# Object Oriented Programming (OOPs)

<https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>

**Why is Java called the ‘Platform Independent Programming Language’?**

Platform independence means that execution of your program does not dependent on type of operating system(it could be any : Linux, windows, Mac ..etc). So compile code only once and run it on any System (In C/C++, we need to compile the code for every machine on which we run it). Java is both compiler(javac) and interpreter(jvm) based lauguage. Your java source code is first compiled into byte code using javac compiler. This byte code can be easily converted to equivalent machine code using JVM. JVM(Java Virtual Machine) is available in all operating systems we install. Hence, byte code generated by javac is universal and can be converted to machine code on any operating system, this is the reason why java is platform independent.

**Object-oriented programming:** As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

**OOPs Concepts:**

* [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)
* [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
* [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
* [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
* [Class](https://www.geeksforgeeks.org/classes-objects-java/)
* [Object](https://www.geeksforgeeks.org/classes-objects-java/)
* [Method](https://www.geeksforgeeks.org/methods-in-java/)
* [Message Passing](https://www.geeksforgeeks.org/message-passing-in-java/)

# Polymorphism in Java

<https://www.geeksforgeeks.org/polymorphism-in-java/>

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

**Real life example of polymorphism:** A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

**In Java polymorphism is mainly divided into two types:**

* Compile time Polymorphism
* Runtime Polymorphism

1. **Compile time polymorphism**: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

**Method Overloading**: When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

**Operator Overloading**: Java also provide option to overload operators. For example, we can make the operator (‘+’) for string class to concatenate two strings. We know that this is the addition operator whose task is to add two operands. So a single operator ‘+’ when placed between integer operands, adds them and when placed between string operands, concatenates them.

In java, Only “+” operator can be overloaded:

* To add integers
* To concatenate strings

1. [**Runtime polymorphism**](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/): It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

[**Method overriding**](https://www.geeksforgeeks.org/overriding-in-java/), on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

**Method Overloading** <https://www.geeksforgeeks.org/different-ways-method-overloading-java/>

<https://www.scientecheasy.com/2019/02/method-overloading-in-java.html/>

**Method overriding** <https://www.geeksforgeeks.org/overriding-in-java/>

# Access Modifiers in Java

<https://www.geeksforgeeks.org/access-modifiers-java/>

# Inheritance in Java

<https://www.geeksforgeeks.org/inheritance-in-java/>

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.  
**Important terminology:**

* **Super Class:**The class whose features are inherited is known as super class(or a base class or a parent class).
* **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:**Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

**Types of Inheritance in Java**

Below are the different types of inheritance which is supported by Java.

* 1. **Single Inheritance :**In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.
  2. **Multilevel Inheritance :**In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).
  3. **Hierarchical Inheritance :**In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.
  4. [**Multiple Inheritance**](https://www.geeksforgeeks.org/java-and-multiple-inheritance/)**(Through Interfaces) :**In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. In java, we can achieve multiple inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/). In image below, Class C is derived from interface A and B
  5. **Hybrid Inheritance(Through Interfaces) :**It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through [Interfaces](http://quiz.geeksforgeeks.org/interfaces-in-java/).

# Encapsulation in Java

<https://www.geeksforgeeks.org/encapsulation-in-java/>  [Encapsulation](https://www.javacodegeeks.com/2013/04/the-three-greatest-paragraphs-ever-written-on-encapsulation.html) provides objects with the ability to hide their internal characteristics and behavior. Each object provides a number of methods, which can be accessed by other objects and change its internal data.

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates.Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

* Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

**Advantages of Encapsulation**:

* **Data Hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user that how the class is storing values in the variables. He only knows that we are passing the values to a setter method and variables are getting initialized with that value.
* **Increased Flexibility:** We can make the variables of the class as read-only or write-only depending on our requirement. If we wish to make the variables as read-only then we have to omit the setter methods like setName(), setAge() etc. from the above program or if we wish to make the variables as write-only then we have to omit the get methods like getName(), getAge() etc. from the above program
* **Reusability:** Encapsulation also improves the re-usability and easy to change with new requirements.
* **Testing code is easy:** Encapsulated code is easy to test for unit testing.

# Abstraction in Java

<https://www.geeksforgeeks.org/abstraction-in-java-2/>

**Abstraction:**Hiding the internal implementation of the feature and only showing the functionality to the users. i.e. what it works (showing), how it works (hiding). Both [abstract class](https://www.geeksforgeeks.org/abstract-classes-in-java/) and [interface](http://quiz.geeksforgeeks.org/interfaces-in-java/) are used for abstraction.

Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

**Abstract classes and Abstract methods :**

1. An abstract class is a class that is declared with [abstract keyword.](https://www.geeksforgeeks.org/abstract-keyword-in-java/)
2. An abstract method is a method that is declared without an implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method defined abstract must always be redefined in the subclass,thus making [overriding](http://contribute.geeksforgeeks.org/overriding-in-java/) compulsory OR either make subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with abstract keyword.
6. There can be no object of an abstract class.That is, an abstract class can not be directly instantiated with the [*new operator*](https://www.geeksforgeeks.org/new-operator-java/).
7. An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

**What are the differences between Abstraction and Encapsulation?**

Abstraction and encapsulation are complementary concepts. On the one hand, abstraction focuses on the behavior of an object. On the other hand, encapsulation focuses on the implementation of an object’s behavior. Encapsulation is usually achieved by hiding information about the internal state of an object and thus, can be seen as a strategy used in order to provide abstraction.

# Interface in Java

<https://www.geeksforgeeks.org/inheritance-in-java/>

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

* Interfaces specify what a class must do and not how. It is the blueprint of the class.
* An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

**Why do we use interface ?**

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
* It is also used to achieve loose coupling.
* Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

|  |
| --- |
| // A simple interface  interface Player  {      final int id = 10;      int move();  } |

* To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

**Marker interface in Java**

<https://www.geeksforgeeks.org/marker-interface-java/>

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Clonnable and Remote interface. All these interfaces are empty interfaces.

# Packages In Java

<https://www.geeksforgeeks.org/packages-in-java/>

**Package** in [Java](https://www.geeksforgeeks.org/java/) is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

* Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
* Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
* Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
* Packages can be considered as data encapsulation (or data-hiding).

# Classes and Objects in Java

<https://www.geeksforgeeks.org/classes-objects-java/>

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

**Class**

A class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers** : A class can be public or has default access (Refer [this](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/) for details).
2. **Class name:** The name should begin with a initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

There are various types of classes that are used in real time applications such as [nested classes](https://www.geeksforgeeks.org/inner-class-java/), [anonymous classes](https://www.geeksforgeeks.org/anonymous-inner-class-java/), [lambda expressions](https://www.geeksforgeeks.org/lambda-expressions-java-8/).

**Object**

It is a basic unit of Object Oriented Programming and represents the real life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

**Ways to create object of a class**

There are four ways to create objects in java.Strictly speaking there is only one way(by using *new* keyword),and the rest internally use *new* keyword.

**Using new keyword**: It is the most common and general way to create object in java. Example:

Test t = new Test();

**Using Class.forName(String className) method**: There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name.We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

Test obj = (Test)Class.forName("com.p1.Test").newInstance();

**Using clone() method**: clone() method is present in Object class. It creates and returns a copy of the object.

t1 = new Test();

Test t2 = (Test)t1.clone();

**Deserialization**: De-serialization is technique of reading an object from the saved state in a file. Refer [Serialization/De-Serialization in java](http://quiz.geeksforgeeks.org/serialization-in-java/)

FileInputStream file = new FileInputStream(filename);

ObjectInputStream in = new ObjectInputStream(file);

Object obj = in.readObject()

# Methods in Java

<https://www.geeksforgeeks.org/methods-in-java/>

A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code.

**Method Declaration**

In general, method declarations has six components :

* **Modifier**-: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  + public: accessible in all class in your application.
  + protected: accessible within the class in which it is defined and in its **subclass(es)**
  + private: accessible only within the class in which it is defined.
  + default (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
* **The return type** : The data type of the value returned by the method or void if does not return a value.
* **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
* **Parameter list**: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
* **Exception list**: The exceptions you expect by the method can throw, you can specify these exception(s).
* **Method body**: it is enclosed between braces. The code you need to be executed to perform your intended operations.

**Memory allocation for methods calls**

Methods calls are implemented through stack. Whenever a method is called a stack frame is created within the stack area and after that the arguments passed to and the local variables and value to be returned by this called method are stored in this stack frame and when execution of the called method is finished, the allocated stack frame would be deleted. There is a stack pointer register that tracks the top of the stack which is adjusted accordingly.

# Constructors in Java

<https://www.geeksforgeeks.org/constructors-in-java/>

Constructors are used to initialize the object’s state. Like [methods](https://www.geeksforgeeks.org/methods-in-java/), a constructor also contains **collection of statements(i.e. instructions)** that are executed at time of Object creation.

**Need of Constructor**  
Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in computer’s memory), then can a box be there with no value defined for its dimensions. The answer is no.  
So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

**When is a Constructor called ?**  
Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members**of the same class.

A constructor is invoked at the time of object or instance creation. For Example:

class Geek

{

.......

// A Constructor

new Geek() {}

.......

}

// We can create an object of the above class

// using the below statement. This statement

// calls above constructor.

Geek obj = new Geek();

**Rules for writing Constructor:**

* Constructor(s) of a class must has same name as the class name in which it resides.
* A constructor in Java can not be abstract, final, static and Synchronized.
* Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

**Types of constructor**

There are two type of constructor in Java:

1. **No-argument constructor:**A constructor that has no parameter is known as default constructor. If we don’t define a constructor in a class, then compiler creates **default constructor(with no arguments)** for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.  
   Default constructor provides the default values to the object like 0, null, etc. depending on the type.
2. **Parameterized Constructor:**A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then use a parameterized constructor.

**Does constructor return any value?**

There are no “return value” statements in constructor, but constructor returns current class instance. We can write ‘return’ inside a constructor.

**Constructor Overloading**

Like methods, we can overload constructors for creating objects in different ways. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

**How constructors are different from methods in Java?**

* Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.
* Constructor(s) do not return any type while method(s) have the return type or **void** if does not return any value.
* Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

**Why Constructors are not inherited in Java?**

Constructor is a block of code that allows you to create an object of class and has same name as class with no explicit return type.

Whenever a class (child class) extends another class (parent class), the sub class inherits state and behavior in the form of variables and methods from its super class but it does not inherit constructor of super class because of following reasons:

* Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.
* Now suppose if constructors can be inherited then it will be impossible to achieving encapsulation. Because by using a super class’s constructor we can access/initialize private members of a class.
* A constructor cannot be called as a method. It is called when object of the class is created so it does not make sense of creating child class object using parent class constructor notation. i.e. Child c = new Parent();
* A parent class constructor is not inherited in child class and this is why super() is added automatically in child class constructor if there is no explicit call to super or this.

# Constructors Interview Questions

<https://www.geeksforgeeks.org/java-interview-questions-constructors/>

1. **What is a**[**Constructor?**](https://www.geeksforgeeks.org/constructors-in-java/)  
   Constructors are used to initialize the object’s state. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.
2. **Do we have**[**Copy Constructor in Java**](https://www.geeksforgeeks.org/copy-constructor-in-java/)**?**  
   Like C++, Java also supports copy constructor. But, unlike C++, Java **doesn’t create a default copy constructor** if you don’t write your own.  
   To copy the values of one object into another in java, you can use:
   * Constructor
   * Assigning the values of one object into another
   * [clone() method](https://www.geeksforgeeks.org/clone-method-in-java-2/) of Object class
3. **What is**[**Constructor Chaining**](https://www.geeksforgeeks.org/constructor-chaining-java-examples/)**?**  
   Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call same class constructor where as super() is used to call super class constructor.

// Java program to illustrate Constructor Chaining

// within same class Using this() keyword

class Temp

{

    // default constructor 1

    // default constructor will call another constructor

    // using this keyword from same class

    Temp()

    {

        // calls constructor 2

        this(5);

        System.out.println("The Default constructor");

    }

    // parameterized constructor 2

    Temp(int x)

    {

        // calls constructor 3

        this(5, 15);

        System.out.println(x);

    }

    // parameterized constructor 3

    Temp(int x, int y)

    {

        System.out.println(x \* y);

    }

    public static void main(String args[])

    {

        // invokes default constructor first

        new Temp();

    }

}

1. **Can we call sub class constructor from super class constructor?**  
   No. There is no way in java to call sub class constructor from a super class constructor.
2. **What happens if you keep a return type for a constructor?**  
   Ideally, Constructor must not have a return type. By definition, if a method has a return type, it’s not a constructor.([JLS8.8 Declaration](http://docs.oracle.com/javase/specs/#8.8)) It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.Example:

class GfG {

int GfG() {

return 0; // Warning for the return type

}

}

1. **What is No-arg constructor?**  
   Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

class GfG {

public GfG() { //No-arg constructor }

}

1. **How a no – argument constructor is different from**[**default Constructor**](https://www.geeksforgeeks.org/g-fact-50/)**?**  
   If a class contains no constructor declarations, then a default constructor with no formal parameters and no throws clause is implicitly declared.

If the class being declared is the primordial class Object, then the default constructor has an empty body. Otherwise, the default constructor simply invokes the superclass constructor with no arguments.

1. **What are**[**private constructors**](https://www.geeksforgeeks.org/private-constructors-and-singleton-classes-in-java/)**and where are they used?**  
   Like any method we can provide access specifier to the constructor. If it’s made private, then it can only be accessed inside the class.  
   The major scenarios where we use private constructor:
   * Internal Constructor chaining
   * Singleton class design pattern
2. **When do we need**[**Constructor Overloading**](https://www.geeksforgeeks.org/constructor-overloading-java/)**?**  
   Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. Different constructors can do different work by implementing different line of codes and are called based on the type and no of parameters passed.  
   According to the situation , a constructor is called with specific number of parameters among overloaded constructors.
3. **Do we have destructors in Java?**  
   No, Because Java is a garbage collected language you cannot predict when (or even if) an object will be destroyed. Hence there is no direct equivalent of a destructor.

# static keyword in java

<https://www.geeksforgeeks.org/static-keyword-java/>

*static* is a non-access modifier in Java which is applicable for the following:

1. blocks
2. variables
3. methods
4. nested classes

To create a static member(block,variable,method,nested class), precede its declaration with the keyword *static*. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in below java program, we are accessing static method *m1()* without creating any object of *Test* class.

**Static blocks**

If you need to do computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded. Consider the following java program demonstrating use of static blocks.

// Java program to demonstrate use of static blocks

class Test

{

    // static variable

    static int a = 10;

    static int b;

    // static block

    static {

        System.out.println("Static block initialized.");

        b = a \* 4;

    }

    public static void main(String[] args)

    {

       System.out.println("from main");

       System.out.println("Value of a : "+a);

       System.out.println("Value of b : "+b);

    }

}

Output:

Static block initialized.

from main

Value of a : 10

Value of b : 40

For Detailed article on static blocks, see [static blocks](https://www.geeksforgeeks.org/g-fact-79/)

**Static variables**

When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Important points for static variables :-**

* We can create static variables at class-level only. See [here](https://www.geeksforgeeks.org/g-fact-47/)
* static block and static variables are executed in order they are present in a program.

Below is the java program to demonstrate that static block and static variables are executed in order they are present in a program.

|  |
| --- |
| // java program to demonstrate execution  // of static blocks and variables  class Test  {      // static variable      static int a = m1();        // static block      static {          System.out.println("Inside static block");      }        // static method      static int m1() {          System.out.println("from m1");          return 20;      }        // static method(main !!)      public static void main(String[] args)      {         System.out.println("Value of a : "+a);         System.out.println("from main");      }      } |

Output:

from m1

Inside static block

Value of a : 20

from main

**Static methods**

When a method is declared with static keyword, it is known as static method. The most common example of a static method is main( ) method.As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to [this](https://www.geeksforgeeks.org/this-reference-in-java/) or [super](https://www.geeksforgeeks.org/super-keyword/) in any way.

Below is the java program to demonstrate restrictions on static methods.

|  |
| --- |
| // java program to demonstrate restriction on static methods  class Test  {      // static variable      static int a = 10;        // instance variable      int b = 20;        // static method      static void m1()      {          a = 20;          System.out.println("from m1");             // Cannot make a static reference to the non-static field b           b = 10; // compilation error             // Cannot make a static reference to the                   // non-static method m2() from the type Test           m2();  // compilation error             //  Cannot use super in a static context           System.out.println(super.a); // compiler error      }        // instance method      void m2()      {          System.out.println("from m2");      }            public static void main(String[] args)      {          // main method      }  } |

**When to use static variables and methods?**

Use the static variable for the property that is common to all objects. For example, in class Student, all students shares the same college name. Use static methods for changing static variables.

Consider the following java program, that illustrate the use of *static* keyword with variables and methods.

|  |
| --- |
| // A java program to demonstrate use of  // static keyword with methods and variables    // Student class  class Student  {      String name;      int rollNo;        // static variable      static String cllgName;        // static counter to set unique roll no      static int counter = 0;          public Student(String name)      {          this.name = name;            this.rollNo = setRollNo();      }        // getting unique rollNo      // through static variable(counter)      static int setRollNo()      {          counter++;          return counter;      }        // static method      static void setCllg(String name){          cllgName = name ;      }        // instance method      void getStudentInfo(){          System.out.println("name : " + this.name);          System.out.println("rollNo : " + this.rollNo);            // accessing static variable          System.out.println("cllgName : " + cllgName);      }  }    //Driver class  public class StaticDemo  {      public static void main(String[] args)      {          // calling static method          // without instantiating Student class          Student.setCllg("XYZ");            Student s1 = new Student("Alice");          Student s2 = new Student("Bob");            s1.getStudentInfo();          s2.getStudentInfo();        }  } |

Output:

name : Alice

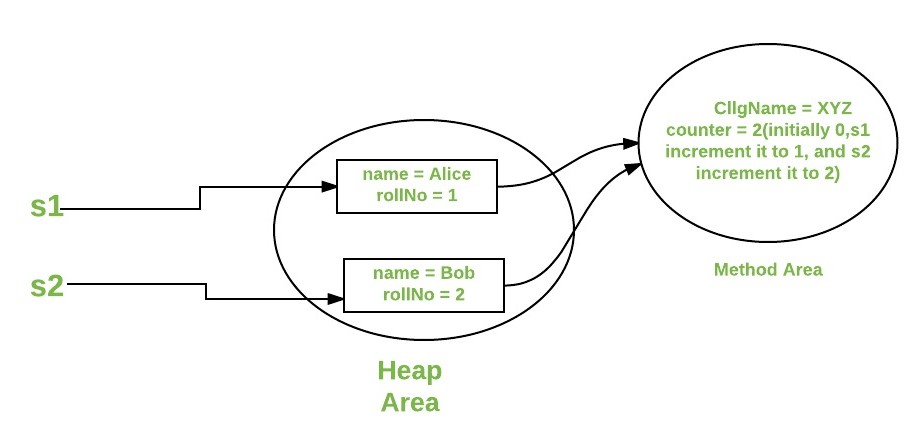
rollNo : 1

cllgName : XYZ

name : Bob

rollNo : 2

cllgName : XYZ



**Static nested classes :**We can not declare top-level class with a static modifier, but can declare [nested classes](https://www.geeksforgeeks.org/nested-classes-java/) as static. Such type of classes are called Nested static classes. For static nested class, see [static nested class in java](https://www.geeksforgeeks.org/static-class-in-java/)

# Java Interview Questions 1

<https://www.geeksforgeeks.org/commonly-asked-java-programming-interview-questions-set-1/?ref=rp>

**Explain Final keyword in java?**

Final keyword in java is used to restrict usage of variable, class and method.  
  
Variable: Value of Final variable is constant, you can not change it.  
Method: you can’t override a Final method.  
Class: you can’t inherit from Final class.

**When is the super keyword used?**  
super keyword is used to refer:

* immediate parent class constructor,
* immediate parent class variable,
* immediate parent class method.

**What is the difference between StringBuffer and String?**

String is an Immutable class, i.e. you can not modify its content once created. While StringBuffer is a mutable class, means you can change its content later. Whenever we alter content of String object, it creates a new string and refer to that,it does not modify the existing one. This is the reason that the performance with StringBuffer is better than with String.  
Refer [this](https://www.geeksforgeeks.org/g-fact-27-string-vs-stringbuilder-vs-stringbuffer/) for details.

**Why multiple inheritance is not supported in java?**

Java supports multiple inheritance but not through classes, it supports only through its interfaces. The reason for not supporting multiple inheritance is to avoid the conflict and complexity arises due to it and keep Java a Simple Object Oriented Language. If we recall [this in C++](https://www.geeksforgeeks.org/multiple-inheritance-in-c/), there is a special case of multiple inheritance (diamond problem) where you have a multiple inheritance with two classes which have methods in conflicts. So, Java developers decided to avoid such conflicts and didn’t allow multiple inheritance through classes at all.

**Can a top level class be private or protected?**

Top level classes in java can’t be private or protected, but inner classes in java can. The reason for not making a top level class as private is very obvious, because nobody can see a private class and thus they can not use it. Declaring a class as protected also doesn’t make any sense. The only difference between default visibility and protected visibility is that we can use it in any package by inheriting it. Since in java there is no such concept of package inheritance, defining a class as protected is no different from default.

**What is the difference between ‘throw’ and ‘throws’ in Java Exception Handling?**

Following are the differences between two:

* throw keyword is used to throw Exception from any method or static block whereas throws is used to indicate that which Exception can possibly be thrown by this method
* If any method throws checked Exception, then caller can either handle this exception(using try catch block )or can re throw it by declaring another ‘throws’ clause in method declaration.
* throw clause can be used in any part of code where you feel a specific exception needs to be thrown to the calling method

E.g.  
**throw**  
throw new Exception(“You have some exception”)  
throw new IOException(“Connection failed!!”)  
**throws**  
throws IOException, NullPointerException, ArithmeticException

**What is finalize() method?**  
   
Unlike c++ , we don’t need to destroy objects explicitly in Java. ‘[Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/)‘ does that automatically for us. Garbage Collector checks if no references to an object exist, that object is assumed to be no longer required, and the memory occupied by the object can be freed. Sometimes an object can hold non-java resources such as file handle or database connection, then you want to make sure these resources are also released before object is destroyed. To perform such operation Java provide protected void finalize() in object class. You can override this method in your class and do the required tasks. Right before an object is freed, the java run time calls the finalize() method on that object. Refer [this](https://www.geeksforgeeks.org/garbage-collection-java/) for more details.

**Difference in Set and List interface?**

Set and List both are child interface of Collection interface. There are following two main differences between them

* List can hold duplicate values but Set doesn’t allow this.
* In List interface data is present in the order you inserted but in the case of Set insertion order is not preserved.

**What will happen if you put System.exit(0) on try or catch block? Will finally block execute?**  
   
By Calling System.exit(0) in try or catch block, we can skip the finally block. System.exit(int) method can throw a SecurityException. If Sysytem.exit(0) exits the JVM without throwing that exception then finally block will not execute. But, if System.exit(0) does throw security exception then finally block will be executed.

# Java Interview Questions 2

<https://www.geeksforgeeks.org/commonly-asked-java-programming-interview-questions-set-2/?ref=rp>

**Q1. Can we [Overload or Override static methods in java](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/)?**

* **Overriding :** Overriding is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class) at runtime.
* **Overloading:** Overloading is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters.
* **Can we overload static methods?**The answer is **‘Yes’**. We can have two ore more static methods with same name, but differences in input parameters
* **Can we Override static methods in java?**We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is **‘No’**. Static methods cannot be overridden because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at compile-time).

Read [more](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/)

**Q2. Why the main method is static in java?**  
The method is static because otherwise there would be ambiguity: which constructor should be called? Especially if your class looks like this:

public class JavaClass

{

protected JavaClass(int x)

{ }

public void main(String[] args)

{

}

}

Should the JVM call new JavaClass(int)? What should it pass for x? If not, should the JVM instantiate JavaClass without running any constructor method? because that will special-case your entire class – sometimes you have an instance that hasn’t been initialized, and you have to check for it in every method that could be called. There are just too many edge-cases and ambiguities for it to make sense for the JVM to have to instantiate a class before the entry point is called. That’s why main is static.

**Q3. What happens if you remove static modifier from the main method?**  
Program compiles successfully. But at runtime throws an error “NoSuchMethodError”.

**Q4. What is the [scope of variables](https://www.geeksforgeeks.org/variable-scope-in-java/) in Java in following cases?**

* **Member Variables** (Class Level Scope) : The member variables must be declared inside class (outside any function). They can be directly accessed anywhere in class
* **Local Variables**(Method Level Scope) : Variables declared inside a method have method level scope and can’t be accessed outside the method.
* **Loop Variables**(Block Scope) : A variable declared inside pair of brackets “{” and “}” in a method has scope withing the brackets only.

Read [more](https://www.geeksforgeeks.org/variable-scope-in-java/)

**Q5. What is[“this” keyword in java](http://quiz.geeksforgeeks.org/this-reference-in-java/)?**  
Within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this.  
Usage of this keyword

* Used to refer current class instance variable.
* To invoke current class constructor.
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* Used to return the current class instance.
* Used to invoke current class method (implicitly)

**Q14. What is [“super” keyword in java](http://quiz.geeksforgeeks.org/super-keyword/)?**  
The super keyword in java is a reference variable that is used to refer parent class objects. The keyword “super” came into the picture with the concept of Inheritance. Whenever you create the instance of a subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.  
Various scenarios of using java super Keyword:

* super is used to refer immediate parent instance variable
* super is used to call parent class method
* super() is used to call immediate parent constructor

**Q6. What is an [abstract class](https://www.geeksforgeeks.org/abstract-classes-in-java/)? How abstract classes are similar or different in Java from C++?**  
Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

* Like C++, in Java, an instance of an abstract class cannot be created, we can have references of abstract class type though.
* Like C++, an abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of a inherited class is created
* In Java, we can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated, but can only be inherited.
* Abstract classes can also have final methods (methods that cannot be overridden). For example, the following program compiles and runs fine.

**Q7. Which class is the superclass for every class?**  
Object class

**Q8. [Can we overload main() method?](https://www.geeksforgeeks.org/gfact-48-overloading-main-in-java/)**  
The main method in Java is no extra-terrestrial method. Apart from the fact that main() is just like any other method & can be overloaded in a similar manner, JVM always looks for the method signature to launch the program.

* The normal main method acts as an entry point for the JVM to start the execution of program.
* We can overload the main method in Java. But the program doesn’t execute the overloaded main method when we run your program, we need to call the overloaded main method from the actual main method only.

**Q9. What is [object cloning](http://quiz.geeksforgeeks.org/cloning-in-java/)?**  
Object cloning means to create an exact copy of the original object. If a class needs to support cloning, it must implement java.lang.Cloneable interface and override clone() method from Object class. Syntax of the clone() method is :

protected Object clone() throws CloneNotSupportedException

If the object’s class doesn’t implement Cloneable interface then it throws an exception ‘CloneNotSupportedException’ .

**Q12. Can we override private methods in Java?**  
No, a private method cannot be overridden since it is not visible from any other class. Read [more](https://www.geeksforgeeks.org/can-override-private-methods-java/)

**Q13. What is [blank final variable](https://www.geeksforgeeks.org/blank-final-in-java/)?**  
A final variable in Java can be assigned a value only once, we can assign a value either in declaration or later.

final int i = 10;

i = 30; // Error because i is final.

A **blank final** variable in Java is a [final](https://www.geeksforgeeks.org/g-fact-48/) variable that is not initialized during declaration. Below is a simple example of blank final.

// A simple blank final example

final int i;

i = 30;

**Q15. What is[static variable in Java](https://www.geeksforgeeks.org/static-class-in-java/)?**  
The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than the instance of the class.

The static can be:

* variable (also known as class variable)
* method (also known as class method)
* block
* nested class

**Q16. Differences between [HashMap and HashTable in Java](http://quiz.geeksforgeeks.org/differences-between-hashmap-and-hashtable-in-java/).**  
1. HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.  
2. HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.  
3. HashMap is generally preferred over HashTable if thread synchronization is not needed

**Q17. How are Java [objects stored in memory](https://www.geeksforgeeks.org/g-fact-46/)?**  
In Java, all objects are dynamically allocated on **Heap**.

In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on the heap

The static variables are provided the memory in the the same memory segment where the code is stored i.e. Class Area. It is independent of the stack or heap segment of memory. It remains accessible during the complete duration of the program.

We have 3 segments in our memory:

1. Stack Segment — contains local variables and Reference variables (variables that hold the address of an object in the heap).
2. Heap Segment — contains all created objects in runtime, objects only plus their object attributes (instance variables).
3. Code Segment — the segment where the actual compiled Java bytecodes resides when loaded. Static members (variables or methods) are called class members, meaning they reside where the class (bytecode) resides, which is in the Code Segment.

# Java Interview Questions 3

<https://www.geeksforgeeks.org/interview-questions-for-java-professionals/?ref=rp>

**Q1. Explain**[**JVM, JRE and JDK**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/)**?**

**JVM (Java Virtual Machine):** JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a Java code. JVM is a part of JRE(Java Runtime Environment).

**JRE (Java Runtime Environment):** JRE refers to a runtime environment in which Java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it’s an implementation of the JVM which physically exists.

**JDK(Java Development Kit):** It is the tool necessary to compile, document and package Java programs. The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.

**Q2. Explain public static void main(String args[]).**

**Public:** Public is an access modifier. Public means that this Method will be accessible by any Class.  
**static :** It is a keyword in java which identifies it is class-based i.e it can be accessed without creating the instance of a Class. Since we want the main method to be executed without any instance also, we use static.  
**Void:** It is the return type of the method. Void defines the method which will not return any value.  
**main:** This is the first method executed by JVM. The signature of the method must be the same.

**In System.out.println(), what is System, out and println?**  
System is a predefined final class, out is a PrintStream object and println is a built-in overloaded method in the out object.

**Q3. Why Java is platform independent?**  
Platform independent practically means “write once run anywhere”. Java is called so because of its **byte codes** which can run on any system irrespective of its underlying operating system.

**Q4. Why is Java not pure Object-oriented?**  
Java is not considered pure Object-oriented because it supports primitive data-types such as boolean, byte, char, int, float, double, long, short.

**Q5. Define class and object. Explain them with an example using java.**

**Class:** A class is a user-defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

**Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

**Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

**Object:** It is a basic unit of Object Oriented Programming and represents the real-life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

**State :** It is represented by attributes of an object. It also reflects the properties of an object.

**Behavior :** It is represented by methods of an object. It also reflects the response of an object with other objects.

**Identity :** It gives a unique name to an object and enables one object to interact with other objects.  
For Example: Employee is an example of a class  
A specific employee with unique identification is an example of an object.

**Q7.Explain the difference between instance variable and a class variable.**

An instance variable is a variable which has one copy per object/instance. That means every object will have one copy of it.  
A class variable is a variable which has one copy per class. The class variables will not have a copy in the object.  
Example :

class Employee {

    int empNo;

    String empName, department;

    double salary;

    static int officePhone;

}

An object referred by empObj1 is created by using the following:  
Employee empObj1 = new Employee();

The objects referred by instance variables empObj1 and empObj2 have separate copies empNo, empName, department, and salary. However, the officePhone belongs to the class(Class Variable) and can be accessed as Employee.officePhone.

**Q8. Which class is the superclass of all classes?**  
[java.lang.Object](https://www.geeksforgeeks.org/object-class-in-java/)is the root class for all the java classes and we don’t need to extend it.

**Q9.What are constructors in Java?**  
In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.  
If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor.  
This default constructor calls the class parent’s no-argument constructor (as it contains only one statement i.e. super();), or the Object class constructor if the class has no other parent (as Object class is a parent of all classes either directly or indirectly).  
There are two types of constructors:

1. Default constructor
2. Parametrized constructor

**Q10. What are the different ways to create objects in Java?**  
There are many different ways to create objects in Java.

<https://www.geeksforgeeks.org/different-ways-create-objects-java/>

1. **Using new Keyword :** Using new keyword is the most basic way to create an object. This is the most common way to create an object in java. Almost 99% of objects are created in this way.
2. **Using**[**New Instance**](https://www.geeksforgeeks.org/new-operator-vs-newinstance-method-java/)**:**If we know the name of the class & if it has a public default constructor we can create an object –**Class.forName**. We can use it to create the Object of a Class. Class.forName actually loads the Class in Java but doesn’t create any Object. To Create an Object of the Class you have to use the new Instance Method of the Class.
3. **Using**[**clone() method:**](https://www.geeksforgeeks.org/clone-method-in-java-2/) Whenever clone() is called on any object, the JVM actually creates a new object and copies all content of the previous object into it. Creating an object using the clone method does not invoke any constructor.  
   To use clone() method on an object we need to implement **Cloneable** and define the clone() method in it

**Note :**

* Here we are creating the clone of an existing Object and not any new Object.
* Class need to implement Cloneable Interface otherwise it will throw **CloneNotSupportedException**.

1. **Using**[**deserialization**](https://www.geeksforgeeks.org/serialization-in-java/)**:** Whenever we serialize and then deserialize an object, JVM creates a separate object. In **deserialization**, JVM doesn’t use any constructor to create the object.  
   To deserialize an object we need to implement the Serializable interface in the class.
2. **Using newInstance() method of Constructor class :** This is similar to the newInstance() method of a class. There is one newInstance() method in the **java.lang.reflect.Constructor** class which we can use to create objects. It can also call parameterized constructor, and private constructor by using this newInstance() method.

Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class internally uses newInstance() method of Constructor class.

**Q11. What’s the purpose of Static methods and static variables?**

When there is a requirement to share a method or a variable between multiple objects of a class instead of creating separate copies for each object, we use static keyword to make a method or variable shared for all objects.  
**Static variable:** 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.  
To access static variables, we need not create an object of that class.  
Static methods: A static method can be accessed without creating objects. Just by using the Class name the method can be accessed. The static method can only access static variables and not local or global non-static variables.

**Q12. Why static methods cannot access non-static variables or methods?**

Ans) A static method cannot access non-static variables or methods because static methods can be accessed without instantiating the class, so if the class is not instantiated the variables are not initialized and thus cannot be accessed from a static method.

**Q13.What is a static class?**  
A class can be said to be static class if all the variables and methods of the class are static and the constructor is private. Making the constructor private will prevent the class to be instantiated. So the only possibility to access is using the Class name only.

Q14. How many types of Variable? Explain.  
There are three types of variables in Java:

1. Local Variables
2. Instance Variables
3. Static Variables

Q15. **What is implicit casting**.

Implicit casting is the process of simply assigning one entity to another without any transformation guidance to the compiler. This type of casting is not permitted in all kinds of transformations and may not work for all scenarios.

**Example**

|  |  |
| --- | --- |
|  | int i = 1000;  long j = i; //Implicit casting |

Q16. **What is explicit casting**.

Explicit casting in the process in which the complier are specifically informed to about transforming the object.

**Example**

|  |  |
| --- | --- |
|  | long i = 700.20;  int j = (int) i; //Explicit casting  Q16.**What is explicit casting**.  Explicit casting in the process in which the complier are specifically informed to about transforming the object. |

**What is the first argument of the String array in main method?**

The String array is empty. It does not have any element. If we do not provide any arguments on the command line, then the String array of main method will be empty but not null.

**How can one prove that the array is not null but empty?**

Print array.length. It will print 0. That means it is empty. But if it would have been null then it would have thrown a NullPointerException on attempting to print array.length.

**When is static variable loaded? Is it at compile time or runtime? When exactly a static block is loaded in Java?**

Static variable are loaded when classloader brings the class to the JVM. It is not necessary that an object has to be created. Static variables will be allocated memory space when they have been loaded. The code in a static block is loaded/executed only once i.e. when the class is first initialized. A class can have any number of static blocks. Static block is not member of a class, they do not have a return statement and they cannot be called directly. Cannot contain this or super. They are primarily used to initialize static fields.

**Explain working of Java Virtual Machine (JVM)?**

JVM is an abstract computing machine like any other real computing machine which first converts .java file into .class file by using Compiler (.class is nothing but byte code file.) and Interpreter reads byte codes.

**How can I swap two variables without using a third variable?**

dd two variables and assign the value into First variable. Subtract the Second value with the result Value. and assign to Second variable. Subtract the Result of First Variable With Result of Second Variable and Assign to First Variable. Example:

|  |  |
| --- | --- |
| 1 | int a=5,b=10;a=a+b; b=a-b; a=a-b; |

**What is transient variable?**

Transient variable can’t be serialize. For example if a variable is declared as transient in a Serializable class and the class is written to an ObjectStream, the value of the variable can’t be written to the stream instead when the class is retrieved from the ObjectStream the value of the variable becomes **null**.

**Shadowing of static functions in Java**

In Java, if the name of a derived class static function is the same as a base class static function then the base class static function shadows (or conceals) the derived class static function. For example, the following Java code prints “A.fun()”  
Note: Static method is a class property, so if a static method is called from a class name or object having a class container then the method of that class is called not the object’s method.

// file name: Main.java

// Parent class

class A {

    static void fun()

    {

        System.out.println("A.fun()");

    }

}

// B is inheriting A

// Base class

class B extends A {

    static void fun()

    {

        System.out.println("B.fun()");

    }

}

// Driver Method

public class Main {

    public static void main(String args[])

    {

        A a = new B();

        a.fun(); // prints A.fun();

        // B a = new B();

        // a.fun(); // prints B.fun()

        // the variable type decides the method

        // being invoked, not the assigned object type

    }

}

**Output:**

A.fun()

**Note:**If we make both A.fun() and B.fun() as non-static then the above program would print “B.fun()”. While both methods are static types, the variable type decides the method being invoked, not the assigned object type

# Strings in Java

Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

[String](https://www.geeksforgeeks.org/string-class-in-java/): String is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created.

**Creating a String**

There are two ways to create string in Java:

* ***String literal***

String s = “GeeksforGeeks”;

* **Using *new* keyword**

String s = new String (“GeeksforGeeks”);

**Memory allotment of String**

Whenever a String Object is created, two objects will be created- one in the Heap Area and one in the String constant pool and the String object reference always points to heap area object.

