

Difference between Procedural Programming and Object Oriented Programming:

Procedural Oriented Programming	Object Oriented Programming
In procedural programming, program is divided into small parts called functions .	In object oriented programming, program is divided into small parts called objects .
Procedural programming follows top down approach .	Object oriented programming follows bottom up approach .
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is less secure .	Object oriented programming provides data hiding so it is more secure .
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on unreal world .	Object oriented programming is based on real world .
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The `java.io` package is a highly cohesive package because it has I/O related classes and interface. However, the `java.util` package is a weakly cohesive package because it has unrelated classes and interfaces.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be unidirectional or bidirectional.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Advantage of OOPs over Procedure-oriented programming language

- 1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
- 2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.
- 3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

Object-Oriented Programming

Characteristics:

- Emphasis on data rather than procedure.
- Programs are divided into entities known as **objects**.
- Data Structures are designed such that they characterize objects.
- Functions that operate on data of an object are tied together in data structures.
- Data is hidden and cannot be accessed by external functions.
- Objects communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows **bottom up design** in program design.

Activ
Go to

Bottom up approach

- Reverse top-down approach.
- Lower level tasks are first carried out and are then integrated to provide the solution of a single program.
- Lower level structures of the program are evolved first then higher level structures are created.
- It promotes code reuse.
- It may allow unit testing.

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

- We can create object of class using following syntax,
- Syntax: `class-name object-name;`
- Here class name is class which is already created. Object name is any user define name. For example, if Student is class,
- Example: `Student ram, sham;`
- In example `ram` and `sham` are name of objects for class `Student`. We can create any number of objects for class.

- Java is a **programming language** and a **platform**. Java is a **high level, robust, object-oriented and secure programming language**.
- Java - The new programming language developed by **Sun Microsystems** (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak.
- Originally called **Oak** by **James Gosling**, one of the inventors of the Java Language.
- Originally created for consumer electronics (TV, VCR, Freeze, Washing Machine, Mobile Phone).
- Java - CPU Independent language
- Internet and Web was just emerging, so Sun turned it into a language of Internet Programming.
- It allows you to publish a webpage with Java code in it.

Sr. No.	Key	JDK	JRE	JVM
1	Definition	JDK (Java Development Kit) is a software development kit to develop applications in Java. In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc.).	JRE (Java Runtime Environment) is the implementation of JVM and is defined as a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming.	JVM (Java Virtual Machine) is an abstract machine that is platform-dependent and has three notions as a specification, a document that describes requirement of JVM implementation, implementation, a computer program that meets JVM requirements, and instance, an implementation that executes Java byte code provides a runtime environment for executing Java byte code.
2	Prime functionality	JDK is primarily used for code execution and has	On other hand JRE is majorly responsible for	JVM on other hand specifies all the implementations and

Sr. No.	Key	JDK	JRE	JVM
		prime functionality of development.	creating environment for code execution.	responsible to provide these implementations to JRE.
3	Platform Independence	JDK is platform dependent i.e for different platforms different JDK required.	Like of JDK JRE is also platform dependent.	JVM is platform independent.
4	Tools	As JDK is responsible for prime development so it contains tools for developing, debugging and monitoring java application.	On other hand JRE does not contain tools such as compiler or debugger etc. Rather it contains class libraries and other supporting files that JVM requires to run the program.	JVM does not include software development tools.
5	Implementation	JDK = Java Runtime Environment (JRE) + Development tools	JRE = Java Virtual Machine (JVM) + Libraries to run the application	JVM = Only Runtime environment for executing the Java byte code.

► JVM

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.
- The JVM performs following main tasks:
 - **Loads code**
 - **Verifies code**
 - **Executes code**
 - **Provides runtime environment**

► JRE

- JRE is an acronym for Java Runtime Environment.

- It is used to provide runtime environment. It is the implementation of JVM.
- It physically exists. It contains set of **libraries + other files** that JVM uses at runtime.
- **JDK**
- JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

First JAVA Program

```
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World !!!");
    }
}
```

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print statement.

System.out.println in Java

Java System.out.println() is used to print an argument that is passed to it. The statement can be broken into 3 parts which can be understood separately as:

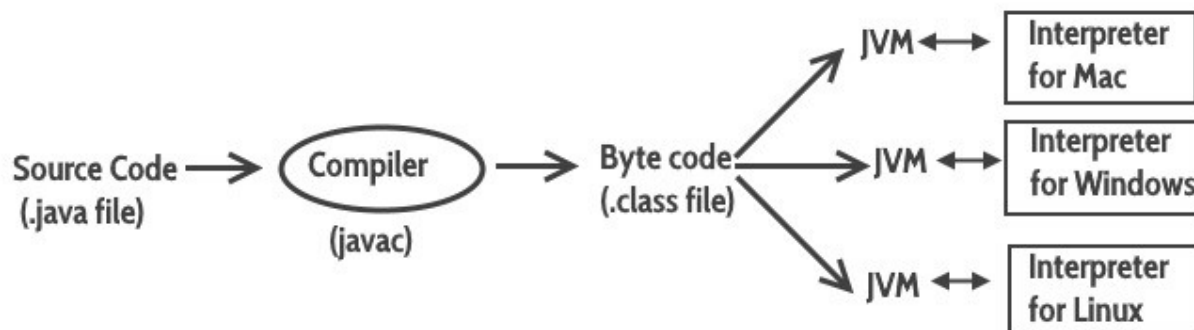
- **System:** It is a final class defined in the java.lang package.
- **out:** This is an instance of PrintStream type, which is a public and static member field of the System class.
- **println():** As all instances of PrintStream class have a public method println(), hence we can invoke the same on out as well. This is an upgraded version of print(). It prints any argument passed to it and adds a new line to the output. We can assume that System.out represents the Standard Output Stream.

Java ByteCode

Java bytecode is the instruction set for the Java Virtual Machine. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

► Just-In-Time(JIT) compiler:

- It is used to improve the performance.
- JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.
- Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.



Beginnersbook.com

► Multithread

- Multithreading in java is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing.
- It shares a common memory area. Threads are important for multi-media, Web applications etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

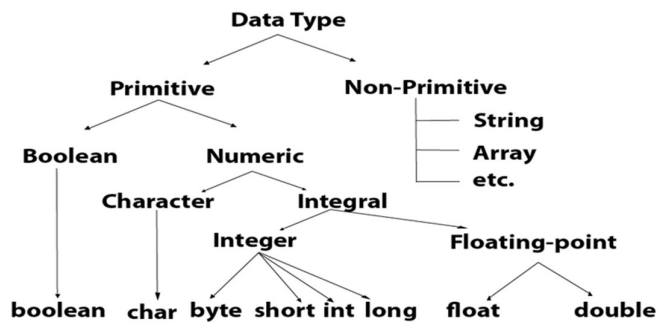
An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.



Use float or double?

The precision of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

	Operator	Type
Unary operator	++, --	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, , !	Logical operator
	&, , <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?: Tutorial4us.com	Ternary or conditional operator

Short Circuiting in Java with Examples

```

class ShortCirAND {
    public static void main(String arg[])
    {

```

```
// Since first operand is false and operator is && Evaluation stops and false is returned.
```

```
if (false && true && true) {  
    System.out.println(  
        "This output will not be printed");  
}  
else {  
    System.out.println(  
        "This output got printed actually, due to short circuit");  
}}
```

AND(&&) short circuit: In case of AND, the expression is evaluated until we get one false result because the result will always be false, independent of the further conditions. If there is an expression with &&(logical AND), and first operand itself is false, then short circuit occurs, the further expression is not evaluated and false is returned.

OR(||) short circuit: In case of OR, the expression is evaluated until we get one true result because the result will always be true, independent of the further conditions. If there is an expression with ||(logical OR), and first operand itself is true, then a short circuit occurs, evaluation stops and true is returned.

In Java logical operators, if the evaluation of a logical expression exit in between before complete evaluation, then it is known as **Short-circuit**. A short circuit happens because the result is clear even before the complete evaluation of the expression, and the result is returned. Short circuit evaluation avoids unnecessary work and leads to efficient processing.

Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !=	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

The **(.) operator** is also known as member operator it is used to access the member of a package or a class.

instanceof operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as –

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example

Keywords In Java

1. abstract	13. double	25. int	37. strictfp
2. assert	14. else	26. interface	38. super
3. boolean	15. enum	27. long	39. switch
4. break	16. extends	28. native	40. synchronized
5. byte	17. final	29. new	41. this
6. case	18. finally	30. package	42. throw
7. catch	19. float	31. private	43. throws
8. char	20. for	32. protected	44. transient
9. class	21. if	33. public	45. try
10. continue	22. implements	34. return	46. void
11. default	23. import	35. short	47. volatile
12. do	24. instanceof	36. static	48. while

Unicode is a 16-bit character encoding standard and is capable to represent almost every character of well-known languages of the world.

Java Naming Conventions:

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
- All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

► Syntax to declare a class:

```
class <class_name>{  
  
    field;  
  
    method;  
  
}
```

Characteristics of Objects:

An object has three characteristics:

1. **State:** State represents properties of an object. It is represented by instance variable/attribute of an object. The properties of an object are important because the outcome of functions depend on the properties.
2. **Behavior:** Behavior represents functionality or actions. It is represented by methods in Java.
3. **Identity:** Identity represents the unique name of an object. It differentiates one object from the other. The unique name of an object is used to identify the object.

Let's take a real-world example to understand all these points clearly.

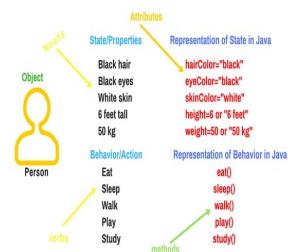
We are taking the example “**person**”. A person has three characteristics: Identity (name), State (properties), and behavior (actions or functionality). Look at the below figure.

In the above figure, a person is an object. First state of the person (object) is black hair which can be represented in Java like this: `hairColor = “black”`.

In the same way, second property of the object is eye color which can be represented in Java like `eyeColor = “black”` and so on. These are called attributes that define the properties of a person.

Let’s consider the behaviors or actions of a person. The actions of a person may be “eat, sleep, walk, play, and study”. These actions are represented in Java like this: `eat()`, `sleep()`, `walk()`, `play()`, and `study()`. These are called methods.

Thus, when properties and actions are combined together of any real-world object make an object in Java.



Realtime Examples of Class in Java

Let us consider two objects Samsung Galaxy S4 and iPhone. Suppose Samsung Galaxy S4 have some properties like width = “6.98 cms”, height = “13.6 cm”, OS = “Android”, brand = “Samsung”, price = “1000\$” and actions are `call()`, `sendMessage()`, `browser()`, `share()`.

Now, suppose iPhone has some properties such as width = “5.86 cm”, height = “12.3 cms”, OS = “iOS”, brand = “Apple”, price = “1200\$” and actions are `call()`, `sendMessage()`, `browse()`, `share()`.

Both objects have some different properties and actions but the type is the same “Phone”. This is the class. i.e the name of the class is “Phone”.

//Java Program to illustrate how to define a class and fields

//Defining a Student class.

```
class Student{  
  
    //defining fields  
  
    int id;//field or data member or instance variable  
  
    String name;  
  
    //creating main method inside the Student class  
  
    public static void main(String args[]){  
  
        //Creating an object or instance  
  
        Student s1=new Student();  
  
        System.out.println(s1.id);//accessing member through reference variable
```

```

System.out.println(s1.name);
}
}

```

```

import java.util.Scanner ← 1. Import Scanner Class

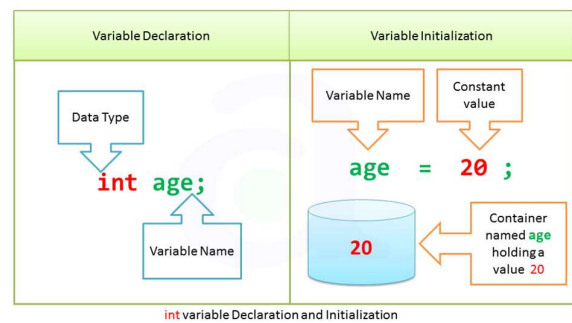
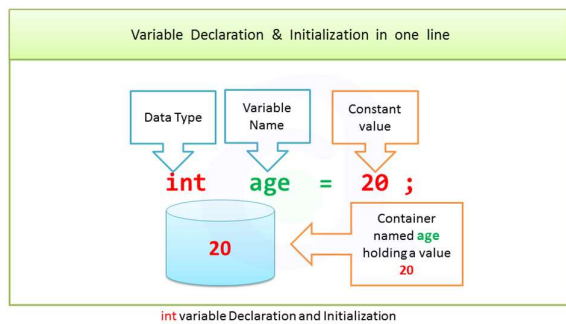
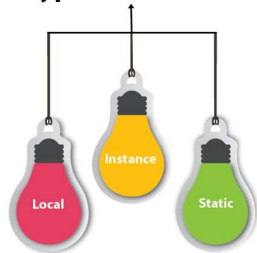
public class ScannerDemo
{
    public static void main(String args[])
    {
        Scanner s=new Scanner(System.in); ← 2. Construct Scanner class Object
        System.out.println("Enter first no= ");

        int num1, num2; ← 3. Define Variable to Receive Input

        num1=s.nextInt(); ← 4. Read Input from Keyboard
        System.out.println("Enter 2nd no ");
        num2=s.nextInt();
        System.out.println("Sum of no is= "+(num1+num2));
    }
}

```

Types of Variables



• Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Instance variable

1. A variable that is declared inside the class but outside the body of the method, constructor, or any block is called instance variable in java.

It is called instance variable because its value is instance specific and is not shared among instances.

2. Instance variables are created when an object is created using the keyword 'new' and destroyed when the object is destroyed.

3. We can also use access modifiers with instance variables. If we do not specify any modifiers, the default access modifiers will be used which can be accessed in the same package only.

4. It is not necessary to initialize the instance variable.

5. Instance variables have default values. For numbers, the default value is 0, for Booleans it is false. Values can be assigned during the declaration or within the constructor.

Static Variables

- Static variables are also known as Class variables.
- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of Static Variable is not Mandatory. Its default value is 0
- If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name to class name automatically.
- If we access the static variable without the class name, Compiler will automatically append the class name.

To access static variables, we need not create an object of that class, we can simply access the variable as

`class_name.variable_name;`

Instance variables	Static (class) variables
Instance variables are declared in a class, but outside a method, constructor or any block.	Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.	Static variables are created when the program starts and destroyed when the program stops.
Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. ObjectReference.VariableName.	Static variables can be accessed by calling with the class name ClassName.VariableName.
Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.	There would only be one copy of each class variable per class, regardless of how many objects are created from it.

METHOD in java

- In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task.
- It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again.
- It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.

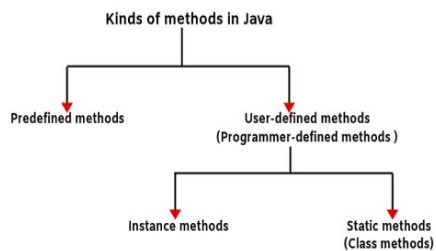
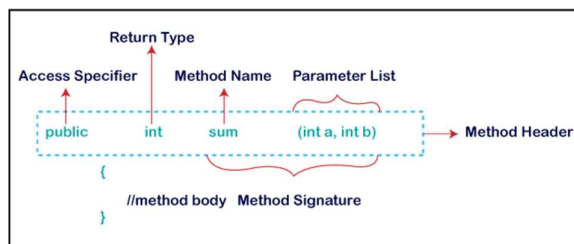


Fig: Basic kinds of methods in Java

Method Declaration



Sr. No	Instance Methods	Static (Class) Methods
1	Instance methods are used to perform repetitive tasks like reading records from the file, reading records from the DBMS etc...	Static methods are used to perform single operation like opening the files, obtaining a DBMS connection etc...
2	Methods definition does not require a static keyword to start. Syntax: void net_salary (parameters if any) { Block of statements; }	Methods definition must start with the static keyword. Syntax: static void basic_salary (parameters if any) { Block of statements; }
3	Each and every instance method must be accessed with the respective Object name	Each and every static method must be accessed with the respective Class name
4	Result of instance method is not shared. Every object has its own copy of instance method.	Results of static method always shared by objects of the same class.

When we declare a static method in Java, the JVM first executes the static method, and then it creates the objects of the class. Since the objects are not available at the time of calling the static method.

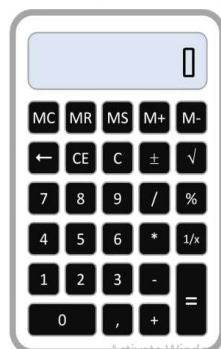
Therefore, the instance variables are also not available to a static method. Due to which a static method cannot access an instance variable in the class.

Real life example of polymorphism in Java

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

Compile-Time Or Static Polymorphism

Compile-time polymorphism refers to behaviour that is resolved when your Java class is compiled. Method overloading is an example of compile-time polymorphism. Method overloading is how you can use method with same name to do different things based on the parameters passed. The add method in Class Calculator for example can add 2 integers or floats based on the types of parameters.



A calculator can add 2 integers. It can also add 2 floats.
The addition method adds differently based on the inputs.

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Advantage of method overloading

- Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{  
  
    static int add(int a,int b){return a+b;}  
  
    static double add(int a,int b){return a+b;}  
  
}  
  
class TestOverloading3{  
  
    public static void main(String[] args){  
  
        System.out.println(Adder.add(11,11));//ambiguity  
  
    }  
}
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading4{  
  
    public static void main(String[] args){System.out.println("main with String[]");}  
  
    public static void main(String args){System.out.println("main with String");}  
  
    public static void main(){System.out.println("main without args");}  
  
}
```

Constructor overloading in Java means to define multiple constructors but each one must have a different signature. It is a technique in Java in which a class can have more than one constructor that differ in the parameters list.

The advantages of using constructor overloading in java programming are as follows:

- 1. Java constructor overloading helps to achieve static polymorphism.
- 2. The main advantage of constructor overloading is to allow an instance of a class to be initialized in various ways.

- 3. It allows to define multiple constructors of a class with different signatures.

Why use super keyword in java?

- Whenever inherit base class data into derived class it is chance to get ambiguity, because may be base class and derived class data are same so to difference these data need to use super keyword
- The super keyword in Java is a reference variable which is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
- Usage of Java super Keyword**
 - super can be used to refer immediate parent class instance variable.
 - super can be used to invoke immediate parent class method.
 - super() can be used to invoke immediate parent class constructor.

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

Run-Time Or Dynamic Polymorphism



Run-time polymorphism refers to behaviour that is resolved when your Java class is run by the JVM. Method overriding by the sub-class is an example of run-time polymorphism. Method overriding allows child classes to provide their own implementation of a method also defined in the parent class. The JVM decides which version of the method (the child's or the parent's) to call based on the object through which the method is invoked.



Consider a SeniorCitizenAccount class. It will provide a calculateInterest method() that overrides the calculateInterest() method in the parent SavingsAccount class

Can we override a static method?

No, we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.

The calling of method depends upon the type of object that calls the static method. It means:

If we call a static method by using the parent class object, the original static method will be called from the parent class.

If we call a static method by using the child class object, the static method of the child class will be called.

In the following example, the ParentClass has a static method named display() and the ChildClass also has the same method signature. The method in the derived class (ChildClass) hides the method in the base class. let's see an example.

Final keyword in java

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level. In java language final keyword can be used in following way.

- Final Keyword at Variable Level
- Final Keyword at Method Level
- Final Keyword at Class Level

Why copy constructor is required?


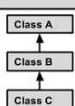
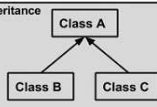
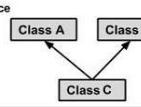
Sometimes, we face a problem where we required to create an exact copy of an existing object of the class. There is also a condition, if we have made any changes in the copy it should not reflect in the original one and vice-versa. For such cases, Java provides the concept of a copy constructor.

Copy Constructor

In Java, a copy constructor is a special type of constructor that creates an object using another object of the same Java class. It returns a duplicate copy of an existing object of the class.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Single Inheritance	 <pre> public class A { } public class B extends A { } </pre>
Multi Level Inheritance	 <pre> public class A { } public class B extends A { } public class C extends B { } </pre>
Hierarchical Inheritance	 <pre> public class A { } public class B extends A { } public class C extends A { } </pre>
Multiple Inheritance	 <pre> public class A { } public class B { } public class C extends A, B { } // Java does not support multiple inheritance </pre>

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

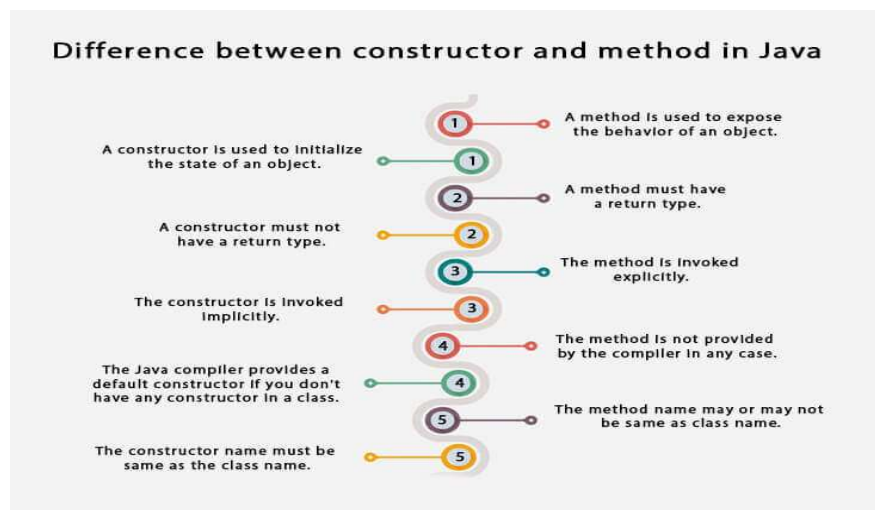
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).



Abstract Class - Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.

- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. **It is used to achieve abstraction and multiple inheritance in Java.**

- **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.
9) Example: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

Why String is Immutable in Java?

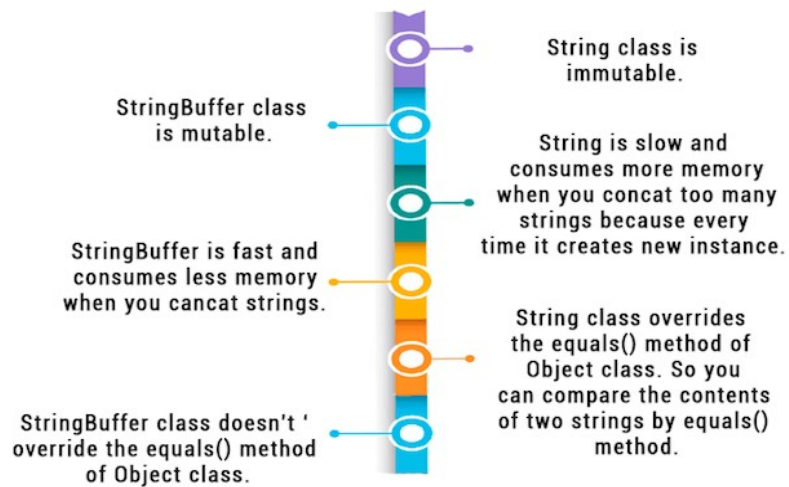
String class in Java is immutable. The meaning of immutable is unchangeable or unmodifiable. i.e. Once we create a string object with value, you are not allowed to perform any changes in that object.

In other words, you cannot modify the value of the string. But if you try to change with a new value, a new string object will be created by storing a new value.

So, we cannot perform any changes with the existing string object. This non-changeable behavior is nothing but an immutability concept in Java.

Java implements this immutability concept to minimize the duplication of string values that tend to exist many times in any application program.

StringBuffer vs String



☐ Exception

- ☐ Exception is an abnormal condition that arises at run time
- ☐ Event that disrupts the normal flow of the program.
- ☐ It is an object which is thrown at runtime.

☐ Exception Handling

- ☐ Exception Handling is a mechanism to handle runtime errors.
- ☐ Normal flow of the application can be maintained.
- ☐ It is an object which is thrown at runtime.
- ☐ Exception handling done with the **exception object**.

	Checked Exception	Un-Checked Exception		Error	Exception
1	checked Exception are checked at compile time	un-checked Exception are checked at run time	1	Can't be handle.	Can be handle.
2	e.g. FileNotFoundException, NumberNotFoundException etc.	e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.	2	Example: OutOfMemoryError	Example: ClassNotFoundException NumberFormatException

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Finally block

- ❑ is a block that is always executed.
- ❑ To perform some important tasks such as closing connection, stream etc.
- ❑ Used to put "cleanup" code such as closing a file, closing connection etc.
- ❑ Finally creates a block of code that will be executed after a try/catch block has completed
- ❑ Finally block will be executed whether or not an exception is thrown.
- ❑ Each try clause requires at least one catch or finally clause.
- **Note:** Before terminating the program, JVM executes finally block(if any).
- **Note:** finally must be followed by try or catch block.

Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Applet

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- Any applet in Java is a class that extends the **java.applet.Applet** class.
- An Applet class does not have any main() method.

- What are the differences between applications and applets ?

	Applications	Applets
main() method	Needed	Not needed
Run stand-alone	Possible	Not possible
Access local files	Can access	Cannot access
Run local programs	Can run	Cannot run
Internet connection	Generally not needed	Needed for Remote applet
GUI	May or may not	GUI based