Short cut for sysout - syso ,CTRl+space

A typical Software Development life cycle consists of the following stages:

* Stage 1: Planning and Requirement Analysis
* Stage 2: Defining Requirements
* Stage 3: Designing the product architecture
* Stage 4: Building or Developing the Product
* Stage 5: Testing the Product
* Stage 6: Deployment in the Market and Maintenance

SDLC Models

There are various software development life cycle models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development.

Following are the most important and popular SDLC models followed in the industry:

* **Waterfall Model**
* Iterative Model
* Spiral Model
* V-Model
* Big Bang Model

Agile model

As I said Set, List and Map are interfaces, which defines core contract e.g. a Set contract says that it can not contain duplicates. Based upon our knowledge of List, Set and Map let's compare them on different metrics.

**Duplicate Objects**

Main [difference between List and Set interface in Java](http://javarevisited.blogspot.com/2012/04/difference-between-list-and-set-in-java.html) is that List **allows duplicates** while Set doesn't allow duplicates. All implementation of Set honor this contract. Map  holds two object per Entry e.g. key and value and It may contain duplicate values but keys are always unique.

**Order**

Another key difference between List and Set is that List is an **ordered collection**, List's contract maintains insertion order or element. Set is an **unordered collection**, you get no guarantee on which order element will be stored. Though some of the Set implementation e.g. LinkedHashSet maintains order. Also SortedSet and SortedMap e.g. [TreeSet and TreeMap](http://java67.blogspot.com/2012/08/difference-between-treemap-and-treeset-java.html) maintains a sorting order, imposed by using [Comparator or Comparable](http://java67.blogspot.com/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html).

**Null elements**

List allows null elements and you can have many null objects in a List, because it also allowed duplicates. Set just allow one nullelement as there is no duplicate permitted while in Map you can have null values and at most one null key. worth noting is that [Hashtable doesn't allow null key or values](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) but HashMap allows null values and one null keys.  This is also the main difference between these two popular implementation of Map interface, aka [HashMap vs Hashtable](http://java67.blogspot.com/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html)

**Popular implementation**

List - ArrayList, LinkedList and Vector

Set - HashSet, TreeSet and LinkedHashSet

Map - HashMap, Hashtable and TreeMap

**When to use List, Set and Map in Java**

Based upon our understanding of *difference between Set, List and Map* we can now decide when to use List, Set or Map in Java.

1) If you need to access elements frequently by using **index**, than List is a way to go. Its implementation e.g. [ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html) provides faster access if you know index.

2) If you want to store elements and want them to maintain an **order** on which they are inserted into collection then go for List again, as [List](http://javarevisited.blogspot.com/2012/03/how-to-loop-arraylist-in-java-code.html) is an ordered collection and maintain insertion order.

3) If you want to create collection of unique elements and **don't want any duplicate** than choose any Set implementation e.g.[HashSet](http://javarevisited.blogspot.com/2012/06/hashset-in-java-10-examples-programs.html), LinkedHashSet or TreeSet. All Set implementation follow there general contract e.g. uniqueness but also add addition feature e.g. TreeSet is a SortedSet and elements stored on TreeSet can be sorted by using [Comparator or Comparable in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html). LinkedHashSet also maintains insertion order.

4) If you store data in form of key and value than Map is the way to go. You can choose from Hashtable, HashMap, TreeMapbased upon your subsequent need. In order to choose between first two see [difference between HashSet and HashMap in Java](http://javarevisited.blogspot.com/2011/09/difference-hashmap-vs-hashset-java.html).

That's all on **difference between Set, List and Map in Java**. All three are most fundamental interface of Java Collection framework and any Java developer should know there distinguish feature and given a situation should be able to pick right Collection class to use. It's also good to remember difference between there implementation e.g. [When to use ArrayList and LinkedList](http://java67.blogspot.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html) ,[HashMap vs Hashtable](http://javarevisited.blogspot.sg/2010/10/difference-between-hashmap-and.html) or [When to use Vector or ArrayList](http://java67.blogspot.com/2012/09/arraylist-vs-vector-in-java-interview.html) etc. Collection API is huge and it's difficult to know every bits and piece but at same time there is no excuse for not knowing fundamentals like Difference between Set, List and Map in Java.

# AOP with Spring Framework

One of the key components of Spring Framework is the **Aspect oriented programming (AOP)**framework. Aspect Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called **cross-cutting concerns** and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, and caching etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java and others.

Spring AOP module provides interceptors to intercept an application, for example, when a method is executed, you can add extra functionality before or after the method execution.

## AOP Terminologies:

**Difference between Comparator and Comparable**

The text below comes from [Comparator vs Comparable](http://www.digizol.com/2008/07/java-sorting-comparator-vs-comparable.html)

**Comparable**

A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface in order to be able to compare its instances.

**Comparator**

A comparator object is capable of comparing two different objects. The class is not comparing its instances, but some other class’s instances. This comparator class must implement the java.util.Comparator interface.

Difference between 1.5 and 1.6

the main difference in the language is that in Java 6 you can add @Override to methods which implement interfaces and in Java 5.0 you cannot.

If you have a problem in migrating its likely to be obscure edge case in a library somewhere. You won't find anything which is likely to cause a problem in the language.

Join method from Thread class is an important method and used to impose order on execution of multiple Threads. Concept of joining multiple threads is very popular on mutithreading interview question. Here is one of such question, “You have three threads T1, T2 and T3, How do you ensure that they finish in order T1, T2, T3 ?. This question illustrate power of join method on multithreaded programming. Unlike classical thread questions like difference between wait and sleep method or solving producer consumer problem in Java, This one is bit tricky. You can do this by using join method, by calling T1.join() from T2 and T2.join() from T3. In this case thread T1 will finish first, followed by T2 and T3. In this Java multithreading tutorial we will have a closer look on join method with a simple example. Idea is to illustrate how join method works in simple words. By the way from Java 5 onwards you can also use CountDownLatch and CyclicBarrier classes to implement scenarios like one thread is waiting for other threads to finish there task.

Read more: <http://javarevisited.blogspot.com/2013/02/how-to-join-multiple-threads-in-java-example-tutorial.html#ixzz2ssCA8ExM>

A wait can be "woken up" by another process calling notify on the monitor which is being waited on whereas a sleep cannot. Also a wait (and notify) must happen in a block synchronized on the monitor object whereas sleep does not:

Object mon = ...;

synchronized (mon) {

mon.wait();

}

At this point the currently executing thread waits and releases the monitor. Another thread may do

synchronized (mon) { mon.notify(); }

(On the same mon object) and the first thread (assuming it is the only thread waiting on the monitor) will wake up.

You can also call notifyAll if more than one thread is waiting on the monitor - this will wake all of them up. However, only one of the threads will be able to grab the monitor (remember that the wait is in a synchronized block) and carry on - the others will then be blocked until they can acquire the monitor's lock.

Another point is that you call wait on Object itself (i.e. you wait on an object's monitor) whereas you call sleep on Thread.

Yet another point is that you can get spurious wakeups from wait (i.e. the thread which is waiting resumes for no apparent reason). You should always wait whilst spinning on some condition as follows:

synchronized {

while (!condition) { mon.wait(); }

}

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class.  
...

Note: A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. **In effect, a static nested class is behaviorally a top-level class that has been nested in another top-level class for packaging convenience.**