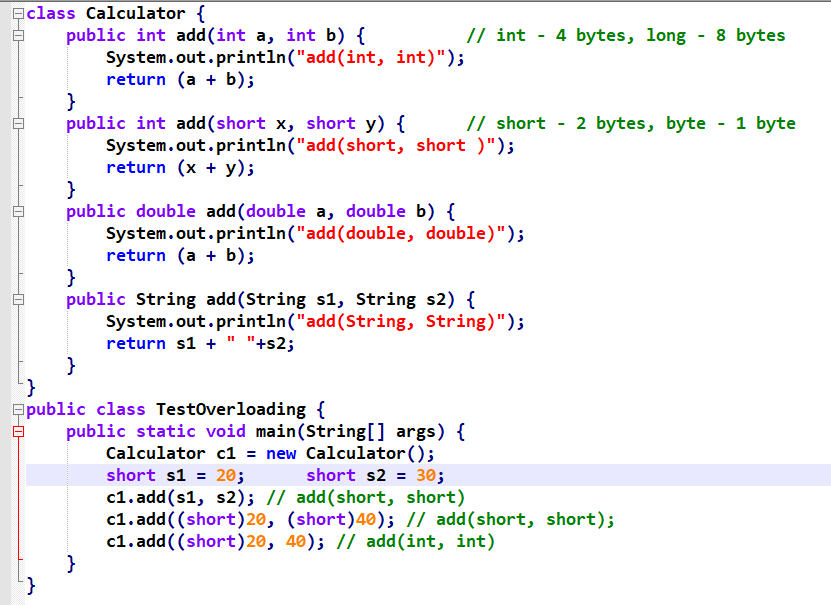


Polymorphism: Same method name with many forms, i.e., a method that behaves differently upon different situation.

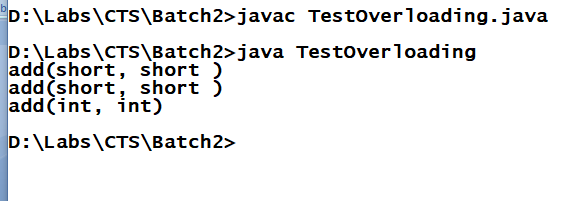
There are 2 types of Polymorphism

* Compile time polymorphism >> Overloading
* Runtime polymorphism >> Overridding

Overloading Example

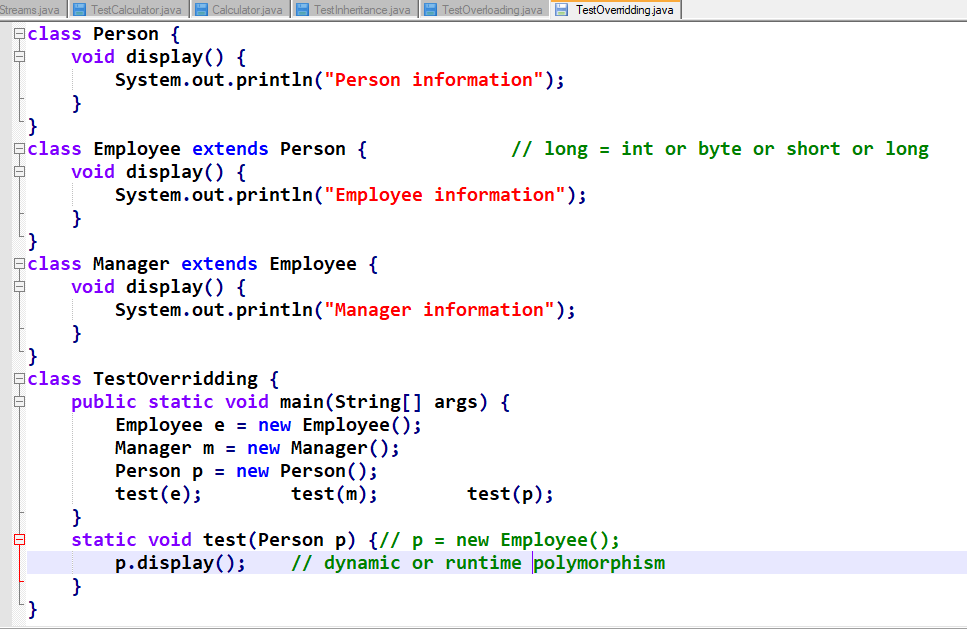


Output:

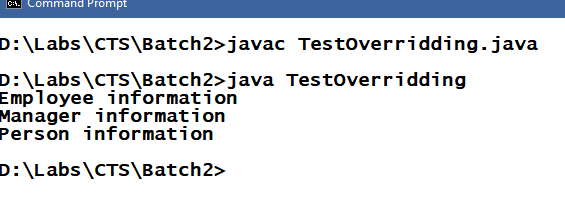


Runtime Polymorphism:

Method invocation is decided at runtime; here overridden methods are decided based on the type of object you are using to invoke the overridden method.

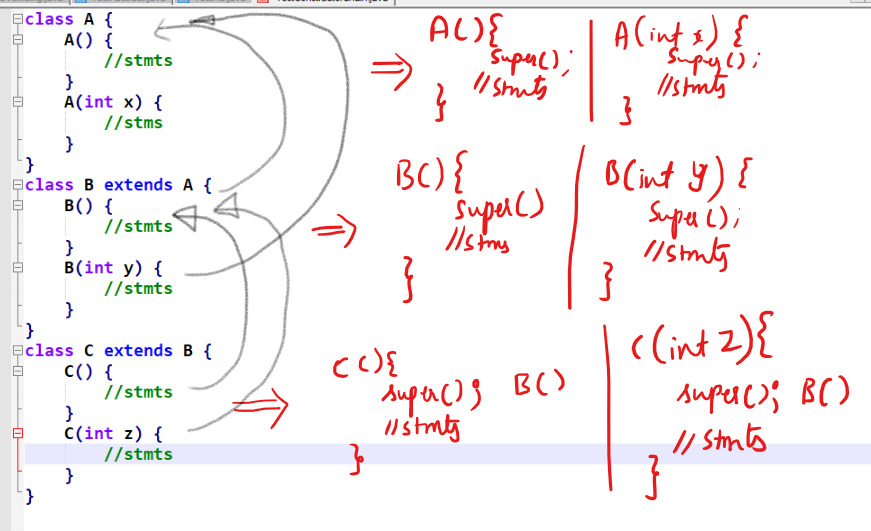


Output:

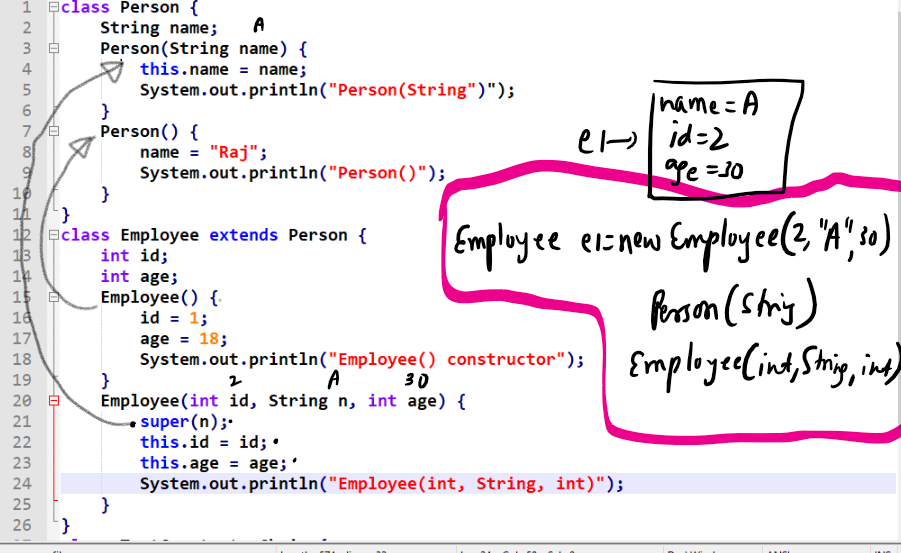


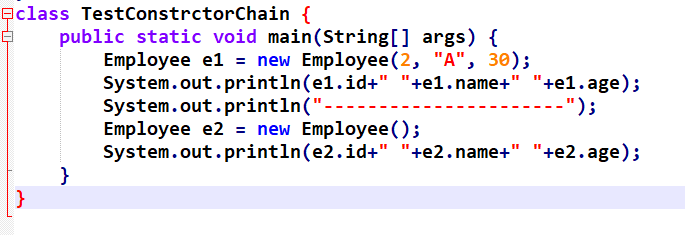
Assignment:

* Understanding about Abstraction - Abstract classes & Interfaces.
* super keyword in the constructor chaining or constructor invocation.

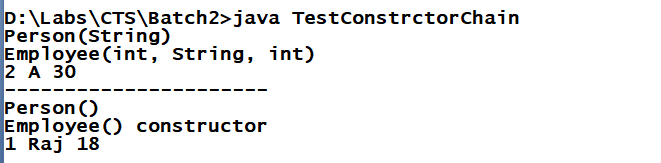


Every subclass constrctors implicitly calls super class default constructor from its first line, you can also explicitly call super class constructors using super() call, but you do that when you want call parameterized constructor of your parent class.





Output:



final keyword:

It is used on variables, methods and classes to restrict modifications.

final variables are constants and can’t be modified

final methods can’t be overridden.

final classes can’t be inherited.

Usage:

final double PI = 3.14; //every object will have their own

copy of PI

final static double PI = 3.14;//all the object will have

//single copy of PI

final void test1() {...}

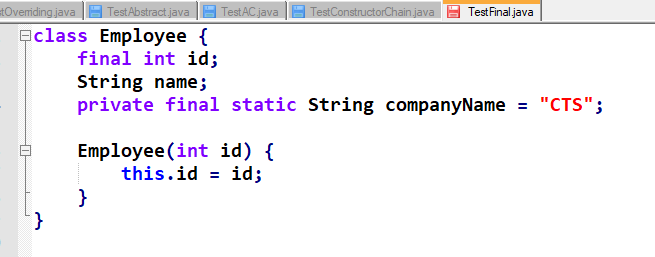
final class Manager extends Employee {....}

Note: final variables must be initialized; the initialized value itself is final because it can’t be modified.

Note: final non-static variables can be initialized inside constructor or at the time declaration.

Note: final static variables must be initialized at the time of declaration itself.

Note: Not initializing final variables will lead to compilation error.



Abstraction:

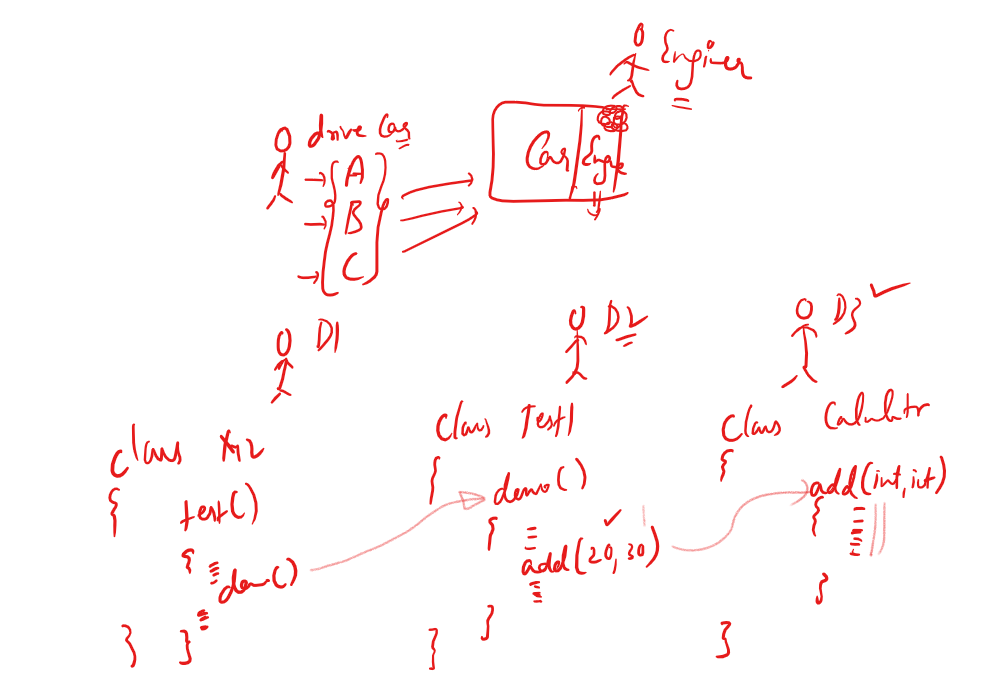
It is used to hide the complexity of an object and expose only the necessary interfaces.

It is mainly for developers who must know what a method does instead of know how it does (how it is implemented).

Example: System.out.println(); // now println() does printing but we know what it does but don’t know how it is implemented.

Example: Integer.parseInt(..); // parseInt() does what we know, but we don’t know its implantation

So here these methods are made us available through abstraction so that we can access them without knowing how it is implemented.



Because of abstraction the code you modify at one place will not affect other codes because other codes must know what the method does instead of how it is implemented.

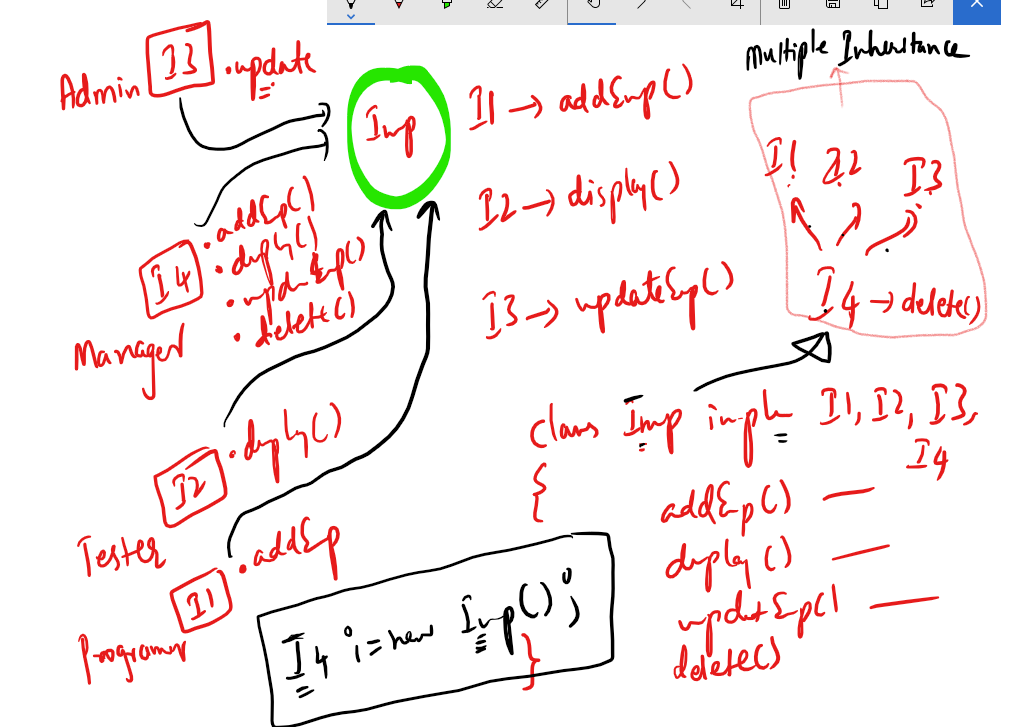
Abstraction can be achieved in 2 ways

1. interfaces
2. abstract classes

Interfaces: It is a kind of class which will have only abstraction methods and constants; everything inside interface is public by default.

Things you can achieve through interfaces are:

1. Complete abstraction, through this modification is going to be less.
2. Multiple inheritances.
3. It is secured because it doesn’t allow you to access the methods that are not present inside interface.



Some points in interface:

* Interface will have only abstract methods and constants.
* Interface will not have constructors.
* All the members of interfaces are public by default (you cannot change the visibility).
* Through interface you can achieve multiple inheritances.
* You cannot create object of interface but you can reference to the object that has implemented the interface.

TestInterfaces.java

interface X {

int add(int a, int b); // add is by default public and abstract

}

interface Y {

int sub(int a, int b); // sub is by default public and abstract

}

interface Z extends X, Y {

int mul(int a, int b); // mul is by default public and abstract

}

class Operations implements X, Y, Z {

public int add(int a, int b) {

System.out.println("add(int, int)");

return (a + b);

}

public int sub(int a, int b) {

System.out.println("sub(int, int)");

return (a - b);

}

public int mul(int a, int b) {

System.out.println("mul(int, int)");

return (a \* b);

}

}

class TestInterfaces {

public static void main(String[] args) {

Operations op = new Operations();

X x = op;

x.add(20, 30);

Z z = op;

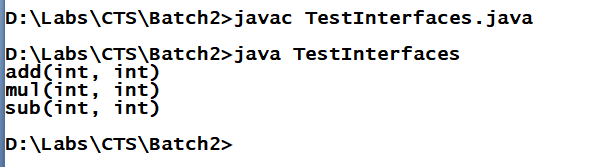
z.mul(2, 3);

z.sub(4, 3);

}

}

Output:



Abstract class: It is a class with abstract keyword, it allows you to have methods with body and without body i.e., method definitions and method declarations.

An abstract method must always be written either inside interfaces or abstract classes.

Usage:

abstract class Car {

abstract void showMileage();

void showWheels() {

// 4 wheels  
}

}

class Innova extends Car {

void showMileage() { 16kmpl }

}

class Swift extends Car {

void showMilage() { 20kmpl }

}

class TestAbstract {

public static void main(String[] args) {

Car c;

Innova i = new Innova();

Swift s = new Swift();

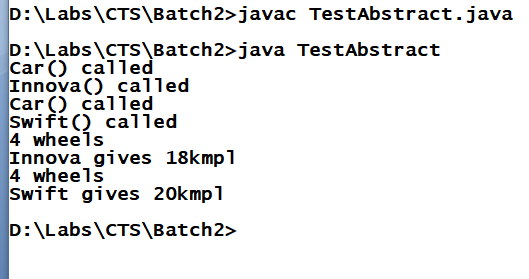
c = i; c.showWheels(); c.showMileage();

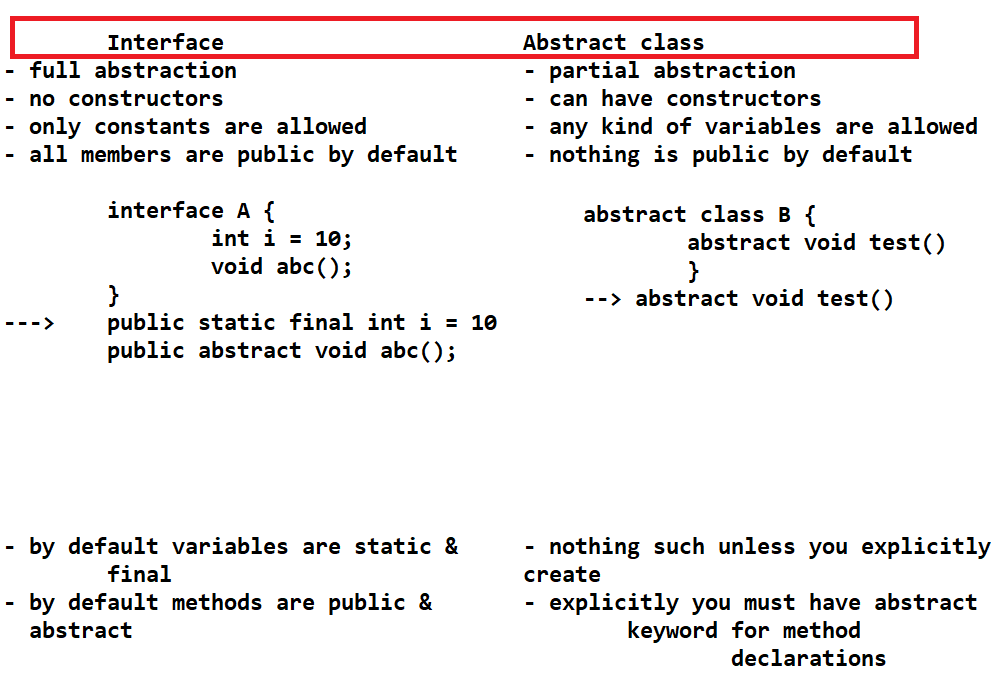
c = s; c.showWheels(); c.showMileage();

}

}

Output:





Similarity between interface and abstract class are: We can’t create object on both.

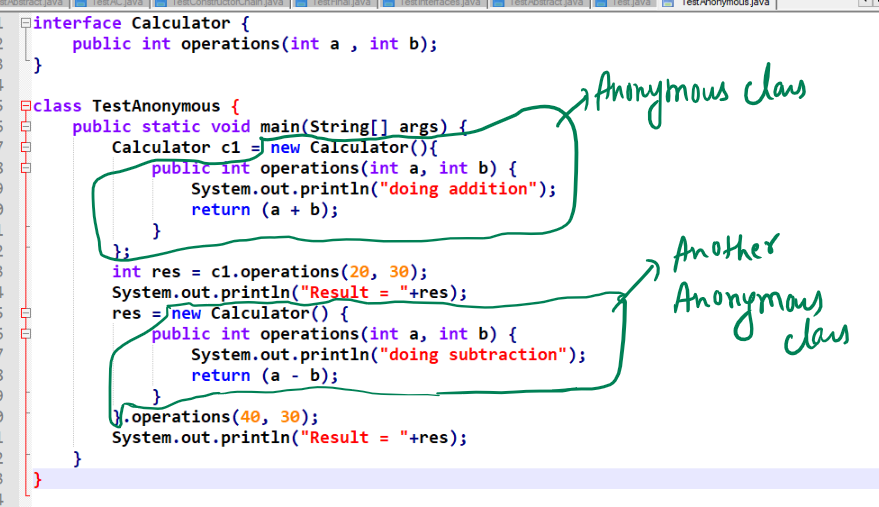
Note: An abstract class may or may not have abstract methods but abstract methods should be inside abstract class, abstract keyword does 2 tasks i.e., declaring method and restricting object creation.

Anonymous inner class

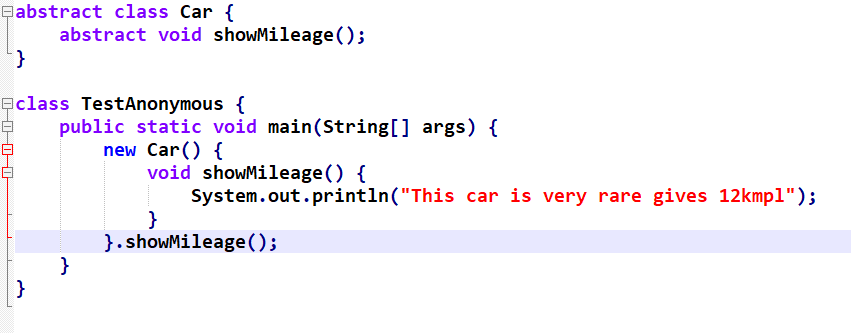
An inner class which doesn’t have any name, you can use anonymous classes when you don’t want to create multiple classes which are of no use outside the class.

An anonymous class is a sub class of a class or interface but it will not have any name.

Anonymous class is a kind of class which is declared & initialized at the same time.



Use anonymous classes when you want the implementations to be used only in one method not reusable in other class methods.



Output:

