**How to sort a collection of custom Objects in Java?**

We need to implement Comparable interface to support sorting of custom objects in a collection. Comparable interface has compareTo(T obj) method which is used by sorting methods and by providing this method implementation, we can provide default way to sort custom objects collection.

However, if you want to sort based on different criteria, such as sorting an Employees collection based on salary or age, then we can create Comparator instances and pass it as sorting methodology. For more details read Java Comparable and Comparator.

**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. Read more with example at [Java Composition](https://www.journaldev.com/1325/composition-in-java-example) example.

**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.

**What are different types of classloaders?**

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

1. Bootstrap Class Loader – It loads JDK internal classes, typically loads rt.jar and other core classes.
2. Extensions Class Loader – It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.
3. 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 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 will happen if you put return statement or System.exit () on try or catch block? Will finally block execute?**

**finally block will execute even if you put a return statement** in the try block or catch block but **finally block won't run if you call System.exit()** from try or catch block.

**Can you override a private or static method in Java?**

## [you can not override a private or static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html), if you create a similar method with same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding. What is the difference between StringBuffer and StringBuilder in Java? StringBuffer methods e.g. length(), capacity() or append() are [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) while corresponding methods in StringBuilder are not synchronized.

## [What is Marker interface? How is it used in Java?](http://www.fromdev.com/2012/02/java-interview-question-answer.html" \l "what-is-marker-interface--how-is-it-used-in-java-" \o "What is Marker interface? How is it used in Java?)

Eg: java.lang.Cloneable, java.io.Serializable

The "instanceof" keyword in java can be used to test if an object is of a specified type. So this keyword in combination with Marker interface can be used to take different actions based on type of interface an object implements.

## What is difference between HashMap and Hashtable?

HashMap and Hashtable both implements Map interface and looks similar, however there are following difference between HashMap and Hashtable.

* HashMap allows null key and values whereas Hashtable doesn’t allow null key and values.
* Hashtable is synchronized but HashMap is not synchronized. So HashMap is better for single threaded environment, Hashtable is suitable for multi-threaded environment.

## What is BlockingQueue?

**java.util.concurrent.BlockingQueue** is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue as it’s handled by implementation classes of BlockingQueue.

Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc.

## Can you make an abstract class/method final in Java?

No, **you cannot make an abstract class or method final in Java** because abstract and final are the mutual exclusive concept. An abstract class is incomplete and can only be instantiated by extending a concrete class and implementing all abstract methods, while a [final class](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) is considered as complete and cannot be extended further. This means when you make an abstract class final, it cannot be extended hence it cannot be used and that's why Java compiler throws a compile time error when you try to make an [abstract class](http://www.java67.com/2014/06/why-abstract-class-is-important-in-java.html) final in Java. In short, an abstract class cannot be final in Java, using both abstract and final modifier with a class is illegal in Java.  
  
Same is true for **abstract methods**, you cannot make them final in Java. An abstract method must be overridden to be useful and called but when you make the *abstract method* final it cannot be overridden in Java, hence there would be no way to use that method.

**What is the difference between Decorator and Proxy pattern in Java?**Another tricky Java design pattern question and trick here is that both Decorator and Proxy implements the interface of the object they decorate or encapsulate. As I said, many Java design pattern can have similar or exactly same structure but they differ in their intent.  
  
Decorator pattern is used to implement functionality on an already created object, while a Proxy pattern is used for controlling access to an object.  
  
One more difference between Decorator and the Proxy design pattern is that Decorator doesn't create an object, instead, it get the object in its constructor, while Proxy actually creates objects. 