**Core Java**

Frequently Asked Questions

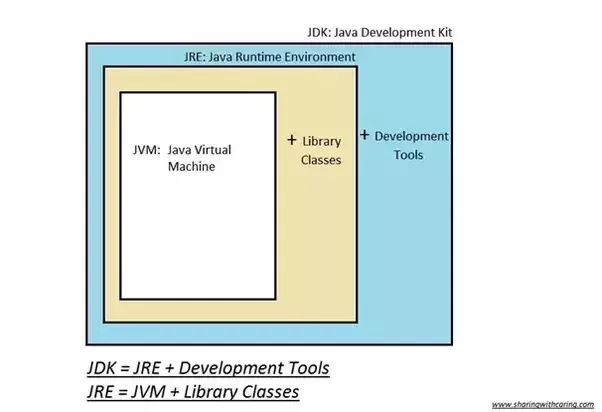
**Java Fundamentals:**

1. What is JVM and Is it platform independent?

Java Virtual Machine (JVM) is the heart of java programming language. JVM is responsible for converting byte code into machine readable code.

JVM is not platform independent, that’s why you have different JVM for different operating systems. so if you are running Mac OS X you will have a different JVM than if you are running Windows or some other operating system. This fact can be verified by trying to download the JVM for your particular machine – when trying to download it, you will given a list of JVM’s corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that you are running. So we can conclude that JVM is platform dependent and it is the reason why Java is able to become “Platform Independent”.

We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It’s called virtual because it provides an interface that doesn’t depend on the underlying OS.

1. What is difference between JDK and JVM and JRE?

* **JDK** – **Java Development Kit** (in short JDK) is Kit which provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) which includes two things
  + 1. Development Tools(to provide an environment to develop your java programs)
    2. JRE (to execute your java program).

**Note :**JDK is only used by Java Developers.

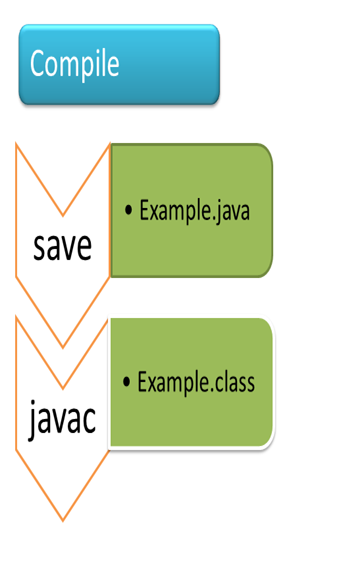
* **JRE** – **Java Runtime Environment** (to say JRE) is an installation package which provides environment to **only run (not develop)** the java program (or application) onto your machine. JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.
* **JVM** – **Java Virtual machine**(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for **executing the java program line by line** hence it is also known as interpreter.

**How does JRE and JDK works:**

**JRE consists:**  
JRE consists of the following components:

* **Deployment technologies**, including deployment, Java Web Start and Java Plug-in.
* **User interface toolkits**, including *Abstract Window Toolkit (AWT), Swing, Java 2D, Accessibility, Image I/O, Print Service, Sound, drag and drop (DnD)* and *input methods*.
* **Integration libraries**, including *Interface Definition Language (IDL), Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), Remote Method Invocation (RMI), Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP)* and *scripting*.
* **Other base libraries**, including *international support, input/output (I/O), extension mechanism, Beans, Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, Override Mechanism, Security, Serialization*and *Java for XML Processing (XML JAXP)*.
* **Lang and util base libraries**, including *lang and util, management, versioning, zip, instrument, reflection, Collections, Concurrency Utilities, Java Archive (JAR), Logging, Preferences API, Ref Objects* and *Regular Expressions*.
* **Java Virtual Machine (JVM)**, including *Java HotSpot Client* and *Server Virtual Machines*.

**How does JRE works:**  
To understand how the JRE works let us consider a Java source file saved as *Example.java*. The file is compiled into a set of Byte Code that is stored in a “*.class*” file. Here it will be “*Example.class*“.

  
  
The following diagram depicts what is done at compile time.  
  
The following actions occur at runtime.

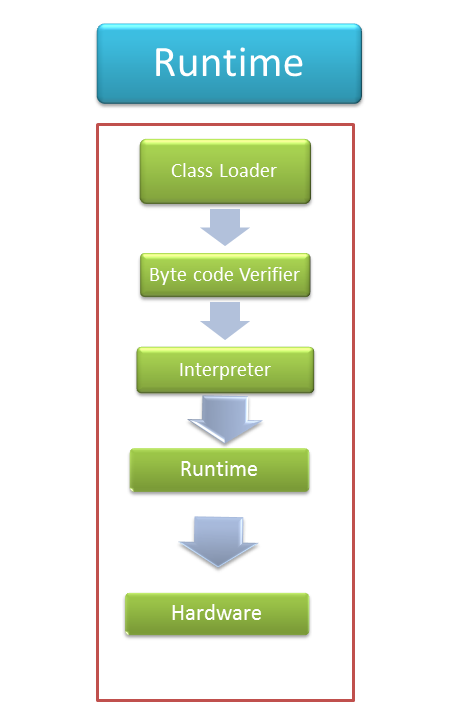
* **Class Loader**

The Class Loader loads all necessary classes needed for the execution of a program. It provides security by separating the namespaces of the local file system from that imported through the network. These files are loaded either from a hard disk, a network or from other sources.

* **Byte Code Verifier**

The JVM puts the code through the Byte Code Verifier that checks the format and checks for an illegal code. Illegal code, for example, is code that violates access rights on objects or violates the implementation of pointers.

The Byte Code verifie1r ensures that the code adheres to the JVM specification and does not violate system integrity.

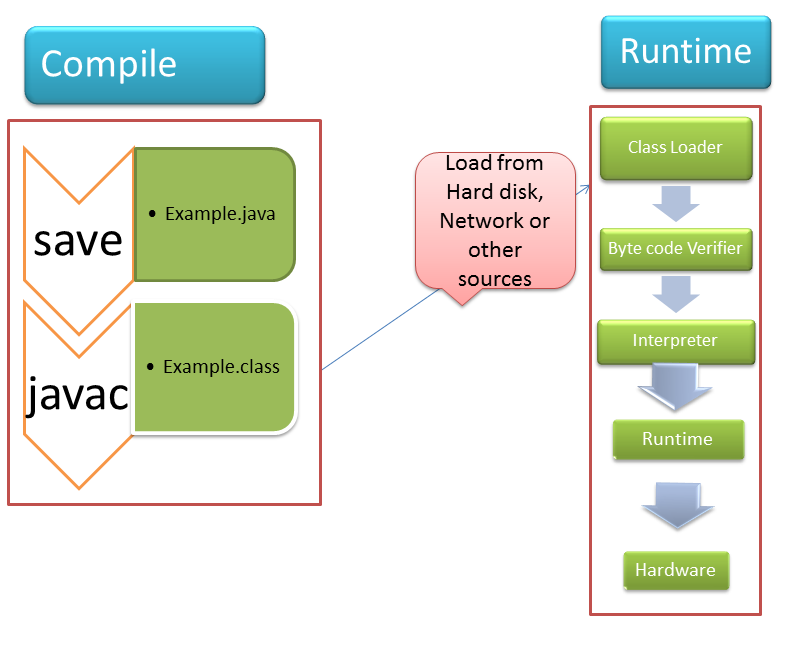


* **Intrepreter**

At runtime the Byte Code is loaded, checked and run by the interpreter. The interpreter has the following two functions:

* + Execute the Byte Code
  + Make appropriate calls to the underlying hardware

Both operations can be shown as:



To understand the interactions between JDK and JRE consider the following diagram.  


How JVM Works – JVM Architecture?

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 Run Environment).

Java applications are called WORA (Write Once Run Everywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java enabled system without any adjustment. This is all possible because of JVM.

When we compile a *.java* file, a *.class* file(contains byte-code) with the same filename is generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.

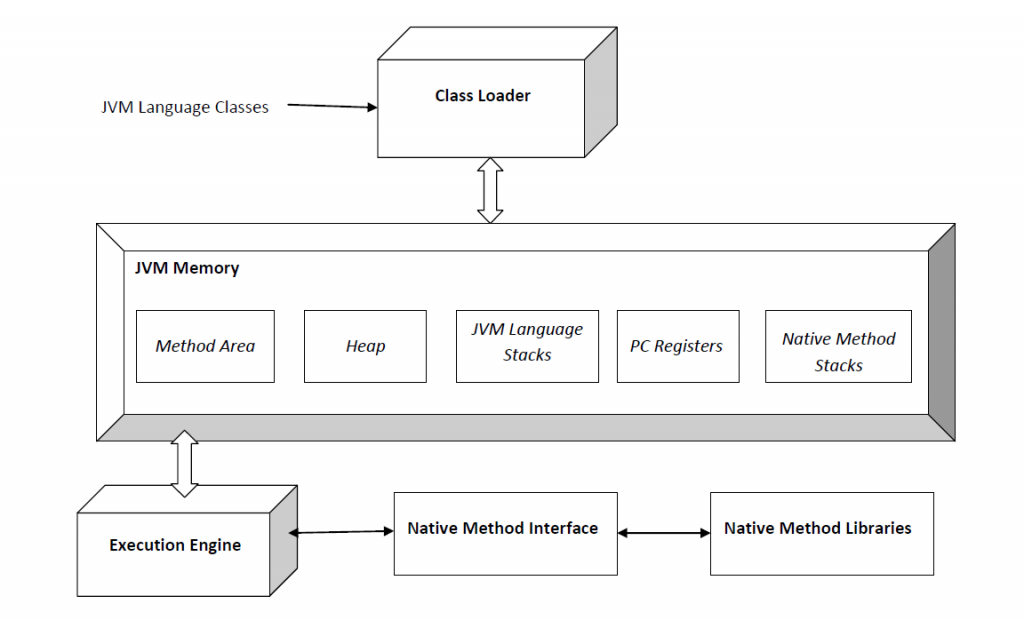
[](https://www.geeksforgeeks.org/wp-content/uploads/JVM.png)

Image: [Java\_virtual\_machine](https://en.wikipedia.org/wiki/Java_virtual_machine)

**Class Loader Subsystem**

It is mainly responsible for three activities.

* Loading
* Linking
* Initialization

**Loading :** The Class loader reads the *.class* file, generate the corresponding binary data and save it in method area. For each *.class* file, JVM stores following information in method area.

* Fully qualified name of the loaded class and its immediate parent class.
* Whether *.class* file is related to Class or Interface or Enum
* Modifier, Variables and Method information etc.

After loading *.class* file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in *java.lang*package. This Class object can be used by the programmer for getting class level information like name of class, parent name, methods and variable information etc. To get this object reference we can use *getClass()* method of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.

1. Why Java is not pure Object Oriented language?

Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language which supports or have features which treats everything inside program as objects. It doesn’t support primitive datatype(like int, char, float, bool, etc.). There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

1. Encapsulation/Data Hiding
2. Inheritance
3. Polymorphism
4. Abstraction
5. All predefined types are objects
6. All user defined types are objects
7. All operations performed on objects must be only through methods exposed at the objects.

Example: Smalltalk

**Java is not a Pure Object Oriented Language. Because:**

Java supports property 1, 2, 3, 4 and 6 but fails to support property 5 and 7 given above. Java language is not a Pure Object Oriented Language as it contain these properties:

* **Primitive Data Type ex. int, long, bool, float, char, etc as Objects:** Smalltalk is a “pure” object-oriented programming language unlike Java and C++ as there is no difference between values which are objects and values which are primitive types. In Smalltalk, primitive values such as integers, booleans and characters are also objects.  
  In Java, we have predefined types as non-objects (primitive types).

int a = 5;

System.out.print(a);

* **The static keyword:**When we declares a class as static then it can be used without the use of an object in Java. If we are using static function or static variable then we can’t call that function or variable by using dot(.) or class object defying object oriented feature.
* **Wrapper Class:** Wrapper class provides the mechanism to convert primitive into object and object into primitive. In Java, you can use Integer, Float etc. instead of int, float etc. We can communicate with objects without calling their methods. ex. using arithmetic operators.

String s1 = "ABC" + "A" ;

Even using Wrapper classes does not make Java a pure OOP language, as internally it will use the operations like Unboxing and Autoboxing. So if you create instead of int Integer and do any mathematical operation on it, under the hoods Java is going to use primitive type int only.

1. What are principle concepts of OOPS?

OOPS simplifies the software development and maintenance by providing some concepts:

1. Object.
2. Class.
3. Inheritance.
4. Polymorphism.
5. Abstraction.
6. Encapsulation.

**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 consist of :

1. **State**: It is represented by attributes of an object. It also reflect 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.

# Inheritance:

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.

**How to use inheritance in Java**

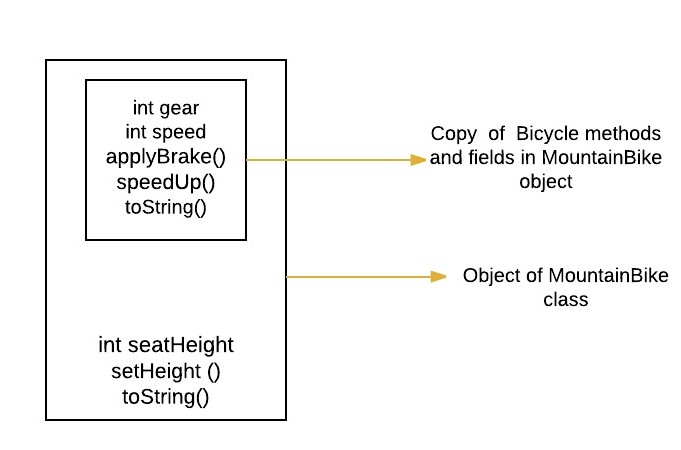
The keyword used for inheritance is **extends**.  
Syntax :

class derived-class extends base-class

{

//methods and fields

}

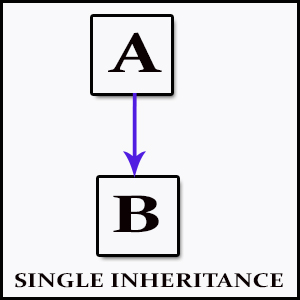
**Illustrative image of the program:**  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/f.jpeg)

In practice, inheritance and [polymorphism](https://www.geeksforgeeks.org/overriding-in-java/) are used together in java to achieve fast performance and readability of code.

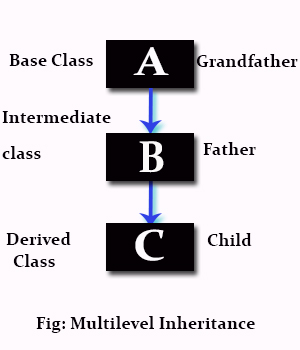
**Types of Inheritance in Java:**

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

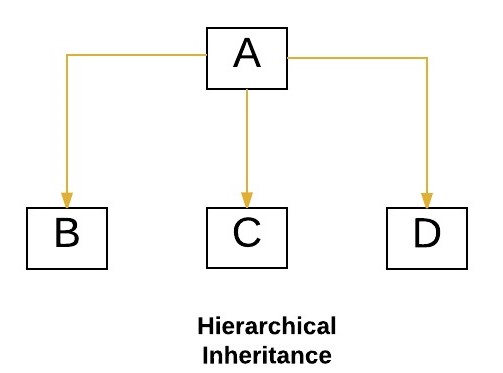
* 1. **Single Inheritance :**In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Single_Inheritance.jpg)

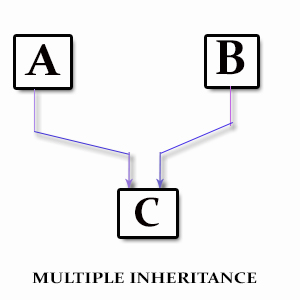
* 1. **Multilevel Inheritance :**In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Multilevel_Inheritance.jpg)

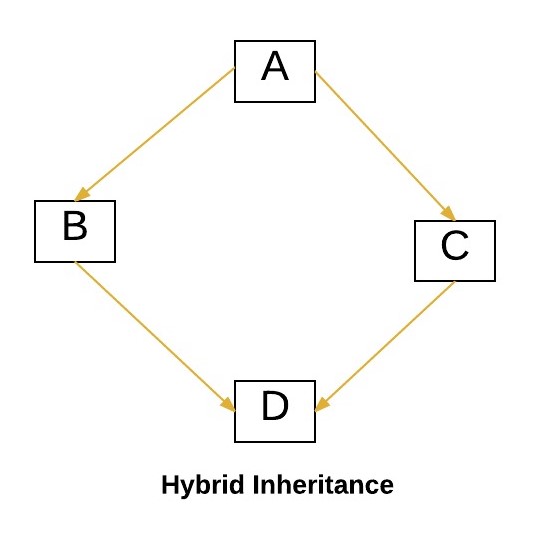
* 1. **Hierarchical Inheritance :**In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/hie.jpeg)

* 1. [**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.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Multiple_Inheritance.jpg)

* 1. **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/).

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/hybrid.jpeg)

**Important facts about inheritance in Java**

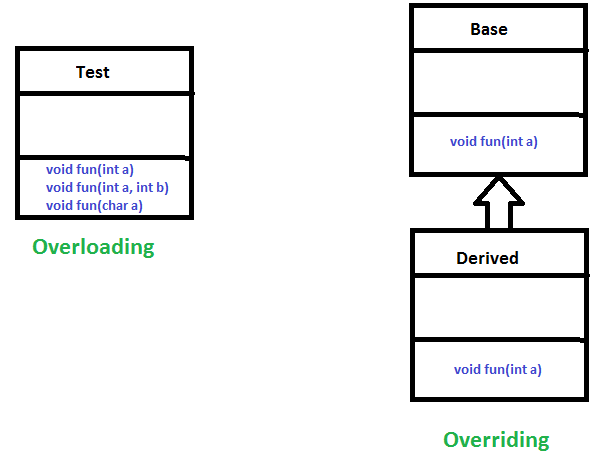
* **Default superclass**: Except [Object](https://www.geeksforgeeks.org/object-class-in-java/) class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.
* **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because Java does not support [multiple inheritance](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. Although with interfaces, multiple inheritance is supported by java.
* **Inheriting Constructors:**A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

**What all can be done in a Subclass?**

In sub-classes we can inherit members as is, replace them, hide them, or supplement them with new members:

* The inherited fields can be used directly, just like any other fields.
* We can declare new fields in the subclass that are not in the superclass.
* The inherited methods can be used directly as they are.
* We can write a new instance method in the subclass that has the same signature as the one in the superclass, thus [overriding](https://www.geeksforgeeks.org/overriding-in-java/) it (as in example above, toString()method is overridden).
* We can write a new static method in the subclass that has the same signature as the one in the superclass, thus [hiding](https://www.geeksforgeeks.org/g-fact-63/) it.
* We can declare new methods in the subclass that are not in the superclass.
* We can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword [super](http://quiz.geeksforgeeks.org/super-keyword/).

#### **Polymorphism:** When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.  
Overloading is about same function have different signatures. Overriding is about same function, same signature but different classes connected through inheritance.  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/OverridingVsOverloading.png)

# 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.

In java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

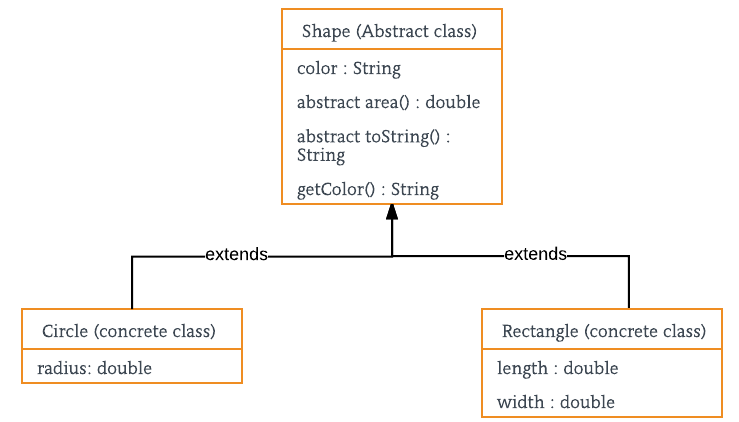
**Abstract classes and Abstract methods :**

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*](http://contribute.geeksforgeeks.org/new-operator-in-java/).
7. An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

**When to use abstract classes and abstract methods with an example**

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

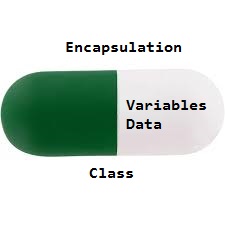
Consider a classic “shape” example, perhaps used in a computer-aided design system or game simulation. The base type is “shape” and each shape has a color, size and so on. From this, specific types of shapes are derived(inherited)-circle, square, triangle and so on – each of which may have additional characteristics and behaviors. For example, certain shapes can be flipped. Some behaviors may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.



# Encapsulation:

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.

* Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
* As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
* 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.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Encapsulation.jpg)

**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.

1. What is Java Package and which package is imported by default?

**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).

All we need to do is put related classes into packages. After that we can simply write a import a class from existing packages and use it in our program. A packages is container of group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.  
We can reuse existing classes from the packages as many time as we need it in our program.

**How packages work:**

Package names and directory structure are closely related. For example if a package name is college.staff.cse, then there are three directories, college, staff and cse such that cse is present in staff and staff is present college. Also, the directory college is accessible through [CLASSPATH](https://en.wikipedia.org/wiki/Classpath_(Java)) variable, i.e., path of parent directory of college is present in CLASSPATH. The idea is to make sure that classes are easy to locate.  
**Package naming conventions:**Packages are named in reverse order of domain names, i.e., org.geeksforgeeks.practice. For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, college.art.history, etc.

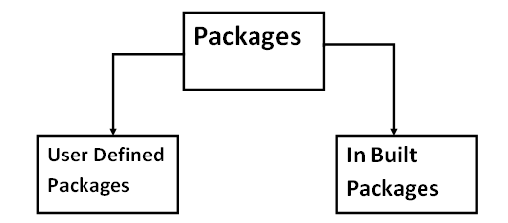
**Adding a class to a Package:** We can add more classes to an created package by using package name at the top of the program and saving it in the package directory. We need a new **java** file to define a public class, otherwise we can add the new class to an existing **.java** file and recompile it.

**Subpackages:**Packages that are inside another package are the **subpackages**. These are not imported by default, they have to imported explicitly. Also, members of a subpackage have no access privileges, i.e., they are considered as different package for protected and default access specifiers.  
**Example :**

import java.util.\*;

**util** is a subpackage created inside **java** package.

**Types of packages:**



**Built-in Packages**  
These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:  
1) **java.lang:**Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.  
2) **java.io:**Contains classed for supporting input / output operations.  
3) **java.util:**Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.  
4) **java.applet:**Contains classes for creating Applets.  
5) **java.awt:**Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).  
6) **java.net:**Contain classes for supporting networking operations.

**User-defined packages**  
These are the packages that are defined by the user. First we create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names**.

**Important points:**

* 1. Every class is part of some package.
  2. If no package is specified, the classes in the file goes into a special unnamed package (the same unnamed package for all files).
  3. All classes/interfaces in a file are part of the same package. Multiple files can specify the same package name.
  4. If package name is specified, the file must be in a subdirectory called name (i.e., the directory name must match the package name).
  5. We can access public classes in another (named) package using: **package-name.class-name**

1. What is Overloading and Overriding in Java?

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

If we have to perform only one operation, having same name of the   
 methods increases the readability of the program.

Method overloading increases the readability of the program.

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

### **Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments

example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

1. By changing the data type

example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

**Overriding:**

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 subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

* Method overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method overriding is used for runtime polymorphism.
* Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.
* Java method overriding example of bank

#### **Rules for Java Method Overriding:**

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).
4. What is the difference between an Inner Class and a Sub-Class?

An Inner class is a class which is nested within another class. An Inner class has access rights for the class which is nesting it and it can access all variables and methods defined in the outer class.

A sub-class is a class which inherits from another class called super class. Sub-class can access all public and protected methods and fields of its super class.

1. What is the difference between abstract class and interface?

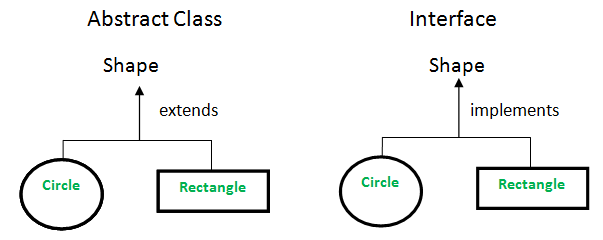
The primary difference between an abstract class and interface is that an interface can only possess declaration of public static methods with no concrete implementation while an abstract class can have members with any access specifiers (public, private etc) with or without concrete implementation.

Another key difference in the use of abstract classes and interfaces is that a class which implements an interface must implement all the methods of the interface while a class which inherits from an abstract class doesn’t require implementation of all the methods of its super class.

A class can implement multiple interfaces but it can extend only one abstract class.

**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.

**Abstract class vs Interface**

1. **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
2. **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
3. **Type of variables:**Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.
4. **Implementation:** Abstract class can provide the implementation of interface. Interface can’t provide the implementation of abstract class.
5. **Inheritance vs Abstraction:** A Java interface can be implemented using keyword “implements” and abstract class can be extended using keyword “extends”.
6. **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
7. **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Abstract-Class-vs-Interface.png)
8. What is static binding and dynamic binding?

**Static Binding:**The binding which can be resolved at compile time by compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time .

**Why binding of static, final and private methods is always a static binding?**  
Static binding is better performance wise (no extra overhead is required). Compiler knows that all such methods **cannot be overridden** and will always be accessed by object of local class. Hence compiler doesn’t have any difficulty to determine object of class (local class for sure). That’s the reason binding for such methods is static.

**Dynamic Binding:**In Dynamic binding compiler doesn’t decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have same method .

1. What is Data Encapsulation and what’s its significance?

Encapsulation is a concept in Object Oriented Programming for combining properties and methods in a single unit.

Encapsulation helps programmers to follow a modular approach for software development as each object has its own set of methods and variables and serves its functions independent of other objects. Encapsulation also serves data hiding purpose.

1. What is Java Bean Class?

JavaBeans are [classes](https://www.geeksforgeeks.org/classes-objects-java/)that [encapsulate](https://www.geeksforgeeks.org/encapsulation-in-java/) many objects into a single object (the bean). It is a java class that should follow following conventions:

1. Must implement [Serializable](https://www.geeksforgeeks.org/serialization-in-java/).
2. It should have a public no-arg constructor.
3. All properties in java bean must be private with public getters and setter methods.

|  |
| --- |
| // Java program to illustrate the  // structure of JavaBean class  public class TestBean {  private String name;  public void setName(String name)      {          this.name = name;      }  public String getName()      {          return name;      }  } |

**Syntax for setter methods:**

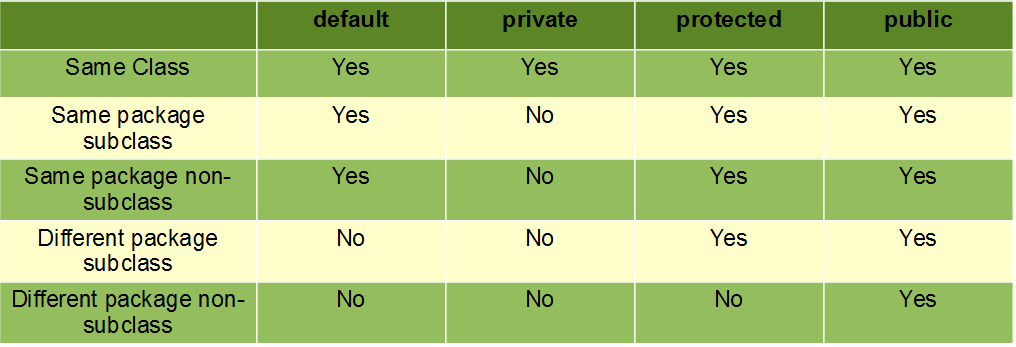
1. It should be public in nature.
2. The return-type should be void.
3. The setter method should be prefixed with set.
4. It should take some argument i.e. it should not be no-arg method.

**Syntax for getter methods:**

1. It should be public in nature.
2. The return-type should not be void i.e. according to our requirement we have to give return-type.
3. The getter method should be prefixed with get.
4. It should not take any argument.
5. What are Access modifiers?

Access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

1. Default – No keyword required
2. Private
3. Protected
4. Public



1. **Default**: When no access modifier is specified for a class , method or data member – It is said to be having the **default** access modifier by default.
   * The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.
2. **Private**: The private access modifier is specified using the keyword **private**.

The methods or data members declared as private are accessible only **within the class** in which they are declared.

Any other **class of same package will not be able to access** these members.

Classes or interface can not be declared as private.

1. **protected**: The protected access modifier is specified using the keyword **protected**.

The methods or data members declared as protected are **accessible within same package or sub classes in different package.**

**4. public**: The public access modifier is specified using the keyword **public**.

The public access modifier has the **widest scope** among all other access modifiers.

Classes, methods or data members which are declared as public are **accessible from every where** in the program. There is no restriction on the scope of a public data members.

1. What’s the benefit of using inheritance?

Key benefit of using inheritance is reusability of code as inheritance enables sub-classes to reuse the code of its super class. Polymorphism (Extensibility ) is another great benefit which allow new functionality to be introduced without effecting existing derived classes.

1. Why multiple Inheritance not supported in Java?

Java doesn’t support multiple inheritance in classes because of “Diamond Problem”. To know more about diamond problem with example, read Multiple Inheritance in Java.

However multiple inheritance is supported in interfaces. An interface can extend multiple interfaces because they just declare the methods and implementation will be present in the implementing class. So there is no issue of diamond problem with interfaces.

1. What is the diamond problem in inheritance?

GrandParent

/ \

/ \

Parent1 Parent2

\ /

\ /

Test

|  |
| --- |
| // A Grand parent class in diamond  class GrandParent  {      void fun()      {          System.out.println("Grandparent");      }  }    // First Parent class  class Parent1 extends GrandParent  {      void fun()      {          System.out.println("Parent1");      }  }    // Second Parent Class  class Parent2 extends GrandParent  {      void fun()      {          System.out.println("Parent2");      }  }  // Error : Test is inheriting from multiple  // classes  class Test extends Parent1, Parent2  {     public static void main(String args[])     {         Test t = new Test();         t.fun();     }  } |

Run on IDE

From the code, we see that: On calling the method fun() using Test object will cause complications such as whether to call Parent1’s fun() or Beta’s fun() method.

Therefore, in order to avoid such complications Java does not support multiple inheritance of classes.

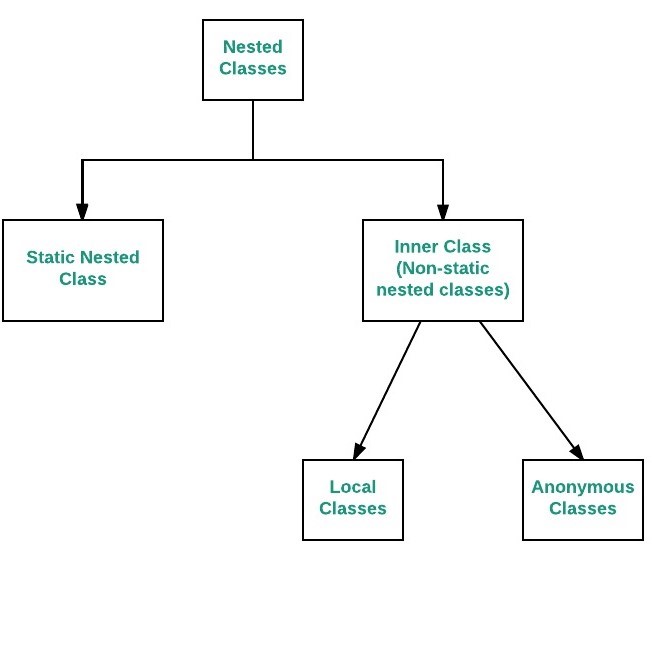
1. What is the difference between break and continue statement?  
    The break keyword is used to breaks(stopping) a loop execution, which may be a for loop, while loop, do while or for each loop.

The continue keyword is used to skip the particular recursion only in a loop execution, which may be a for loop, while loop, do while or for each loop.

1. What is nested class?

In java, it is possible to define a class within another class, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of [encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/), and create more readable and maintainable code.

* The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class NestedClass does not exist independently of class OuterClass.
* A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.
* A nested class is also a member of its enclosing class.
* As a member of its enclosing class, a nested class can be declared private, public, protected, or package private(default).
* Nested classes are divided into two categories:
  1. **static nested class :**Nested classes that are declared static are called static nested classes.
  2. **inner class :**An inner class is a non-static nested class.



**Difference between static and inner(non-static nested) classes**

* Static nested classes do not directly have access to other members(non-static variables and methods) of the enclosing class because as it is static, it must access the non-static members of its enclosing class through an object. That is, it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.
* Non-static nested classes (inner classes) has access to all members(static and non-static variables and methods, including private) of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

1. How are this() and super() used with Constructor?

this() is used to specify the current class or the child's class constructor always  
and super() refers to the parent's class constructor  
  
To use them follow the rules:  
  
 super()has to be the first line in any case if you are using the parent's class constructor in the child's class because your code might be using the parent's class variable and usng super() in the midway doesn't make any sense.So it has to be the first line always in the child's constructor.you can also supply super with arguments if ur parent class  has defined such constructor  
 this() also has to be the first line and is generally used when u r using the concept of constructor overloading then suppose u have 2 constructors eg:  
public class Manager extends Employee  
{

private String department;  
     public Manager(String name,double salary)  
    {  
       super(name,salary); // always first line  
           
    }  
    public Manager(String name,String dept)  
    {  
        this (name,dept) // always first line and this refers to above const.  
         department=dept;  
    }  
}  
 use this() and super() as shown above  
and note that whatever happens the constructor of the parent class will always be executed first and then child's class

1. What is Serialization and Deserialization?

We can convert a Java object to an Stream that is called Serialization. Once an object is converted to Stream, it can be saved to file or send over the network or used in socket connections.

The object should implement Serializable interface and we can use java.io.ObjectOutputStream to write object to file or to any OutputStream object. Read more at Java Serialization.

The process of converting stream data created through serialization to Object is called deserialization. Read more at Java Deserialization.

1. What is difference between Heap and Stack Memory?
2. Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.
3. Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.
4. Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.
5. Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at [Java Garbage Collection](https://www.journaldev.com/2856/java-jvm-memory-model-memory-management-in-java).
6. Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.
7. We can use **-Xms** and **-Xmx** JVM option to define the startup size and maximum size of heap memory. We can use **-Xss** to define the stack memory size.
8. When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.
9. Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.
10. How garbage collection is done in Java?

Garbage Collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. In Java, process of deallocating memory is handled automatically by the garbage collector.

We can run the garbage collector with code Runtime.getRuntime().gc() or use utility method System.gc().

**Array:**

1. **What do you mean by an Array? How to create?**

Array is a set of similar data type. Arrays objects store multiple variables with the same type. It can hold primitive types and object references. Arrays are always fixed

An Array is declared similar to how a variable is declared, but you need to add [] after the type.

Example: int [] intArray;

We can declare Java array as a field, static field, a local variable, or parameter, like any other variable. An array is a collection of variables of that type. Here are a few more Java array declaration examples:

String [] stringArray;

MyClass [] myClassArray;

1. **What are Advantages and disadvantages of Array?**

**Advantages:**

* We can put in place other data structures like stacks, queues, linked lists, trees, graphs, etc. in Array.
* Arrays can sort multiple elements at a time.
* We can access an element of Array by using an index.

**Disadvantages:**

* We have to declare Size of an array in advance. However, we may not know what size we need at the time of array declaration.
* The array is static structure. It means array size is always fixed, so we cannot increase or decrease memory allocation.

1. **What is the meaning of anonymous array? Explain with an example?**

An [array in Java](https://www.geeksforgeeks.org/arrays-in-java/) **without any name** is anonymous array. It is an array just for creating and using instantly.

* We can create an array without name, such type of nameless arrays are called anonymous array.
* The main purpose of anonymous array is just for instant use (just for one time usage) .
* Anonymous array is passed as an argument of method

**Syntax:**

// anonymous int array

new int[] { 1, 2, 3, 4};

// anonymous char array

new char[] {'x', 'y', 'z');

// anonymous String array

new String[] {"Geeks", "for", "Geeks"};

// anonymous multidimensional array

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

1. **What are “jagged” arrays in java?**

Jagged Arrays are Arrays are containing arrays of different length. Jagged arrays are also known as multidimensional arrays.

1. **How to copy an array into another array?**

Using “For loop”

Using “Arrays.copyOf()” method

Using “System.arraycopy()” method

Using “clone()” method

1. **What is the step to access elements of an array in Java?**

We can access using “index”.

Index starts from Zero(0), so the first element is stored in location zero and the last element will be Array.length – 1.

Example:-String strArr[] = new String []{“A”, “B”, “C”, “D”, “E”};

strArr[0] means “A” and strArr[2] means “C”.

1. What is difference between ArrayIndexOutOfBounds and ArrayStoreException?

It is a runtime exception. For example, we can store only string elements in a String Array. If anybody tries to insert integer element in this String Array, then we will get ArrayStoreException at run time.

It is a run time exception. It will occur when the program tries to access invalid index of an array. Index higher than the size of the array or negative index.

1. **How to check array contains value or not?**

Here is a String[] with values

public static final String[] myNames = new String[] {“B”,”A”,”K”,”C”};

If myNames contains that value then it will return true otherwise false.

Here is two methods isExists() and contains()

both the methods return true if the value is available otherwise false.

**First Method:**

It is converting an array to ArrayList.

After that it will test if an array contains any value then it will return  
 true otherwise false.

**Second Method:**

This method loop through an array and use equal() method to search element.

This actually performs a linear search over an array in Java. It will return true if an array has provided value.

1. Where does array stored in memory?

In Java, memory for arrays is always allocated on the heap as arrays in Java are objects.

**String:**

1. What is String in Java?

String is a Class in java and defined in java.lang package. It’s not a primitive data type like int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String in immutable and final in Java and JVM uses String Pool to store all the String objects. Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation.

1. What are different ways to create String Object?

We can create String object using new operator like any normal java class or we can use double quotes to create a String object. There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.

String str = new String("abc"); String str1 = "abc";

When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool. When we use new operator, JVM creates the String object but don’t store it into the String Pool. We can use intern() method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.

1. What is String subSequence method?

Java 1.4 introduced CharSequence interface and String implements this interface, this is the only reason for the implementation of subSequence method in String class. Internally it invokes the String substring method.

1. How to convert String to char and vice versa?

We can use use charAt method to get the character at given index or we can use toCharArray() method to convert String to character array.

1. How to convert String to byte array and vice versa?

We can use String getBytes() method to convert String to byte array and we can use String constructor new String(byte[] arr) to convert byte array to String.

1. What are difference between String, StringBuffer and StringBuilder?

String is immutable whereas StringBuffer and StringBuider are mutable classes.

StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why [StringBuilder is more faster than StringBuffer](https://www.journaldev.com/137/stringbuffer-vs-stringbuilder).

String concat + operator internally uses StringBuffer or StringBuilder class.

For String manipulations in non-multi threaded environment, we should use StringBuilder else use StringBuffer class.

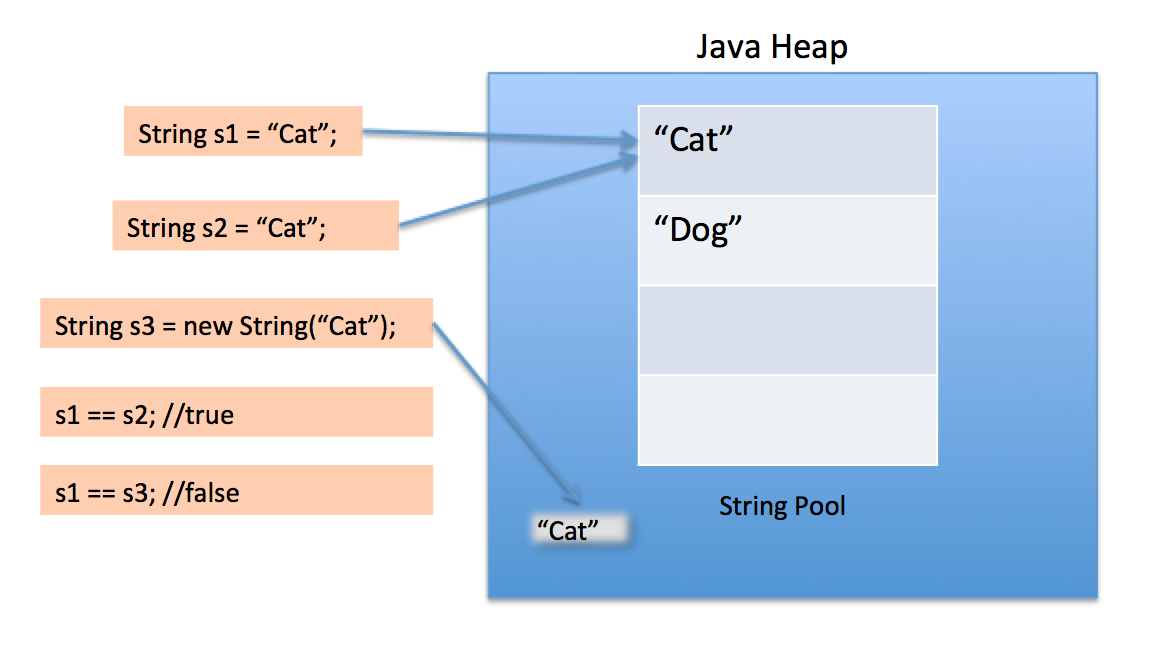
1. Why String is immutable or final in Java?
2. [String pool](https://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool. If String would not have been immutable, then String interning would not have been possible because if any variable would have changed the value, it would have been reflected to other variables also.
3. If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](https://www.journaldev.com/741/java-socket-programming-server-client) host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.
4. Since String is immutable, it is safe for multithreading and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.
5. Strings are used in [java classloader](https://www.journaldev.com/349/java-classloader) and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.
6. Since String is immutable, its **hashcode** is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

Above are some of the reasons I could think of that shows benefits of String immutability. It’s a great feature of [Java String](https://www.journaldev.com/16928/java-string) class and makes it special.

1. What is String Pool?

**String Pool in java** is a pool of Strings stored in [Java Heap Memory](https://www.journaldev.com/4098/java-heap-space-vs-stack-memory). We know that String is special class in java and we can create String object using new operator as well as providing values in double quotes.

Here is a diagram which clearly explains how String Pool is maintained in java heap space and what happens when we use different ways to create Strings.

[[](https://www.journaldev.com/wp-content/uploads/2012/11/String-Pool-Java1.png)](https://www.journaldev.com/wp-content/uploads/2012/11/String-Pool-Java1.png)

String Pool is possible only because [String is immutable in Java](https://www.journaldev.com/802/string-immutable-final-java) and it’s implementation of [String interning](https://en.wikipedia.org/wiki/String_interning) concept. String pool is also example of [Flyweight design pattern](https://www.journaldev.com/1562/flyweight-design-pattern-java).

String pool helps in saving a lot of space for Java Runtime although it takes more time to create the String.

When we use double quotes to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference.

However using new operator, we force String class to create a new String object in heap space. We can use intern() method to put it into the pool or refer to other String object from string pool having same value.

1. What does String intern() method do?

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

This method always return a String that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

1. Why String is popular HashMap key in Java?

Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and its processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

**Collections:**

1. What is Java Collections Framework? List out some benefits of Collections framework?

A Collection is a group of individual objects represented as a single unit. Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are two main root interfaces of Java collection classes.

**Hierarchy of Collection Framework**

Collection Map

/ / \ \ |

/ / \ \ |

Set List Queue Dequeue SortedMap

/

/

SortedSet

**Core Interfaces in Collections**

**Collection:** Root interface with basic methods like add(), remove(), contains(), isEmpty(), addAll(), ... etc.

**Set:** Doesn't allow duplicates. Example implementations of Set interface are HashSet (Hashing based) and TreeSet (balanced BST based). Note that TreeSet implements **SortedSet**.

**List:** Can contain duplicates and elements are ordered. Example implementations are LinkedList (linked list based) and

[ArrayList](https://www.geeksforgeeks.org/array-vs-arraylist-in-java/) (dynamic array based)

**Queue:** Typically order elements in FIFO order except exceptions

like PriorityQueue.

**Deque:** Elements can be inserted and removed at both ends. Allows

both LIFO and FIFO.

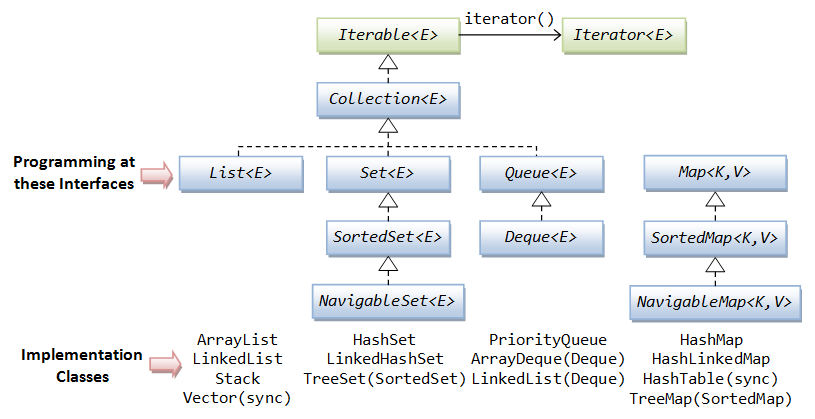
**Map:** Contains Key value pairs. Doesn't allow duplicates. Example

implementation are [HashMap and TreeMap](https://www.geeksforgeeks.org/hashmap-treemap-java/).

TreeMap implements **SortedMap**.

The difference between Set and Map interface is, in Set we have only

keys, but in Map, we have key value pairs.



**Advantages of Collection Framework:**

1. Consistent API : The API has basic set of interfaces like Collection, Set, List, or Map. All those classes (such as ArrayList, LinkedList, Vector etc) which implements, these interfaces have some common set of methods.
2. Reduces programming effort: The programmer need not to worry about design of Collection rather than he can focus on its best use in his program.
3. Increases program speed and quality: Increases performance by providing high-performance implementations of useful data structures and algorithms.
4. What is the benefit of Generics in Collections Framework?

There are mainly 3 advantages of generics. They are as follows:

**1) Type-safety :** We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

1. What are the basic interfaces of Java Collections Framework?

**Collection:** Collection is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

**Set:** Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

**List:** List is an ordered collection and can contain duplicate elements. You can access any element from its index. List is more like array with dynamic length.

**Map:** A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are Queue, Dequeue, Iterator, SortedSet, SortedMap and ListIterator.

1. What are common algorithms implemented in Collections Framework?

Java Collections Framework provides algorithm implementations that are commonly used such as sorting and searching. Collections class contain these method implementations. Most of these algorithms work on List but some of them are applicable for all kinds of collections. Some of them are sorting, searching, shuffling, min-max values.

1. Why Collection doesn’t extend Cloneable and Serializable interfaces?

Collection interface specifies group of Objects known as elements. How the elements are maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don’t.

A lot of the Collection implementations have a public clone method. However, it doesn’t really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation. The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized. So mandating cloning and serialization in all implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

1. What is difference between Array and ArrayList? When will you use Array over ArrayList?

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects. Arrays are fixed size whereas ArrayList size is dynamic. Arrays doesn’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc.

Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

If the size of list is fixed and mostly used to store and traverse them. For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types. If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>.

1. What are similarities and difference between ArrayList, Linked List and Vector?

ArrayList and Vector are similar classes in many ways:

1. Both are index based and backed up by an array internally.

2. Both maintains the order of insertion and we can get the elements in the order of insertion.

3. The iterator implementations of ArrayList and Vector both are fail-fast by design.

4. ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector:

1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.

2. ArrayList is faster than Vector because it doesn’t have any overhead because of synchronization.

3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

ArrayList and LinkedList both implement List interface but there are some differences between them:

1. ArrayList is an index based data structure backed by Array, so it provides random access to its elements with performance as O(1) but LinkedList stores data as list of nodes where every node is linked to its previous and next node. So even though there is a method to get the element using index, internally it traverse from start to reach at the index node and then return the element, so performance is O(n) that is slower than ArrayList.

2. Insertion, addition or removal of an element is faster in LinkedList compared to ArrayList because there is no concept of resizing array or updating index when element is added in middle.

3. LinkedList consumes more memory than ArrayList because every node in LinkedList stores reference of previous and next elements.

1. Why Map interface doesn’t extend Collection interface?

Map interface and its implementations are part of Collections Framework, Map are not collections and collections are not Map. Hence it doesn’t make sense for Map to extend Collection or vice versa.

If Map extends Collection interface, then where are the elements? Map contains key-value.

pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

1. What is difference between Enumeration and Iterator interface?

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.

Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make its functionality clear.

1. What is difference between Stack and Queue?

Both Stack and Queue are used to store data before processing them.

java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it’s not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.  
 Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order. Stack is a class that extends Vector whereas Queue is an interface.

1. What is difference between Comparable and Comparator interface?

Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can’t chose the field on which we want to sort the Object.

Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

**Difference between Comparable and Comparator interface:**

Comparable and Comparator interfaces are used to sort collection or array of objects.

Comparable interface is used to provide the natural sorting of objects and we can use it to provide sorting based on single logic.

Comparator interface is used to provide different algorithms for sorting and we can chose the comparator we want to use to sort the given collection of objects.

1. How HashMap works in Java?

HashMap stores key-value pair in Map.Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.

When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.

When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it’s value. Below image will explain these detail clearly.

The other important things to know about HashMap are capacity, load factor, threshold resizing. HashMap initial default capacity is 16 and load factor is 0.75. Threshold is capacity multiplied by load factor and whenever we try to add an entry, if map size is greater than threshold, HashMap rehashes the contents of map into a new array with a larger capacity. The capacity is always power of 2, so if you know that you need to store a large number of key-value pairs, for example in caching data from database, it’s good idea to initialize the HashMap with correct capacity and load factor.

1. What are different Collection views provided by Map interface?

Map interface provides three collection views:

**1. Set keySet():** Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.

**2. Collection values():** Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations. 3. Set<Map.Entry<K, V>> entrySet(): Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

**Multi-Threading and Concurrency:**

1. What is the difference between Process and Thread?

A process is a self-contained execution environment and it can be seen as a program or application whereas Thread is a single task of execution within the process. Java runtime environment runs as a single process which contains different classes and programs as processes. Thread can be called lightweight process. Thread requires less resources to create and exists in the process, thread shares the process resources.

1. What are the benefits of multi-threaded programming?

In Multi-Threaded programming, multiple threads are executing concurrently that improves the performance because CPU is not idle in case some thread is waiting to get some resources. Multiple threads share the heap memory, so it’s good to create multiple threads to execute some task rather than creating multiple processes. For example, Servlets are better in performance than CGI because Servlet support multi-threading but CGI doesn’t.

1. What is difference between user Thread and daemon Thread?

When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

1. How can we create a Thread in Java?

There are two ways to create Thread in Java – first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class.

1. What are different states in lifecycle of Thread?

When we create a Thread in java program, its state is New. Then we start the thread that change its state to Runnable. Thread Scheduler is responsible to allocate CPU to threads in Runnable thread pool and Blocked and Dead. Read this post to learn more about life cycle of thread.

1. What is Thread Scheduler and Time Slicing?

Thread Scheduler is the Operating System service that allocates the CPU time to the available runnable threads. Once we create and start a thread, its execution depends on the implementation of Thread Scheduler. Time Slicing is the process to divide the available CPU time to the available runnable threads. Allocation of CPU time to threads can be based on thread priority or the thread waiting for longer time will get more priority in getting CPU time. Thread scheduling can’t be controlled by java, so it’s always better to control it from application itself.

1. What is context-switching in multi-threading?

Context Switching is the process of storing and restoring of CPU state so that Thread execution can be resumed from the same point at a later point of time. Context Switching is the essential feature for multitasking operating system and support for multi-threaded environment.

1. How does thread communicate with each other?

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource.

1. Why thread communication methods wait(), notify() and notifyAll() are in Object class?

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

1. Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

1. Why Thread sleep() and yield() methods are static?

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

1. How can we achieve thread safety in Java?

There are several ways to achieve thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes. Learn more at thread safety tutorial.

1. What is Deadlock? How to analyze and avoid deadlock situation?

Deadlock is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

To analyze a deadlock, we need to look at the java thread dump of the application, we need to look out for the threads with state as BLOCKED and then the resources it’s waiting to lock, every resource has a unique ID using which we can find which thread is already holding the lock on the object.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation, read this post to learn how to analyze deadlock in java with sample program.

1. What is Java Timer Class? How to schedule a task to run after specific interval?

java.util.Timer is a utility class that can be used to schedule a thread to be executed at certain time in future. Java Timer class can be used to schedule a task to be run one-time or to be run at regular intervals.

java.util.TimerTask is an abstract class that implements Runnable interface and we need to extend this class to create our own TimerTask that can be scheduled using java Timer class.

1. What is Thread Pool? How can we create Thread Pool in Java?

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.

A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.

java.util.concurrent.Executors provide implementation of java.util.concurrent.Executor interface to create the thread pool in java. Thread Pool Example program shows how to create and use Thread Pool in java. Or read ScheduledThreadPoolExecutor Example to know how to schedule tasks after certain delay.

**Exception Handling:**

1. What is Exception & Exception Handling in Java?

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then JRE tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as catching the exception. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

Java Exception handling framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework

1. What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

**1. throw:** Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. throw keyword is used to throw exception to the runtime to handle it.

**2. throws:** When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to its caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.

**3. try-catch:** We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.

4. finally: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. Finally block gets executed always, whether exception occurs or not.

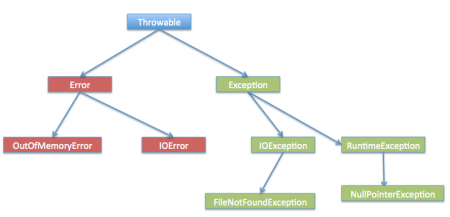
1. Explain Java Exception Hierarchy?

Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exception.

**Errors:** Errors are exceptional scenarios that are out of scope of application and it’s not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error.

**Checked Exceptions:** Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. Exception is the parent class of all Checked Exceptions.

**Runtime Exceptions:** Runtime Exceptions are caused by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions.

[](https://www.journaldev.com/wp-content/uploads/2013/07/exception-hierarchy.png)

1. What are important methods of Java Exception Class?

Exception and all of its subclasses doesn’t provide any specific methods and all of the methods are defined in the base class Throwable.

1. String getMessage() – This method returns the message String of Throwable and the message can be provided while creating the exception through it’s constructor.

2. String getLocalizedMessage() – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.

3. synchronized Throwable getCause() – This method returns the cause of the exception or null id the cause is unknown.

4. String toString() – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.

5. void printStackTrace() – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

1. What is difference between Checked and Unchecked Exception in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else main() method should use throws keyword to let JRE know about these exception that might be thrown from the program. Unchecked Exceptions are not required to be handled in the program or to mention them in throws clause.

2. Exception is the super class of all checked exceptions whereas RuntimeException is the super class of all unchecked exceptions.

3. Checked exceptions are error scenarios that are not caused by program, for example FileNotFoundException in reading a file that is not present, whereas Unchecked exceptions are mostly caused by poor programming, for example NullPointerException when invoking a method on an object reference without making sure that it’s not null.

1. What is difference between throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of program and handing over the exception object to runtime to handle it.

1. How to write custom exception in Java?

We can extend Exception class or any of its subclasses to create our custom exception class. The custom exception class can have its own variables and methods that we can use to pass error codes or other exception related information to the exception handler.

A simple example of custom exception is shown below.

package com.journaldev.exceptions;

import java.io.IOException;

public class MyException extends IOException {

private static final long serialVersionUID = 4664456874499611218L;

private String errorCode="Unknown\_Exception";

public MyException(String message, String errorCode){ super(message);

this.errorCode=errorCode;

}

public String getErrorCode() {

return this.errorCode;

} }

1. What are different scenarios causing “Exception in thread main”?

Some of the common main thread exception scenarios are:

 Exception in thread main java.lang.UnsupportedClassVersionError:

This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.

 Exception in thread main java.lang.NoClassDefFoundError:

There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.

 Exception in thread main java.lang.NoSuchMethodError: main:

This exception comes when you are trying to run a class that doesn’t have main method.

 Exception in thread “main” java.lang.ArithmeticException:

Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

1. What is difference between final, finally and finalize in Java?

final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can’t be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it’s great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

1. What happens when exception is thrown by main method?

When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

**Regular Expressions:**

1. What is Regex ? Why we go for regex?

Regular Expressions or Regex (in short) is an API for defining String patterns that can be used for searching, manipulating and editing a text. It is widely used to define constraint on strings such as password. Regular Expressions are provided under java.util.regex package.

1. What are the classes in Java that helps to deal with regular expressions?

**The java.util.regex** package primarily consists of the following three classes:

1. **util.regex.Pattern** – Used for defining patterns
2. **util.regex.Matcher** – Used for performing match operations on text using patterns
3. **PatternSyntaxException**– Used for indicating syntax error in a regular expression pattern.

**java.util.regex.Pattern** Class

1. **matches()**­­­– It is used to check if the whole text matches a pattern. Its output is boolean.
2. **compile()**– Used to create a pattern object by compiling a given string that may contain regular expressions. Input may also contains flags like Pattern.CASE\_INSENSITIVE, Pattern.COMMENTS, .. etc (See [this](https://docs.oracle.com/javase/tutorial/essential/regex/pattern.html) for details).
3. **split()**– It is used to split a text into multiple strings based on a delimiter pattern.

**java.util.regex.Matcher** Class

1. **find()**–It is mainly used for searching multiple occurrences of the regular expressions in the text.
2. **start()**– It is used for getting the start index of a match that is being found using find() method.
3. **end()**–It is used for getting the end index of a match that is being found using find() method. It returns index of character next to last matching character.
4. What is a metacharacter?

This API also supports a number of special characters that affect the way a pattern is matched. Change the regular expression to cat. and the input string to cats.

The output will appear as follows:

Enter your regex: cat.

Enter input string to search: cats

I found the text cats starting at index 0 and ending at index 4.

The match still succeeds, even though the dot "." is not present in the input string. It succeeds because the dot is a *metacharacter* — a character with special meaning interpreted by the matcher. The metacharacter "." means "any character" which is why the match succeeds in this example.

The metacharacters supported by this API are: <([{\^-=$!|]})?\*+.>

There are two ways to force a metacharacter to be treated as an ordinary character:

* precede the metacharacter with a backslash, or
* enclose it within \Q (which starts the quote) and \E (which ends it).

When using this technique, the \Q and \E can be placed at any location within the expression, provided that the \Q comes first.

04. What are predefined character classes?

Predefined character classes are useful shorthand notations available for commonly used regular expressions.

Predefined Character Class Description:

. Any character

d 0-9

s Whitespace character

w A word character, ie [A-Za-z\_0-9]

D Non-digit character

W Non-word character

S Non-whitespace character

05. Which is regex engine class?

Matcher class is the engine of regex and it implements MatchResult(I), used perform matching operations on a character sequence.

06. Which is compiler of regex?

Pattern class the compiled version of regex, used to define a pattern for the regex engine.

07. Advantages of regex?

With smart code completion, safe refactoring, and better support for Node.js, angular

**Annotations:**

1. What are annotations? What are their typical use cases?

Annotations are metadata bound to elements of the source code of a program and have no effect on the operation of the code they operate.

Their typical uses cases are:

* **Information for the compiler** – with annotations, the compiler can detect errors or suppress warnings
* **Compile-time and deployment-time processing** – software tools can process annotations and generate code, configuration files, etc.
* **Runtime processing** – annotations can be examined at runtime to customize the behavior of a program

1. Describe some useful annotations from the standard library.

There are several annotations in the *java.lang* and   
 *java.lang.annotation* packages, the more common ones include but not limited to:

* *@Override –* marks that a method is meant to override an element declared in a superclass. If it fails to override the method correctly, the compiler will issue an error
* *@Deprecated*– indicates that element is deprecated and should not be used. The compiler will issue a warning if the program uses a method, class, or field marked with this annotation
* *@SuppressWarnings*– tells the compiler to suppress specific warnings. Most commonly used when interfacing with legacy code written before generics appeared
* *@FunctionalInterface* – introduced in Java 8, indicates that the type declaration is a functional interface and whose implementation can be provided using a Lambda Expression

1. How can you create an annotation?

Annotations are a form of an interface where the keyword *interface* is preceded by *@,* andwhose body contains *annotation type element* declarations that look very similar to methods:

public @interface SimpleAnnotation {

    String value();

  int[] types();

After the annotation is defined, yon can start using it in through your code:

@SimpleAnnotation(value = "an element", types = 1)

public class Element {

    @SimpleAnnotation(value = "an attribute", types = { 1, 2 })

    public Element nextElement;

Note that, when providing multiple values for array elements, you must enclose them in brackets.

Optionally, default values can be provided as long as they are constant expressions to the compiler:

public @interface SimpleAnnotation {

    String value() default "This is an element";

    int[] types() default { 1, 2, 3 };

Now, you can use the annotation without those elements:

@SimpleAnnotation

public class Element {

    // ...  
}

Or only some of them:  
@SimpleAnnotation(value = "an attribute")

public Element nextElement;

1. What object types can be returned from an annotation method declaration?

The return type must be a primitive, String, Class, Enum, or an array of one of the previous types. Otherwise, the compiler will throw an error.

Here’s an example code that successfully follows this principle:

enum Complexity {

    LOW, HIGH

}

public @interface ComplexAnnotation {

    Class<? extends Object> value();

     int[] types();

     Complexity complexity();

}

The next example will fail to compile since Object is not a valid return type:

public @interface FailingAnnotation {

    Object complexity();

}

1. Which program elements can be annotated?

Annotations can be applied in several places throughout the source code. They can be applied to declarations of classes, constructors, and fields:

@SimpleAnnotation

public class Apply {

    @SimpleAnnotation

    private String aField;

    @SimpleAnnotation

    public Apply() {

        // ...

    }

}

Methods and their parameters:

@SimpleAnnotation

public void aMethod(@SimpleAnnotation String param) {

    // ...

}  
Local variables, including a loop and resource variables:

@SimpleAnnotation

int i = 10;

for (@SimpleAnnotation int j = 0; j < i; j++) {

    // ...

}

 try (@SimpleAnnotation FileWriter writer = getWriter()) {

    // ...

} catch (Exception ex) {

    // ...

}

Other annotation types:

@SimpleAnnotation

public @interface ComplexAnnotation {

    // ...

}

And even packages, through the package-info.java file:

@PackageAnnotation

package com.[baeldung](http://www.baeldung.com/java-annotations-interview-questions).interview.annotations;

As of Java 8, they can also be applied to the *use* of types. For this to work, the annotation must specify an *@Target* annotation with a value of *ElementType.USE*:

|  |
| --- |
|  |

@Target(ElementType.TYPE\_USE)

public @interface SimpleAnnotation {

    // ...

}

Now, the annotation can be applied to class instance creation: new @SimpleAnnotation Apply();

Type casts:

aString = (@SimpleAnnotation String) something;

Implements clause:

public class SimpleList<T>

  implements @SimpleAnnotation List<@SimpleAnnotation T> {

    // ...

}

And throws clause:

void aMethod() throws @SimpleAnnotation Exception {

    // ...

}

1. What are meta-annotations?

Are annotations that apply to other annotations.

All annotations that aren’t marked with @Target, or are marked with it but include ANNOTATION\_TYPE constant are also meta-annotations:

@Target(ElementType.ANNOTATION\_TYPE)

public @interface SimpleAnnotation {

    // ...

}

1. What are repeating annotations?

These are annotations that can be applied more than once to the same element declaration.

For compatibility reasons, since this feature was introduced in Java 8, repeating annotations are stored in a container annotation that is automatically generated by the Java compiler. For the compiler to do this, there are two steps to declared them.

First, we need to declare a repeatable annotation:

@Repeatable(Schedules.class)

public @interface Schedule {

    String time() default "morning";

}

Then, we define the containing annotation with a mandatory value element, and whose type must be an array of the repeatable annotation type:

public @interface Schedules {

    Schedule[] value();

}

Now, we can use @Schedule multiple times:

@Schedule

@Schedule(time = "afternoon")

@Schedule(time = "night")

void scheduledMethod() {

}

**Enum:**

* + 1. What is enum and why we go for enum?

It is a data type that contains fixed set of constants.

It can be used for days of the week , directions etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

02. Can Enum implement interface in Java?

Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implement interface. This gives a lot of flexibility to use Enum as specialized implementation in some cases.

03. Can Enum extends class in Java?

No, Enum can not extend class in Java. Since all Enum by default extend abstract base class java.lang.Enum, obviously they can not extend another class, because Java doesn't support multiple inheritance for classes. Because of extending java.lang.Enum class, all enum gets methods like ordinal(), values() or valueOf().

04. Can we declare Constructor inside Enum in Java?

Yes, but remember you can only declare either private or package-private constructor inside enum. public and protected constructors are not permitted inside enum.

05. Can we override toString() method for Enum? What happens if we don't?

Yes we can override toString in Enum, as like any other class it also extends java.lang.Object and has toString()method available, but even if you don't override, you will not going to regret much, because abstract base class of enum does that for you and return name, which is name of the enum instance itself. here is the code of toString() method from Enum class.

06. How do you create Enum without any instance?

yes, we can create Enum without any instance in Java, say for creating a utility class.