<https://www.tutorialspoint.com/java/java_unicode_system.htm>

<https://javaconceptoftheday.com/>

<https://www.programiz.com/java-programming/operators>

**Java - Classes and Objects:**

Class : Blue print for creating the objects. It is logically exists.

Object: instance of a class. It is physically exists.

Blue print: its template. Eg: House blue print. It is logical.



Is it real? Is it existing? Is physical or logical?

There is a House.

Animal. 🡪 4 legs, 1 tail, 2 eyes, white teeth.

Running(), walkng(), eating(), makeSounds(), sleeping()….

All these are ogically exists. what

Dog, cat, Tiger, lion..etc 🡪 4 legs, 1 tail, 2 eyes, white teeeth.

Running(), walkng(), eating(), makeSounds(), sleeping()….

All these are physically existing. Whatever physically exists in the world is called as Object.

What ever you see with your eyes is called as Object.

**Object: every object should contain below properties.**

1. State 🡪 means : **properties / fields/ variables**.
2. Behavior: **methods / functions / actions**
3. Identity. -OPTIONAL.: object name / object reference name.

**State:**

Every object has some **properties / fields/ variables**.

Those properties are called as state.

For example:

Book properties: width, size, weight, price, no:of pages, author name.

Bike properties: color, weight, price, speed, milage, lights, bikeCompanyName, wheels type, etc…

Etc..examples.

**Behavior:**

Every object performs some actions; those actions are called as **behavior / methods / functions / actions**.

For example:

Book actions: reading(), writing(), playing(), beating(),

Bike actions: riding(), engineBeating(), spped(), stunts(), etc….

**Identity:**

Means giving a name to the object. It is optional. If an object doesn’t have identity means we can not say it is not an object. Even though it is not having identity but still it is having STATE and BEHAVIOR.

Class and objects both have state, behavior and identity. A class have logical state, behavior and identity.

But an object have physical state, behavior and identity.

Now our target is to convert every object state and behavior as program.

Without class there is No object. Means first we need to create class, then after using that class we need to create object.

<https://www.tutorialspoint.com/java/java_oops_concepts.htm>

OOP’s concepts:



Class

Object

**P**olymorphism.

**I**nheritance.

**E**ncapsulation. (**PIE**).

Abstraction.

**What a class contains?**

1. Constructors.
2. Properties / Fields / Variables.
3. Methods – Abstract methods, concrete methods, static methods.
4. Enums.

What types of Classes we have?

1. Classes with business logic.
2. Class for storing the data and transferring the data. (**POJO**-Plain Old Java Object **class**)

**Creating MAVEN project in STS / Eclipse:**

STS 🡪 File 🡪 new 🡪 Maven project🡪 next 🡪 type “**maven-archetype-quickstart**” 🡪 click next button 🡪



Next fill the required details.



Here “**Group Id**” means package name.

“**artifcatId**” means project name.

And click on finish button. Your Maven project will be created.

Below is the maven project structure created.



Here

**src/main/java** 🡪 this the place where we are going to write java code under given package. (com.skh is package).

src/test/java 🡪 this is the place where we are going to write test cases.

Pom.xml file 🡪 it is MAVEN related file. 🡪 which is used for downloading external jars.

**Jar?**

Whenever any developer develops the project. That project will be released in different formats..exe, .bat, ;.pdf, .text, .apk,

In java, developed projects are released in below different formats.

1. .**Jar** (Java Archives). 🡪 99.99%
2. .war (Web Archive). 🡪 0.1%
3. .ear (Enterprise Archive) 🡪 0.0%

Developed software will be release in above formats.

Above are compressed files. They will club/ combine all file together in compressed file.

**How to create our project .jar file in STS / Eclipse?**

**OR**

**How to run maven build in STS?**

Go to project in STS 🡪 right click on project 🡪 Select “Run AS” 🡪 Maven build 🡪 it will open on popup 🡪 In pop up go to “Goals” option 🡪 type there “**clean install**” 🡪 Apply 🡪 Run.



Once the build is completed, on STS console we can see where our project .jar file is generated.

Generally jar file will be created in our “project root folder” inside “**target**” folder.



We can see there is a .jar file created.



**Question?**

**I don’t know where is my project is created by using STS, how to find project root folder in STS?**

Right click on the project 🡪 properties🡪 click on “resource” option 🡪



**Question?**

**What is meant by build? 🡪 [target folder creation + code compile + by combining all .class files creating .jar file]**

When we run build, all java classes in project are compiled and generated all .class files kept inside **target** folder of project root folder.

We no need to worry about **target** folder, when run maven build, automatically **target** folder will be created.

We can delete “target” folder, why because when run maven build, automatically **target** folder will be created again and again.

**What .jar file contains?**

.jar files contains all compiled java classes, means a .jar file contains all .class files only.



We are working with Spring boot means we are working with their .jar files.

We are working with servlets means we are working with their .jar files.

We are working with Hibernate means we are working with their .jar files.

**How to download spring boot or servlets or hibernate external jar files / dependencies in our project?**

To download other external jar files in our project **we need to make use of “maven – pom.xml” tool or “gradle – build.gradle” tool.**

In maven/gradle we need to add external projects **dependencies**.

**Currently “gradle” is tending in the market.**

Build tools:

1. Ant
2. Maven
3. Gradle
4. Jenkins – we use in Server.

The main drawbacks of Ant include −

* XML is used as a format to write the build scripts.
* Being hierarchical is not good for procedural programming, and
* XML is relatively unmanageable.

The main drawbacks of Maven include −

* It does not handle the conflicts between versions of the same library.
* Complex customised build scripts are difficult to write in Maven, as compared to writing the build scripts in **ANT**.

<https://www.tutorialspoint.com/gradle/gradle_overview.htm>



<https://mvnrepository.com/>

Go to above website and copy the required dependency and paste it in either maven pom.xml file or gradle build.gradle file. Then it will download all required jar files into your project.

If we add dependencies .jar files will be downloaded.

**How many “public” classes a .java file contains?**

A .java file can contain only one public class. If we write more than one public class, we are going to get compilation error.

**To work with class.**

Create a class, declare variables if required, create constructors if required, create methods. Write the logic inside methods.

-Class name constructor name should be same.

-“void” means NO Return type.

Create another class, create main() method in that class and create object to above class in main() method and call methods. Now run this 2nd class.

**What is the difference between method and constructor?**

-Method have return type where as constructor don’t have return type.

-class name and constructor name always should same, methods name we can give any names as per our choice.

-when ever we created object that time only Constructor will be called and executed, but methods we need to call after creating object.

- constructor will be called only once while creating the object, but methods we can call any number of times using object reference.

<https://www.programiz.com/java-programming/variables-literals>

<https://www.tutorialspoint.com/java/java_date_time.htm>

**Variables: varies/ changeable. Over the time values can be change.**

* + 1. Class level / global level 🡪 we create outside the methods. Scope is complete class. Class level variables we can use in all methods.
    2. Method / construcot level. 🡪 we create variables inside the method. Scope is only that declared method. Variable data we can use only in that particular method.

**How to display the content on console?**

int a = 10;  
int b = 34;  
int c =45;  
float ff = 12.23f;

**1st approach: using “concatenation”.**  
 System.*out*.println("The value of a is " + a+ " "+ b +" "+ c + " "+ ff);  
**2nd approach: using String class format() method.**

System.*out*.println**(String.*format*("**The value of a is **%d %d %d %f", a, b, c, ff**));

**8 Primitive Data Types:**

BSIL 🡪 bite short int long





**Operators in Java can be classified into 5 types:**

1. Arithmetic Operators.
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators

Only for primitive types we use **==** operator. For objects comparison we always use .equals() method.





Java instanceof Operator.

Java Ternary Operator.

<https://techvidvan.com/tutorials/java-virtual-machine/>

**Static keyword:**

I want to call other class variables or methods... without creating object?  
How to do it? --> we can do it by using "**static**" keyword.  
If a variable or method is static type.. **we can call them by using className, we don’t need to create object to call them**.  
  
eg: Employe.**name**, Student.**id**, Student.**fetchFullName();**  
  
For calling non-static variables or methods. We must create object and call them with object referecne.

|  |  |
| --- | --- |
| public class CoreJavaTest {  int a = 10;  static String *name* = "Azad";  public static void m1(){  System.*out*.println("This is a method..!");  } } | public class App {   public static void main(String[] args) {  System.*out*.println(CoreJavaTest.*name*);  CoreJavaTest.*m1*();  CoreJavaTest javaTest = new CoreJavaTest();  System.*out*.println(javaTest.a);  } } |

**We have 8 primitive data types. [BSIL, float, double, char, boolean]**

**Apart from JDK given primitive data types we can use any kind of class/interface/abstract class as Datatype.**

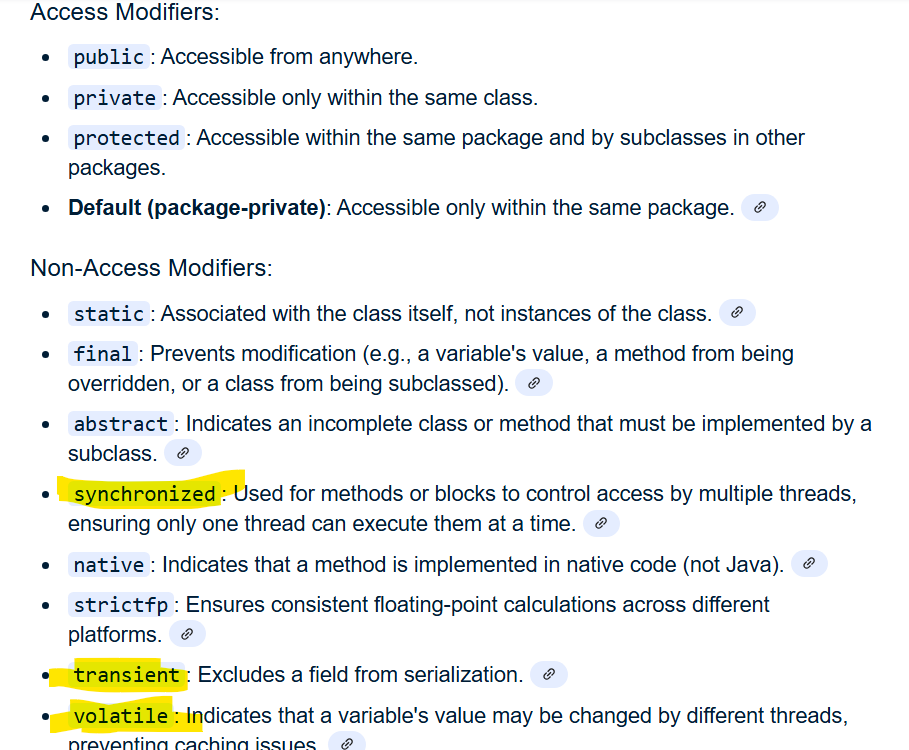
**Means predefined classes we can use as data types and user defined classes also we can use as datatypes.**

|  |
| --- |
| public **int** *i* = 0; public **String** *name* = null; public **PrintStream** *out* = null; public **Calculator** *cal* = null; |

Data types:

1. Primitive datatype. 🡪 **[BSIL, float, double, char, boolean]**
2. Non-Primitive datatype. 🡪 Wrapper classes + All predefined (classes/interfaces/abstract classes) +

All user defined (classes/interfaces/abstract classes) + arrays….etc.



<https://www.programiz.com/java-programming/class-objects>

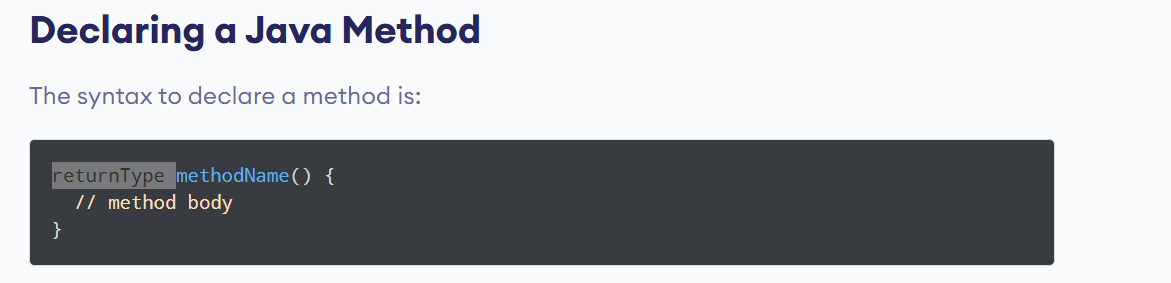
Java is an object-oriented programming language. Means everything in java is treated as Object.

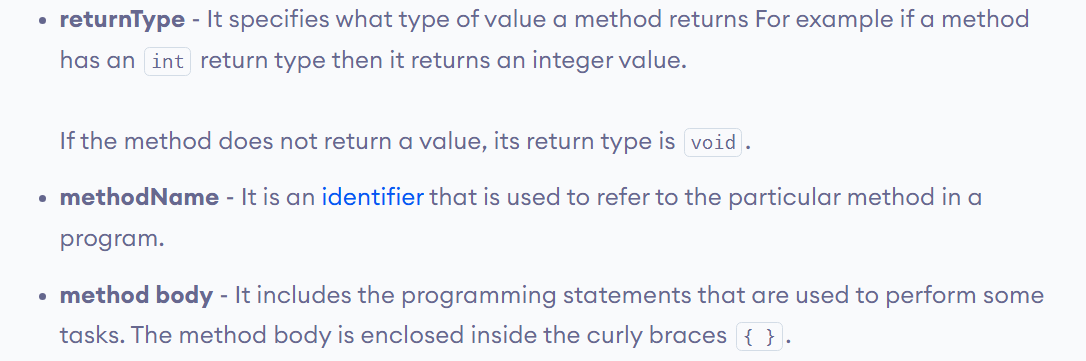
**Java Methods:**

A method is a block of code that performs a specific task.

In Java, there are two types of methods:

* **User-defined Methods**: We can create our own method based on our requirements. Programmer created.
* **Standard Library Methods**: These are built-in methods in Java that are available to use.





|  |
| --- |
| public class Test {   public String fetchName() {  return "Arafarth...";  }  int finalSalary(int salary) {  int finalSalary = salary + 2000;  *// 1000* return finalSalary;  }  int add(int a, int b) {  return a + b;  }  int mul(int a, int b) {  return a \* b;  }  void myMetod() {  } } |

**Java Method Return Type**

A Java method may or may not return a value to the function call. We use the **return statement** to return any value.

**Method Parameters in Java**

A method parameter is a value accepted by the method. As mentioned earlier, a method can also have any number of parameters

**What are the advantages of using methods?**

**1.** The main advantage is **code reusability**. We can write a method once, and use it multiple times. We do not have to rewrite the entire code each time. Think of it as, "**write once, reuse multiple times**".

|  |
| --- |
| public String percentageCalculation(String stuName,Integer telMarks, Integer engmarks, Integer hinMarks) {  Integer totalPercntage = ((telMarks + engmarks + hinMarks) \* 100) / 300;  return "Hi "+stuName +" your marks percentage is : "+totalPercntage;] } |

**Java Method Overloading:**

**Same method performing multiple tasks called as method overloading.**

When we declare same method multiple times we will get error. We get method duplicate error.

Below is duplicate method.

|  |
| --- |
| void calculateMarksSum(){  }  void calculateMarksSum(int tMarks){  } |

But still I want to use same method multiple times, then what to do?

If you want to use same method multiple times we need to use the Method overloading concept.

Same method is doing multiple tasks is called as Method overloading.

When we are working Method overloading we need to consider below points.

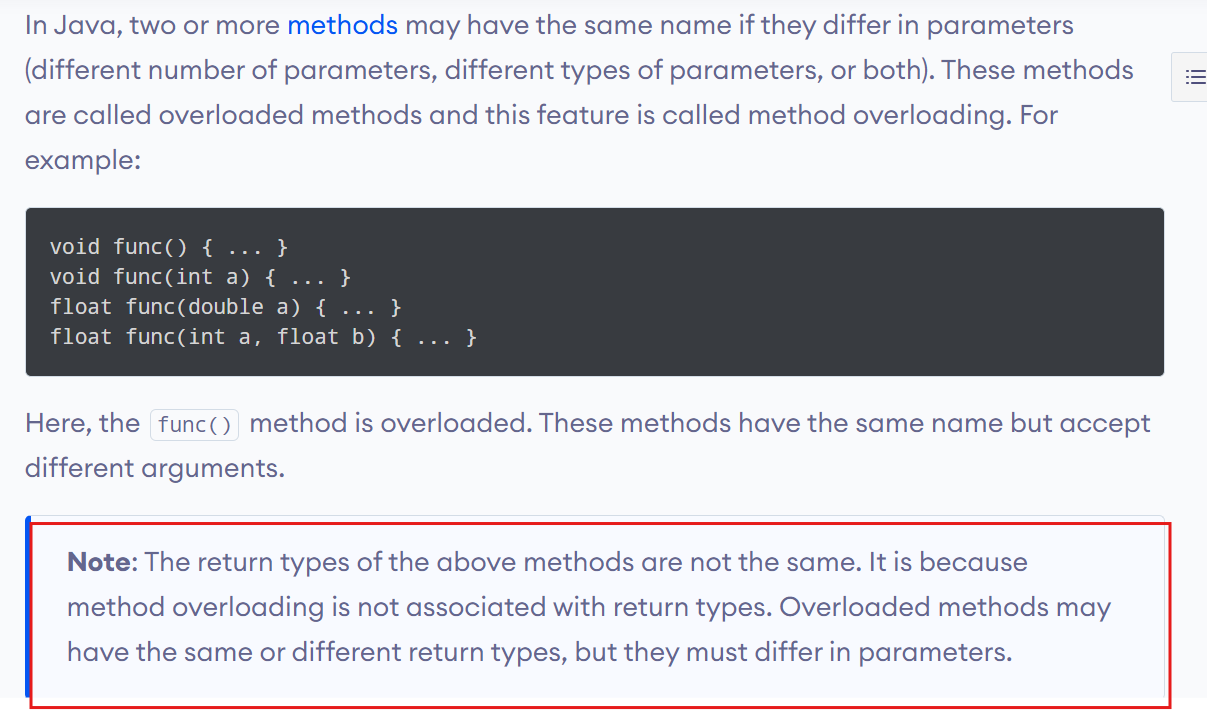
Use **same method** multiple times **with**.

1. **Number of** parameters should be different. 🡪 use same method, but with no:of different parameters
2. **Types** **of** parameters should be different. . 🡪 use same method, but with different data type params.
3. **Order of** parameters should be different.

|  |  |  |
| --- | --- | --- |
| Number of parameters should be different | Types of parameters should be different | Order of parameters should be different |
| void m1(){  }  void m1(int a, int b){  }  void m1(int a){  } | void m1(){  }  void m1(int a){  }  void m1(String a){  } | void m1(){  }  void m1(int a, String b){  }  void m1(String a, int b){  } |

No: of parameters are same = Duplicate method.

No:of Parameters same + Types of parameters same = Duplicate method.



Method overloading always talks about method name + method parameters.

Method return type Is not involved in method overloading.

What is meant by method signature?

**Method signature = Method name + method parameters.**

Method return type is not involved in method signature.

|  |
| --- |
| void **m1(int a, String b**){ } |

Here m1( )method and int, String datatypes combination is called as method signature.

**Method overloading always works with method signature only, it will not worry about method return types.**

<http://localhost:9000> 🡪 local code.

<https://123.45.23.34:9000> 🡪 name 🡪 Domain name 🡪 <https://azadtravels.com>

<protocol>://<IP\_ADDRESS>:<PORT\_NUMBRE>

Linux:

ssh skh@192.168.162.5 -p 22

**Java Constructors: Constructor name and className should be same.**

When ever we are creating object to a class that time only constructors will be executed. Only one time it will be executed while creating object.

If we are creating object to class 5 times, then 5 times constructor will be called. If you are creating object a class N number of times, then N number of tiles constructor will be called and executed.

**A constructor should not have return type. If it is having return type then that will be considered as normal method only.**

|  |  |
| --- | --- |
| public class Calculator {  **public Calculator(){**  System.*out*.println("This is Calculator class constructor..!");  }  } | public class Calculator {  **public void Calculator(){**  System.*out*.println("This is Calculator class constructor..!");  } } |

**What is the difference between constructor and Method?**

Constructor will be called only one time when we created object to a class, but once object is created by using that object reference we can call that method N number of times.

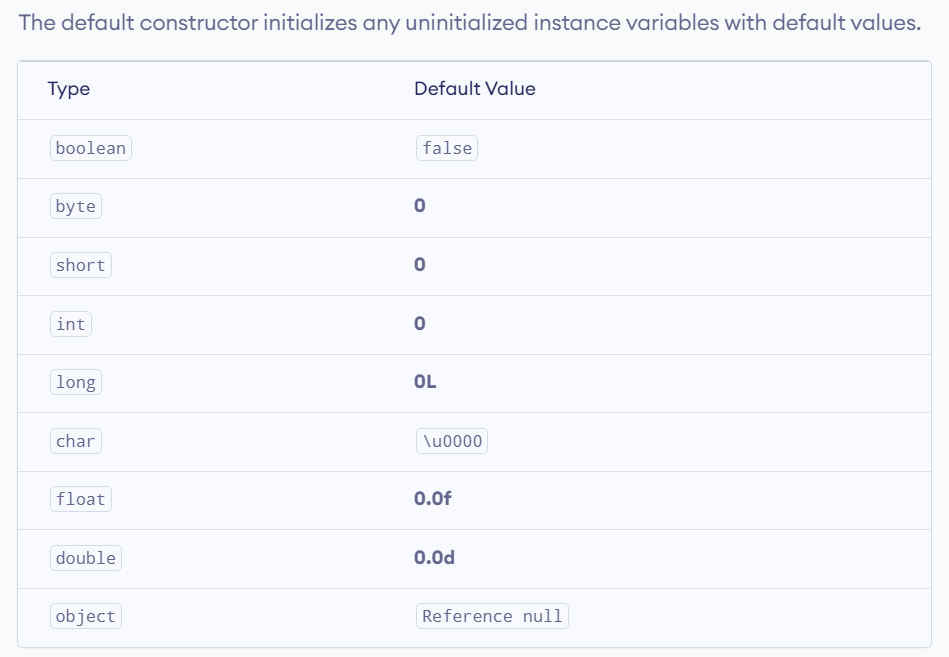
|  |  |
| --- | --- |
| public **class Calculator** {  public **Calculator()**{  System.*out*.println("This is Calculator class constructor..!");  }  void m1(){  System.*out*.println("I am method..!");  } } | public class App {  public static void main(String[] args) {  Calculator c1 = new **Calculator();**  c1.m1();  c1.m1();  c1.m1();  } } |

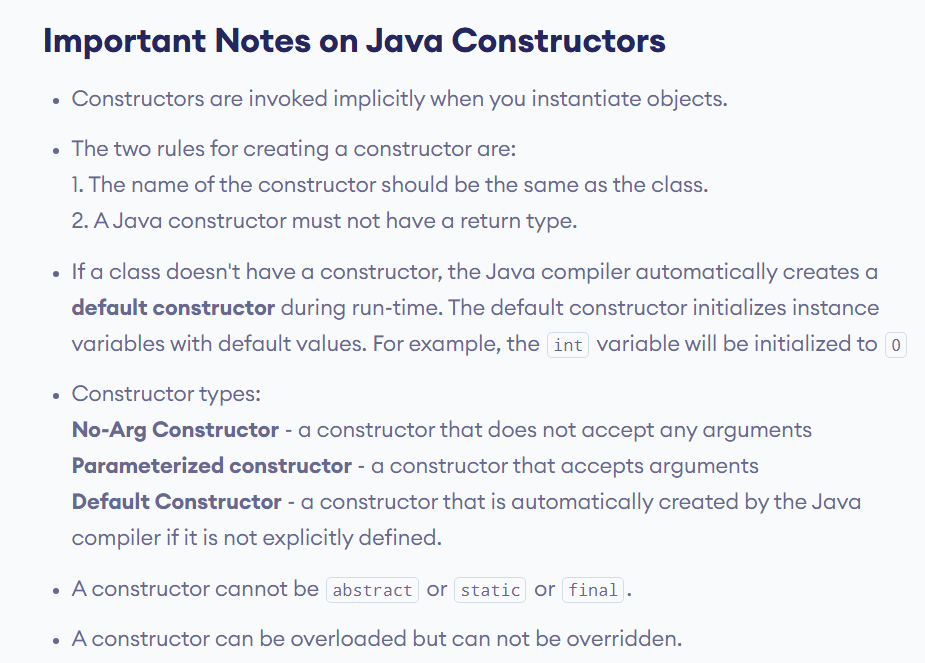
**Types of Constructor:**

In Java, constructors can be divided into three types:

1. **No-Arg** Constructor
2. **Parameterized** Constructor
3. Default Constructor

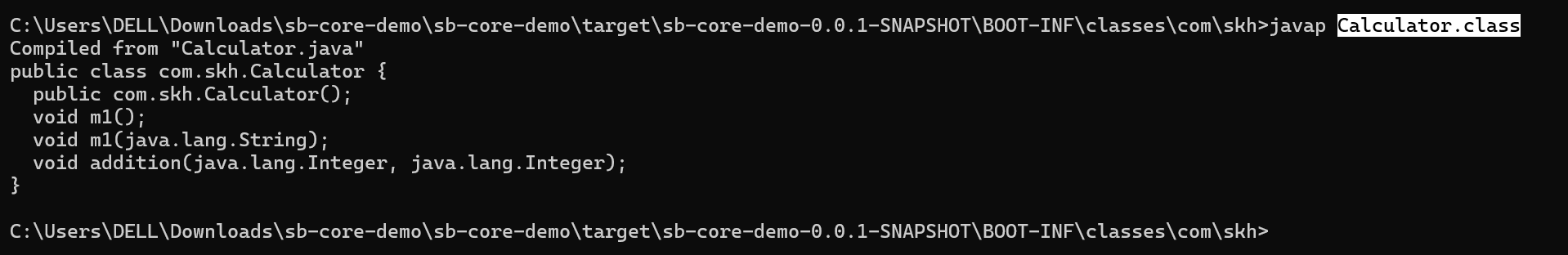
If we do not create any constructor, the Java compiler automatically creates a no-arg constructor during the execution of the program.

****

****

**By default constructor will be added at compile time. If and only if you didn’t add manually.**

**$ javap Employee.class**

****

<https://www.programiz.com/java-programming/constructors>

**Constructors Overloading in Java:**

Similar to method overloading, same constructor performing different tasks, based on your interest you call different constructors.

|  |  |
| --- | --- |
| public class Calculator {  public Calculator() {  System.*out*.println("P-less ctr..");  }  public Calculator(String empName) {  System.*out*.println("Hi..: " + empName);  }  public Calculator(String empName, String collegeName) {  System.*out*.println("Hi..: " + empName + ", your college is : " + collegeName);  } } | public class App {  public static void main(String[] args) {  new Calculator("Kamal", "Nova");  } } |

What is the difference between method overloading and constructor overloading?

**What is the mail task of constructor.?**

1. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* To initialize instance[class level] variables.**
2. To execute the logic when we create object.

|  |
| --- |
| private String calName; *// variable declaration. No value assigned.*  int a = 10;*// declaration + initialization. Means assigning the values also to the variable.* |

|  |
| --- |
| public class Calculator {  private String calName; *// variable declaration.* private Double calPrice;   public Calculator(Double calPrice, String calName) {  this.calPrice = calPrice;  this.calName = calName;  }  void m1() {  System.*out*.println("HI ... : " + this.calName);  } } |

**Instantiation** means creating the object. Eg: Employee e = new Employee();

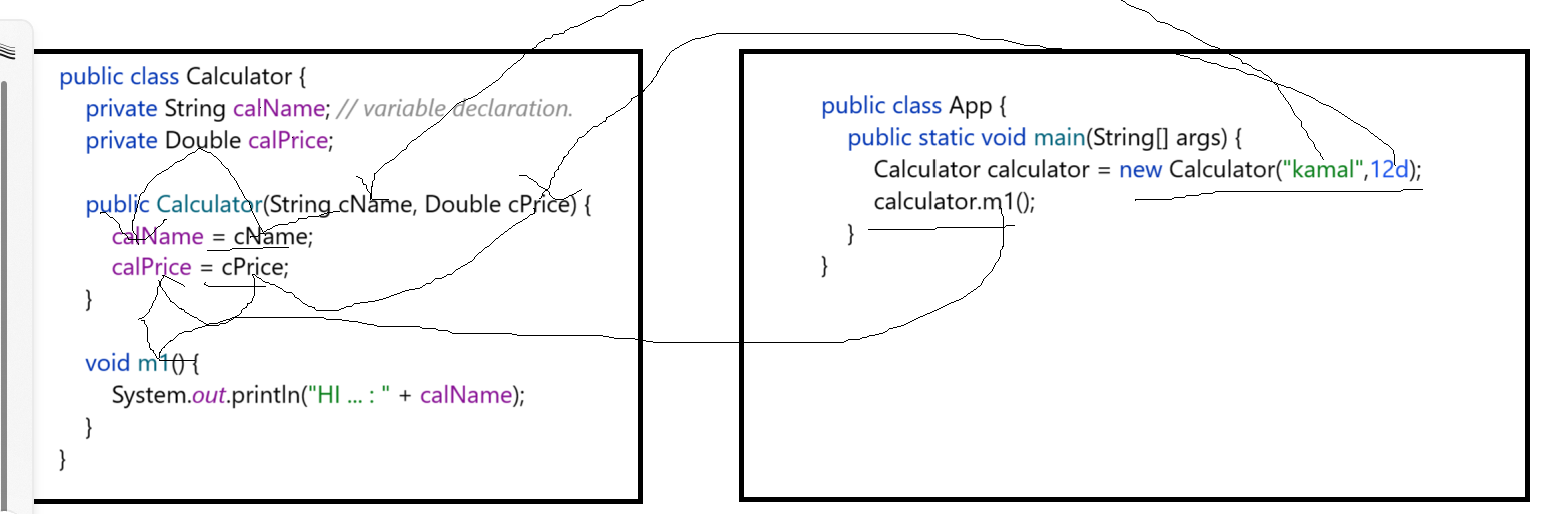
**Initialization** means assigning the value to a variable. Eg: String name = “Kamal”;

|  |  |
| --- | --- |
| public class Calculator {  private String calName; *// variable declaration.* private Double calPrice;   public Calculator(String cName, Double cPrice) {  **calName = cName;  calPrice = cPrice;**  }   void m1() {  System.*out*.println("HI ... : " + calName);  } } | public class App {  public static void main(String[] args) {  Calculator calculator = new Calculator("kamal",12d);  calculator.m1();  } } |

**In java initializing the instance variables can be done in 2 ways.**

1. **By using Constructors.**
2. **By using setter methods.**

Why are we using “this” in pojo class?



We cannot assign direct values to instance variables, why because we declare instance variables always as “**private**” only.

Constructor field are mandatory to provide. We can escape passing constructor parameters. We can not ignore unnecessary values in case of constructors.

**What is the purpose of constructor and setter methods?**

To set the data to **instance** variables. / To initialize the instance varialbes.

**What is the purpose of getter methods?**

To get the data from **instance** variables we use getter methods.

|  |  |
| --- | --- |
| **Employee.java:** | **App.java** |
| public class Employee {   private String empName;  private int empId;  private double empSalary;  private String aadharNumber;   public Employee(String empName, int empId, double empSalary, String aadharNumber){  this.aadharNumber = aadharNumber;  this.empSalary = empSalary;  this.empId = empId;  this.empName = empName;  }  public String **getEmpName**(){  **return this.empName;**  } } | Employee e = new Employee("Kamal", 12,323232, "AAASAS23232"); System.*out*.println(e.getEmpName()); |

**Setter methods always have “void” as return type. NO return type.**

**Getters always have any return type based on fields data type.**

**Java Static Keyword:**

**What is a static keyword in Java?**

The main purpose of the static keyword is to call methods, variables of a class without creating object, we can access static methods, static variables in other classes by using class name directly.

|  |  |
| --- | --- |
| **Non-Static class members** | **Static class members** |
| public class Calculator {   public String myName = "Kamal";   public void m1(){  System.*out*.println("Method m1()");  } } | public class Calculator {   public **static** String *myName* = "Kamal";   public **static** void m1(){  System.*out*.println("Method m1()");  } } |
| public class App {  public static void main(String[] args) {  Calculator calculator = new Calculator();  calculator.m1();  System.*out*.println(calculator.myName);  } } | public class App {  public static void main(String[] args) {  Calculator.*m1*();  System.*out*.println(Calculator.*myName*);  } } |

**For calling Non-static class members we need object creation and its reference.**

**For calling status class members we just need class Name only, no need to create object in this scenario.**

**Static Methods: Accessing class members without creating object.**

In Java, if we want to access class members, we must first create an instance of the class. But there will be situations where we want to access class members without creating any variables. In those situations, we can use the **static** keyword in Java.  If we want to access class members without creating an instance of the class, we need to declare the class members static.

Static methods are also called class methods. It is because a static method belongs to the class rather than the object of a class.

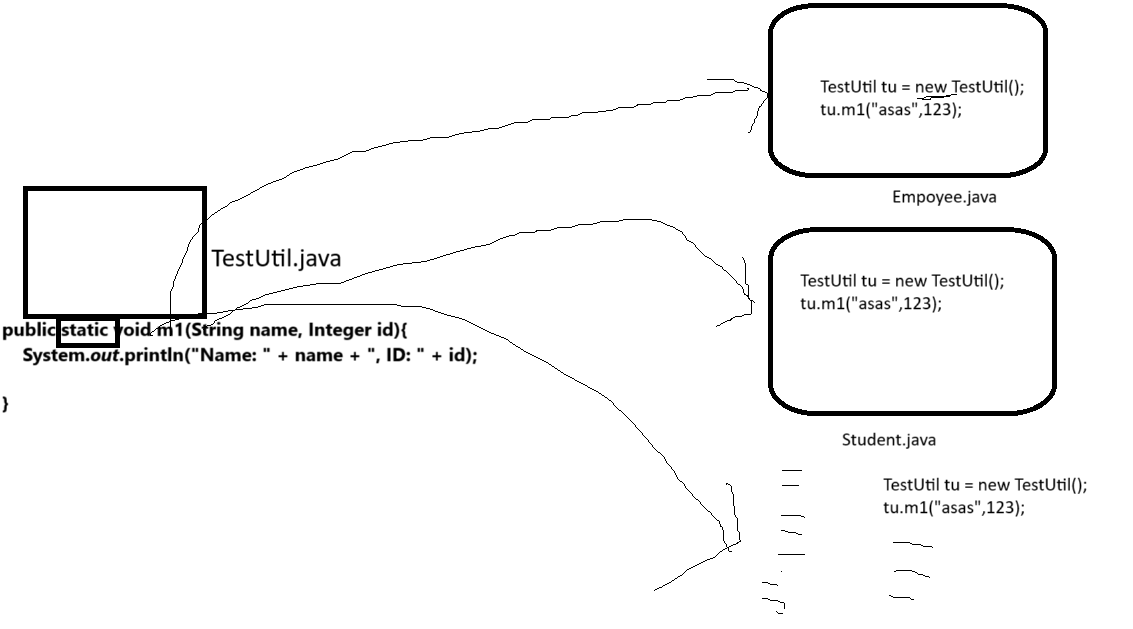
And we can invoke static methods directly using the class name.

We can use the static keyword in Java. If we want to access **class members [variables, methods] without creating an instance of the class**, we need to declare the class members static.

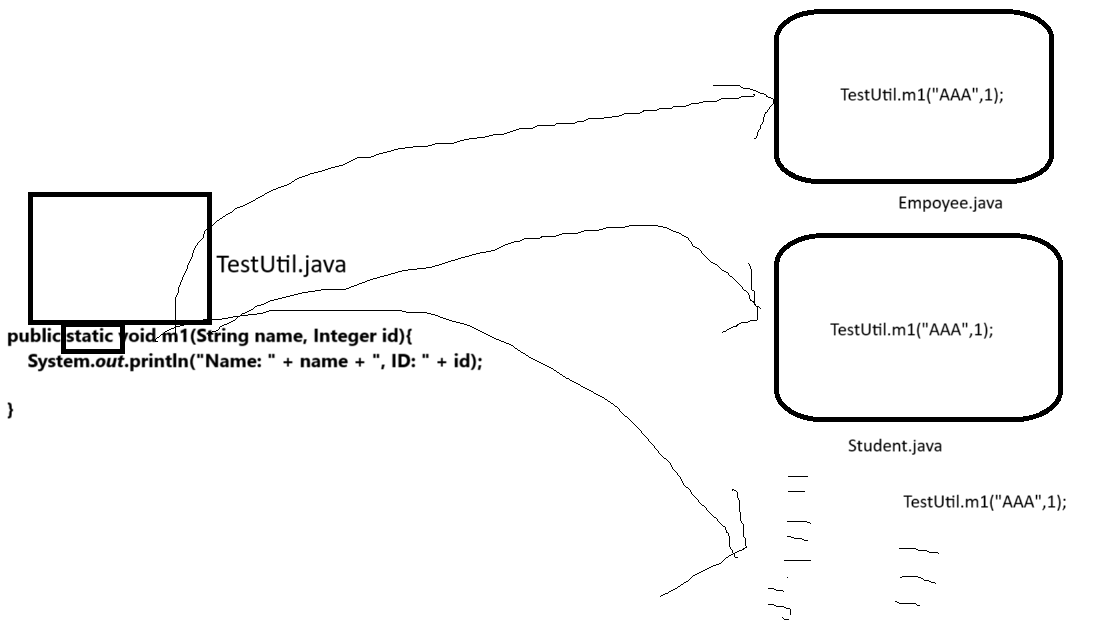
When to use static? When we have common data to objects or sharable to object then we can use static keyword for class members[methods or varaibles].

|  |
| --- |
| **Ctrl + F12** 🡪 To see all methods list in a class on popup. In IntelliJ.  **Double shift 🡪** to find the class globally in IntelliJ. |

**Before using “static” keyword:**



**After using “static” keyword:**



**In programming we should always try avoiding creating object using “new” keyword.**

Even in JavaScript also we should not use “new” keyword multiple times. We should use alternative approaches to avoid “new” keyword.

If the logic in class is common to the number of classes, or fields value is common to the number of classes, then declare that method and fields as static. And access those static method and static fields in other classes by directly calling by using class name;

Load on JVM will be reduced.

In java we can avoid object creation multiple times using below approaches.

1. Static keyword.
2. **Singleton Desing pattern. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

For entire application only one object will be created.

**To avoid creating object multiple times.**

Core java:

**Interfaces.**

**Abstract classes.**

Inheritance

Polymorphism.

**Interfaces:**

<https://www.tutorialspoint.com/java/java_interfaces.htm>

<https://www.digitalocean.com/community/tutorials/interface-in-java>

<https://www.simplilearn.com/tutorials/java-tutorial/java-interface>

It is like rules, or instructions class.

|  |
| --- |
| public **interface** ICalculator {  **int add(int fValue, int sValue);** // unimplemented methods / abstract methods **int multiplication(int a, int b, int c);** **int subtraction(int a, int b);** } |

Interface always have rules to follow.

Interface mainly contains “**abstract methods**”, Abstract methods mean methods with immediate termination[**;**];

Body less methods with no logic with immediate termination. We can see above example we have 3 abstract methods.

Now, whoever implements above interface they must provide implementation to all methods in their class.

Providing implements means writing the logic to those abstract methods.

|  |  |  |
| --- | --- | --- |
| **Interface** | Implementation class. | Main class. |
| public interface ICalculator {  int add(int fValue, int sValue);  int multiplication(int a, int b, int c);  int subtraction(int a, int b); } | public class MyCalculator implements ICalculator {  public int add(int fValue, int sValue) {  System.*out*.println("add()method");   int result = fValue + sValue;  return result;  }  public int multiplication(int a, int b, int c) {  System.*out*.println("multiplication()method");  int result = a \* b \* c;  return result;  }  public int subtraction(int a, int b) {  System.*out*.println("subtraction()method");  int result = a - b;  return result;  } } | public class App {  public static void main(String[] args) {  MyCalculator myCalculator = new MyCalculator();  int addResult = myCalculator.add(112,3434);  System.*out*.println(addResult);  } } |

Fully implemented classes are called “concrete” classes. Those methods are called as “concrete” methods.

Partial implemented classes we need to declare as “**abstract**” classes. Those methods are called as “abstract” methods.

What is the difference between concrete methods and abstract methods?

|  |  |
| --- | --- |
| **abstract method** | **concrete method** |
| void m1(); | void m1(){  System.*out*.println("HI…….");  } |

Can we create object to abstract class? NO.

Then how to utilize abstract class?? 🡪 Inherit[**create child class**] the abstract to another class by using **extend** keyword and override or implement abstract method.

**\*\*\*\*\*\*\*We cannot create object to interfaces and abstract classes.**

**We can create object to concrete classes only. A class without abstract methods, or a class without abstract declaration is called as Concrete class.**

Partially implemented class is called as abstract class.

|  |  |  |  |
| --- | --- | --- | --- |
| **interface** | **Abstract class** | **Concrete class** | **Main class** |
| public **interface** IColony {  void singleStair();  void door();  void windows();  void landSize(); } | **abstract** **public** **class** AzadHouse implements IColony {  public void singleStair() {  }  public void door() {  }  public void windows() {  } } | public class HamidHouse **extends** AzadHouse{  public void landSize() {  } } | public class App {  public static void main(String[] args) {  HamidHouse a = new HamidHouse();  a.windows();  } } |

**Concrete class?** A class which doesn’t have abstract methods in that class and its parent class also should not have abstract methods.

**If all abstract methods are implemented then only that class will become concrete class.**

**Abstraction?**

Showing necessary details and hiding unnecessary details it is called as abstraction.

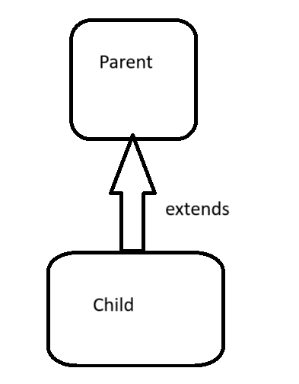
Eg: ATM machine, Car, Human body….etc.

**Interface –** contains abstract methods.

**Class –** contains concreate methods.

**Abstract class –** contains **abstract methods + concreate** methods / or all **concrete methods + abstract keyword declaration** / all **abstract methods + abstract keyword declaration** / or no methods **+ abstract keyword declaration**.

Abstract class must be declared with “**abstract**” keyword.



**Marker Interface:** Interface without any abstract methods is called as Marker interface.

\*\*\*\*\*\*\*\*\*\*\*\***Functional Interface:** Interface with **SINGLE ABSTRACT METHOD** is called as functional interface.

It is **java 8** feature, it will be used in functional programming.

|  |  |  |
| --- | --- | --- |
| **Marker Interface** | **Functional Interface \*\*\*\*\*\*\*\*\*\*\*\*\*** | **Lambda expression for FInterface.** |
| public **interface** ICar {  } | public **interface** ICar {  String carSpeed(); } | **ICar ref = () -> "Azad...";**  System.*out*.println(ref.m1()); |

**By using Functional interface only, we are going work with LAMBDA expressions.**

For single interface there can be any number of implementation classes.

ICAR 🡪 TataCar, NanoCar, NexonCar,…..BreezaCar………………

Along with abstract methods, an interface may also contain constants, default [methods](https://www.tutorialspoint.com/java/java_methods.htm), static methods, private static methods, and nested types. **Method bodies exist only for default methods and static methods in interfaces.**

If an interface contains abstract methods and default methods and static methods then in the child class it **must implement “abstract” methods. We no need to override and implement default and static methods. If need we can do it.**

**\*\*\* By default all variables in interface are CONSTANTS. Means once value is assigned we cannot change that value.**

**By default interface variables internally assigned with “public static final” modifiers. We define constant variable name with CAPITAL Letters only.**

**Eg:**

**Public static final COUNTRY\_NAME = “INDIA”;**

**Public static final STATE\_NAME = “AP”;**

**public static final int *MIN\_PRIORITY* = 1;**

**public static final int *NORM\_PRIORITY* = 5;**

**public static final int *MAX\_PRIORITY* = 10;  
  
Static –** if we declare methods or fields with static keyword, we can call those methods are variables with class name or interface name without creating object.

**Public –** If we declare public fields or methods or constructors, we can access them anywhere in the project.

**Final -** if we declare fields as final we cannot change the value, if we declare method as final in parent class then we cannot override that methods in child class.

Below is example for final method in parent class.

|  |  |
| --- | --- |
| public class Parent {   public **final** void m1(){  System.*out*.println("This is m1() methods...");  } } | public class ChildOne extends Parent{  **@Override  public void m1() {**  System.*out*.println("from child class...");  } }  ERROR class, we can not override final method. |

|  |
| --- |
| public interface ICar {  String *NAME* = "NANO CAR";   void carSpeed();  default void iCarDefaltMethod(){  System.*out*.println("iCarDefaltMethod");  }   static String iCarStaticMethod(){  *iCarPrivateMethod*();  return "Dummy..";  }   private static void iCarPrivateMethod(){  System.*out*.println("iCarPrivateMethod");   } } |

**Finally an interface can contain.**

1. **Abstract methods.** 🡪 NO BODY / No LOGIC.
2. **Default methods.** 🡪 Can have body & we can write logic.
3. **Static methods.** 🡪 Can have body & we can write logic.
4. **Private static methods.** 🡪 Can have body & we can write logic.
5. **CONSTANTS**. 🡪 internally interface variable are **public static final**, so they are called as constants.

**Below are for interface declarations:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Abstract methods.** | **Default methods.** | **Static methods.** | **Private static methods.** | **CONSTANTS** |
| void carSpeed(); | **default** void iCarDefaltMethod(){  System.*out*.println("iCarDefaltMethod"); } | **static** String iCarStaticMethod(){  *iCarPrivateMethod*();  return "Dummy.."; } | **private static** void iCarPrivateMethod(){  System.*out*.println("iCarPrivateMethod");  } | String *NAME* = "NANO CAR"; |

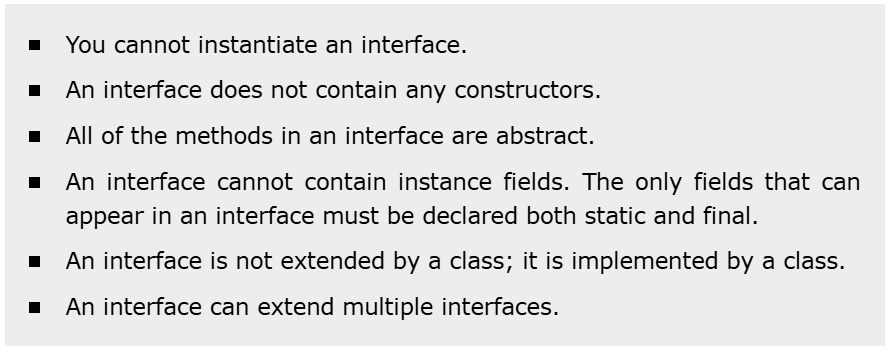
If we are not implementing all abstract methods method’s in implemented class, then we need to declare that class as abstract class. Means we use abstract keyword Infront of class name.

|  |  |
| --- | --- |
| **Interface cannot implements another interface.** | **Interface can EXTENDS another interface.** |
| interface A{ void m1(); } interface B implements A{  } | interface A{  void m1(); } interface B extends A{  void m2(); } |

If anyone is going to implement interface B, then that class must implement abstract methods of interface A and interface B also.

For example: A class which implements B interface.

|  |
| --- |
| public class IABImpl **implements B**{  @Override  public void m2() {   }   @Override  public void m1() {   } } |



-Interface internally **abstract** only and all its methods internally **public and abstract** only.

|  |  |
| --- | --- |
| public interface MyInterface {  void m1(); } | public **abstract** interface MyInterface {  **public abstract** void m1(); } |

**Class to class we always use 🡪 extends.**

**Interface to interface we always use 🡪 extends.**

**Interface to class we always use 🡪 implements.**

Class to interface 🡪 **NOT POSSIBLE.**

**Extending Multiple Java Interfaces:**