**What are the important features of Java 10 release?**

Java 10 is the first every-six-months from Oracle Corporation, so it’s not a major release like earlier versions. However some of the important features of Java 10 are:

* Local-Variable Type Inference
* Enhance java.util.Locale and related APIs to implement additional Unicode extensions of BCP 47 language tags.
* Enable the HotSpot VM to allocate the Java object heap on an alternative memory device, such as an NV-DIMM, specified by the user.
* Provide a default set of root Certification Authority (CA) certificates in the JDK.

Java 10 is mostly a maintenance release, however I really liked the local variable type inference feature.

**What are the important features of Java 9 release?**

Java 9 was a major release and brought a lot of features. Some of the important features are:

* Java 9 REPL (JShell)
* Java 9 Module System
* Factory Methods for Immutable List, Set, Map and Map.Entry
* Private methods in Interfaces
* Reactive Streams
* GC (Garbage Collector) Improvements

**List of most commonly used Java 8 new features:**

Java 8 has been released in March 2014, so it’s one of the hot topic in java interview questions. If you answer this question clearly, it will show that you like to keep yourself up-to-date with the latest technologies.

Java 8 has been one of the biggest release after Java 5 annotations and generics. Some of the important features of Java 8 are:

* Interface changes with default and static methods
* Functional interfaces
* Lambda Expressions
* Java Stream API for collection classes
* Java Date Time API
* Pipelines and Streams
* Type Annotations
* Concurrent Accumulators
* Parallel operations

**List of most commonly used Java 7 new features:**

* String in Switch Expression
* Underscores Between Digits in Numeric Literals
* Integral Types as Binary Literals
* Handling multiple exceptions in a single catch block
* Try-with-resources Statement
* Automatic Type Inference in Generic object instantiation

**List of most commonly used Java 5 new features:**

* Generics for Collections
* Enhanced for Loop (for-each loop)
* Autoboxing/Unboxing
* Typesafe Enums
* Varargs/Vargs (Variable-length Argument Lists)
* Static Import
* Metadata (Annotations)
* Formatting
* Scanner

**What do you mean by platform independence of Java?**

Platform independence means that you can run the same Java Program in any Operating System. For example, you can write java program in Windows and run it in Mac OS.

**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. 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.

**JVM ,JRE and JDK – A basic explanation.**

**JVM (Java Virtual Machine)** is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

Loads code

Verifies code

Executes code

Provides runtime environment

**JRE**

JRE = JVM + Other Runtime Libraries (like java.util, java.lang, java.math and many more).

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs is also actively released by other companies besides Sun Micro Systems.

**JDK**

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

JDK = JRE + Tools to code a java program ( e.g. Java Compiler ‘javac’, tools.jar, Java DB etc.) Java is both a compiled as well as an interpreted language, The Compiler inside the JDK (javac compiler) actually compiles the JAVA source code into Byte Code which the JVM finally interprets and gives us the output.

**There are three types of variables in java**

Local variable

Instance variable

Static variable

Java Variable Types

**In Java there are four types of variables:**

**An Instance field** is a variable that belongs to an object. Objects keep their internal state in non-static fields. Non-static fields are also called instance variables, because they belong to instances (objects) of a class. Non-static fields are covered in more detail in the text on Java fields.

**A static field** is a variable that belongs to a class. A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on Java fields.

**A local variable** is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on Java methods.

**Is Java Pass by Value or Pass by Reference?**

Java is Pass by Value, and does not support Pass by Reference.

**Is constructor inherited?**

No, constructor is not inherited.

**Can you make a constructor final?**

No, constructor can't be final.

**What does the "abstract" keyword mean in front of a method? A class?**

Abstract keyword declares either a method or a class.

If a method has an abstract keyword in front of it, it is called abstract method. Abstract method has no body. It has only arguments and return type. Abstract methods act as placeholder methods that are implemented in the subclasses.

Abstract classes can't be instantiated. If a class is declared as abstract, no objects of that class can be created. If a class contains any abstract method it must be declared as abstract

**How to run a JAR file through command prompt?**

We can run a jar file using java command but it requires Main-Class entry in jar manifest file. Main-Class is the entry point of the jar and used by java command to execute the class. Learn more at java jar file.

**Why Java is not pure Object Oriented language?**

Java is not said to be pure object oriented because it support primitive types such as int, byte, short, long etc. I believe it brings simplicity to the language while writing our code. Obviously java could have wrapper objects for the primitive types but just for the representation, they would not have provided any benefit.

As we know, for all the primitive types we have wrapper classes such as Integer, Long etc. that provides some additional methods.

**What if I do not provide the String array as the argument to the method?**

JVM would again fail, not being able to find the correct method.

**What if I write static public void main, instead of public static void main?**

It’s just fine. No issues here.

**If I do not provide any arguments on the command line, then the String array of Main method will be empty or null?**

It would be an empty array, NOT NULL.

**Can we import same package/class twice?**

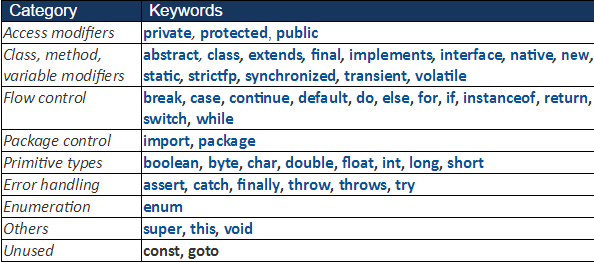
Yes

**What is difference between path and classpath variables?**

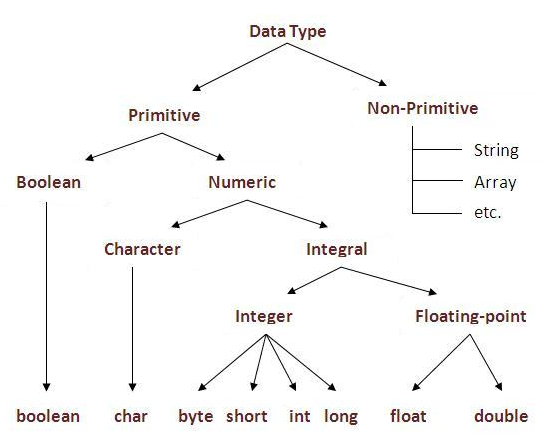
PATH is an environment variable used by operating system to locate the executable. That’s why when we install Java or want any executable to be found by OS, we need to add the directory location in the PATH variable. If you work on Windows OS, read this post to learn how to setup PATH variable on Windows.

Class path is specific to java and used by java executable to locate class files. We can provide the classpath location while running java application and it can be a directory, ZIP files, JAR files etc.

**Reserved Keywords in Java**



**Data Types in Java**



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | False | 1 bit |
| Char | '\u0000' | 2 byte |
| Byte | 0 | 1 byte |
| Short | 0 | 2 byte |
| Int | 0 | 4 byte |
| Long | 0L | 8 byte |
| Float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

**Object Oriented Programming (OOPS) Concepts**

Object Oriented Programming Approach has become a standard in the programming world, and every code written today uses OOP. OOP gives a structure to your code, makes it more modulated, easy to reuse and most importantly it keeps your code simple. Here are the chief OOP concepts we would use in our everyday coding:

**Classes and Objects:** The entry point into OOP world is class and objects. OOP states that every piece of code must fall inside a class, and to use any piece of code an object of the class has to be created, and the code can only be invoked via the object (except for 1 exception of static code, which is invoked by the class itself)

**Abstraction**

Now let’s talk about Abstraction; Abstraction is an implementation model over Data Hiding. It’s a technique for achieving Information Hiding. Abstraction says about extracting out the complex details and exposing out the necessary details (i.e. what Data Hiding proposes). Abstraction therefore gives a blue print of an idea, and concrete implementations would later be built over it. Example of Abstraction is Java’s Abstract class, where we only give the blue print of what is in our mind, without the actual code. The code is later build on top of it.

**Data Hiding**

Data Hiding is a proposal, asking us to hide as much vital things as possible, and to expose/reveal only the most necessary things to the outer world. Data hiding does justice to its name, and only talks about hiding the data, not on how you’re going to implement it.

**Encapsulation**

Encapsulation is simple, as simple as its name is. Encapsulation simply says about putting all things in a capsule, exactly what we do while creating Java classes. People also tend to confuse Encapsulation with Data Hiding. Well everything that is encapsulated may not be hidden. But yes encapsulation can be used to achieve Data Hiding also, by adding private methods or fields etc., but that is an additional step taken by the programmer, Encapsulation does not demand data hiding from you.

**Polymorphism**

The concept of polymorphism tells about multiple forms a particular code can take. It’s like one thing taking multiple forms, capable of doing multiple tasks. There are two kinds of Polymorphism supported by java, Static Polymorphism (Compile time polymorphism) and Dynamic Polymorphism (Runtime polymorphism). Method Overloading is an example of compile time polymorphism; whereas Method Overriding is an example of runtime polymorphism in java.

**Inheritance**

Inheritance is the OOP principle where any class can extend one class as its parent. Inheritance enables code reuse in our programs, where any child class gets all the extendable code from its parent, exactly similar how the word inheritance itself works.

**What is the advantage of OOP?**

You will get varying answers to this question depending on whom you ask. Major advantages of OOP are:

1. Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear;

2. Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system;

3. Modifiability: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods;

4. Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones;

5. Maintainability: objects can be maintained separately, making locating and fixing problems easier;

6. Re-usability: objects can be reused in different programs

**Relationships**

IS-A Relationship

HAS-A Relationship:

USES-A Relationship:

**IS-A Relationship:** IS-A is denotes an Inheritance relation between two entities. IS-A is used to specify a more specific type of the entity. Eg. Dog IS-A Animal, or SuperMan IS-A SuperHero. Where the parent classes Animal and SuperHero are more general, we have more specific subclasses Dog and Superman. The concept of IS-A is all about Inheritance.

**HAS-A Relationship:** The HAS-A Relationship tells about containing some entity inside another entity. It’s like Zoo HAS Animal. The Animal can be considered an instance variable inside the Zoo class. The HAS-A Relationship is also known as the Aggregation/Composition relation.

**USES-A Relationship**: USES-A is a simple relationship where one entity is not related to the other entity but just wants to use it for some purpose. It’s more like you using a second class inside your first class’s code. The USES-A Relationship is also known as the Association relation.

class Users{ }

class FaceBookWall{

public void postOnWall(String message){

}}

class SocialNetwork{ }

class FaceBook extends SocialNetwork

{ List<Users> millionUsers;

public void updateStatus(){

FaceBookWall wall = new FaceBookWall();

wall.postOnWall("I Hate Weekdays");

}

}

Let’s sync our understanding with the above code:

FaceBook IS-A SocialNetwork website. (Inheritance).

FaceBook HAS Users(actually Million users).(Aggregation/Composition).

FaceBook USES-A FaceBookWall to post any status updates. (Association).

**What is composition in java?**

Composition is the design technique to implement has-a relationship in classes. We can use Object composition for code reuse.

Java composition is achieved by using instance variables that refers to other objects. Benefit of using composition is that we can control the visibility of other object to client classes and reuse only what we need.

**What is the benefit of Composition over Inheritance?**

One of the best practices of java programming is to “favor composition over inheritance”. Some of the possible reasons are:

• Any change in the superclass might affect subclass even though we might not be using the superclass methods. For example, if we have a method test() in subclass and suddenly somebody introduces a method test() in superclass, we will get compilation errors in subclass. Composition will never face this issue because we are using only what methods we need.

• Inheritance exposes all the super class methods and variables to client and if we have no control in designing superclass, it can lead to security holes. Composition allows us to provide restricted access to the methods and hence more secure.

• We can get runtime binding in composition where inheritance binds the classes at compile time. So composition provides flexibility in invocation of methods.

You can read more about above benefits of composition over inheritance at java composition vs inheritance.

**Difference between association aggregation and composition in java.**

Both Composition and Aggregation are Associations.

Composition -> Strong Has-A relationship

Aggregation -> Weak Has-A relationship.

Composition: Since Engine is-part-of Car, the relationship between them is Composition.

A class contains students. A student cannot exist without a class. There exists composition between class and students.

Aggregation: Since Organization has Person as employees, the relationship between them is Aggregation.

A Library contains students and books. Relationship between library and student is aggregation.

Relationship between library and book is composition.

A student can exist without a library and therefore it is aggregation. A book cannot exist without a library and therefore its a composition.

**Explain the usage of Java packages.**

This is a way to organize files when a project consists of multiple modules. It also helps resolve naming conflicts when different packages have classes with the same names. Packages access level also allows you to protect data from being used by the non-authorized classes.

java.lang — basic language functionality and fundamental types

java.util — collection data structure classes

java.io — file operations

java.math — multiprecision arithmetic

java.sql — Java Database Connectivity (JDBC) to access databases

**What is the importance of main method in Java?**

main() method is the entry point of any standalone java application. The syntax of main method is public static void main(String args[]).

main method is public and static so that java can access it without initializing the class. The input parameter is an array of String through which we can pass runtime arguments to the java program.

**Can we have multiple public classes in a java source file?**

We can’t have more than one public class in a single java source file. A single source file can have multiple classes that are not public.

**What is the purpose of abstract class?**

It is not an insatiable class. It provides the concrete implementation for some/all the methods. So that they can reuse the concrete functionality by inheriting the abstract class.

**What is constructor?**

Constructor in java is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

**Rules for creating java constructor**

There are basically two rules defined for the constructor.

Constructor name must be same as its class name

Constructor must have no explicit return type

**What if the difference between && and &??**

& is bitwise and && is logical.

& evaluates both sides of the operation.

&& evaluates the left side of the operation, if it’s true, it continues and evaluates the right side.

**What is the significance of a static modifier?**

static keyword can be used with class level variables to make it global i.e all the objects will share the same variable.

**Static variable**: One per class, shared variable. Can be accessed without creating class’s object by ‘ClassName.variableName’ directly.

**Static Method**: Belongs to class, similar to static variable. It can access only static data and static methods directly. Else we need to create object of the class and call the methods from the object (exactly what we do in our main method most of the times).A static method cannot use this and super keyword inside itself. It can also be accessed via the ClassName.methodName() directly.

**Static Block:** Your program can also have an interesting thing known as the static block. A static block is a piece of code which you want to execute before any part of your class is used. A static blocks is the 1st thing that would be executed by the JVM, even before the constructor. The static block would be executed only once. It’s generally used for initializing variables before the objects can be initialized.

**What is the volatile modifier for?**

The volatile modifier is used to identify variables whose values should not be optimized by the Java Virtual Machine, by caching the value for example. The volatile modifier is typically used for variables that may be accessed or modified by numerous independent threads and signifies that the value may change without synchronization.

**What is a final modifier?**

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can’t extend it.

We can use final keyword with methods to make sure child classes can’t override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

**Can main method be declared as private?**

Yes, we can declare main method as private. It compiles without any errors, but in runtime, it says main method is not public.

**What if the static modifier is removed from the main method signature?**

Again JVM won’t be able to call the method as ClassName.main(String\*+ arguments) for a non-static method.

**Explain all access modifiers?**

**private**

If a method or variable is marked as private, then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.

If a class is marked as private then no external class an access the class. This doesn’t really make so much sense for classes though. Therefore, the access modifier private is mostly used for fields, constructors and methods.

**default**

The default access level is declared by not writing any access modifier at all. Default access levels means that code inside the class itself + code inside classes in the same package as this class, can access the class, field, constructor or method. Therefore, the default access modifier is also sometimes called a package access modifier.

Subclasses cannot access methods and member variables in the superclass, if they have default accessibility declared, unless the subclass is located in the same package as the superclass.

**Protected**

The protected access modifier does the same as the default access, except subclasses can also access protected methods and member variables of the superclass. This is true even if the subclass is not located in the same package as the superclass.

**Public**

The public access modifier means that all code can access the class, field, constructor or method, regardless of where the accessing code is located.

# [Can we overload the main method in Java?](http://stackoverflow.com/questions/3759315/can-we-overload-the-main-method-in-java)

You can overload the main() method, but only public static void main(String[] args) will be used when your class is launched by the JVM. For example:

public class Test {

public static void main(String[] args) {

System.out.println("main(String[] args)");

}

public static void main(String arg1) {

System.out.println("main(String arg1)");

}

public static void main(String arg1, String arg2) {

System.out.println("main(String arg1, String arg2)");

}

}

### Why main() in java is declared as public static void?

### Why public? main method is public so that it can be accessible everywhere and to every object which may desire to use it for launching the application. Here, i am not saying that JDK/JRE had similar reasons because java.exe or javaw.exe (for windows) use Java Native Interface (JNI) calls to invoke method, so they can have invoked it either way irrespective of any access modifier.

### Why static? Let’s suppose we do not have main method as static. Now, to invoke any method you need an instance of it. Right? Java can have overloaded constructors, we all know. Now, which one should be used and from where the parameters for overloaded constructors will come.

### Why void? Then there is no use of returning any value to JVM, who actually invokes this method. The only thing application would like to communicate to invoking process is: normal or abnormal termination. This is already possible using System.exit(int). A non-zero value means abnormal termination otherwise everything was fine.

**What are the differences b/w Interface and Abstract class?**

1). abstract class can extend only one class or one abstract class at a time interface can extend any number of interfaces at a time

2). abstract class can extend from a class or from an abstract class interface can extend only from an interface

3). abstract class can have both abstract and concrete methods interface can have only abstract methods

4). A class can extend only one abstract class A class can implement any number of interfaces

5). In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract In an interface keyword ‘abstract’ is optional to declare a method as an abstract

6). abstract class can have protected, public and public abstract methods Interface can have only public abstract methods i.e. by default

7). abstract class can have static, final or static final variable with any access specifier interface can have only static final (constant) variable i.e. by default.

**What are all the different ways to create an object in Java?**

***A. Using new keyword***

This is the most common way to create an object in java. Almost 99% of objects are created in this way.

MyObject object = new MyObject();

***B. Using Class.forName()***

If we know the name of the class & if it has a public default constructor we can create an object in this way.

MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();

***C. Using clone()***

The clone() can be used to create a copy of an existing object.

MyObject anotherObject = new MyObject();

MyObject object = (MyObject) anotherObject.clone();

***D. Using object deserialization***

Object deserialization is nothing but creating an object from its serialized form.

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyObject object = (MyObject) inStream.readObject();

**Abstract class must have only abstract methods. True or false?**

False. Abstract methods can also have concrete methods.

**Is it compulsory for a class which is declared as abstract to have at least one abstract method?**

Not necessarily. Abstract class may or may not have abstract methods.

**Can we use “abstract” keyword with constructor, Instance Initialization Block and Static Initialization Block?**

No. Constructor, Static Initialization Block, Instance Initialization Block and variables cannot be abstract.

**Why final and abstract cannot be used at a time?**

Because, final and abstract are totally opposite in nature. A final class or method cannot be modified further whereas abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

**What is the use of private constructor?**

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it’s objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures only one instance of a class exist at any point of time. Here is an example of singleton pattern using private constructor.

class MyClass

{

private static MyClass object = null;

private MyClass() { } //private constructor

public MyClass getObject()

{

if(object == null)

{

object = new MyClass(); //Creating object using private constructor

}

return object;

}

}

**Can we use this() and super() in a method?**

No, We can’t use this() and super() in a method.

class SuperClass

{

public SuperClass()

{

System.out.println ("Super Class Constructor");

}

}

class SubClass extends SuperClass

{

public SubClass()

{

System.out.println("Sub Class Constructor");

}

void method()

{

this(); //Compile time error

super(); //Compile time error

}

}

**Can we instantiate a class which does not have even a single abstract methods but declared as abstract?**

No, We can’t instantiate a class once it is declared as abstract even though it does not have abstract methods.

**Can we declare abstract methods as private? Justify your answer?**

No. Abstract methods cannot be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

**Can we call sub class constructor from super class constructor?**

No. There is no way in java to call sub class constructor from a super class constructor.

**What are non-access modifiers in java?**

These are the modifiers which are used to achieve other functionalities like,

a) static: This modifier is used to specify whether a member is a class member or an instance member.

b) final : It is used to restrict the further modification of a class or a method or a field.

c) abstract : abstract class or abstract method must be enhanced or modified further. (For more on abstract, click here).

d) synchronized : It is used to achieve thread safeness. Only one thread can execute a method or a block which is declared as synchronized at any given time.

**Can we use a field or a method declared without access modifiers outside the package?**

No, we can’t use a field or a method with no-access (default) specifiers outside the package in which their class is defined.

**Can a method or a class be final and abstract at the same time?**

No, it is not possible. A class or a method cannot be final and abstract at the same time. final and abstract are totally opposite in nature. final class or final method must not be modified further whereas abstract class or abstract method must be modified further.

**Can we access non-static members of outer class inside a static nested class?**

No, we can’t access non-static members of outer class inside a static nested class. We can access only static members of outer class inside a static nested class.

**What are member inner classes in java?**

Member inner classes are the classes which are declared as non-static members of another class. Member inner classes can be accessed only by instantiating the outer class.

**Can member inner classes have static members in them?**

No, member inner classes can’t have static members in them. They can have only non-static members. But, exception being the static and final field. i.e member inner class can have static and final field, but it must be initialized at the time of declaration only.

**Can we access all the members of outer class inside a member inner class?**

Yes, we can access all the members, both static and non-static, of outer class inside a member inner class.

**Can we declare local inner classes as static?**

No. Local inner classes can’t be static.

**Can we use local inner classes outside the method or block in which they are defined?**

No. Local inner classes are local to method or block in which they are defined. We can’t use them outside the method or block in which they are defined.

**Can we declare local inner classes as private or protected or public?**

No. Local inner classes can’t be declared with access modifiers. They can’t be private or protected or public.

**What is the condition to use local variables inside a local inner class?**

The condition is that local variables must be final. We can’t use non-final local variables inside a local inner class.

**What is the main difference between static and non-static nested classes?**

The main difference between static and non-static nested classes is that you need not to instantiate the outer class to access static nested classes. But, to access non-static nested classes, you have to instantiate the outer class.

**Can we declare one overloaded method as static and another one as non-static?**

Yes. Overloaded methods can be either static or non-static.

**How do compiler differentiate overloaded methods from duplicate methods?**

Compiler uses method signature to check whether the method is overloaded or duplicated. Duplicate methods will have same method signatures i.e same name, same number of arguments and same types of arguments. Overloaded methods will also have same name but differ in number of arguments or else types of arguments.

**Is it possible to have two methods in a class with same method signature but different return types?**

No, compiler will give duplicate method error. Compiler checks only method signature for duplication not the return types. If two methods have same method signature, straight away it gives compile time error.

**In “MyClass” , there is a method called “myMethod” with four different overloaded forms. All four different forms have different visibility ( private, protected, public and default). Is “myMethod” properly overloaded?**

Yes. Compiler checks only method signature for overloading of methods not the visibility of methods.

**Can overloaded methods be synchronized?**

Yes. Overloaded methods can be synchronized.

**Can we overload main() method?**

Yes, we can overload main() method. A class can have any number of main() methods but execution starts from public static void main(String[] args) only.

**Can we declare overloaded methods as final?**

Yes, we can declare overloaded methods as final.

**Can overloaded method be overrided?**

Yes, we can override a method which is overloaded in super class.

**Can we override static methods?**

No, Static methods cannot be overridden. If we try to override them they will be hidden in the sub class.

**What happens if we change the arguments of overriding method?**

If we change the arguments of overriding method, then that method will be treated as overloaded not overridden.

**Can we override protected method of super class as public method in the sub class?**

Yes. You can increase the visibility of overriding methods but can’t reduce it.

**Can we change the return type of overriding method from Number type to Integer type?**

Yes. You can change as Integer is a sub class of Number type.

**Can we override a super class method without throws clause as a method with throws clause in the sub class?**

Yes, but only with unchecked type of exceptions.

**Can we change an exception of a method with throws clause from SQLException to NumberFormatException while overriding it?**

Yes. Overridden method may throw SQLException or it’s sub class exception or any unchecked type of exceptions.

**Can we change an exception of a method with throws clause from unchecked to checked while overriding it?**

No. We can’t change an exception of a method with throws clause from unchecked to checked.

**How do you refer super class version of overridden method in the sub class?**

Using super keyword, we can refer super class version of overridden method in the sub class.

**Can we override private methods?**

No question of overriding private methods. They are not at all inherited to sub class.

**Can we remove throws clause of a method while overriding it?**

Yes. You can remove throws clause of a method while overriding it.

**Is it possible to override non-static methods as static?**

No. You can’t override non-static methods as static.

**Can we change an exception of a method with throws clause from checked to unchecked while overriding it?**

Yes. We can change an exception from checked to unchecked but reverse is not possible.

**Can we change the number of exceptions thrown by a method with throws clause while overriding it?**

Yes, we can change. But, exceptions must be compatible with throws clause in the super class method.final?

No, constructors cannot be final.

**Can we declare a class as private?**

We can’t declare an outer class as private. But, we can declare an inner class (class as a member of another class) as private.

**Can we declare an abstract method as private also?**

No, abstract methods cannot be private. They must be public or protected or default so that they can be modified further.

**Can we declare a class as protected?**

We can’t declare an outer class as protected. But, we can declare an inner class (class as a member of another class) as protected.

**We can’t instantiate an abstract class. Then why constructors are allowed in abstract class?**

It is because, we can’t create objects to abstract classes but we can create objects to their sub classes. From sub class constructor, there will be an implicit call to super class constructor. That’s why abstract classes should have constructors. Even if you don’t write constructor for your abstract class, compiler will keep default constructor.

**Can we declare abstract methods as static?**

No, abstract methods can not be static.

**Can a class contain an abstract class as a member?**

Yes, a class can have abstract class as it’s member.

**Abstract classes can be nested. True or false?**

True. Abstract classes can be nested i.e an abstract class can have another abstract class as it’s member.

**Can we declare abstract methods as synchronized?**

No, abstract methods cannot be declared as synchronized. But methods which override abstract methods can be declared as synchronized.

**Can we declare local inner class as abstract?**

Yes. Local inner class can be abstract.

**Can abstract method declaration include throws clause?**

Yes. Abstract methods can be declared with throws clause.

**Can abstract class have constructors in Java?**

Yes, abstract class can declare and define constructor in Java. Since you can not create instance of abstract class, constructor can only be called during constructor chaining, i.e. when you create instance of concrete implementation class. Now some interviewer, ask what is the purpose of constructor, if you cannot instantiate abstract class? Well, it can still be used to initialize common variables, which are declared inside abstract class, and used by various implementation. Also even if you don’t provide any constructor, compiler will add default no argument constructor in abstract class, without that your subclass will not compile, since first statement in any constructor implicitly calls super(), default super class constructor in Java.

**Can abstract class implements interface in Java? Does they require to implement all methods?**

Yes, abstract class can implement interface by using implements keyword. Since they are abstract, they don’t need to implement all methods. It’s good practice to provide an abstract base class, along with an interface to declare Type. One example of this is java.util.List interface and corresponding java.util.AbstractList abstract class. Since AbstractList implements all common methods,concrete implementations like LinkedList and ArrayList are free from burden of implementing all methods, had they implemented List interface directly. It’s best of both world, you can get advantage of interface for declaring type, and flexibility of abstract class to implement common behavior at one place. Effective Java has a nice chapter on how to use interface and abstract class in Java, which is worth reading.

**Can you create instance of abstract class?**

No, you cannot create instance of abstract class in Java, they are incomplete. Even though, if your abstract class don’t contain any abstract method, you cannot create instance of it. By making a class abstract, you told compiler that, it’s incomplete and should not be instantiated. Java compiler will throw error, when a code tries to instantiate abstract class.

**Is it necessary for abstract class to have abstract method?**

No, it’s not mandatory for an abstract class to have any abstract method. You can make a class abstract in Java, by just using abstract keyword in class declaration. Compiler will enforce all structural restriction, applied to abstract class, e.g. now allowing to create any instance. By the way, it’s debatable whether you should have abstract method inside abstract class or interface. In my opinion, abstract class should have abstract methods, because that’s the first thing programmer assumes, when he see that class. That would also go nicely along principle of least surprise.

**Can abstract class be final in Java?**

No, abstract class cannot be final in Java. Making them final will stop abstract class from being extended, which is the only way to use abstract class. They are also opposite of each other, abstract keyword enforces to extend a class, for using it, on the other hand, final keyword prevents a class from being extended. In real world also, abstract signifies incompleteness, while final is used to demonstrate completeness. Bottom line is, you cannot make your class abstract and final in Java, at same time, it’s a compile time error.

**A class cannot be declared with synchronized keyword. Then, why we call classes like Vector, StringBuffer are synchronized classes?**

Any classes which have only synchronized methods and blocks are treated as synchronized classes. Classes like Vector, StringBuffer have only synchronized methods. That’s why they are called as synchronized classes.

**Can abstract class have static methods in Java?**

Yes, abstract class can declare and define static methods, nothing prevents from doing that. But, you must follow guidelines for making a method static in Java, as it’s not welcomed in a object oriented design, because static methods cannot be overridden in Java. It’s very rare, you see static methods inside abstract class, but as I said, if you have very good reason of doing it, then nothing stops you.

**Can abstract class contains main method in Java?**

Yes, abstract class can contain main method, it just another static method and you can execute Abstract class with main method, until you don’t create any instance.

**Can an interface implement or extend another interface?**

Interfaces don’t implement another interface, they extend it. Since interfaces can’t have method implementations, there is no issue of diamond problem. That’s why we have multiple inheritance in interfaces i.e an interface can extend multiple interfaces.

**What is Dynamic Method Dispatching?**

Dynamic method dispatch is a mechanism to resolve overridden method call at run time instead of compile time. It is based on the concept of up-casting (A super class reference variable can refer subclass object.).

class Student {

public void show(){

System.out.println("Student details.");

} }

public class CollegeStudent extends Student {

public void show(){

System.out.println("College Student details."); }

public method static void main(String args[]){

Student obj = new CollegeStudent();

obj.show();

}

}

**Can a class be declared private or protected?**

No

**What is serialization?**

Serialization is the process of turning an object in memory into a stream of bytes so you can do stuff like store it on disk or send it over the network. Deserialization is the reverse process: turning a stream of bytes into an object in memory.

**What are inner/nested classes?**

Java inner class or nested class is a class i.e. declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

**Advantage of java inner classes**

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

3) Code Optimization: It requires less code to write.

**Types of Nested classes**

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

Non-static nested class(inner class)

a) Member inner class

b) Annomynous inner class

c) Local inner class

d) Static nested class

**When do you use varargs in Java?**

**Variable Argument (Varargs):**

The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

***Advantage of Varargs:***

We don't have to provide overloaded methods so less code.

***Syntax of varargs:***

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

return\_type method\_name(data\_type... variableName){}

**What are Wrapper classes?**

Java wrapper classes are the Object representation of eight primitive types in java. All the wrapper classes in java are immutable and final. Java 5 autoboxing and unboxing allows easy conversion between primitive types and their corresponding wrapper classes.

***Importance of Wrapper classes***

There are mainly two uses with wrapper classes.

To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.

To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

**What is Enum in Java?**

Enum was introduced in Java 1.5 as a new type whose fields consists of fixed set of constants. For example, in Java we can create Direction as enum with fixed fields as EAST, WEST, NORTH, SOUTH.

enum is the keyword to create an enum type and similar to class. Enum constants are implicitly static and final. Read more in detail at java enum.

**What is Java Annotations?**

Java Annotations provide information about the code and they have no direct effect on the code they annotate. Annotations are introduced in Java 5. Annotation is metadata about the program embedded in the program itself. It can be parsed by the annotation parsing tool or by compiler. We can also specify annotation availability to either compile time only or till runtime also. Java Built-in annotations are @Override, @Deprecated and @SuppressWarnings. Read more at java annotations.

**What is Java Reflection API? Why it’s so important to have?**

Java Reflection API provides ability to inspect and modify the runtime behaviour of java application. We can inspect a java class, interface, enum and get their methods and field details. Reflection API is an advanced topic and we should avoid it in normal programming. Reflection API usage can break the design pattern such as Singleton pattern by invoking the private constructor i.e violating the rules of access modifiers.

Even though we don’t use Reflection API in normal programming, it’s very important to have. We can’t have any frameworks such as Spring, Hibernate or servers such as Tomcat, JBoss without Reflection API. They invoke the appropriate methods and instantiate classes through reflection API and use it a lot for other processing.

**What does super keyword do?**

super keyword can be used to access super class method when you have overridden the method in the child class.

We can use super keyword to invoke super class constructor in child class constructor but in this case it should be the first statement in the constructor method.

package com.journaldev.access;

public class SuperClass {

public SuperClass(){

}

public SuperClass(int i){}

public void test(){

System.out.println("super class test method");

}

}

Use of super keyword can be seen in below child class implementation.

package com.journaldev.access;

public class ChildClass extends SuperClass {

public ChildClass(String str){

//access super class constructor with super keyword

super();

//access child class method

test();

//use super to access super class method

super.test();

}

@Override

public void test(){

System.out.println("child class test method");

}

}

**What is break and continue statement?**

We can use break statement to terminate for, while, or do-while loop. We can use break statement in switch statement to exit the switch case. You can see the example of break statement at java break. We can use break with label to terminate the nested loops.

The continue statement skips the current iteration of a for, while or do-while loop. We can use continue statement with label to skip the current iteration of outermost loop.

**What is this keyword?**

this keyword provides reference to the current object and it’s mostly used to make sure that object variables are used, not the local variables having same name.

//constructor

public Point(int x, int y) {

this.x = x;

this.y = y;

}

We can also use this keyword to invoke other constructors from a constructor.

public Rectangle() {

this(0, 0, 0, 0);

}

public Rectangle(int width, int height) {

this(0, 0, width, height);

}

public Rectangle(int x, int y, int width, int height) {

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

**What is default constructor?**

No argument constructor of a class is known as default constructor. When we don’t define any constructor for the class, java compiler automatically creates the default no-args constructor for the class. If there are other constructors defined, then compiler won’t create default constructor for us.

**When do you favor abstract class over interface?**

This is the follow-up of previous interview questions on abstract class and interface. If you know syntactical difference, you can answer this question quite easily, as they are the one, which drives the decision. Since it’s almost impossible to add a new method on a published interface, it’s better to use abstract class, when evolution is concern. Abstract class in Java evolves better than interface. Similarly, if you have too many methods inside interface, you are creating pain for all it’s implementation, consider providing an abstract class for default implementation. This is the pattern followed in Java collection package, you can see AbstractList provides default implementation for List interface.

**What is instanceof keyword?**

We can use instanceof keyword to check if an object belongs to a class or not. We should avoid it’s usage as much as possible. Sample usage is:

public static void main(String args[]){

Object str = new String("abc");

if(str instanceof String){

System.out.println("String value:"+str);

}

if(str instanceof Integer){

System.out.println("Integer value:"+str);

}

}

Since str is of type String at runtime, first if statement evaluates to true and second one to false.

**What is abstract method in Java?**

An abstract method is a method without body. You just declare method, without defining it and use abstract keyword in method declaration. All method declared inside Java Interface are by default abstract. Here is an example of abstract method in Java

public void abstract printVersion();

Now, In order to implement this method, you need to extend abstract class and override this method.

**What are wrapper classes?**

All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number. The object of the wrapper class contains or wraps its respective primitive data type. converting primitive data types into object is called boxing, and this is taken care by the compiler.

**Difference between equals() and == operator.**

**1).** equals will only compare what it is written to compare, no more, no less.

**2).** if a class does not override the equals method, then it defaults to the equals(Object o) method of the closest parent class that has overridden this method.

**3).** If no parent classes have provided an override, then it defaults to the method from the ultimate parent class, Object, and so you're left with the Object#equals(Object o) method. Per the Object API this is the same as ==; that is, it returns true if and only if both variables refer to the same object, if their references are one and the same. Thus you will be testing for **object equality** and not **functional equality**.

**4).** Always remember to override hashCode if you override equals so as not to "break the contract". As per the API, the result returned from the hashCode() method for two objects **must** be the same if their equals methods shows that they are equivalent. The converse is not necessarily true.

**What is java cloning? What are the different types of cloning?**

A clone is an exact copy of the original. In java, it essentially means the ability to create an object with similar state as the original object. The clone() method provides this functionality.

**Shallow copies duplicate as little as possible.**  By default, java cloning is shallow copy or ‘field by field copy’ i.e. as the Object class does not have idea about the structure of class on which clone() method will be invoked. So, JVM when called for cloning, do following things:

1) If the class has only primitive data type members then a completely new copy of the object will be created and the reference to the new object copy will be returned.

2) If the class contains members of any class type then only the object references to those members are copied and hence the member references in both the original object as well as the cloned object refer to the same object.

**Deep copies duplicate everything.** A deep copy of a collection is two collections with all of the elements in the original collection duplicated. Here, we want a clone which is independent of original and making changes in clone should not affect original.

**Deep cloning requires satisfaction of following rules.**

No need to separately copy primitives.

All the member classes in original class should support cloning and in clone method of original class in context should call super.clone() on all member classes.

If any member class does not support cloning then in clone method, one must create a new instance of that member class and copy all its attributes one by one to new member class object. This new member class object will be set in cloned object.

**Which class is the superclass of all classes?**

java.lang.Object is the root class for all the java classes and we don’t need to extend it.

**Why Java doesn’t support multiple inheritance?**

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.

**What is Java Package and which package is imported by default?**

Java package is the mechanism to organize the java classes by grouping them. The grouping logic can be based on functionality or modules based. A java class fully classified name contains package and class name. For example, java.lang.Object is the fully classified name of Object class that is part ofjava.lang package.

java.lang package is imported by default and we don’t need to import any class from this package explicitly.

**What is final keyword?**

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can’t extend it.

We can use final keyword with methods to make sure child classes can’t override it.

final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on.

Java interface variables are by default final and static.

### Can we execute a program without main() method?

Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

**Can we declare a class as static?**

We can’t declare a top-level class as static however an inner class can be declared as static. If inner class is declared as static, it’s called static nested class.

Static nested class is same as any other top-level class and is nested for only packaging convenience.

**What is static import?**

If we have to use any static variable or method from other class, usually we import the class and then use the method/variable with class name.

import java.lang.Math;

//inside class

double test = Math.PI \* 5;

We can do the same thing by importing the static method or variable only and then use it in the class as if it belongs to it.

import static java.lang.Math.PI;

//no need to refer class now

double test = PI \* 5;

Use of static import can cause confusion, so it’s better to avoid it. Overuse of static import can make your program unreadable and unmaintainable.

**What is anonymous inner class?**

A local inner class without name is known as anonymous inner class. An anonymous class is defined and instantiated in a single statement. Anonymous inner class always extend a class or implement an interface.

Since an anonymous class has no name, it is not possible to define a constructor for an anonymous class. Anonymous inner classes are accessible only at the point where it is defined.

**What is Classloader in Java?**

Java Classloader is the program that loads byte code program into memory when we want to access any class. We can create our own classloader by extending ClassLoader class and overriding loadClass(String name) method. Learn more at java classloader.

**What are different types of classloaders?**

There are three types of built-in Class Loaders in Java:

.Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.

A. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.

B. System Class Loader – It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.

**What is ternary operator in java?**

Java ternary operator is the only conditional operator that takes three operands. It’s a one liner replacement for if-then-else statement and used a lot in java programming. We can use ternary operator if-else conditions or even switch conditions using nested ternary operators. An example can be found at java ternary operator.

**What is garbage collection? Can we enforce it?**

Garbage collection is an automatic memory management feature in many modern programming languages, such as Java and languages in the .NET framework. Languages that use garbage collection are often interpreted or run within a virtual machine like the JVM. In each case, the environment that runs the code is also responsible for garbage collection. A GC has two goals: any unused memory should be freed, and no memory should be freed unless the program will not use it anymore.

**Can you force garbage collection??**

Nope, System.gc() is as close as you can get. Your best option is to call System.gc() which simply is a hint to the garbage collector that you want it to do a collection. There is no way to force and immediate collection though as the garbage collector is non-deterministic. Also, under the documentation for OutOfMemoryError it declares that it will not be thrown unless the VM has failed to reclaim memory following a full garbage collection. So if you keep allocating memory until you get the error, you will have already forced a full garbage collection.

**What is Serialization and Deserialization?**

We can convert a Java object to a 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.

**What is the use of System class?**

Java System Class is one of the core classes. One of the easiest way to log information for debugging is System.out.print() method.

System class is final so that we can’t subclass and override it’s behavior through inheritance. System class doesn’t provide any public constructors, so we can’t instantiate this class and that’s why all of it’s methods are static.

Some of the utility methods of System class are for array copy, get current time, reading environment variables. Read more at Java System Class.

**What is difference between Heap and Stack Memory?**

Major difference between Heap and Stack memory are as follows:

•Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

•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.

• Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally.

For a detailed explanation with a sample program, read Java Heap vs Stack Memory.

**How to create an immutable object in Java? Count all benefits?**

**To create immutable class in java, you have to do following steps.**

Declare the class as final so it can’t be extended.

Make all fields private so that direct access is not allowed.

Don’t provide setter methods for variables

Make all mutable fields final so that it’s value can be assigned only once.

Initialize all the fields via a constructor performing deep copy.

### Following benefits of immutable class.

### Immutable classes –

### Are simple to construct, test, and use

### are automatically thread-safe and have no synchronization issues

### do not need a copy constructor

### do not need an implementation of clone

### allow hashCode to use lazy initialization, and to cache its return value

### do not need to be copied defensively when used as a field

### make good Map keys and Set elements (these objects must not change state while in the collection)

### have their class invariant established once upon construction, and it never needs to be checked again

### always have “failure atomicity” (a term used by Joshua Bloch) : if an immutable object throws an exception, it’s never left in an undesirable or indeterminate state.

### Why there are two Date classes; one in java.util package and another in java.sql?

### A java.util.Date represents date and time of day, a java.sql.Date only represents a date. The complement of java.sql.Date is java.sql.Time, which only represents a time of day.

### The java.sql.Date is a subclass (an extension) of java.util.Date. So, what changed in java.sql.Date:

### -- toString() generates a different string representation: yyyy-mm-dd

### -- a static valueOf(String) methods to create a date from a string with above representation

### -- the getters and setter for hours, minutes and seconds are deprecated

### The java.sql.Date class is used with JDBC and it was intended to not have a time part, that is, hours, minutes, seconds, and milliseconds should be zero… but this is not enforced by the class.

### Explain marker interfaces?

### The marker interface pattern is a design pattern in computer science, used with languages that provide run-time type information about objects. It provides a means to associate metadata with a class where the language does not have explicit support for such metadata. In java, it is used as interfaces with no method specified.

### A good example of use of marker interface in java is Serializable interface. A class implements this interface to indicate that its non-transient data members can be written to a byte steam or file system.

### A major problem with marker interfaces is that an interface defines a contract for implementing classes, and that contract is inherited by all subclasses. This means that you cannot “un-implement” a marker. In the example given, if you create a subclass that you do not want to serialize (perhaps because it depends on transient state), you must resort to explicitly throwing NotSerializableException.

### How does substring () inside String works?

### String in java are like any other programming language, a sequence of characters. This is more like a utility class to work on that char sequence. This char sequence is maintained in following variable:

### /\*\* The value is used for character storage. \*/

### private final char value[];

### To access this array in different scenarios, following variables are used:

### /\*\* The offset is the first index of the storage that is used. \*/

### private final int offset;

### /\*\* The count is the number of characters in the String. \*/

### private final int count;

### Whenever we create a substring from any existing string instance, substring() method only set’s the new values of offset and count variables. The internal char array is unchanged. This is a possible source of memory leak if substring() method is used without care.

**Give the list of Java Object class methods.**

public final Class getClass()

returns the Class class object of this object. The Class class can further be used to get the metadata of this class.

public int hashCode()

returns the hashcode number for this object.

public boolean equals(Object obj)

compares the given object to this object.

protected Object clone() throws CloneNotSupportedException

creates and returns the exact copy (clone) of this object.

public String toString()

returns the string representation of this object.

public final void notify()

wakes up single thread, waiting on this object's monitor.

public final void notifyAll()

wakes up all the threads, waiting on this object's monitor.

public final void wait(long timeout)throws InterruptedException

causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).

public final void wait(long timeout,int nanos)throws InterruptedException

causes the current thread to wait for the specified miliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).

public final void wait()throws InterruptedException

causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

protected void finalize()throws Throwable

is invoked by the garbage collector before object is being garbage collected.

**Usage of hashCode() and equals()**

**hashCode()** method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.

**equals()** method, as name suggest, is used to simply verify the equality of two objects. Default implementation simply check the object references of two objects to verify their equality.

**When do you override hashCode() and equals()?**

hashCode() and equals() methods have been defined in Object class which is parent class for java objects. For this reason, all java objects inherit a default implementation of these methods.

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable like data structure. By default, Object’s hashCode() method returns and integer representation of memory address where object is stored.

equals() method, as name suggest, is used to simply verify the equality of two objects. Default implementation simply check the object references of two objects to verify their equality.

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode() method, which states that equal objects must have equal hash codes.

equals() must define an equality relation (it must be reflexive, symmetric and transitive). In addition, it must be consistent (if the objects are not modified, then it must keep returning the same value). Furthermore, o.equals(null) must always return false.

hashCode() must also be consistent (if the object is not modified in terms of equals(), it must keep returning the same value).

The relation between the two methods is:

Whenever a.equals(b) then a.hashCode() must be same as b.hashCode().

**Explain transient and volatile keywords in java?**

**Transient**

“The transient keyword in Java is used to indicate that a field should not be serialized.” According to language specification: Variables may be marked transient to indicate that they are not part of the persistent state of an object. For example, you may have fields that are derived from other fields, and should only be done so programmatically, rather than having the state be persisted via serialization.

For example, in class BankPayment.java fields like principal and rate can be serialized while interest can be calculated any time even after de-serialization.

If we recall, each thread in java has its own local memory space as well and it does all read/write operations in its local memory. Once all operations are done, it write back the modified state of variable in main memory from where all threads access this variable. Normally, this is the default flow inside JVM. But, the volatile modifier tells the JVM that a thread accessing the variable must always reconcile its own private copy of the variable with the master copy in memory. It means every time thread want to read the state of variable, it must flush its local memory state and update the variable from main memory.

**Volatile**

volatile is most useful in lock-free algorithms. You mark the variable holding shared data as volatile when you are not using locking to access that variable and you want changes made by one thread to be visible in another, or you want to create a “happens-after” relation to ensure that computation is not re-ordered, again, to ensure changes become visible at the appropriate time.

The volatile should be used to safely publish immutable objects in a multi-threaded Environment. Declaring a field like public volatile ImmutableObject foo secures that all threads always see the currently available instance reference.

**Method Overloading Rules**

1) First and important rule to overload a method in java is to change method signature. Method signature is made of number of arguments, type of arguments and order of arguments if they are of different types.

2) Return type of method is never part of method signature, so only changing the return type of method does not amount to method overloading.

3) Thrown exceptions from methods are also not considered when overloading a method. So your overloaded method throws the same exception, a different exception or it simply does no throw any exception; no effect at all on method loading.

**Method Overriding Rules**

1) The method argument list in overridden and overriding methods must be exactly same If they don’t match, you will end up with an overloaded method.

2) The return type of overriding method can be child class of return type declared in overridden method.

3) private, static and final methods cannot be overridden in java in any way. As simple as that!!

4) Overriding method cannot throw checked Exception higher in hierarchy than thrown by overridden method. Let’s say for example overridden method in parent class throws FileNotFoundException, the overriding method in child class can throw FileNotFoundException; but it is not allowed to throw IOException or Exception, because IOException or Exception are higher in hierarchy i.e. super classes of FileNotFoundException.

**Difference between method Overloading and Overriding in java.**

Overloading vs Overriding in Java

1). Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.

2). Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.

3). The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.

4). Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.

5). Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.

6). private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.

7). Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the overriding method can have more specific return type (refer this).

8). Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

**What is synchronization? Object level locking and class level locking?**

Synchronization refers to multi-threading. A synchronized block of code can only be executed by one thread at a time. Java supports multiple threads to be executed. This may cause two or more threads to access the same fields or objects. Synchronization is a process which keeps all concurrent threads in execution to be in synch. Synchronization avoids memory consistence errors caused due to inconsistent view of shared memory. When a method is declared as synchronized; the thread holds the monitor for that method’s object If another thread is executing the synchronized method, your thread is blocked until that thread releases the monitor.

Synchronization in java is achieved using synchronized keyword. You can use synchronized keyword in your class on defined methods or blocks. Keyword can not be used with variables or attributes in class definition.

Object level locking is mechanism when you want to synchronize a non-static method or non-static code block such that only one thread will be able to execute the code block on given instance of the class. This should always be done to make instance level data thread safe.

Class level locking prevents multiple threads to enter in synchronized block in any of all available instances on runtime. This means if in runtime there are 100 instances of DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads. This should always be done to make static data thread safe.

**What is native keyword? Explain in detail?**

The native keyword is applied to a method to indicate that the method is implemented in native code using JNI. It marks a method, that it will be implemented in other languages, not in Java.

Native methods were used in the past to write performance critical sections but with Java getting faster this is now less common. Native methods are currently needed when

You need to call a library from Java that is written in other language.

You need to access system or hardware resources that are only reachable from the other language (typically C). Actually, many system functions that interact with real computer (disk and network IO, for instance) can only do this because they call native code.

The downsides of using native code libraries are also significant:

1). JNI / JNA have a tendency to destabilize the JVM, especially if you try to do something complicated. If your native code gets native code memory management wrong, there’s a chance that you will crash the JVM. If your native code is non-reentrant and gets called from more than one Java thread, bad things will happen … sporadically. And so on.

2). Java with native code is harder to debug than pure Java or pure C/C++.

3). Native code can introduce significant platform dependencies / issues for an otherwise platform independent Java app.

4). Native code requires a separate build framework, and that may have platform / portability issues as well.

# [What are all the different ways to create an object?](http://stackoverflow.com/questions/95419/what-are-all-the-different-ways-to-create-an-object-in-java)

There are four different ways to create objects in java:

**A. Using new keyword**

This is the most common way to create an object in java. Almost 99% of objects are created in this way.

MyObject object = new MyObject();

**B. Using Class.forName()**

If we know the name of the class & if it has a public default constructor we can create an object in this way.

MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();

**C. Using clone()**

The clone() can be used to create a copy of an existing object.

MyObject anotherObject = new MyObject();

MyObject object = (MyObject) anotherObject.clone();

**D. Using object deserialization**

Object deserialization is nothing but creating an object from its serialized form.

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyObject object = (MyObject) inStream.readObject();