## TreeSet vs TreeMap in Java

Now let's see some differences between TreeSet vs TreeMap in Java:

1) Major difference between TreeSet and TreeMap is that TreeSet implements Set interface while TreeMap implements Map interface in Java.

2) Second difference between TreeMap and TreeSet is the way they store objects.

TreeSet stores only one object while TreeMap uses two objects called key and Value.

Objects in TreeSet are sorted while keys in TreeMap remain in sorted Order based on keys.

3) Fourth difference is that duplicate objects are not allowed in TreeSet but duplicates values are allowed in TreeMap.

**Abstraction vs Encapsulation – Java OOPS**

1) First difference between Abstraction and Encapsulation is that, Abstraction is implemented in Java using [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and abstract class while Encapsulation is implemented using [private](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html), package-private and protected access modifier.

2) Encapsulation is also called data hiding.

### Difference between JIT and JVM in Java

### Main difference between JVM and JIT is there purpose, main goal of JVM is to provide platform independence while objective of JIT is to improve performance of JVM, by compiling more code into machine language.

### Another difference between JIT and JVM is that, JIT is part of JVM.

### At last, JVM is older concept than JIT. JIT actually invented to improve performance of JVM after years of initial release of Java, while Java virtual machine was part of initial Java release.

**Difference between Hashtable and HashMap in Java**  
1) First and most significant different between Hashtable and HashMap is that, [HashMap is not thread-safe](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)  while Hashtable is a [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) collection.  
2) Second important difference between Hashtable and HashMap is performance, since HashMap is not synchronized it perform better than Hashtable.

**What is Aggregation in java? (** composition,HAS-A relationship)

Aggregation is a special form of [**association**](http://beginnersbook.com/2013/05/association/).

It is also a relationship between two classes like association,

however its a **directional** association, which means it is strictly a **one way association.**

It represents a **Has-A** relationship.

For example consider two classes Student class and Address class. each student must have an address so the relationship between student and address is a Has-A relationship**. But if you consider its vice versa then it would not make sense as an Address doesn’t need to have a Student necessarily**. Below example shows this theoretical explanation in a sample java program.

**Student Has-A Address**

**class Address**

**{**

**int streetNum;**

**String city;**

**String state;**

**String country;**

**Address(int street, String c, String st, String coun)**

**{**

**this.streetNum=street;**

**this.city =c;**

**this.state = st;**

**this.country = coun;**

**}**

**}**

**class StudentClass**

**{**

**int rollNum;**

**String studentName;**

**Address studentAddr;**

**StudentClass(int roll, String name, Address addr){**

**this.rollNum=roll;**

**this.studentName=name;**

**this.studentAddr = addr;**

**}**

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

**Address ad = new Address(55, "Agra", "UP", "India");**

**StudentClass obj = new StudentClass(123, "Chaitanya", ad);**

**System.out.println(obj.rollNum);**

**System.out.println(obj.studentName);**

**System.out.println(obj.studentAddr.streetNum);**

**System.out.println(obj.studentAddr.city);**

**System.out.println(obj.studentAddr.state);**

**System.out.println(obj.studentAddr.country);**

**}**

**}**

**Output:**

**123**

**Chaitanya**

**55**

**Agra**

**UP**

**India**

The above example shows the **Aggregation** between Student and Address classes. we can see that in Student class I have used Address class to obtain student address. Its a typical example of Aggregation in Java.

**Why we need Aggregation?**

**To maintain code re-usability**. To understand this lets consider the above example again. Suppose there are two other classes College and Staff along with above two classes Student and Address.

In order to maintain Student’s address, College Address and Staff’s address we don’t need to use the same code again and again. We just have to use the reference of Address class while defining each of these classes like:

**Student Has-A Address (Has-a relationship between student and address)**

**College Has-A Address (Has-a relationship between college and address)**

**Staff Has-A Address (Has-a relationship between staff and address)**

Hence we can improve code re-usability by using Aggregation relationship.

***HAS-A relationship***

* Has-A means an instance of one class “has a” reference to an instance of another class or another instance of same class.
* It is also known as “composition” or “aggregation”.
* There is no specific keyword to implement HAS-A relationship but mostly we are depended upon “new” keyword.

**Composition :**

* Without existence of container object, if there is no chance of existence of contained objects then container and contained objects are said to be strongly associated and this strong association is known as *composition*.

Eg: A “university” has several “departments”. Without existence of “university” there is no chance for the “departments” to exist. Hence “university” and “departments” are strongly associated and this strong association is known as ***composition***.

**Aggregation**

* Without existence of container object, if there is a chance of existence of contained objects then container and contained objects are said to be loosely associated and this strong association is known as *aggregation.*

Eg: A  “department” has several “professors”. Without existence of “departments” there is good chance for the “professors” to exist. Hence “professors” and “department” are loosely associated and this loose association is known as *Aggregation*.

### *IS-A RELATIONSHIP*

**package com;**

**class Parent**

**{**

**public void m1()**

**{**

**System.*out*.println("from m1()");**

**}**

**}**

**class Child extends Parent**

**{**

**public void m2()**

**{**

**System.*out*.println("from m2()");**

**}**

**}**

**public class ISARelationship**

**{**

**public static void main(String[] args)**

**{**

**//case: 1**

**Parent p = new Parent ();**

**p.m1();**

**//p.m2();// Will get a compile time error as "Cannot find m2()"**

**//case: 2**

**Child c1 = new Child();**

**c1.m1();**

**c1.m2();**

**//case: 3**

**Parent s2 = new Child();**

**s2.m1();**

**//s2.m2();// Will get a compile time error as "Cannot find m2()"**

**//case : 4**

**//Child c = new Parent(); // Compiler error as "incompatible types"**

**}**

**}**

**SEE CASE: 2**

Whatever the parent class has is by default available to the child. Hence by using child reference, we can call both parent and child class methods.

**SEE CASE: 3**

Whenever the child class has, by default is not available to parent, hence on the parent class reference we can only call parent class methods but not child specific methods.

**SEE CASE: 3**

Parent class reference can be used to hold child class objects, but by using that reference we can call only parent class methods but not child specific methods.

**SEE CASE: 4**

We cannot use child class reference to hold parent class objects

**IS-A Relationship:**

* This refers to inheritance or implementation.
* Expressed using keyword “extends”.
* Main advantage is code reusability.

Entire java API is implemented based on inheritance.

Every java class extends from Object class which has most common and basic methods required for all java classes.

Hence we can say “Object” class is root class of all java methods.

A point to remember on inheritance..

 A java class cannot extend more than one class at a time so it wont provide support for multiple inheritance in classes, but it can extend more than one interface at a time so we can say java provides support for multiple inheritance w.r.t. interfaces.

## *What is encapsulation?*

The whole idea behind encapsulation is to hide the implementation details from users.

If a data member is private it means it can only be accessed within the same class.

No outside class can access private data member (variable) of other class.

However if we setup public getter and setter methods to update (for e.g. void setSSN(int ssn))and read (for e.g.  int getSSN()) the private data fields then the outside class can access those private data fields via public methods.

This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes.

That’s why encapsulation is known as **data hiding.**Lets see an example to understand this concept better.

**public class EncapsulationDemo{**

**private int ssn;**

**private String empName;**

**private int empAge;**

**//Getter and Setter methods**

**public int getEmpSSN(){**

**return ssn;**

**}**

**public String getEmpName(){**

**return empName;**

**}**

**public int getEmpAge(){**

**return empAge;**

**}**

**public void setEmpAge(int newValue){**

**empAge = newValue;**

**}**

**public void setEmpName(String newValue){**

**empName = newValue;**

**}**

**public void setEmpSSN(int newValue){**

**ssn = newValue;**

**}**

**}**

**public class EncapsTest{**

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

**EncapsulationDemo obj = new EncapsulationDemo();**

**obj.setEmpName("Mario");**

**obj.setEmpAge(32);**

**obj.setEmpSSN(112233);**

**System.out.println("Employee Name: " + obj.getEmpName());**

**System.out.println("Employee SSN: " + obj.getEmpSSN());**

**System.out.println("Employee Age: " + obj.getEmpAge());**

**}**

**}**

**Output:**

Employee Name: Mario

Employee SSN: 112233

Employee Age: 32

In above example all the three data members (or data fields) are private which cannot be accessed directly. These fields can be accessed via public methods only. Fields empName, ssn and empAge are made hidden data fields using encapsulation technique of OOPs.

#### Advantages of encapsulation:

1. It improves maintainability and flexibility and re-usability: for e.g. In the above code the implementation code of void setEmpName(String name) and String getEmpName() can be changed at any point of time. Since the implementation is purely hidden for outside classes they would still be accessing the private field empName using the same methods (setEmpName(String name) and getEmpName()). Hence the code can be maintained at any point of time without breaking the classes that uses the code. This improves the re-usability of the underlying class.
2. The fields can be made read-only (If we don’t define setter methods in the class) or write-only (If we don’t define the getter methods in the class). For e.g. If we have a field(or variable) which doesn’t need to change at any cost then we simply define the variable as private and instead of set and get both we just need to define the get method for that variable. Since the set method is not present there is no way an outside class can modify the value of that field.
3. User would not be knowing what is going on behind the scene. They would only be knowing that to update a field call set method and to read a field call get method but what these set and get methods are doing is purely hidden from them.

Encapsulation is also known as “**data Hiding**”.

1. Objects encapsulate data and implementation details. To the outside world, an object is a black box that exhibits a certain behavior.
2. The behavior of this object is what which is useful for the external world or other objects.
3. An object exposes its behavior by means of public methods or functions.
4. The set of functions an object exposes to other objects or external world acts as the interface of the object.

***Abstraction([http://www.visionsdeveloper.com/tutorial/java/java-abstraction.jsp#](http://www.visionsdeveloper.com/tutorial/java/java-abstraction.jsp),*** ***http://www.functionx.com/java/)***

**What is Abstraction?**  
Abstraction is process of **hiding the implementation details** and showing only the functionality.  
  
Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.  
  
**What is Abstract class in Java?**  
 A class that is declared as **abstract** is known as abstract class.  
  
Syntax:  
abstract class <class-name>{}  
  
An abstract class is something which is incomplete and you cannot create instance of abstract class.  
If you want to use it you need to make it complete or concrete by extending it.

A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended.

**What is Abstract method in Java?**  
  
A method that is declare as abstract and **does not have implementation** is known as abstract method.  
If you define abstract method than class must be abstract.  
  
Syntax:  
  
abstract return\_type method\_name ();  
  
An abstract method in Java doesn't have body, it’s just a declaration. In order to use abstract method you need to **override** that method in Subclass.  
  
Example 1 :( Without abstract method)

**class Employee extends Person {**

**private String empCode;**

**public String getEmpCode() {**

**return empCode;**

**}**

**public void setEmpCode(String empCode) {**

**this.empCode = empCode;**

**}**

**}**

**abstract class Person {**

**private String name;**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**}**

**public class Main{**

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

**//INSTIATING AN ABSTRACT CLASS GIVES COMPILE TIME ERROR**

**//Person p = new Person() ;**

**//THIS REFERENCE VARIABLE CAN ACESS ONLY THOSE METHOD WHICH ARE OVERRIDDEN**

**Person person = new Employee();**

**person.setName("Jatin Kansagara");**

**System.out.println(person.getName());**

**}**

**}**

Example 2: (with abstract method)

public class Main{

public static void main(String args[]){

TwoWheeler test = new Honda();

test.run();

}

}

abstract class TwoWheeler {

public abstract void run();

}

class Honda extends TwoWheeler{

public void run(){

System.out.println("Running..");

}

}

**When do you use abstraction?**  
When you know something needs to be there but not sure how exactly it should look like.

**Advantages of Abstraction**  
By using abstraction, we can s**eparate the things** that can be grouped to another type.  
  
Frequently changing properties and methods can be grouped to a separate type so that the main type need not undergo changes. This adds strength to the OOAD principle -"**Code should be open for Extension but closed for Modification**".  
  
Simplifies the representation of the **domain models**.  
  
  
**Summary:**  
-    Use abstraction if you know something needs to be in class but implementation of that varies.  
-     In Java you **cannot create instance of abstract class**, its compiler error.  
-    abstract is a keyword in java.  
-    A class automatically becomes abstract class when any of its method declared as abstract.  
-     abstract method doesn't have method body.  
-    **Variable cannot be made abstract**, its only behavior or methods which would be abstract.  
-    If a class extends an abstract class or interface it has to provide implementation to all its abstract method to be a concrete class. Alternatively this class can also be abstract.

PFA stands for Please Find Attached (email attachments)

--------------------------------------------------------------------

**Why threads are called light-weight ?**

Ans).

Threads are light-weight because they utilize minimum resources of the system.

This means they take less memory and less processor time

**What is the difference between single tasking and multitasking ?**

Ans).

Executing only one job at a time is called single tasking.

Executing several jobs at a time is called multi tasking.

In single tasking, the processor time is wasted, but in multi tasking, we can utilize the processor time in an optimum way.

**What is the difference between ‘extends Thread’ and ‘implements**

**Runnable’ ? Which one is advatageous ?**

Ans).

extends Thread and implements Runnable – both are functionally same. But when we write

extends Thread, there is no scope to extend another class, as multiple inheritance is not supported in

Java.

Class Myclass extends Thread, AnotherClass //invalid

If we write implements Runnable, then still there is scope to extend another class.

class Myclass extends AnotherClass implements Runnable //valid

This is definitely advantageous when the programmer wants to use threads and also wants to access

the features of another class.

**Which Java framework meets these requirements? What set of frameworks would be best suited to meet these requirements?**

The requirements are:

* oriented for web
* support for transactions
* support for creating RESTful web services
* support for security, levels of security
* integration with some kind of ORM framework like Hibernate
* ability to change front-end without need of making changes to back-end. Firstly, I want to develop a flex based front-end but if I've ever wanted to change it to HTML5 then I don't want to make changes to my back-end
* ready for cloud
* free to use for commercial purposes

To my mind comes Spring but are there any other alternatives meeting these requirements? What about if it is not necessary java? Do you know framework, set of frameworks combined together in other language which meet these requirements best?

**Frequentyl used plugins in java eclipse.**

 [Subversive - SVN Team Provider](http://marketplace.eclipse.org/content/subversive-svn-team-provider/metrics)

The Subversive project is aimed to integrate the Subversion (SVN) version control system with the Eclipse platform. Using the Subversive plug-in, you can work with projects stored in Subversion repositories directly from the Eclipse workbench in a way similar to work with other Eclipse version control providers, such as CVS and Git.

# Code Quality Plugins

## [FindBugs](http://findbugs.sourceforge.net/)

FindBugs plug-in is used with eclipse IDE and it is the software used to find bugs in Java programs. It’s free software licensed under Lesser GNU Public License. This is one of the most popular eclipse plugins and also available on Maven, Netbeans and IntelliJ.

## [Checkstyle](http://eclipse-cs.sourceforge.net/" \o "Checkstyle)

Checkstyle plug-ins can be integrated to IDE Eclipse and it ensures the Java code follows the coding standards.

# Framework Development

## [Spring Tool Suite](http://www.springsource.com/products/eclipse-downloads" \o "Spring Tool Suite)

Spring Tool Suite provides development environment and tools such as XML file wizards and graphical spring configuration editor, which can be used for easy development.

## [Spring IDE](http://springide.org/" \o "Spring IDE)

Spring IDE helps in developing Spring Application and provides interface to use the graphical editing framework. What’s more, Spring IDE display graphs of relationships and beans using configuration files. (Update site - http://springide.org/updatesite

## [Hibernator](http://hibernator.sourceforge.net/" \o "Hibernator)

Hibernator helps to create or update schema, run hibernate queries and create mapping documents. The user can synchronize a Java class to a related mapped file

## What is Framework?

A **Framework** provides readymade architecture and represents set of classes and interface

***When to use Interface and Abstract?***

---------------------------------------------------------

If we do not know anything about implementation just we have requirement

Specification then we should go for interface.

If we are talking about implementation but not completely (partial implementation) then we should go for Abstract class.

Class Loader in java

BootStrap class loader is responsible to load classes from BootStrap class path

i.e

BootStrap class path is nothing but location of rt.jar.

All core java api class are loaded by BootStrap class loader at run time.

Extention class loader

--------------

The classes which are present in ext folder jdk, jre, lib, ext folder

Application level class loader

Is responsible to load the classes from the application lever class path

i.e Environment path