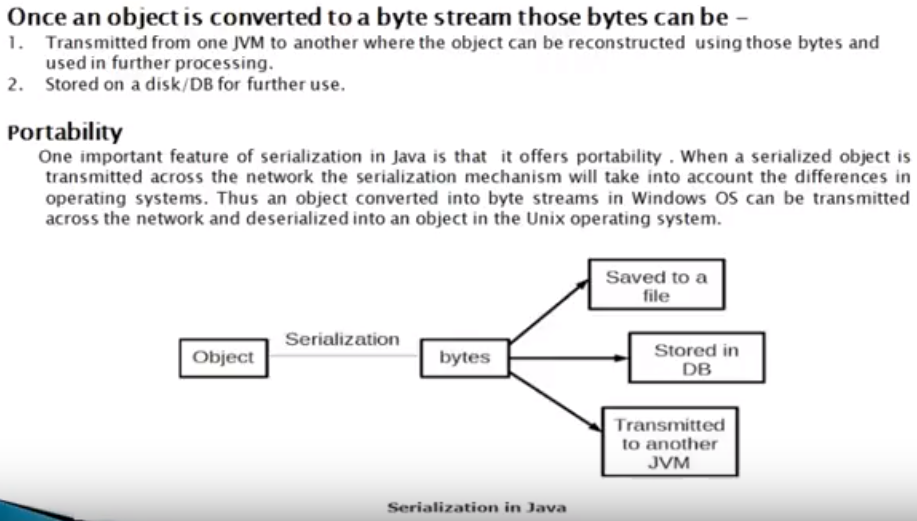
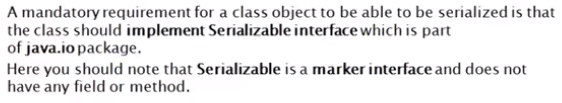
Serializable: If you want a class object to be serializable, all you need to do it implement the java.io.Serializable interface. Serializable in java is a marker interface and has no fields or methods to implement. It’s like an Opt-In process through which we make our classes serializable.

Serialization in java is implemented by ObjectInputStream and ObjectOutputStream, so all we need is a wrapper over them to either save it to file or send it over the network.



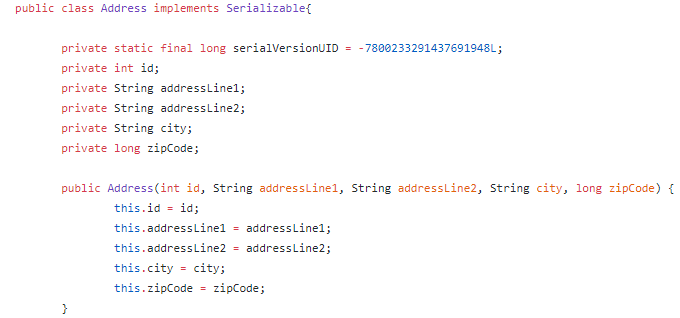




To *serialize* an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object. A Java object is *serializable* if its class or any of its super classes implements either

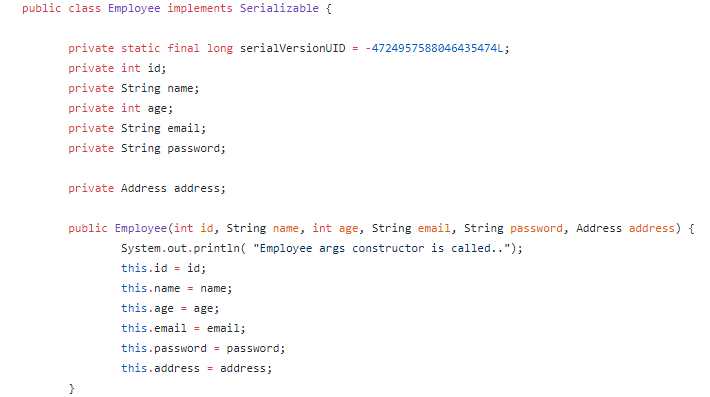
the java.io.Serializable interface or its sub interface, java.io.Externalizable. *Deserialization* is the process of converting the serialized form of an object back into a copy of the object.

Example:



//Setter and getters

}

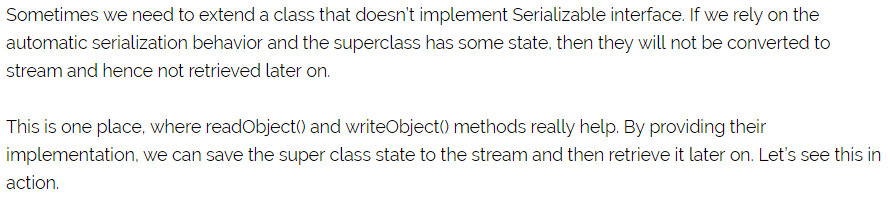


//Setter and Getter

}



Example 2:



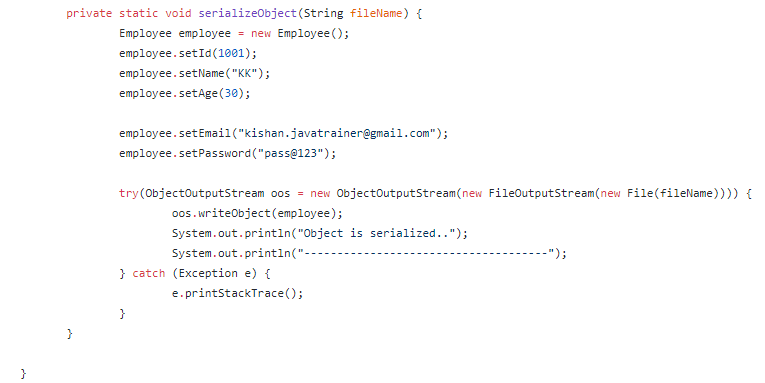
If parent class not serialize then we have to do something in child class below is the example->

Person.java is the parent class:

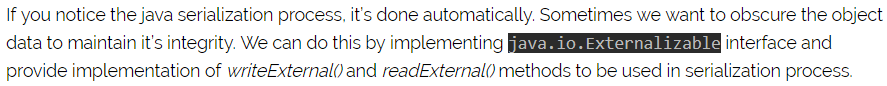




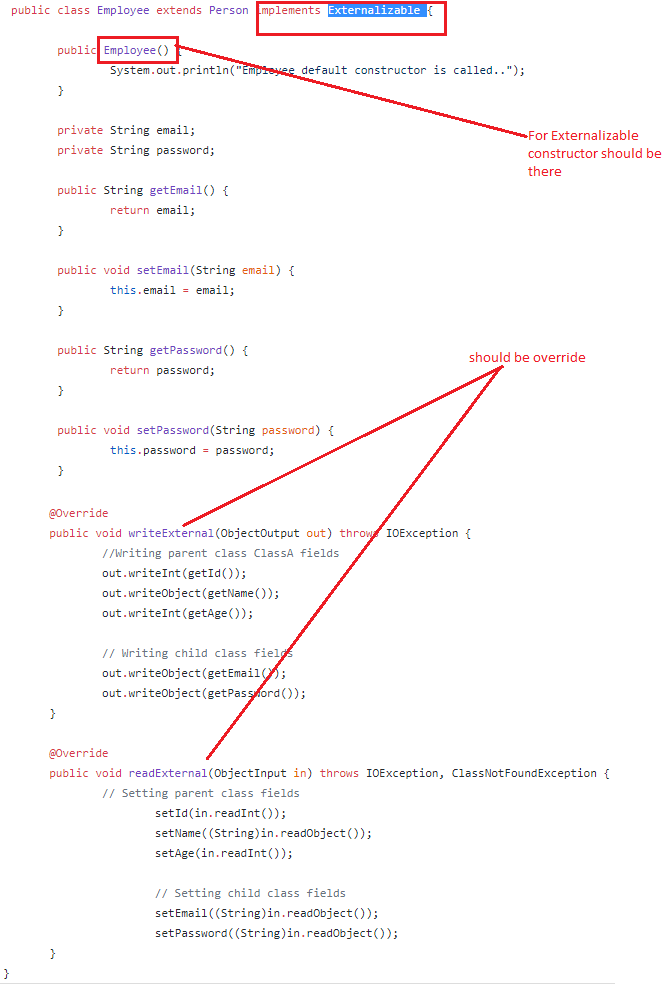




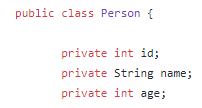
Externalizable:



example:



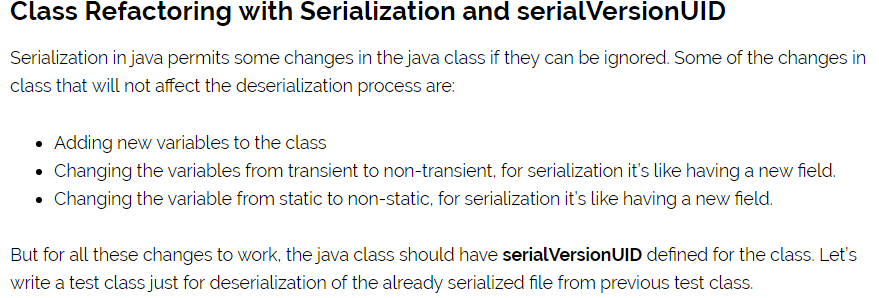
Person class not implementing externalization.



//setter and getter

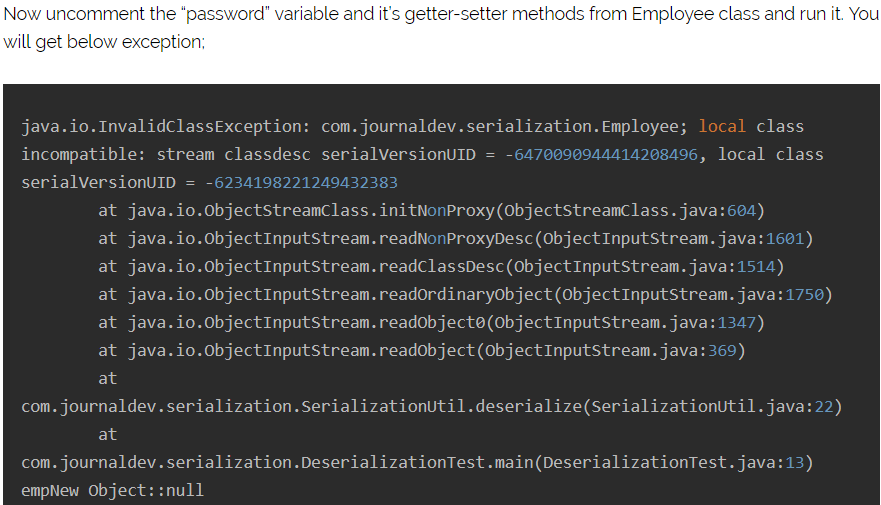
}

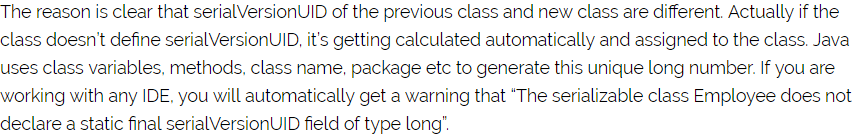


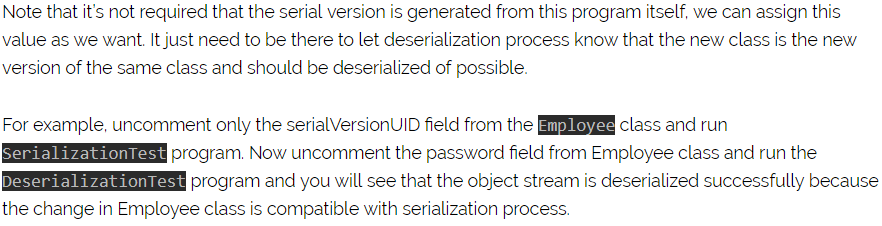


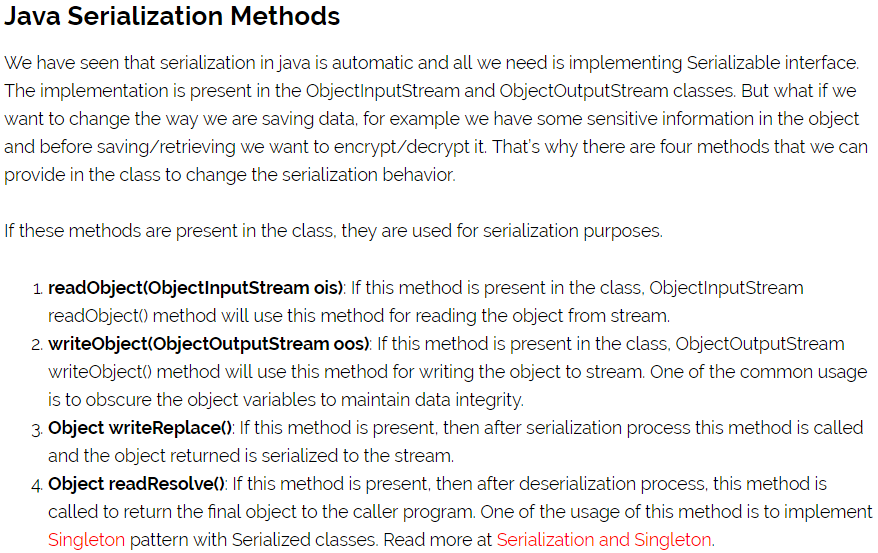
The serialVersionUID is a universal version identifier for a Serializable class. Deserialization uses this number to ensure that a loaded class corresponds exactly to a serialized object. If no match is found, then an InvalidClassException is thrown.





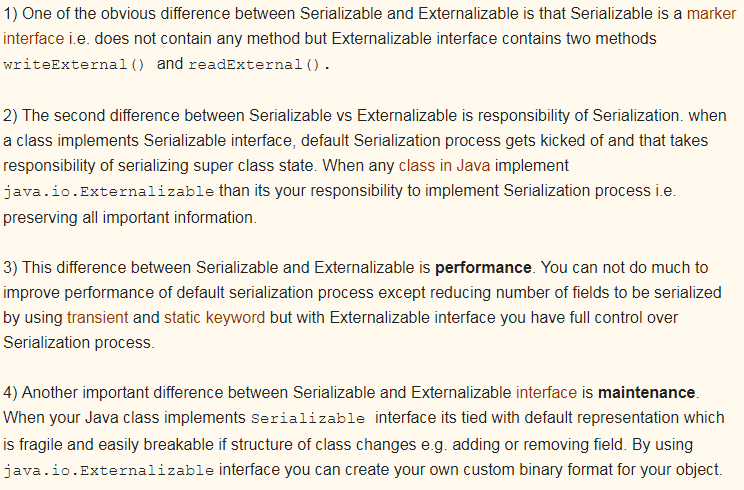


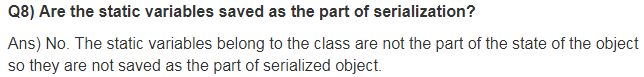


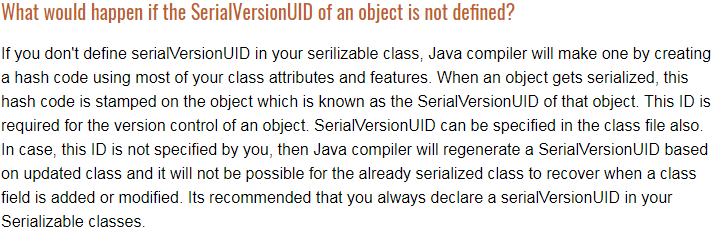


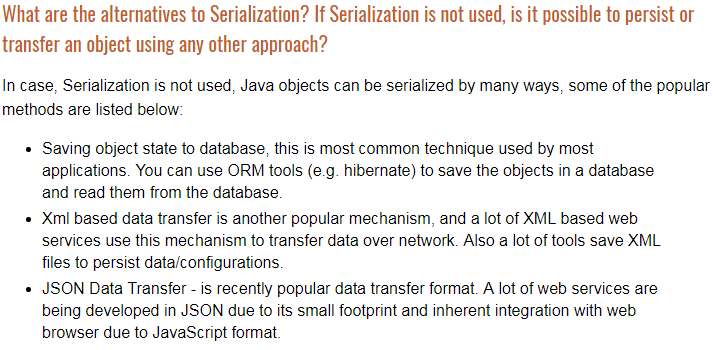


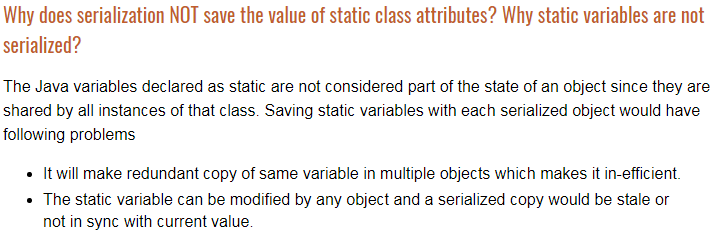
Difference between Serializable and Externalizable:



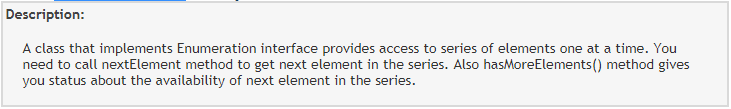


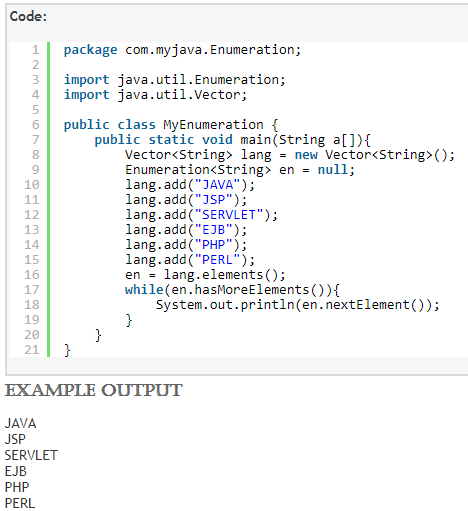


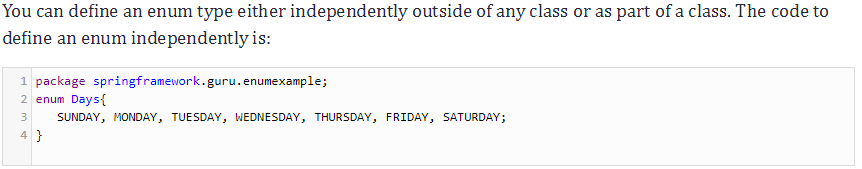


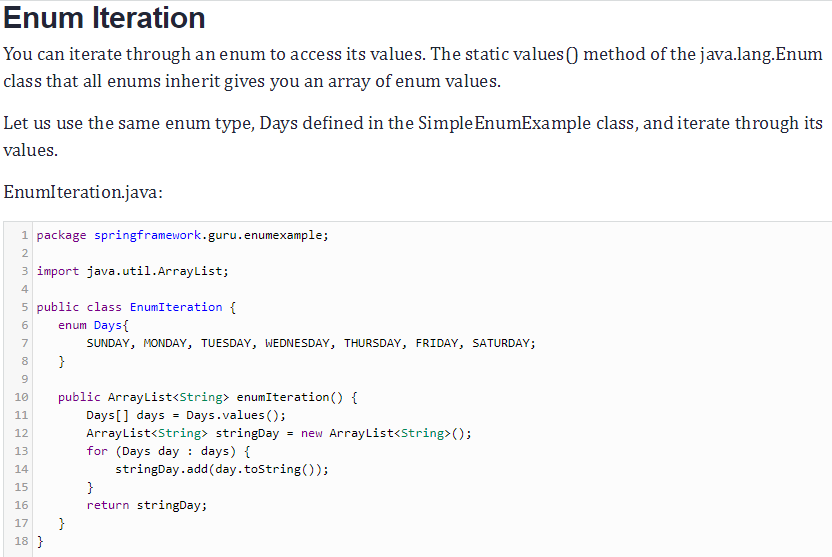


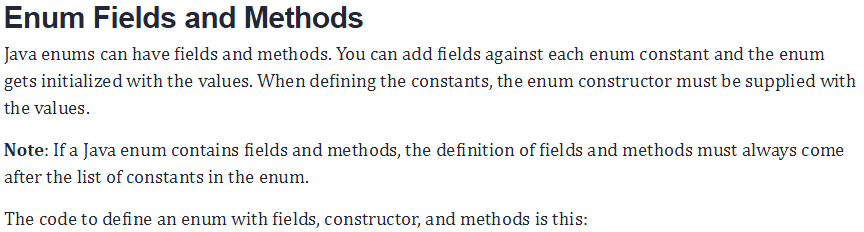
Enumeration: Enumeration is a recent (from JDK5) inclusion into the family of [Java](http://www.developer.com/java) APIs. It basically represents a list of named constants.



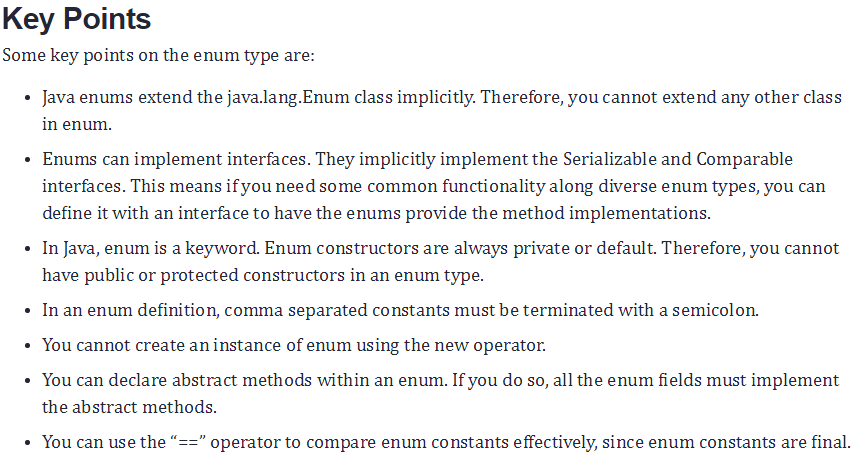




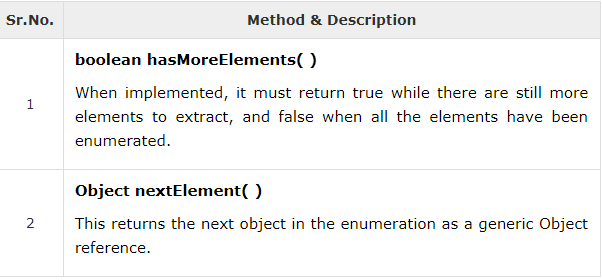






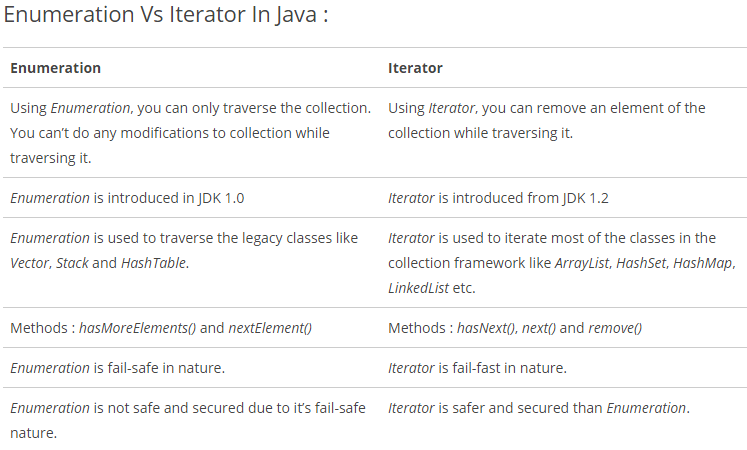


Enumeration Interface: The Enumeration interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects.

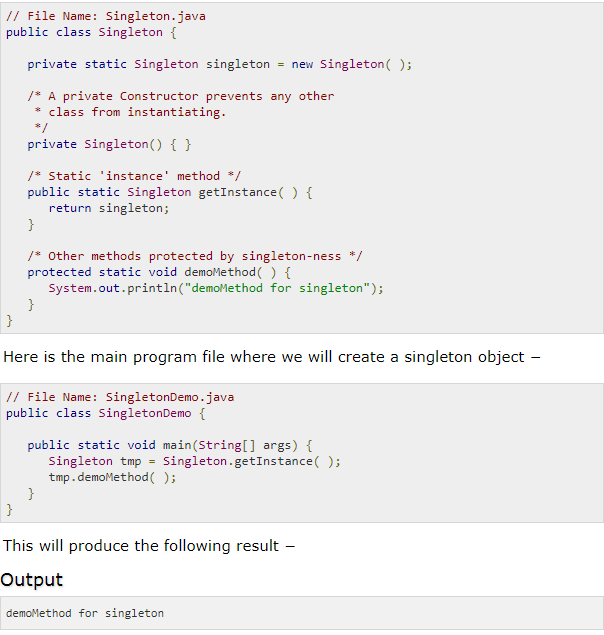


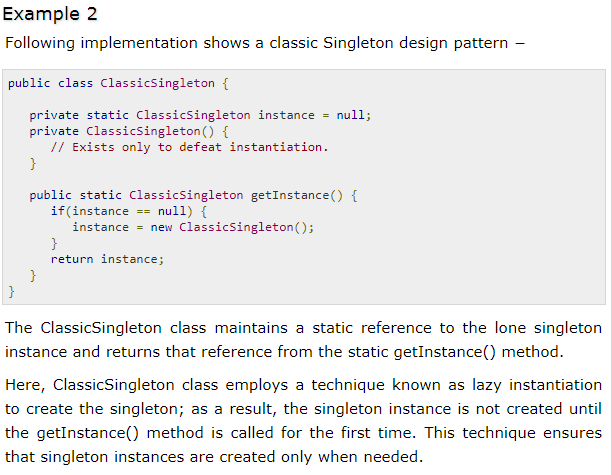
Example:

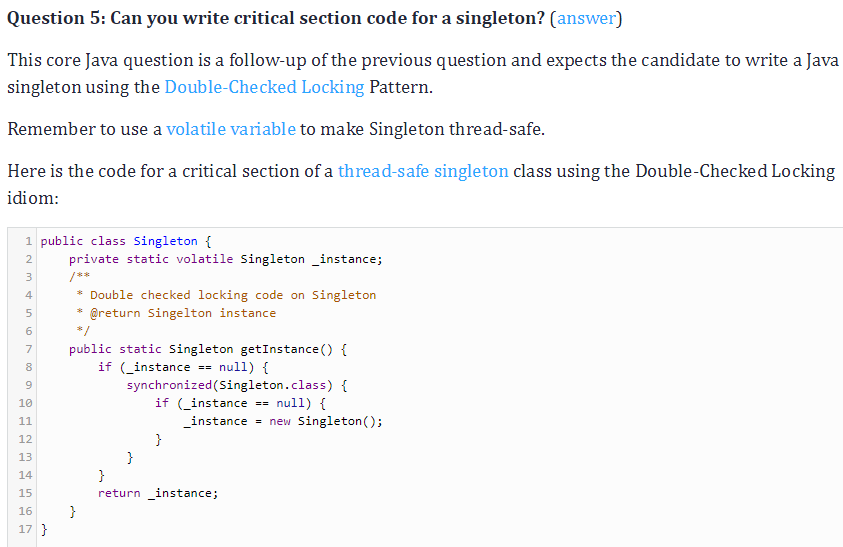




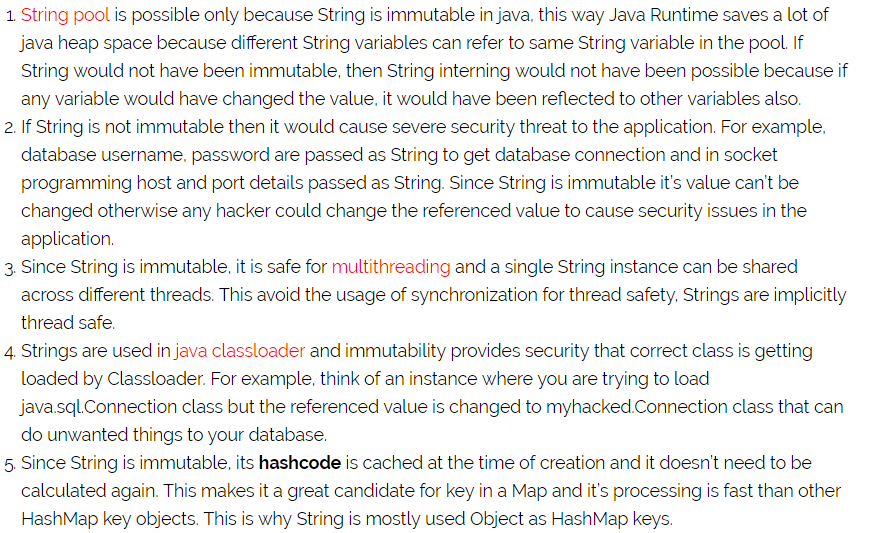
Singleton Class: The Singleton's purpose is to control object creation, limiting the number of objects to only one. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields. Singletons often control access to resources, such as database connections or sockets.

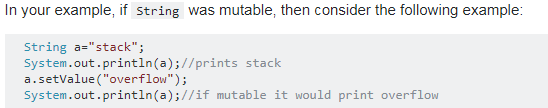


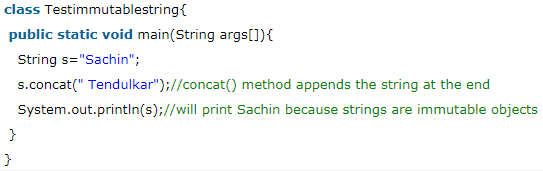


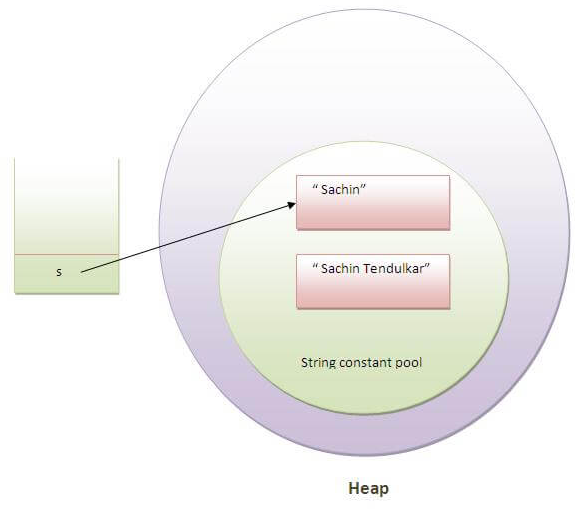


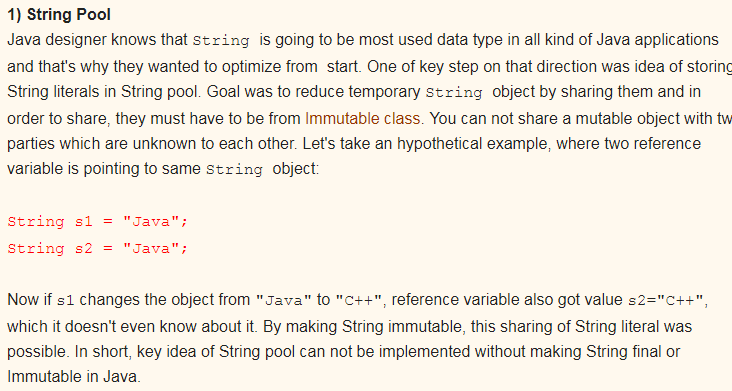
[Why is String immutable in Java?](https://stackoverflow.com/questions/22397861/why-is-string-immutable-in-java)

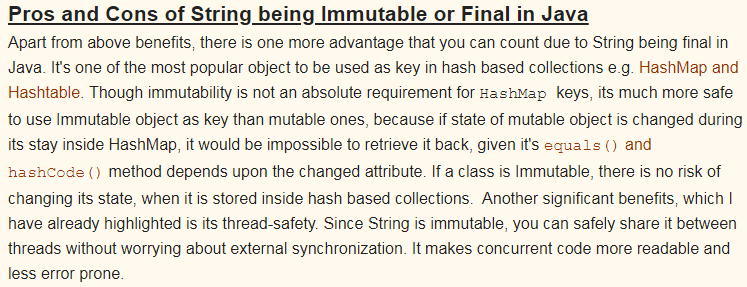


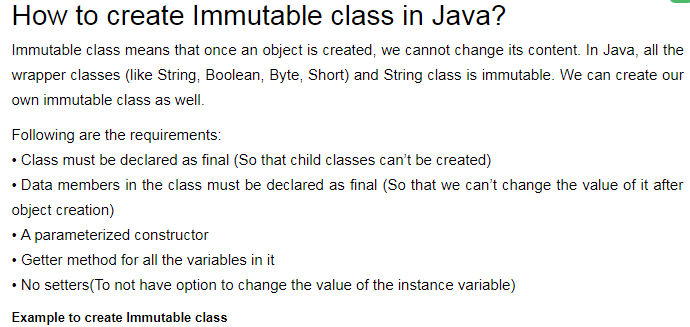


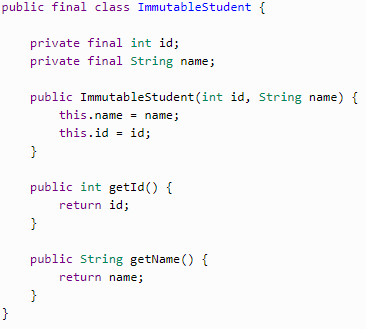






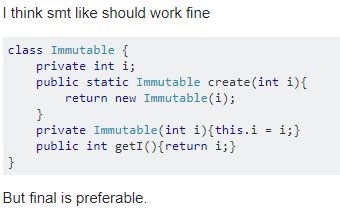




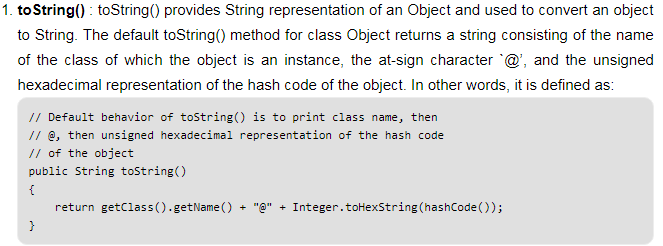


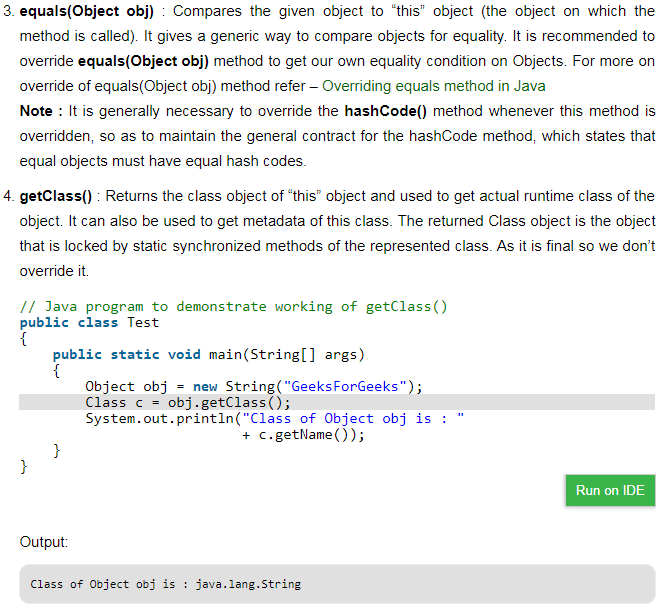
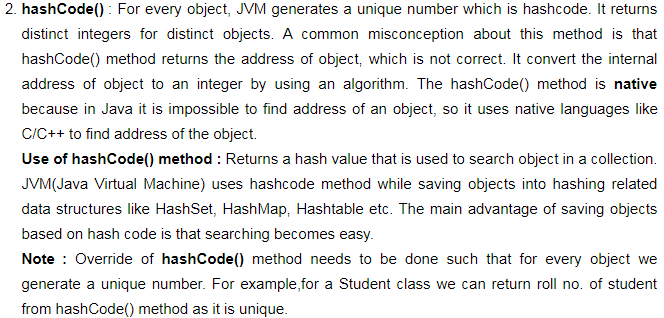
How to create an immutable class in Java without using final keyword:

By creating private constructor we can make immutable class:

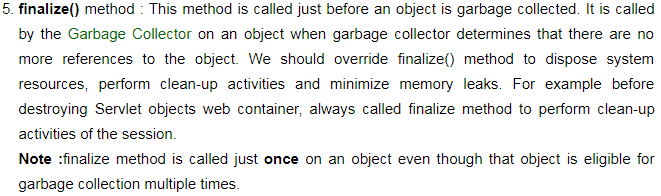


Object class: **Object** class is present in **java.lang** package. Every class in Java is directly or indirectly derived from the **Object** class. If a Class does not extend any other class then it is direct child class of **Object** and if extends other class then it is an indirectly derived. Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of inheritance hierarchy in any Java Program.

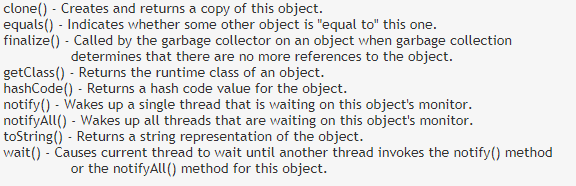




Note :After loading a .class file, JVM will create an object of the type java.lang.Class in the Heap area. We can use this class object to get Class level information. It is widely used in Reflection



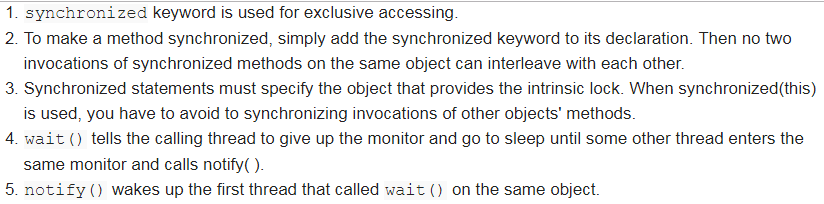


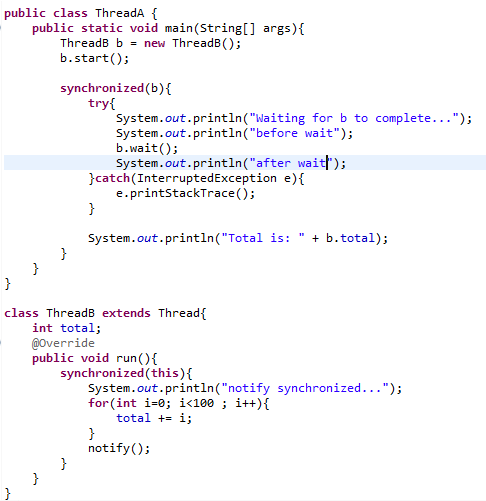


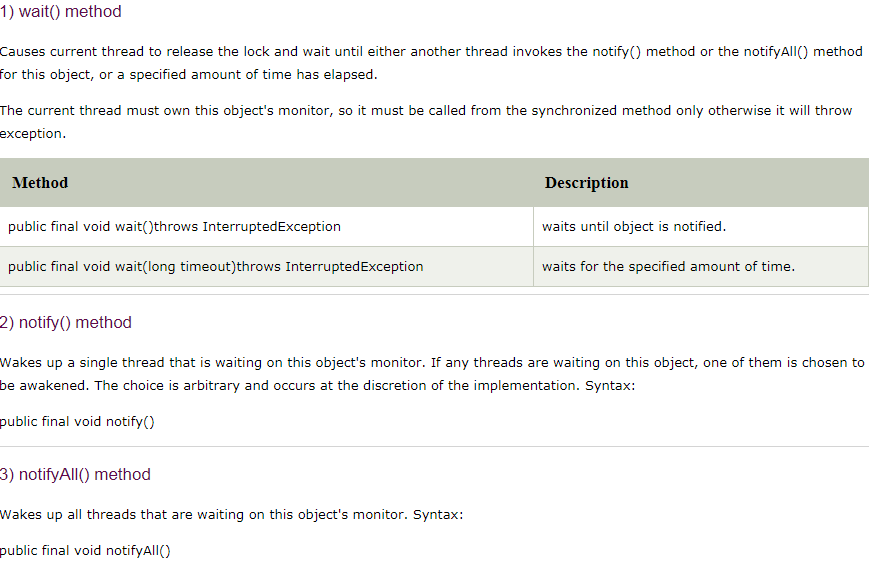
Object class wait and notify():

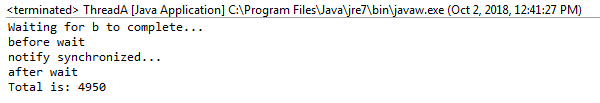
**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

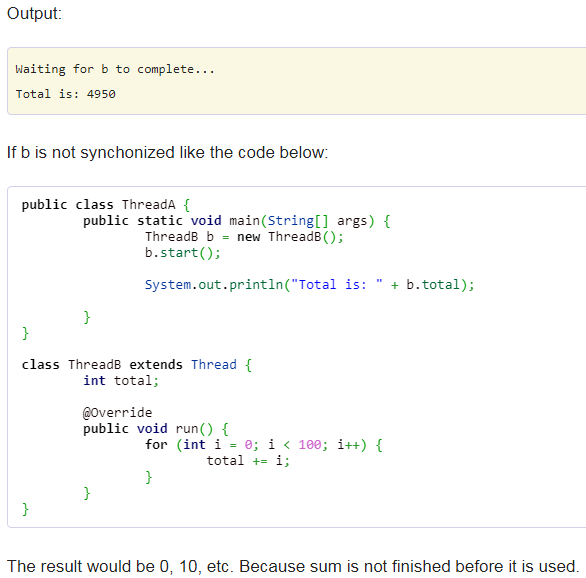
Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**



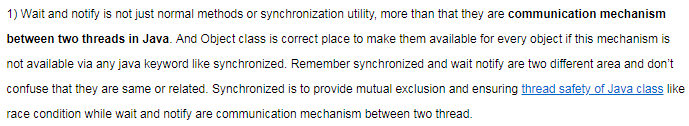


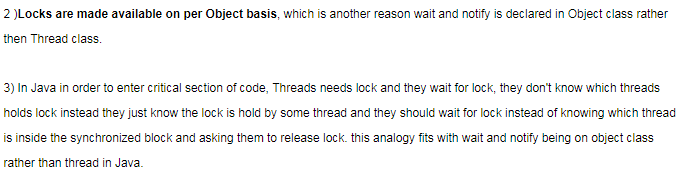


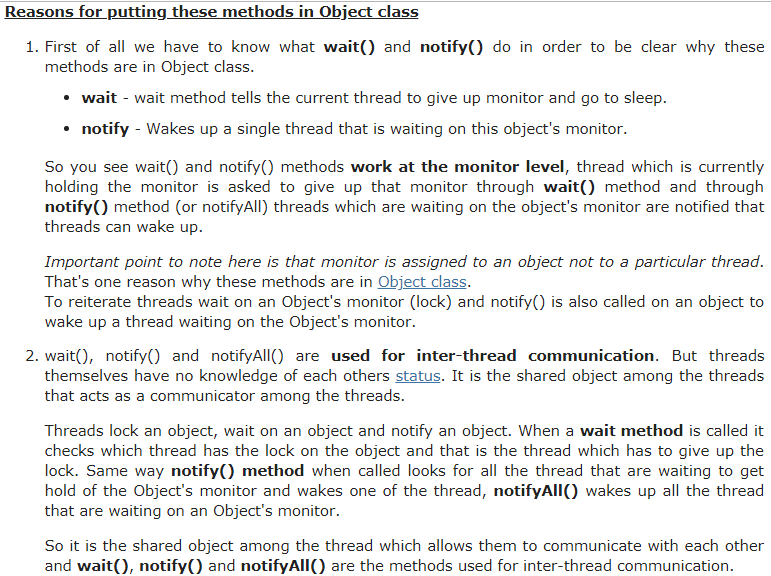


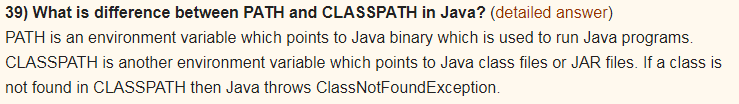


Why wait(),notify() and notifyAll() methods are in Object Class not in Thread Class?: As every object in the java has only one lock or monitor and notify() and notifyAll() are used for monitor sharing. That’s why it’s part of object class rather than thread class.









What is Interface in Java: An interface in java is a blueprint of a class. It has static constants and abstract methods only.  
The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.  
It cannot be instantiated just like abstract class.

There are mainly three reasons to use interface. They are given below.   
• It is used to achieve fully abstraction.  
• By interface, we can support the functionality of multiple inheritance.  
• It can be used to achieve loose coupling.  
In other words, Interface fields are public, static and final bydefault, and methods are public and abstract.

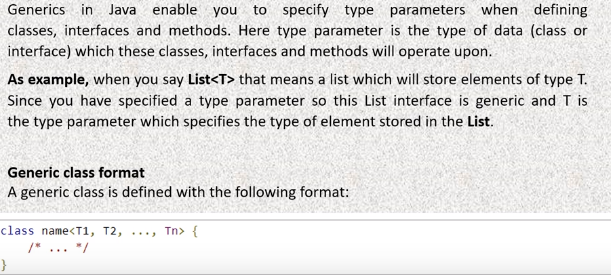
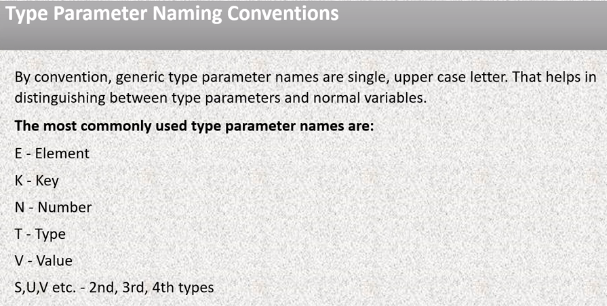
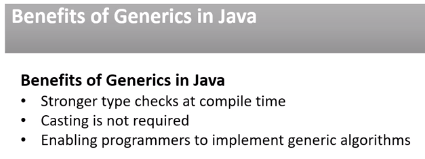
What is marker or tagged interface: An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

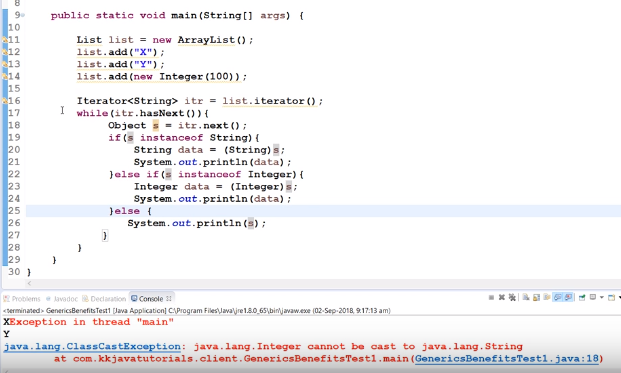
//How Serializable interface is written?

public interface Serializable{

}

Generics:

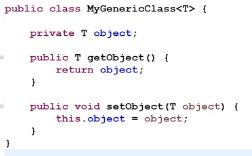


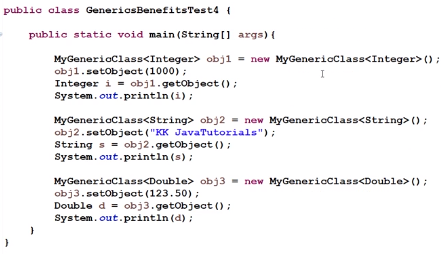
For generic we don’t need for casting:

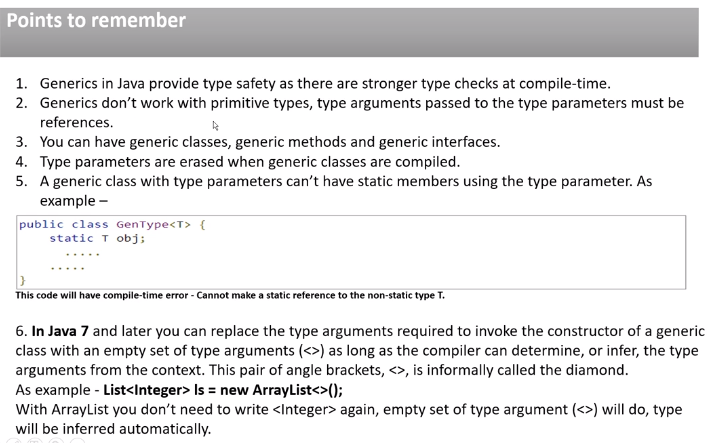


Custom generic class:

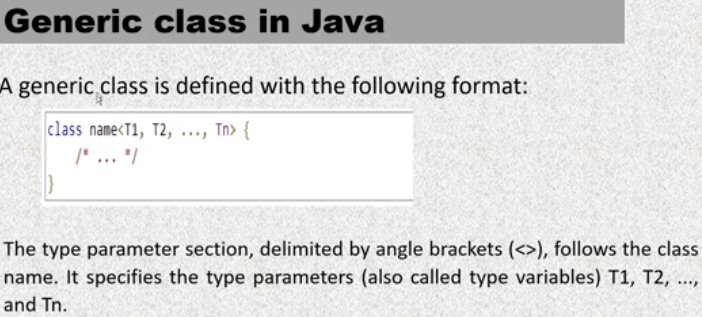
Example 1:



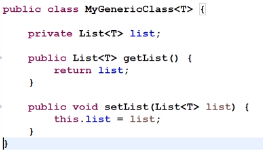




Generic Class,Interface and Methods:

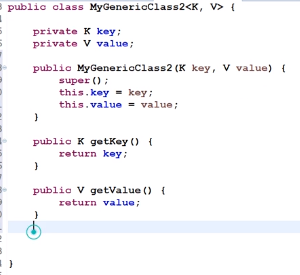


Example1:



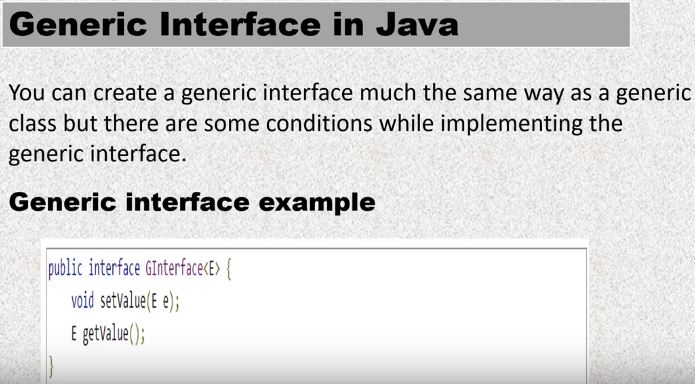


Example2:



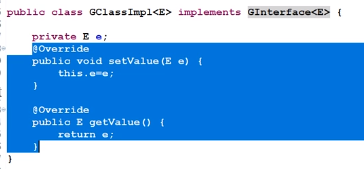
In client program:

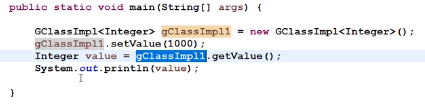




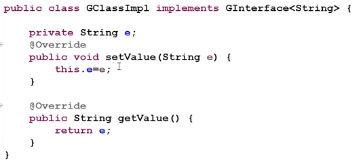
Example1:

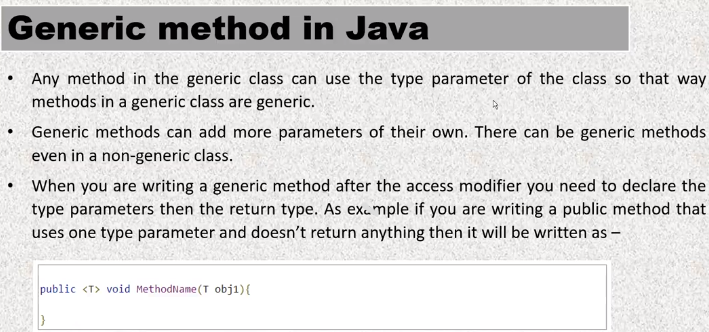




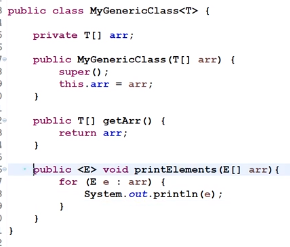


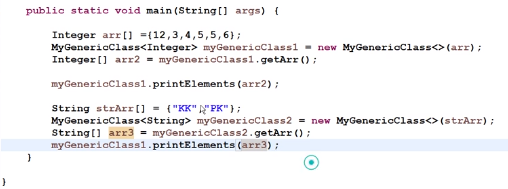
Example2:



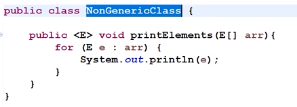


Example1:





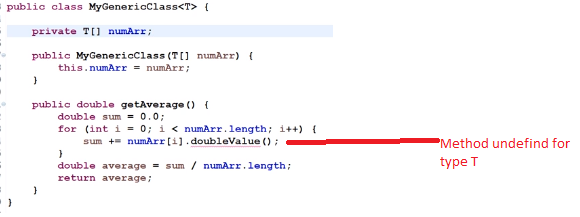
Example2:

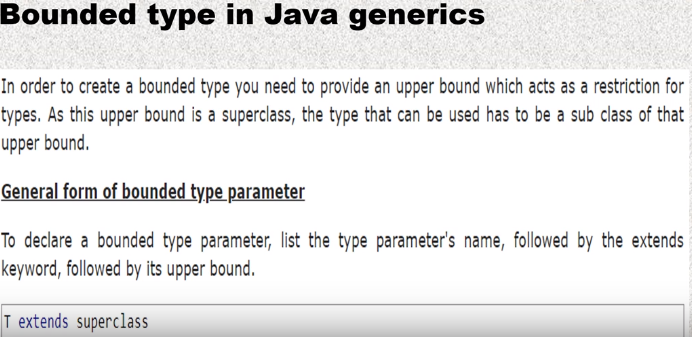


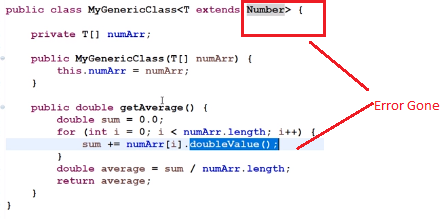


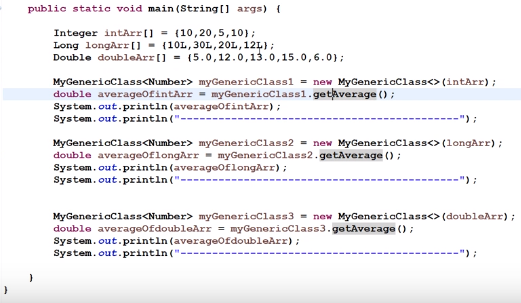


Example1: without bounded type problem is:

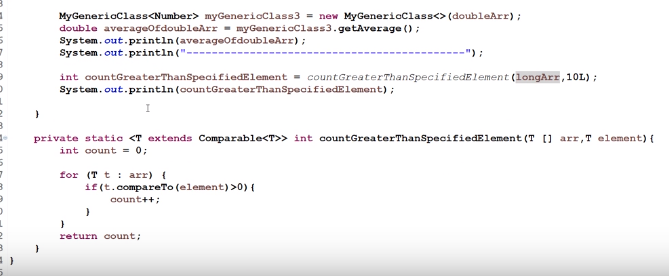


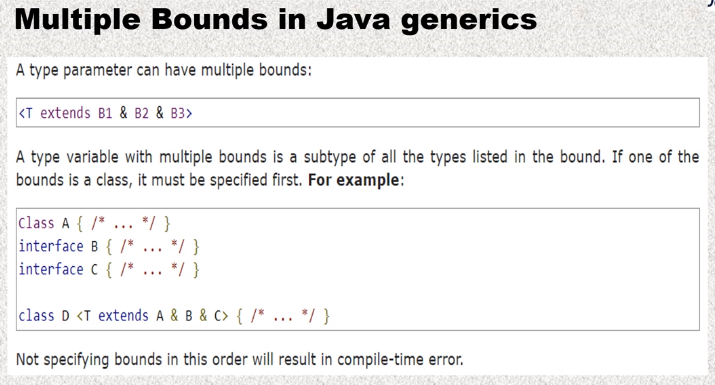


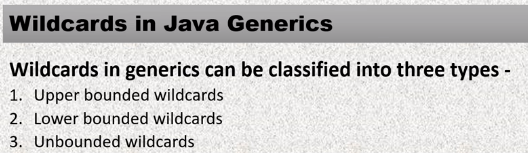


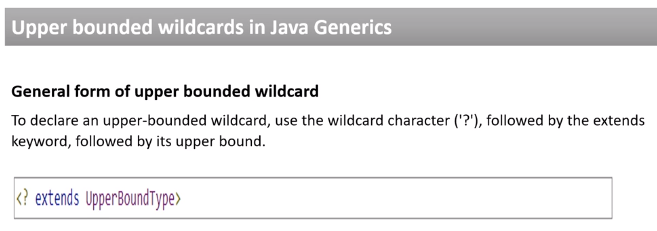


Example2: continue with above example



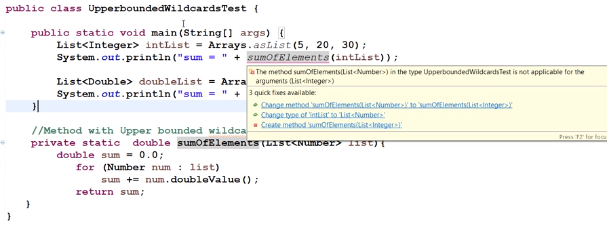




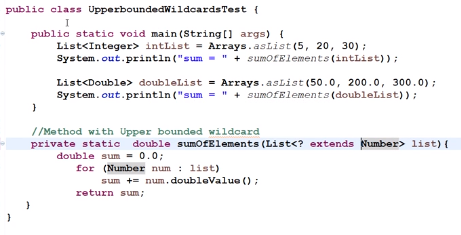


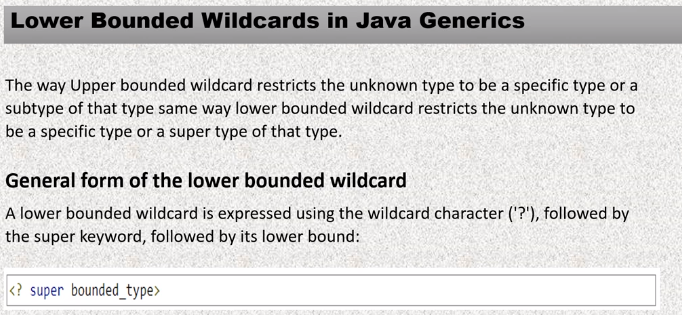
Example upper bounded:

Without :

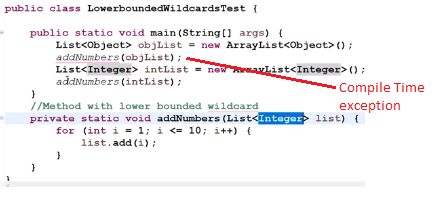


With upper bound:

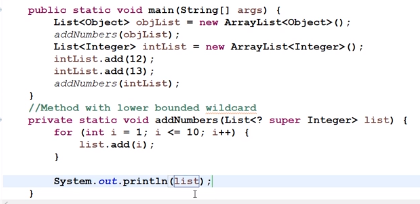


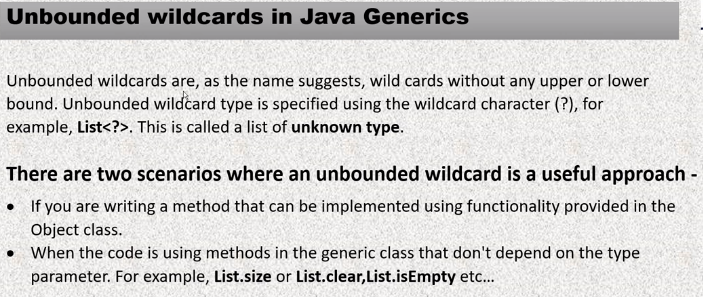


Example:without lower bound:

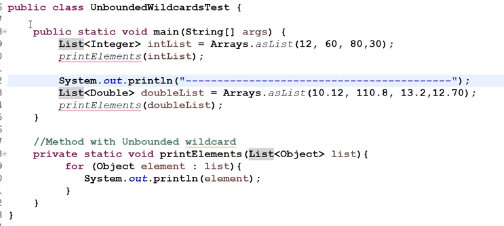


With lower bound:

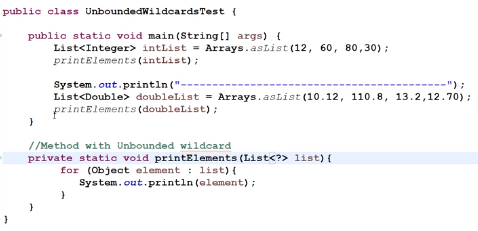


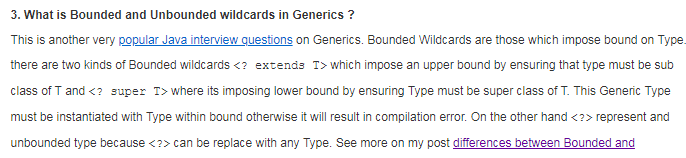


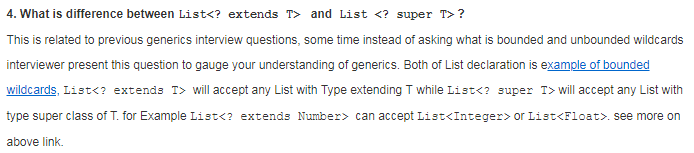
Example1 : without unbounded:

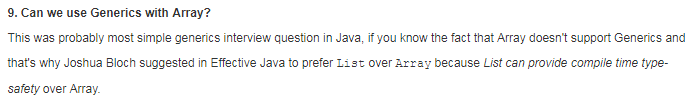


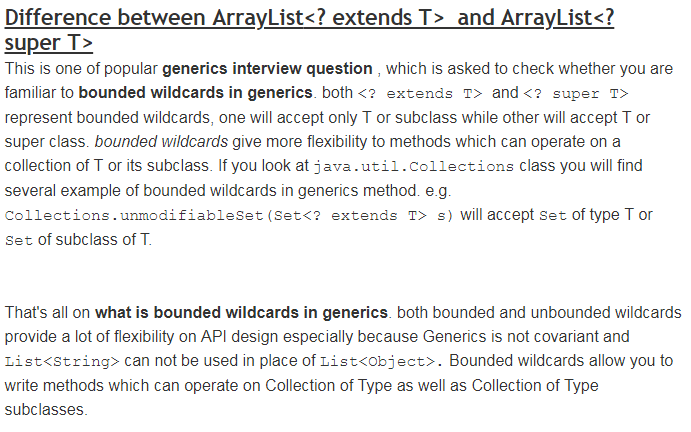
With unbouned:

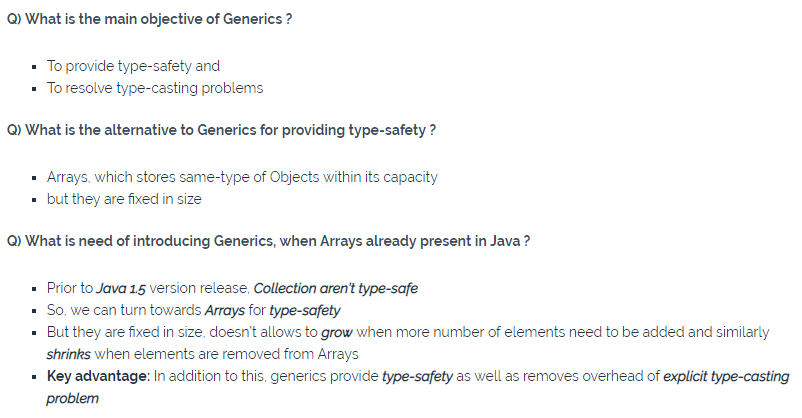


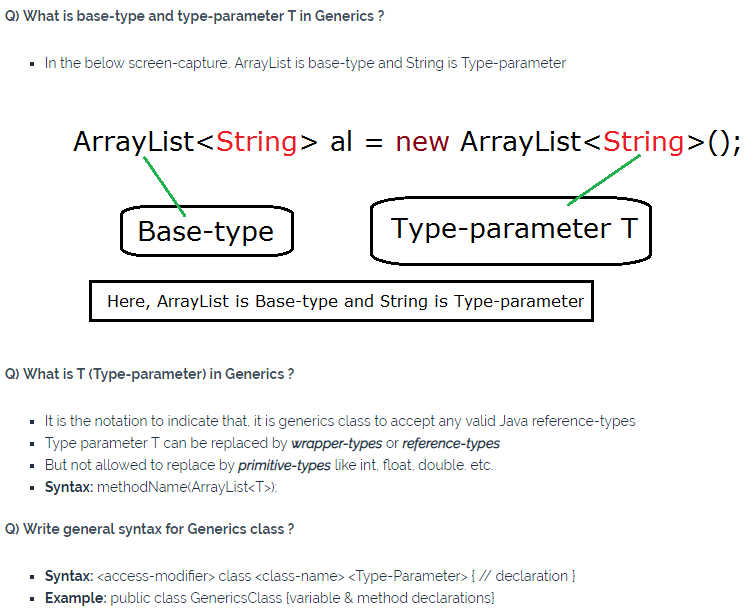
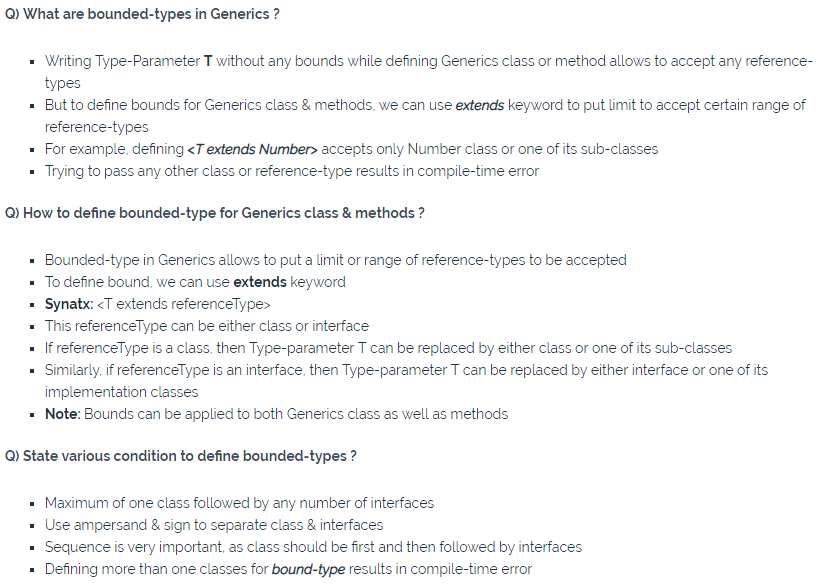
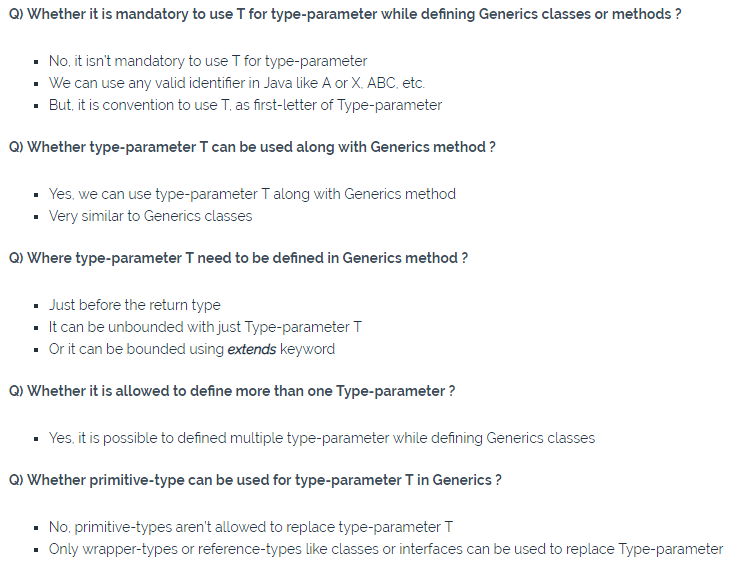










# **Inner class and nested Static Class in Java:**

Where should you use nested class in Java? This is one of the tricky Java question, If a class can only be useful to one particular class, it make sense to keep that inside the class itself  otherwise declare it as top level class. Nested class improves Encapsulation and help in maintenance. I actually look JDK itself for examples and if you look HashMap class, you will find that Map.Entry is a good example of nested class in Java.

## **Inner Class:**

Any class which is not a top level or declared inside another class is known as nested class and out of those nested classes, class which are declared [non static](http://javarevisited.blogspot.sg/2012/02/why-non-static-variable-cannot-be.html) are known as Inner class in Java. there are three kinds of Inner class in Java:

1. Local inner class
2. Anonymous inner class
3. Member inner class

Local inner class is declared inside a code block or method. Anonymous inner class is a class which doesn't have name to reference and initialized at same place where it gets created. Member inner class is declared as non static member of outer class.

**Here are some important properties of Inner classes in Java:**

1) In order to create instance of Inner class, an instance of outer class is required. Every instance of inner class is bounded to one instance of Outer class.

2) Member inner class can be make [private, protected or public](http://javarevisited.blogspot.sg/2012/10/difference-between-private-protected-public-package-access-java.html). its just like any other member of class.

3) Local inner class can not be private, protected or public because they exist only inside of local block or method. You can only use [final modifier](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with local inner class.

4) Anonymous Inner class are common to implement [Runnable](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) or CommandListener where you just need to implement one method. They are created and initialized at same line.

5) You can access current instance of Outer class, inside inner class as Outer.this variable.