1. What is copy of array .
2. How to improve application performance on server side.

https://blog.profitbricks.com/application-performance-management-tools/s

1. Why there is not a reverse method in String class in java, but available in StringBuffer and StringBuilder.

String is immutable, meaning it can't be changed. When you reverse a String, that each letter is switched on it's own, means it will always create the new object each times.

Let us see with example: This means that for instance Hello becomes as below

Example: elloH lloeH loleH olleH

and you end up with 4 new String objects on the heap. So, think if you have thousands of letter (word) of string or more, then how much object will be created.... it will be really a very expensive. So too much memory will be occupied. So because of this String class not having reverse() method.

1. Can constructor has return statement inside? No
2. Can method name can be same as class name. No
3. Can we run a java application in a system without JRE? No
4. If JDK got installed along with it JRE also got installed so if I removed JRE now then can I run my java appl. No
5. JVM is platform independent or platform dependent.

Java is platform independent (Java Virtual Machine) is platform dependent.

JVM converts byte code into machine language and execute the machine language instruction.

Java source code->java compiler->Java object code (class file). Class file is platform independent.

Java Object Code->JVM (Java Interpreter) ->Machine language.

1. Can we have class inside interface.

Yes, we can have class inside interface and those class only can use that class who implements the interface.

<http://way2java.com/java-lang/inner-classes-java-lang/class-inside-interface/>

**interface** Outer1 {

**public** **abstract** **void** show();

**class** Inner1 {

**public** **void** display() {

System.***out***.println("Hello 1");

}

}

}

**public** **class** Test **extends** Outer1.Inner1 **implements** Outer1 {

**public** **void** show() {

System.***out***.println("Hello 2");

}

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

Test t1 = **new** Test();

t1.display();

t1.show();

}

}

**Name the methods of Object Class?**

clone() – This method helps to create and return a copy of the object.  
equals() – This method helps to compare reference of two object.  
finalize() – Called by the garbage collector on an object  
getClass() – It helps to return the runtime class of an object.  
hashCode() – Helps to return a hash code value for the object.  
toString() – helps to return a string representation of the object.  
notify(), notifyAll(), and wait() – It helps to synchronize the activities of independently running threads in a program.

**What is Generic?**

Introduced in java 1.5, To provide type-safety and to resolve type-casting problems.

Bounded-types: We can bound the type parameter for a particular range by using ‘extends’ keyword such types are called bounded types.

**Class Test<T> {}:** As the type parameter we can pass any type and there are no restrictions, hence it is un-bounded type.

**Class Test<T extends Number> {}:** As the type parameter we can pass ‘T’ type or child class of number, hence it is bounded up to number.

**Class Test<T extends Number & Runnable> {}:** As the type parameter we can pass ‘T’ type or child class of Number and Runnable.

|  |  |
| --- | --- |
| Class Test<T>{} | Class Test< T extends X>  X: can be class or interface. |

Generic ‘?’ (Wild card): We can pass any argument.

1. Method m1(ArrayList<String> al): We can call this method by passing ArrayList of only String type.

Exe, al.add(“abc”);

al.add(10) CTE

1. Method m1(ArrayList<?> al): By passing ArrayList of any type.

Exe, al.add(“abc”);

al.add(10) ;

1. Method m1(ArrayList<? extends Number> al ): By passing Number type or it’s child class.
2. Method m1(ArrayList<? super X> al): By passing X type or it’s super class.

Various combination and example:

ArrayList<String> al = **new** ArrayList<String>()

ArrayList<?> al = **new** ArrayList<String>()

ArrayList<?> al = **new** ArrayList<Integer>()

ArrayList<? **extends** Number> al = **new** ArrayList<Integer>()

ArrayList<? **extends** Number> al = **new** ArrayList<String>(): CTE-Incompatible type

ArrayList<? **super** Number> al = **new** ArrayList<Object>()

ArrayList<? > al = **new** ArrayList< ? >(): CTE-Unexpected type

ArrayList<? > al = **new** ArrayList< ? **extends** Number>(): CTE-Unexpected type

**Find second highest salary from Employee table.**

**SELECT** **MAX**(salary) **FROM** Employee **WHERE** Salary NOT **IN** ( **SELECT** **Max**(Salary) **FROM** Employee);  
**SELECT** **MAX**(Salary) **FROM** Employee **WHERE** Salary **<** ( **SELECT** **Max**(Salary) **FROM** Employee);

**Find third highest salary from Employee table.**

**SELECT** **MAX**(salary) **FROM** Employee **WHERE** Salary NOT **IN** ( **SELECT** **Max**(Salary) **FROM** Employee NOT IN (**SELECT** **Max**(Salary) **FROM** Employee)); It will return only highest salary not third salary.

**SELECT** **MAX**(Salary) **FROM** Employee **WHERE** Salary **<** ( **SELECT** **Max**(Salary) **FROM** Employee < (( **SELECT** **Max**(Salary) **FROM** Employee));  
  
**Use Interfaces when…**

· You see that something in your design will change frequently.

· If various implementations only share method signatures then it is better to use Interfaces.

· you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to

just implement and make use of the methods defined in the interface.

**Use Abstract Class when…**

· If various implementations are of the same kind and use common behavior or status then abstract class is better to use.

· When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.

· Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

# [Difference between association, aggregation and composition](https://stackoverflow.com/questions/885937/difference-between-association-aggregation-and-composition)

* **Association:** is a relationship where all objects have their own lifecycle and there is no owner.

Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers, but there is no ownership between the objects, and both have their own lifecycle. Both can be created and deleted independently.

* **Aggregation:** is a specialized form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object.

Let's take an example of Department and teacher. A single teacher cannot belong to multiple departments, but if we delete the department, the teacher object will not be destroyed. We can think about it as a “has-a” relationship.

* **Composition:** is again specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.

Let's take again an example of relationship between House and Rooms. House can contain multiple rooms - there is no independent life of room and any room cannot belong to two different houses. If we delete the house - room will automatically be deleted.

Association - I have a relationship with an object. Foo uses Bar

public class Foo {

void Baz(Bar bar) {

}

};

Aggregation - I have an object which I've borrowed from someone else. When Foo dies, Bar may live on.

public class Foo {

private Bar bar;

Foo(Bar bar) {

this.bar = bar;

}

}

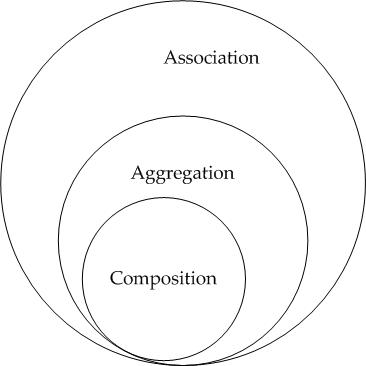
Composition - I own an object and I am responsible for its lifetime, when Foo dies, so does Bar

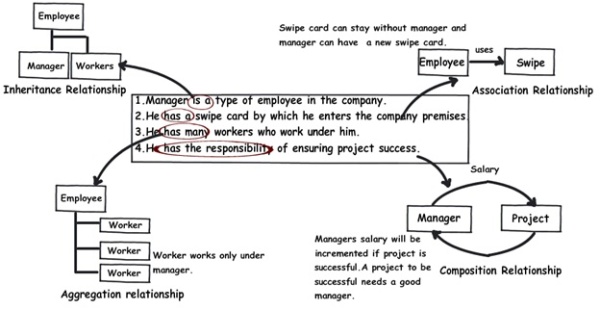
public class Foo {

private Bar bar = new Bar();

}

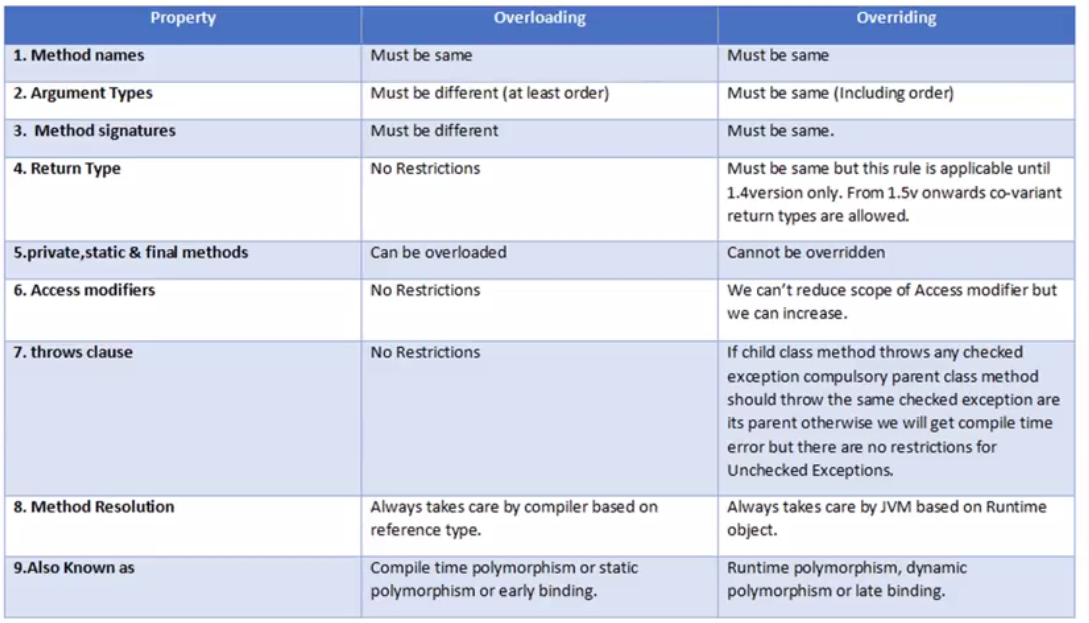
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Association** | **Aggregation** | **Composition** |
| **Owner** | No Owner | Single Owner | Single Owner |
| **Lifetime** | Have their own lifetime | Have their own lifetime | Owner’s lifetime |
| **Child Object** | Child objects all are independent | Child objects belong to a single parent | Child objects belong to a single parent |





## What is polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.



1. How to call default method of the interface.

**Default**: MyClass.super.test()

**Static**: MyClass.test()

1. How java verifies errors available in java codes.

**Verifier** (It checks whether line contains any logical or typing issues)

1. Read repeated value from Array and only print unique value, count repeated value from an array.
2. What is spring boot and its main class.

Spring Boot application’s main class is a class that contains a public static void main() method that starts up the Spring ApplicationContext. By default, if the main class isn’t explicitly specified, Spring will search for one in the classpath at compile time and fail to start if none or multiple of them are found.

Unlike in conventional Java applications, the main class discussed in this tutorial does not appear as the Main-Classmetadata property in META-INF/MANIFEST.MF of the resulting JAR or WAR file.

-Spring Boot expects the artifact’s Main-Class metadata property to be set to org.springframework.boot.loader.JarLauncher (or WarLauncher) which means that passing our main class directly to the java command line won’t start our Spring Boot application correctly.

1. Wait,sleep,yield,join

**yield**(): If a thread wants to pass its execution to give chance to remaining threads of same priority then we should go for yield()

**Join**(): If a thread wants to wait until completing of some other thread then we should go for join()

**Sleep**(): If a thread does not want to perform any operation for a particular amount of time, then it goes for sleep()

1. How to write exception in spring boot and custom exception (using annotation)

<https://www.baeldung.com/exception-handling-for-rest-with-spring>

1. How to configure datasource in spring boot to connect with db (configuration level)

@Repository

**public** **interface** UserRepository **extends** CrudRepository<User, Long> {}

@Repository

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {}

1. Your application flow from controller to db.
2. How to write custom exception in java (@interface, @target, @Retention)
3. Java 8 splitIterator (difference between Iterator,ListIterator,SplitIterator)

An Iterator always has an unknown size: you can traverse elements only via hasNext/next; a Spliterator can provide the size (thus improving other operations too internally); either an exact one via getExactSizeIfKnown or a approximate via estimateSize.

On the other hand, tryAdvance is what hasNext/next is from an Iterator, but it's a single method, much easier to reason about, IMO. Related to this, is forEachRemaining which in the default implementation delegates to tryAdvance

A Spliterator also is a "smarter" Iterator, via its internal properties like DISTINCT or SORTED

List<String> list = Arrays.asList("Apple", "Banana", "Orange");

Iterator<String> i = list.iterator();

i.next();

i.forEachRemaining(System.out::println);

#output

Banana

Orange

List<String> list = Arrays.asList("Apple", "Banana", "Orange");

Spliterator<String> s = list.spliterator();

s.tryAdvance(System.out::println);

System.out.println(" --- bulk traversal");

s.forEachRemaining(System.out::println);

System.out.println(" --- attempting tryAdvance again");

boolean b = s.tryAdvance(System.out::println);

System.out.println("Element exists: "+b);

Output:

Apple

--- bulk traversal

Banana

Orange

--- attempting tryAdvance again

Element exists: false

1. Contract between hashcode and equals method

## hashCode() Method

public int hashCode()

This method returns an **integer** value, which is referred to as the hash code value of an object. Every Object, at the time of creation assigned with a unique 32-bit, signed int value. This value is the hash code value of that object.

## General contract associated with hashCode() method

* The**hashCode()** method should return the same integer value for the same object for each calling of this method unless the value stored in the object is modified.
* If two objects are equal(according to **equals()** method) then the **hashCode()** method should return the same integer value for both the objects.
* But, it is not necessary that the**hashCode()** method will return the distinct result for the objects that are not equal (according to **equals()** method).

## equals() Method

public boolean equals(Object obj)

The **equals()** method of Object class checks the equality of the objects and accordingly it **returns** **true** or **false**. The default implementation, as provided by Object class, checks the equality of the objects on the basis if both references refer to the same object. It does not check the value or state of the objects. But we can override this method to provide own implementation to compare the state or value of the objects.

## General contract associated with equals() method

For any non-null reference variables a, b and c

* **a.equals(a)** should always return **true**.
* **a.equals(b)** should return **true** if and only if **b.equals(a)** returns true.
* If **a.equals(b)** returns **true** and **b.equals(c)** returns **true** then **a.equals(c)** should return **true**.
* Multiple calling of **a.equals(b)** should consistently return **true** or consistently return **false** If the value of the object is not modified for either object.
* **a.equals(null)** should return **false**.

# What is the difference between getter/setter methods and constructor in Java?

***getters() and setters():***

* actually getters() and setters available in POJO/Bean classes.
* The main reason used for getters() setters() in java classes is **To get the Java Encapsulation Mechanism.**
* In POJO/Bean classes we declare all attributes as a **private.** that means these class attributes can't use in other classes and packages, so in this, we can achieve Encapsulation.

***Constructors():***

* I think you know the definition of constructor, **The constructor is used for initializing the attributes giving our own values rather than storing the default values**
* We can say another way i.e **Constructor used for creating an object and setters used for changing the values inside object, getters() user for getting the values, this is only the main difference.**

1. Difference of equals methods in objects class and String equals methods, why overrided

## The equals() method of the String class

The equals() method of the String class is not same as the equals() method of the Object class. It is overridden, this method accepts a String value and compares it with the current object and returns true only if the character sequences in the both String objects are exactly same.

The equals() method is defined in [java.lang.Object class](https://www.knowprogram.com/java/object-class-java/) and **compare two objects based on their reference**. If both have the same reference then it returns true else it returns false.

public boolean equals(Object obj)

1. What is functional interface, how to right own functional interface, Does it can have one abstract method or multiple and what if it has multiple defaults methods and static methods

Functional interface has been introduced in java 8 and it contains only one abstract method but it can have any number of default method and static method.

*Static method*: are basically used for creating instances of utils, db etc where as

*Default method*: to achieve backward compatibility.

1. When we call third party api using webservice what we have to produce and consume.

# What is the difference between a web API and a web service? https://www.decipherzone.com/blog-detail/api-vs-web-services

Web Service:

1) It is a SOAP-based service and returns data as XML.

2) It only supports the HTTP protocol.

3) It is not open source but can be used by any client that understands XML.

5) It requires a SOAP protocol to receive and send data over the network, so it is not a light-weight architecture.

Web API:

1) A Web API is an HTTP based service and returns JSON or XML data by default.

2) It supports the HTTP protocol.

3) It can be hosted within an application or IIS.

4) It is open source and it can be used by any client that understands JSON or XML.

5) It has light-weight architecture and good for devices which have limited bandwidth, like mobile devices.

# or

1. All Web services are APIs but all APIs are not Web services.
2. Web services might not perform all the operations that an API would perform.
3. A Web service uses only three styles of use: SOAP, REST and XML-RPC for communication whereas API may use any style for communication.
4. A Web service always needs a network for its operation whereas an API doesn’t need a network for its operation
5. Consume Rest and soap api in java or in spring.

Using Spring boot or Spring:

RestTemplate restTemplate = **new** RestTemplate();

YourBean obj = restTemplate.getForObject(

"http://gturnquistquoters.cfapps.io/api/random", YourBean.**class**);

Using java:

**package** restclient;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**public** **class** NetClientGet {

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

**try** {

URL url = **new** URL(

"http://localhost:3002/RestWebserviceDemo/rest/json/product/dynamicData?size=5");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

**if** (conn.getResponseCode() != 200) {

**throw** **new** RuntimeException("Failed : HTTP Error code : " + conn.getResponseCode());

}

InputStreamReader in = **new** InputStreamReader(conn.getInputStream());

BufferedReader br = **new** BufferedReader(in);

String output;

**while** ((output = br.readLine()) != **null**) {

System.***out***.println(output);

}

conn.disconnect();

} **catch** (Exception e) {

System.***out***.println("Exception in NetClientGet:- " + e);

}

}}

1. Thread parallel processing without using synchronized keyword.
2. how to implement if else in sql querry.

IF ((SELECT COUNT(\*) FROM table1 WHERE project = 1) > 0)

SELECT product, price FROM table1 WHERE project = 1

ELSE IF ((SELECT COUNT(\*) FROM table1 WHERE project = 2) > 0)

SELECT product, price FROM table1 WHERE project = 2

ELSE IF ((SELECT COUNT(\*) FROM table1 WHERE project = 3) > 0)

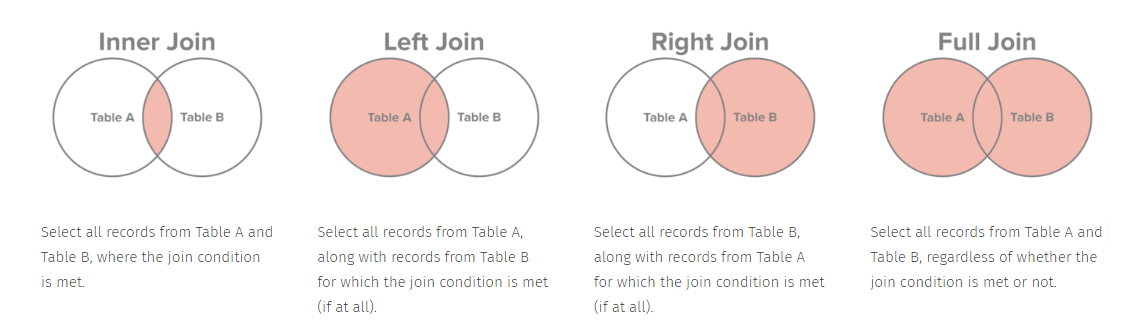
SELECT product, price FROM table1 WHERE project = 3

SELECT OrderID, Quantity, IF(Quantity>10, "MORE", "LESS") FROM OrderDetails;

1. Select top five results from the table.

**SELECT TOP 10 Id, ProductName, UnitPrice, Package FROM Product ORDER BY UnitPrice DESC**

1. **Joins**



SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

SELECT student.first\_name, student.last\_name, course.name FROM student

JOIN student\_course ON student.id = student\_course.student\_id

JOIN course course.id = student\_course.course\_id;

1. Implement synchronous vs asynchronous in java

<https://www.geeksforgeeks.org/asynchronous-synchronous-callbacks-java/>

ThreadPoolTaskExecutor executor = **new** ThreadPoolTaskExecutor();

@EnableAsync

@Async

Simply put – annotating a method of a bean with @Async will make it **execute in a separate thread** i.e. the caller will not wait for the completion of the called method.

1. How to implement job using spring, spring boot

@EnableScheduling

@Scheduled(fixedRate = 5000)

## PUT vs POST in REST WebService

PUT and POST can be used to create and update an entity, POST is usually preferred for creating and PUT is preferred for updating an existing entity.  
1) You should use POST to create new resources and PUT to update existing resources.  
2) Use PUT when you know the "id" of the object e.g. Order, Book, Employee  
3) Use POST when you need the server to be in control of URL generation of your resources.  
  
4) Examples  
PUT /items/1 update  
POST /items create  
**PUT** implies putting a resource - completely replacing whatever is available at the given URL with a different thing. By definition, a PUT is idempotent. Do it as many times as you like, and the result is the same. x=5 is idempotent. You can PUT a resource whether it previously exists, or not (eg, to Create, or to Update)!

**POST** updates a resource, adds a subsidiary resource, or causes a change. A POST is not idempotent, in the way that x++ is not idempotent.

**Distinguish between static loading and dynamic class loading?**

Static loading – Classes are loaded statically with operator “new”.Dynamic class loading – It is a technique for programmatically invoking the functions of a class loader at run time. The syntax isClass.forName (Test className);

**What is the difference between transient and volatile variable in Java?**

Transient: In Java, it is used to specify the variable is not being serialized. Serialization is a process of saving an object’s state in Java. When we want to persist and the object’s state by default, all instance variables in the object are stored. In some cases, if we want to avoid persisting few variables because we don’t have the necessity to transfer across the network. So, declare those variables as transient.

If the variable is confirmed as transient, then it will not be persisted. Transient keyword is used with that instance variable which will not participate in the serialization process. We cannot use static with a transient variable as they are part of the instance variable.

Volatile: Volatile keyword is used with only variable in Java and it guarantees that the value of the volatile variable will always be read from main memory and not from Thread’s local cache, it can be static.

<https://intellipaat.com/interview-question/java-interview-questions/>

# SOAP vs REST Web Services

|  |  |
| --- | --- |
| SOAP is a protocol. | REST is an architectural style. |
| SOAP stands for Simple Object Access Protocol. | REST stands for REpresentational State Transfer. |
| SOAP can't use REST because it is a protocol. | REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP. |
| SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

**What is Handler Mapping?**

Handler mapping is responsible for mapping incoming web requests to appropriate handler that can process the request. DispatcherServlet delegates the request to Handler Mapping to inspect the request and return appropriate HandlerExecutionChain that can handle the request.

**What are the different built in Handler Mapping implementations available in Spring**

* BeanNameUrlHandlerMapping
* SimpleUrlHandlerMapping
* ControllerClassNameHandlerMapping
* CommonsPathMapHandlerMapping
* DefaultAnnotationHandlerMapping
* RequestMappingHandlerMapping

**What is a ViewResolver?**

DispatcherServlet uses the ViewResolver for mapping and dispatching the request to an appropriate view. ModelAndView object contains the logical view name. In the previous example ‘DeleteEmployee’ is the logical view name. DispatcherServlet will map the logical view name to view using ViewResolver.

**What are the different built in ViewResolver implementations available in Spring 3?**

* UrlBasesViewResolver
* InternalResourceViewResolver
* ResourceBundleViewResolver
* BeanNameViewResolver
* XmlViewResolver

<http://answersz.com/spring-mvc-interview-questions-for-experienced/>

**How to execute stored procedure in hibernate.**

<sql­query name="selectAllEmployees\_SP" callable="true">

<**return** alias="emp" **class**="employee">

<return­property name="empid" column="EMP\_ID"/>

<return­property name="name" column="EMP\_NAME"/>

<return­property name="address" column="EMP\_ADDRESS"/>

{ ? = call selectAllEmployees() }

</**return**>

</sql­query>

Code:

SQLQuery sq = (SQLQuery)

session.getNamedQuery("selectAllEmployees\_SP");

List results = sq.list();

**Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**What are the benefits does HibernateTemplate provide?**

HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.

Common functions are simplified to single method calls. Sessions are automatically closed.

Exceptions are automatically caught and converted to runtime exceptions.

### [**can we use super and this keyword at same time?**](https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&ved=0ahUKEwjw-YSO-efTAhVIM48KHRuhCO0QFghoMA0&url=https%3A%2F%2Fwww.javatpoint.com%2Fq%2F5531%2Fcan-we-use-super-and-this-keyword-at-same-time%3F&usg=AFQjCNFRecGtEI08i34DnJTGEBZml_7b1Q&sig2=PG7Qe7wUHrltIikB0F7zDg&cad=rja)

this(...) will call another constructor in the same class whereas

super() will call a super constructor.

If there is no super() in a constructor the compiler will add one implicitly.

Thus if both were allowed you could end up calling the super constructor twice.

super()- calls the base class constructor whereas  
this()- calls current class constructor.

Both this() and super() are constructor calls.  
Constructor must always be the first statement. SO we can not have two statements as first statement, hence either we can call super() or we can call this() from the constructor, but not both.

class A {

public A() {

this( false );

}

public A(boolean someFlag) {

}

}

class B extends A {

public B() {

super();

}

public B( boolean someFlag ) {

super( someFlag );

}

public B ( int someNumber ) {

this(); //

}

}

**Why non-static variable cannot be referenced from a static context?  
  
“non-static variable cannot be referenced from a static context**" is biggest nemesis of some one who has just

started programming and that too in Java. Since [main method in java](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) is most popular method among all beginners and

they try to put program code there they face "*non-static variable cannot be referenced from a static context*" **compiler error**when they  try to access a non static member variable inside main in Java which is static. if you want to know

Note: We can access but we have to create a object then only we can call non static data under static.

public class **StaticTest** {

    private int count=0;

    public static void main(String args[]) throws IOException {

        count++; //**compiler error: non-static variable count cannot be referenced from a static context**

    }

}

1. **What is the difference between List<? super T> and List<? extends T> ?**

Use extends if you need to read from the collection (i.e. List<? extends A>;). This will ensure that the collection itself contains items which extends A. This is read-only because there is no way to determine the exact type to add to the collection (the parameter could be List<B> and there would be no way of ensure the type safety of an addition).

Use super  if you want to write to the collection (i.e. List<? super A>;). In this case, the collection can support addition of A types as we know the specified type of the collection is a super class of A. Therefore, A typed items can always be added to the collection.

**class** Holder<T> {

T v;

T get() {

**return** v;

}

**void** set(T n) {

v = n;

}

}

**class** A {

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

Holder<? **extends** Number> he = **new** Holder();

Holder<? **super** Number> hs = **new** Holder();

Integer i;

Number n;

Object o;

i = hs.get(); // <? super Number> cannot be converted to Integer

n = hs.get(); // <? super Number> cannot be converted to Number

o = hs.get();

hs.set(i);

hs.set(n);

hs.set(o); // Object cannot be converted to <? super Number>

i = he.get(); // <? extends Number> cannot be converted to Integer

n = he.get();

o = he.get();

he.set(i); // Integer cannot be converted to <? extends Number>

he.set(n); // Number cannot be converted to <? extends Number>

he.set(o); // Object cannot be converted to <? extends Number>

}

}

1. **Difference between ClassNotFoundException vs NoClassDefFoundError**

**ClassNotFoundException** is an exception that occurs when you try to load a class at run time using **Class.forName()** or **loadClass()** methods and mentioned classes are not found in the classpath.

**NoClassDefFoundError** is an error that occurs when a particular class is present at compile time, but was missing at run time.

|  |  |
| --- | --- |
| **ClassNotFoundException** | **NoClassDefFoundError** |
| It is an exception. It is of type java.lang.Exception. | It is an error. It is of type java.lang.Error. |
| It occurs when an application tries to load a class at run time which is not updated in the classpath. | It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at run time. |
| It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass(). | It is thrown by the Java Runtime System. |
| It occurs when classpath is not updated with required JAR files. | It occurs when required class definition is missing at runtime. |

|  |  |
| --- | --- |
| public class MainClass  {  public static void main(String[] args)  {  try  {  Class.forName("oracle.jdbc.driver  .OracleDriver");  }catch (ClassNotFoundException e)  {  e.printStackTrace();  }  }  } | class A  {  // some code  }  public class B  {  public static void main(String[] args)  {  A a = new A();  }  } |

# [What happens if String Pool runs out of memory?](http://stackoverflow.com/questions/30911800/what-happens-if-string-pool-runs-out-of-memory)

String Pool is a Collection of references that points to the String Objects.

When you write String = "hello" it creates that an String Object "hello" on the heap and will place an reference to this object in the String Literal Pool ( provided no Object is already there on the heap named "Hello")

*Point to Note* **"hello" is** **added to the constant pool of the corresponding class**. Therefore, it can be garbage collected only after the class is unloaded. So when the class is unloaded that Objects gets GC

What will happens?

String pooling is done through a process called string canonicalisation Which is a weakHashMap.This weakHashMap automatically clears out mapping when there is no other references to the keys or values. .ie the string will be garbage collected from the JVM.

Does it Grow in size?

**NO STRING POOL DOESNOT GROW IN SIZE- It's is Compile Time Constant**

How it Grow in Size ?

You need to specify **-XX:StringTableSize=N**, at the compile time where N is the string pool map size

At and at Last your question :

What happens if String Pool runs out of memory?

Simplest Answer : You get java.lang.OutOfMemoryError:java.lang.OutOfMemoryError: Java heap space from java 7 onwards . While java.lang.OutOfMemoryError: PermGen space in older version of java like 6

# String is a final class every time you user "+" and "=" new object will be created. For Variable assignments, if value already exists in pool then reference will be returned along with Object.

# [Difference between system.gc() and runtime.gc()](https://stackoverflow.com/questions/6197306/difference-between-system-gc-and-runtime-gc)

Both are same. System.gc() is effectively equivalent to Runtime.gc(). System.gc()internally calls Runtime.gc().

The only difference is System.gc() is a class method where as Runtime.gc() is an instance method. So, System.gc() is more convenient.

**System.gc():**

1: It is a class method(static method).

2: Non-Native method.(Code which doesn't directly interacts with Hardware and System Resources).

3: System.gc(), Internally calls Runtime.getRuntime().gc().

**Runtime.gc():**

1: Instance method.

2: Native method(A programming language which directly interacts with Hardware and System Resources.).

# [When will a string be garbage collected in java](https://stackoverflow.com/questions/18406703/when-will-a-string-be-garbage-collected-in-java)

# In Java, when an object has got no live reference, it is eligible for garbage collection. Now in case of a string, this is not the case because string will go into the string pool and JVM will keep the object alive for resuse. So that means a string once created will 'never' be garbage collected?

Whenever you call new in JAVA it create an object in heap but in case of String *literals*, it will go into the **String Constant Pool**.

Sample code:

String value = new String("ABC");

value = "xyz";

Now in the above sample code "ABC" and "xyz" string literals will go to the String Constant Pool and will not be garbage collected but finally value is referring to "xyz" from the String Constant Pool.

So basically there are 3 objects, 2 in the String Constant Pool and 1 in the heap.

at which point previously created String object will be garbage collected?

The object is created by new will be garbage collected once its scope/life is finished or there is no reference to access it. It's applicable similarly for all the objects including String as well.

Since the value reference will be pointed to the existing object with the value "xyz" within the string constant poll in the next line, so that previously created object using new in the heap is eligible for garbage collection but not "ABC" string literal that is still in the string constant pool.

# C:\Users\Priyadarshi\Desktop\71l1O.jpg

[What happens if two different objects have the same hashcode?](http://stackoverflow.com/questions/11195027/what-happens-if-two-different-objects-have-the-same-hashcode)

1) If two Objects are equal according to equal(), then calling the hashcode method on each of those two objects should produce same hashcode.

and other one is

2) It is not required that if two objects are unequal according to the equal(), then calling the hashcode method on each of the two objects must produce distinct values.

public class Test {

public static void main(String[] args) {

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

map.put(1, 11);

map.put(4, 11);

System.out.println(map.hashCode());

Map<Integer, Integer> map1 = new HashMap<Integer, Integer>();

map1.put(1, 11);

map1.put(4, 11);

System.out.println(map1.hashCode());

if (map.equals(map1)) {

System.out.println("equal ");

}

}

}

1. **How to throw and throws exception**

**Example 1: How to throw your own exception explicitly using throw keyword**

package beginnersbook.com;

class MyOwnException extends Exception {

public MyOwnException(String msg){

super(msg);

}

}

class EmployeeTest {

static void employeeAge(int age) throws MyOwnException{

if(age < 0)

throw new MyOwnException("Age can't be less than zero");

else

System.out.println("Input is valid!!");

}

public static void main(String[] args) {

try {

employeeAge(-2);

}

catch (MyOwnException e) {

e.printStackTrace();

}

}

}

**Example2: How to throw an already defined exception using throw keyword**

package beginnersbook.com;

class Exception2{

static int sum(int num1, int num2){

if (num1 == 0)

throw new ArithmeticException("First parameter is not valid");

else

System.out.println("Both parameters are correct!!");

return num1+num2;

}

public static void main(String args[]){

int res=sum(0,12);

System.out.println(res);

System.out.println("Continue Next statements");

}

}

Throw Throws

|  |  |
| --- | --- |
| Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. | Throws is followed by class. |
| Throw is used within the method. | Throws is used with the method signature. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions |

***throw*keyword:**  
The ***throw***keyword is used to explicitly throw an exception.  
We can throw either checked or **unchecked exception.**The ***throw***keyword is mainly used to throw custom exception.

**class** AgeValidator {

**static** **void** validateStudent(**int** age) {

**if** (age < 5)

**throw** **new** ArithmeticException("age to play more");

**else**

System.***out***.println("welcome to school");

}

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

*validateStudent*(4);

System.***out***.println("rest of the code...");

}

}

***Throws***: this is to be used when you are not using the try catch statement in your code but you know that this particular class is capable of throwing so and so exception(only checked exceptions). In this you do not use try catch block but write using the throw clause at appropriate point in you code and the exception is thrown to caller of the method and is handled by it.

# [Can constructors throw exceptions in Java?](http://stackoverflow.com/questions/1371369/can-constructors-throw-exceptions-in-java)

# Constructors are nothing more than special methods, and can throw exceptions like any other method. A constructor CAN throw any exception. But if any subclass constructor calls a super class constructor which throws an exception, then the subclass constructor must either catch the exception or throw it.

**public** **class** ConstructorTest {

**public** ConstructorTest() **throws** InterruptedException {

System.***out***.println("Preparing object....");

Thread.*sleep*(1000);

System.***out***.println("Object ready");

}

**public** **static** **void** main(String... args) {

**try** {

ConstructorTest test = **new** ConstructorTest();

} **catch** (InterruptedException e) {

System.***out***.println("Got interrupted...");

}

}

}

1. HashMap and CurncurrentHashMap
2. Fail Fast and Fail Safe in Java

**How to Synchronized any collections**

Map<Integer, String> syncHashMap =Collections.synchronizedMap(new HashMap<>());

List<String> list = Collections.synchronizedList(new ArrayList<String>());

Set<String> synset = Collections.synchronizedSet(set);

### **Fail Fast And Fail Safe Iterators In Java :**

### Iterators in java give us the facility to traverse over the Collection objects.

### Iterators returned by the Collection are either fail-fast in nature or fail-safe in nature.

### Fail-Fast iterators immediately throw ConcurrentModificationException if a collection is modified while iterating over it.

### Where as Fail-Safe iterators don’t throw any exceptions if a collection is modified while iterating over it. Because, they operate on the clone of the collection, not on the actual collection. Let’s see Fail-Fast and Fail-Safe Iterators in detail.

### **Fail-Fast Iterators In Java :**

Fail-Fast iterators, returned by most of the collection types, doesn’t tolerate any structural modifications to a collection while iterating over it. (Structural modifications means add, remove or updating an element in the collection) They throw ConcurrentModificationException if a collection is structurally modified while iteration is going on the collection. But, they don’t throw any exceptions if the collection is modified by the iterator’s own methods like remove().

**public** **class** FailFastIteratorExample

{

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

{

ArrayList<Integer> list = **new** ArrayList<Integer>();

list.add(1452);

list.add(6854);

list.add(8741);

list.add(6542);

list.add(3845);

//Getting an Iterator from list

Iterator<Integer> it = list.iterator();

**while** (it.hasNext())

{

Integer integer = (Integer) it.next();

list.add(8457); //This will throw ConcurrentModificationException

}

}

}

**How Fail-Fast Iterators Work?**

All Collection types maintain an internal array of objects ( Object[] ) to store the elements. Fail-Fast iterators directly fetch the elements from this array. They always consider that this internal array is not modified while iterating over its elements. To know whether the collection is modified or not, they use an internal flag called modCount which is updated each time a collection is modified. Every time when an Iterator calls the next() method, it checks the modCount. If it finds the modCount has been updated after this Iterator has been created, it throws ConcurrentModificationException.

### **Fail-Safe Iterators In Java :**

Fail-Safe iterators don’t throw any exceptions if the collection is modified while iterating over it. Because, they iterate on the clone of the collection not on the actual collection. So, any structural modifications done on the actual collection goes unnoticed by these iterators. But, these iterators have some drawbacks. One of them is that it is not always guaranteed that you will get up-to-date data while iterating. Because any modifications to collection after the iterator has been created is not updated in the iterator. One more disadvantage of these iterators is that there will be additional overhead of creating the copy of the collection in terms of both time and memory.

**public** **class** FailSafeIteratorExample

{

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

{

//Creating a ConcurrentHashMap

ConcurrentHashMap<String, Integer> map = **new** ConcurrentHashMap<String, Integer>();

map.put("ONE", 1);

map.put("TWO", 2);

map.put("THREE", 3);

map.put("FOUR", 4);

//Getting an Iterator from map

Iterator<String> it = map.keySet().iterator();

**while** (it.hasNext())

{

String key = (String) it.next();

System.*out*.println(key+" : "+map.get(key));

map.put("FIVE", 5); //This will not be reflected in the Iterator

}

}}

|  |  |
| --- | --- |
| **Fail-Fast Iterators** | **Fail-Safe Iterators** |
| Fail-Fast iterators doesn’t allow modifications of a collection while iterating over it. | Fail-Safe iterators allow modifications of a collection while iterating over it. |
| These iterators throw ConcurrentModificationException if a collection is modified while iterating over it. | These iterators don’t throw any exceptions if a collection is modified while iterating over it. |
| They use original collection to traverse over the elements of the collection. | They use copy of the original collection to traverse over the elements of the collection. |
| These iterators don’t require extra memory. | These iterators require extra memory to clone the collection. |
| Ex : Iterators returned by ArrayList, Vector, HashMap. | Ex : Iterator returned by ConcurrentHashMap. |

1. Race condition in java

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

When more than one thread try to access same resource without synchronization causes race condition.

So we can solve race condition by using either [synchronized block or synchronized method](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html).

**Race condition:**

**package** race.condition;

**class** TicketBooking **implements** Runnable

{

**int** ticketsAvailable=1;

**public** **void** run()

{

System.*out*.println("Waiting to book ticket for : "+Thread.*currentThread*().getName());

**if**(ticketsAvailable>0)

{

System.*out*.println("Booking ticket for : "+Thread.*currentThread*().getName());

//Let's say system takes some time in booking ticket (here we have taken 1 second time)

**Try**{

Thread.*sleep*(1000);

}**catch**(Exception e){

}

ticketsAvailable--;

System.*out*.println("Ticket BOOKED for : "+ Thread.*currentThread*().getName());

System.*out*.println("currently ticketsAvailable = "+ticketsAvailable);

} **else** {

System.*out*.println("Ticket NOT BOOKED for : "+ Thread.*currentThread*().getName());

}

}

}

/\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/

**public** **class** MyClass

{

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

{

TicketBooking obj=**new** TicketBooking();

Thread thread1=**new** Thread(obj,"Passenger1 Thread");

Thread thread2=**new** Thread(obj,"Passenger2 Thread");

thread1.start();

thread2.start();

}

}

/\*OUTPUT

Waiting to book ticket for : Passenger1 Thread

Waiting to book ticket for : Passenger2 Thread

Booking ticket for : Passenger1 Thread

Booking ticket for : Passenger2 Thread

**Ticket BOOKED for : Passenger1 Thread**

currently ticketsAvailable = 0

**Ticket BOOKED for : Passenger2 Thread**

**currently ticketsAvailable = -1**

\*/

**Resolving the issue using synchronizing the code**

**class** TicketBooking1 **implements** Runnable

{

**int** ticketsAvailable=1;

**public** **void** run()

{

System.*out*.println("Waiting to book ticket for : "+Thread.*currentThread*().getName());

**synchronized** (**this**)

{

**if**(ticketsAvailable>0)

{

System.*out*.println("Booking ticket for : "+Thread.*currentThread*().getName());

//Let's say system takes some time in booking ticket (here we have taken 1 second time)

**try**

{

Thread.*sleep*(1000);

}**catch**(Exception e)

{

}

ticketsAvailable--;

System.*out*.println("Ticket BOOKED for : "+ Thread.*currentThread*().getName());

System.*out*.println("currently ticketsAvailable = "+ticketsAvailable);

}

**else**

{

System.*out*.println("Ticket NOT BOOKED for : "+ Thread.*currentThread*().getName());

}

}//End synchronization block

}

}

/\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/

**public** **class** MyClass1

{

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

{

TicketBooking1 obj=**new** TicketBooking1();

Thread thread1=**new** Thread(obj,"Passenger1 Thread");

Thread thread2=**new** Thread(obj,"Passenger2 Thread");

thread1.start();

thread2.start();

}

}

/\*OUTPUT

Waiting to book ticket for : Passenger2 Thread

Waiting to book ticket for : Passenger1 Thread

Booking ticket for : Passenger2 Thread

**Ticket BOOKED for : Passenger2 Thread**

currently ticketsAvailable = 0

**Ticket NOT BOOKED for : Passenger1 Thread**

\*/

# Producer-Consumer solution using threads in Java

In computing, the producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue.

* The producer’s job is to generate data, put it into the buffer, and start again.
* At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time.

**Problem**  
To make sure that the producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.

**Solution**  
The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.  
An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

**Recommended Reading-** [Multithreading in JAVA](http://quiz.geeksforgeeks.org/multithreading-in-java/), [Synchronized in JAVA](http://quiz.geeksforgeeks.org/synchronized-in-java/) , [Inter-thread Communication](http://www.geeksforgeeks.org/inter-thread-communication-java/)

**Implementation of Producer Consumer Class**

* A **LinkedList list** – to store list of jobs in queue.
* **A Variable Capacity** – to check for if the list is full or not
* A mechanism to control the insertion and extraction from this list so that we do not insert into list if it is full or remove from it if it is empty.

*http://www.geeksforgeeks.org/producer-consumer-solution-using-threads-java/*

**import** java.util.LinkedList;

**public** **class** Threadexample

{

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

{

// Object of a class that has both produce() and consume() methods

**final** PC pc = **new** PC();

// Create producer thread

Thread t1 = **new** Thread(**new** Runnable()

{

@Override

**public** **void** run()

{

**try**

{

pc.produce();

}

**catch**(InterruptedException e)

{

e.printStackTrace();

}

}

});

// Create consumer thread

Thread t2 = **new** Thread(**new** Runnable()

{

@Override

**public** **void** run()

{

**try**

{

pc.consume();

}

**catch**(InterruptedException e)

{

e.printStackTrace();

}

}

});

// Start both threads

t1.start();

t2.start();

// t1 finishes before t2

t1.join();

t2.join();

}

// This class has a list, producer (adds items to list and consumber (removes items).

**public** **static** **class** PC

{

// Create a list shared by producer and consumer Size of list is 2.

LinkedList<Integer> list = **new** LinkedList<>();

**int** capacity = 2;

// Function called by producer thread

**public** **void** produce() **throws** InterruptedException

{

**int** value = 0;

**while** (**true**)

{

**synchronized** (**this**)

{

// producer thread waits while list is full

**while** (list.size()==capacity)

wait();

System.*out*.println("Producer produced-" + value);

// to insert the jobs in the list

list.add(value++);

// notifies the consumer thread that now it can start consuming

notify();

// makes the working of program easier to understand

Thread.*sleep*(1000);

}

}

}

// Function called by consumer thread

**public** **void** consume() **throws** InterruptedException

{

**while** (**true**)

{

**synchronized** (**this**)

{

// consumer thread waits while list is empty

**while** (list.size()==0)

wait();

//to retrive the ifrst job in the list

**int** val = list.removeFirst();

System.*out*.println("Consumer consumed-" + val);

// Wake up producer thread

notify();

// and sleep

Thread.*sleep*(1000);

}

}

}

}

}

**Important Points**

* In **PC class** (A class that has both produce and consume methods), a linked list of jobs and a capacity of the list is added to check that producer does not produce if the list is full.
* In **Producer class**, the value is initialized as 0.
  + Also, we have an infinite outer loop to insert values in the list. Inside this loop, we have a synchronized block so that only a producer or a consumer thread runs at a time.
  + An inner loop is there before adding the jobs to list that checks if the job list is full, the producer thread gives up the intrinsic lock on PC and goes on the waiting state.
  + If the list is empty, the control passes to below the loop and it adds a value in the list.
* In the **Consumer class**, we again have an infinite loop to extract a value from the list.
  + Inside, we also have an inner loop which checks if the list is empty.
  + If it is empty then we make the consumer thread give up the lock on PC and passes the control to producer thread for producing more jobs.
  + If the list is not empty, we go round the loop and removes an item from the list.
* In both the methods, we use notify at the end of all statements. The reason is simple, once you have something in list, you can have the consumer thread consume it, or if you have consumed something, you can have the producer produce something.
* sleep() at the end of both methods just make the output of program run in step wise manner and not display everything all at once so that you can see what actually is happening in the program.

# try-with-resources example in JDK 7

In Java, normally we open a file in a try block, and close the file in the finally block.

try{

//open file or resources

}catch(IOException){

//handle exception

}finally{

//close file or resources

}

Since JDK 7, a new “**try-with-resources**” approach is introduced. When a try block is end, it will close or release your opened file automatically.

**try**(open file or resource here)

{

//...

}

//after try block, file will close automatically.

## JDK 6 or early version

**public** **class** Example1 {

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

BufferedReader br = **null**;

**try** {

String line;

br = **new** BufferedReader(**new** FileReader("C:\\testing.txt"));

**while** ((line = br.readLine()) != **null**) {

System.*out*.println(line);

}

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (br != **null**)

br.close();

} **catch** (IOException ex) {

ex.printStackTrace();

}

}

}

}

## JDK 7

**public** **class** Example2 {

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

**try** (BufferedReader br = **new** BufferedReader(**new** FileReader (" C:\\ testing.txt"))) {

String line;

**while** ((line = br.readLine()) != **null**) {

System.*out*.println(line);

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**ORACLE:**

1. Find out 2nd max salary of and 2nd less salary of the employee.

2. What is JOIN and how many type of joins are there.

## **JOIN**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

## **Different Types of SQL JOINs**

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

**What is difference between stored procedures and application procedures, stored function and application function?**

**Answer:** Stored procedures are sub programs stored in the database and can be called & execute multiple times where in an application procedure is the one being used for a particular application same is the way for function

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs

1) Stored subprograms: These are coded and compiled from within the Oracle database via SQL Plus etc.  
2) Application subprograms: These are coded and compiled from within the Oracle development tools such as Oracle Developer.

Syntax:

Create procedure <procedure\_Name>

As

Begin

<SQL Statement>

End

CREATE PROCEDURE SelectCustomerstabledata

AS

SELECT \* FROM Testdb.Customers

GO

Create Procedure Employee details

As

Begin

Select \* from Employee

End

**What is the difference between peek(), poll() and remove() method of the Queue interface?**

Both poll() and remove() method is used to remove head object of the Queue.

The main difference lies when the Queue is empty().

If Queue is empty then poll() method will return null. While in similar case, remove() method will throw NoSuchElementException.

peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

# Difference between remove() method in iterator and collection,

# [Collection - Iterator.remove() vs Collection.remove()](http://stackoverflow.com/questions/14200489/collection-iterator-remove-vs-collection-remove)

Collection.remove()

you can get runtime errors (specifically ConcurrentModifcationException) because you're changing the state of the object used previously to construct the explicit series of calls necessary to complete the loop.

Iterator.remove()

you tell the runtime that you would like to change the underlying collection AND re-evaluate the explicit series of calls necessary to complete the loop.

The problem solved by Iterator.remove() is illustrated below:

List<Integer> l = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4));

for (int el : l) {

if (el < 3) {

l.remove(el);

}

}

This code is invalid since l.remove() is called during iteration over l.

The following is the correct way to write it:

Iterator<Integer> it = l.iterator();

while (it.hasNext()) {

int el = it.next();

if (el < 3) {

it.remove();

}

}

"Iterator.remove is the only safe way to modify a collection during iteration; the behavior is unspecified if the underlying collection is modified in any other way while the iteration is in progress."

# [Program to print prime numbers in java](http://www.instanceofjava.com/2014/12/program-to-print-prime-numbers-in-java.html)

package com.instaceofjava;  
public class primenumbers {  
 public static void main(String[] args) {  
 int num = 50;  
 int count = 0;  
 for (int i = 2; i <= num; i++) {  
 count = 0;  
 for (int j = 2; j <= i / 2; j++) {  
 if (i % j == 0) {  
 count++;  
 break;  
 }  
 }  
 if (count == 0) {  
 System.out.println(i);  
 }  
 }  
 }  
}  
  
package com.instaceofjava;  
public class ReverseString {  
 public static void main(String[] args) {  
 String str = "Hello world";  
 String revstring = "";  
 for (int i = str.length() - 1; i >= 0; --i) {  
 revstring += str.charAt(i);  
 }  
 System.out.println(revstring);  
 }  
}

### **Palindrome program in java using for loop**

package com.instaceofjava;  
public class PalindromeDemo {  
 public static void main(String[] args) {  
 String str = "MADAM";  
 String revstring = "";  
 for (int i = str.length() - 1; i >= 0; --i) {  
 revstring += str.charAt(i);  
 }  
 System.out.println(revstring);  
 if (revstring.equalsIgnoreCase(str)) {  
 System.out.println("The string is Palindrome");  
 } else {  
 System.out.println("Not Palindrome");  
 }  
 }  
}

# [Reverse number program in java](http://www.instanceofjava.com/2014/12/reverse-number-program-in-java.html)

package com.instaceofjavaforus;  
public class Reversenum {  
 public static void main(String[] args) {  
 int rev = 0;  
 int num = 1234;  
 while (num > 0) {  
 int rem = num % 10;  
 rev = rem + (rev \* 10);  
 num = num / 10;  
 }  
 }  
}

# [Fibonacci series without using recursion in java](http://www.instanceofjava.com/2014/12/fibonacci-series-without-using.html)

package com.instaceofjava;  
public class Fibanaciwithoutrecursive {  
 public static void main(String[] args) {  
 int n1 = 0;  
 int n2 = 1;  
 System.out.println(n1);  
 System.out.println(n2);  
 for (int i = 0; i <= 100; i++) {  
 int sum = n1 + n2;  
 if (sum <= 100) {  
 n1 = n2;  
 n2 = sum;  
 System.out.println(sum);  
 }  
 }  
 }  
}

# [How to find largest element in an array with index and value using array?](http://www.instanceofjava.com/2014/12/how-to-find-largest-element-in-array.html)

package com.instanceofjava;  
public class Array {  
 public static void main(String[] args) {  
 int arr[] = {1, 120, 56, 78, 87};  
 int largest = arr[0];  
 int smallest = arr[0];  
 int small = 0;  
 int index = 0;  
 for (int i = 1; i < arr.length; i++) {  
 if (arr[i] > largest) {  
 largest = arr[i];  
 index = i;  
 } else if (smallest > arr[i]) {  
 smallest = arr[i];  
 small = i;  
 }  
 }  
 System.out.println(largest);  
 System.out.println(index);  
 System.out.println(smallest);  
 System.out.println(small);  
 }  
}

# [Check armstrong number or not](http://www.instanceofjava.com/2014/12/check-armstrong-number-or-not.html)

package com.instanceofjavaTutorial;  
import java.util.Scanner;  
public class ArmstrongNumber {  
 public static void main(String args[]) {  
 int n, sum = 0, temp, r;  
 Scanner in = new Scanner(System.in);  
 System.out.println("Enter a number to check if it is an Armstrong number or not");  
 n = in.nextInt();  
 temp = n;  
 while (temp != 0) {  
 r = temp % 10;  
 sum = sum + r \* r \* r;  
 temp = temp / 10;  
 }  
 if (n == sum)  
 System.out.println(n + "is an Armstrong number.");  
 else  
 System.out.println(n + " is not an Armstrong number.");  
 }  
}

Interview questions starts with "**What is deadlock?**"  
Answer is simple, when two or more threads are waiting for each other to release lock and get stuck for infinite time, situation is called deadlock . It will only happen in case of multitasking.

## How do you detect deadlock in Java ?

Though this could have many answers , my version is first I would look the code if I see nested synchronized block or calling one synchronized method from other or trying to get lock on different object then there is good chance of deadlock if developer is not very careful.  
  
Other way is to find it when you actually get locked while running the application , try to take thread dump , in Linux you can do this by command **"kill -3"** , this will print status of all the thread in application log file and you can see which thread is locked on which object.

Other way is to use **jconsole**, it will show you exactly which threads are get locked and on which object.

## Write a Java program which will result in deadlock?

Once you answer this , they may ask you to **write code which will result in deadlock ?**  
here is one of my version

/\*\*Java program to create a deadlock by imposing circular wait.\*/

public class DeadLockDemo {

//This method request two locks, first String and then Integer

public void method1() {

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

}

}

}

/\*

\* This method also requests same two lock but in exactly

\* Opposite order i.e. first Integer and then String.

\* This creates potential deadlock, if one thread holds String lock

\* and other holds Integer lock and they wait for each other, forever.

\*/

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

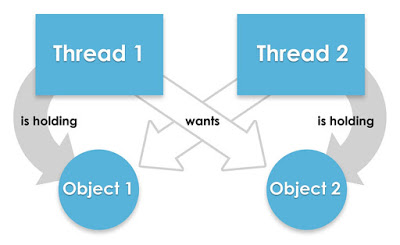
}

}

}

}

If method1() and method2() both will be called by two or many threads , there is a good chance of deadlock because if thread 1 acquires lock on Sting object while executing method1() and thread 2 acquires lock on Integer object while executing method2() both will be waiting for each other to release lock on Integer and String to proceed further which will never happen.  
  
This diagram exactly demonstrate our program, where one thread holds lock on one object and waiting for other object lock which is held by other thread.

[](http://2.bp.blogspot.com/-63RZ-BTlAFs/VfGeHMnGdFI/AAAAAAAADuw/gwqtrVliMsM/s1600/Deadlock+of+Threads.jpg)

## How to avoid deadlock in Java?

Now interviewer comes to final part, one of the most important in my view; *How do you fix deadlock?*or **How to avoid deadlock in Java?**  
  
If you have looked above code carefully then you may have figured out that real reason for deadlock is not multiple threads but ***the way they are requesting lock*** , if you provide an ordered access then problem will be resolved , here is my fixed version, which avoids deadlock by avoiding circular wait with no preemption.

public class DeadLockFixed {

/\*\*

\* Both method are now requesting lock in same order, first Integer and then String.

\* You could have also done reverse e.g. first String and then Integer,

\* both will solve the problem, as long as both method are requesting lock

\* in consistent order.

\*/

public void method1() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

}

# Now there would not be any deadlock because both methods are accessing lock on Integer and String class literal in same order. So, if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock, same way thread A will not be blocked even if thread B holds String lock because now thread B will not expect thread A to release Integer lock to proceed further. How to sort Object in Java - Comparator and Comparable Example

**Java Object Sorting Example**

Comparator and Comparable interface along with Collections.sort() method are used to sort the list of object in Java. compare() and compareTo() method of Comparator and Comparable interface provides comparison logic needed for sorting objects.

[compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) is used to provide Object's natural order sorting and

compare() method is used to sort Object with any arbitrary field. Almost all value classes in Java library e.g. [String](http://javarevisited.blogspot.sg/2012/08/how-to-format-string-in-java-printf.html), [Integer](http://javarevisited.blogspot.sg/2011/08/convert-string-to-integer-to-string.html), [Double](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html), BigDecimal implement compareTo() to specify their natural sorting order. While overriding compareTo method String is sorted lexicographically and Integers are sorted numerically. Just beware that it must inconsistent with equals method i.e. two objects which are equal by [equals method in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html), compareTo() method must return zero for them.  Anyway,  sorting standard value Object is not a problem for many Java programmer but some of them really struggle when it comes to [sorting custom Objects](http://javarevisited.blogspot.sg/2011/06/comparator-and-comparable-in-java.html) or domain Objects. In this Java sorting tutorial, we will create a custom object and sort list of Object in ascending and descending order.

## **Sorting list of Object in ascending and descending Order**

1) Create Order object as custom or domain object

2) Implement [Comparable and Comparator](http://javarevisited.blogspot.in/2011/06/comparator-and-comparable-in-java.html) interface to define sorting logic

3) Sort list of Object using Collections.sort method

**package** test;  
**import** java.util.ArrayList;  
**import** java.util.Collections;  
**import** java.util.Comparator;  
**import** java.util.List;  
  
/\*\*  
 \*  
 \* **Java program to test Object sorting in Java**. This Java program  
 \* test Comparable and Comparator implementation provided by Order  
 \* class by sorting list of Order object in ascending and descending order.  
 \* Both in natural order using Comparable and custom Order using Comparator in Java  
 \*  
 \* @author http://java67.blogspot.com  
 \*/  
**public** **class** ObjectSortingExample {  
  
    **public** **static** **void** main([**String**](http://java67.blogspot.sg/2012/08/difference-between-string-and-stringbuffer-in-java.html) args[]) {  
        
        *//Creating Order object to demonstrate Sorting of Object in Java*  
        Order ord1 = **new** Order(101,2000, "Sony");  
        Order ord2 = **new** Order(102,4000, "Hitachi");  
        Order ord3 = **new** Order(103,6000, "Philips");  
        
        *//putting Objects into Collection to sort*  
        [**List**](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html)<Order> orders = **new** **ArrayList**<Order>();  
        orders.add(ord3);  
        orders.add(ord1);  
        orders.add(ord2);  
        
        *//printing unsorted collection*  
        **System**.out.println("Unsorted Collection : " + orders);  
        
        *//Sorting Order Object on natural order - ascending*  
        **Collections**.sort(orders);  
        
        *//printing sorted collection*  
        **System**.out.println("List of Order object sorted in natural order : " + orders);  
        
        *// Sorting object in descending order in Java*  
        **Collections**.sort(orders, **Collections**.reverseOrder());  
        **System**.out.println("List of object sorted in descending order : " + orders);  
                
        *//Sorting object using Comparator in Java*  
        **Collections**.sort(orders, **new** Order.OrderByAmount());  
        **System**.out.println("List of Order object sorted using Comparator - amount : " + orders);  
        
        *// Comparator sorting Example - Sorting based on customer*  
        **Collections**.sort(orders, **new** Order.OrderByCustomer());  
        **System**.out.println("Collection of Orders sorted using Comparator - by customer : " + orders);  
    }  
}  
  
*/\*  
 \* Order class is a domain object which implements  
 \* Comparable interface to provide sorting on the natural order.  
 \* Order also provides couple of custom Comparators to  
 \* sort object based upon amount and customer  
 \*/*  
**class** Order **implements** **Comparable**<Order> {  
  
    **private** **int** orderId;  
    **private** **int** amount;  
    **private** **String** customer;  
  
    */\*  
     \* Comparator implementation to Sort Order object based on Amount  
     \*/*  
    **public** **static** **class** OrderByAmount **implements** **Comparator**<Order> {  
  
        @**Override**  
        **public** **int** compare(Order o1, Order o2) {  
            **return** o1.amount > o2.amount ? 1 : (o1.amount < o2.amount ? -1 : 0);  
        }  
    }  
  
    */\*  
     \* Anohter implementation or Comparator interface to sort list of Order object  
     \* based upon customer name.  
     \*/*  
    **public** **static** **class** OrderByCustomer **implements** **Comparator**<Order> {  
  
        @**Override**  
        **public** **int** compare(Order o1, Order o2) {  
            **return** o1.customer.compareTo(o2.customer);  
        }  
    }  
  
    **public** Order(**int** orderId, **int** amount, **String** customer) {  
        **this**.orderId = orderId;  
        **this**.amount = amount;  
        **this**.customer = customer;  
    }  
  
    
    **public** **int** getAmount() {**return** amount; }  
    **public** **void** setAmount(**int** amount) {**this**.amount = amount;}  
  
    **public** **String** getCustomer() {**return** customer;}  
    **public** **void** setCustomer(**String** customer) {**this**.customer = customer;}  
  
    **public** **int** getOrderId() {**return** orderId;}  
    **public** **void** setOrderId(**int** orderId) {**this**.orderId = orderId;}  
  
    */\*  
     \* Sorting on orderId is natural sorting for Order.  
     \*/*  
    @**Override**  
    **public** **int** compareTo(Order o) {  
        **return** **this**.orderId > o.orderId ? 1 : (**this**.orderId < o.orderId ? -1 : 0);  
    }  
    
    */\*  
     \* implementing toString method to print orderId of Order  
     \*/*  
    @**Override**  
    **public** **String** toString(){  
        **return** **String**.valueOf(orderId);  
    }  
}  
  
**Output**  
Unsorted **Collection** : [103, 101, 102]  
**List** of Order object sorted in natural order : [101, 102, 103]  
**List** of object sorted in descending order : [103, 102, 101]  
**List** of Order object sorted using **Comparator** - amount : [101, 102, 103]  
**Collection** of Orders sorted using **Comparator** - by customer : [102, 103, 101]

**Important points to note:**

1) If you implement Comparable interface and override compareTo() method it must be consistent with equals() method i.e. for equal object by equals() method compareTo() must return zero. failing to so will affect contract of SortedSet e.g. [TreeSet](http://java67.blogspot.sg/2012/08/difference-between-treemap-and-treeset-java.html) and SortedMap like [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html), which uses compareTo() method for checking equality

2) Some programmer use Integer subtraction to implement compareTo() in Java, which can cause overflow issue if both integers are not positive. See [How compareTo works in Java](http://javarevisited.blogspot.in/2011/11/how-to-override-compareto-method-in.html) for more details.

3) This example of sorting Object in Java also shows a good example of Where to use [nested static class in Java](http://java67.blogspot.sg/2012/10/nested-class-java-static-vs-non-static-inner.html). In this example We have created custom Comparator as a static inner class So that they can access properties of Order for comparison and also they are only used in the context of Order class.

4) Don't forget to revise [Difference between Comparator and Comparable in Java](http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html), which is one of the most asked Java questions on Interviews.

5) Remember to use Collections.reverseOrder() comparator for sorting Object in reverse order or [descending order](http://javarevisited.blogspot.sg/2012/01/how-to-sort-arraylist-in-java-example.html), as shown in this example.

6) Use [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html) while implementing Comparator and Comparable interface, that prevents error of accidentally [overloading](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) compareTo() and compare() method instead of [overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) it because both of these methods accept Object as a parameter. By using Generics and @Override annotation we effectively remove that subtle error.

7) This Object Sorting Example in Java also teaches us [Why you should override toString() in Java](http://javarevisited.blogspot.sg/2012/09/override-tostring-method-java-tips-example-code.html). If you are going to store your Object in Collection like List, Set or Map than printing Collection will call toString() method of each stored Object. By providing readable String format you can see What is stored in a collection in logs. This is also a useful [logging tip in Java](http://javarevisited.blogspot.sg/2011/05/top-10-tips-on-logging-in-java.html).

Write a program to find out duplicate characters in a string. –

package com.java2novice.algos;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

public class DuplicateCharsInString {

    public void findDuplicateChars(String str){

        Map<Character, Integer> dupMap = new HashMap<Character, Integer>();

        char[] chrs = str.toCharArray();

        for(Character ch:chrs){

            if(dupMap.containsKey(ch)){

                dupMap.put(ch, dupMap.get(ch)+1);

            } else {

                dupMap.put(ch, 1);

            }

        }

        Set<Character> keys = dupMap.keySet();

        for(Character ch:keys){

            if(dupMap.get(ch) > 1){

                System.out.println(ch+"--->"+dupMap.get(ch));

            }

        }

    }

    public static void main(String a[]){

        DuplicateCharsInString dcs = new DuplicateCharsInString();

        dcs.findDuplicateChars("Java2Novice");

    }

}

- See more at: <http://www.java2novice.com/java-interview-programs/duplicate-string->

# How to get distinct elements from an array by avoiding duplicate elements?

package com.java2novice.algos;  
  
public class MyDisticntElements {  
 public static void printDistinctElements(int[] arr) {  
 for (int i = 0; i < arr.length; i++) {  
 boolean isDistinct = false;  
 for (int j = 0; j < i; j++) {  
 if (arr[i] == arr[j]) {  
 isDistinct = true;  
 break;  
 }  
 }  
 if (!isDistinct) {  
 System.out.print(arr[i] + " ");  
 }  
 }  
 }  
  
 public static void main(String a[]) {  
 int[] nums = {5, 2, 7, 2, 4, 7, 8, 2, 3};  
 MyDisticntElements.*printDistinctElements*(nums);  
 }  
}

# How will you make a custom class object as Hashmap Key

There are two methods that a class needs to override to make objects of that class work as hash map keys.

1. public int hashCode();
2. public boolean equals(Object o);

But you must obey the contract between hashCode and equals method while overriding them in your new Class.

### **Contract for hashCode and equals method**

1. The equals() method must return true if the fields of the current object equal those of the object passed in, else return false. By “equal”, we generally mean that primitive fields match via the == operator, and objects are either both null or both non-null and match via the equals() method. Note two important constraints on equals():
2. if x.equals(y) returns true, then the hash codes of x and y must be identical;
3. it must be reflexive and transitive: that is, x.equals(y) must return the same value as y.equals(x), and if x.equals(y) and y.equals(z), then x.equals(z) must also be true (see below for what this actually means in real terms!).

If you do not obey the Contract Rules between hashCode and equals method then unpredictable results can happen, for example you may not be able to get newly inserted item back from hashmap or in worst case a wrong object could be returned for the given key.

**public** **class** DefinedObjectAsAKeyInHashMap {

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

HashMap<Price, String> hm = **new** HashMap<Price, String>();

hm.put(**new** Price("Banana", 20), "Banana");

hm.put(**new** Price("Apple", 40), "Apple");

hm.put(**new** Price("Orange", 30), "Orange");

*printMap*(hm);

Price key = **new** Price("Banana", 20);

System.*out*.println("Does key available? " + hm.containsKey(key));

}

**public** **static** **void** printMap(HashMap<Price, String> map) {

Set<Price> keys = map.keySet();

**for** (Price p : keys) {

System.*out*.println(p + "==>" + map.get(p));

}

}

}

**class** Price {

**private** String item;

**private** **int** price;

**public** Price(String itm, **int** pr) {

**this**.item = itm;

**this**.price = pr;

}

**public** **int** hashCode() {

**int** hashcode = 0;

hashcode = price \* 20;

hashcode += item.hashCode();

**return** hashcode;

}

**public** **boolean** equals(Object obj) {

System.*out*.println("In equals");

**if** (obj **instanceof** Price) {

Price pp = (Price) obj;

**return** (pp.item.equals(**this**.item) && pp.price == **this**.price);

} **else** {

**return** **false**;

}

}

**public** String getItem() {

**return** item;

}

**public** **void** setItem(String item) {

**this**.item = item;

}

**public** **int** getPrice() {

**return** price;

}

**public** **void** setPrice(**int** price) {

**this**.price = price;

}

**public** String toString() {

**return** "item: " + item + " price: " + price;

}

}

**Difference between Struts and Spring.**

we use both Spring and Struts, they are not an either/or proposition. Struts is the front end, ActionForms and Actions. Actions will call the Spring Service which has the Spring DAO injected into it. Spring makes unit testing sooooo much easier, you can inject your fake dao and not have to test with the database at all if you do not want to. we use it with Hibernate and my code is quite small as Spring handles expceptions quite well and you are relieved of all the Hibernate repetitive code.   
  
Differences:

1) Struts is follows OOP and spring is based on AOP.  
1) Struts is a web framework while Spring is not.   
2) Spring is a Layered Architecture while Struts is not.   
3) Struts is heavy weight while Spring is light weight.   
4) Struts supports tag Library while Spring does not.   
5) Spring is loosely coupled while Struts is tightly coupled.   
6) Spring provides easy integration with ORM technologies while in struts, we need to do coding manually.   
7) Struts easily integrate with other client side technologies. It is not easy in case of spring.

1. Spring don't have any action from, it bind the http form values directly into pojo. Instead of initializing the form bean spring directly initialize the domain object.   
2. ActionForward in struts is replace with the ModelAndView in Spring. Model component contain the business object to be displayed via view component.   
3. Unlike Struts Spring don't provide any separate tag library.   
4. What Action class do in struts, Controller does in Spring. And action in Struts is a Abstract class but Controller in Spring is an interface, This is very good advantage of the spring.   
5.Struts implement MVC Design Patten where as Spring implements IOC Design Pattren and addreses AOP Cross cutting concerns.   
6.Struts is heavy weight where as Spring is light weight framework.