Java/J2EE with related framework (FAQ for Interview questions and answers)

1**) How to mark the fields not to override?**

We can create normal methods.

**2) Can we override static methods in Java?**We can declare static methods with the same signature in a subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.  
If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

**3) What are the OOPS concept?**

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding, polymorphism** etc.

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

**Object**

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

**Class**

**Collection of objects** is called class. It is a logical entity.

**Inheritance**

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**



When **one task is performed by different ways** i.e. known as polymorphism. For example: to convenes the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

**Abstraction**

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

**Encapsulation**



**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Advantage of OOPs over Procedure-oriented programming language**

|  |
| --- |
| 1) OOPs makes development and maintenance easier where as in Procedure-oriented programming language, it is not easy to manage if code grows as the project size grows. |
| 2) OOPs provides data hiding, whereas in Procedure-oriented programming language a global data can be accessed from anywhere. |
| 3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of a real world problem if we are using the Object-Oriented Programming language. |

|  |  |
| --- | --- |
| Global Data | Object Data |

**What is the difference between object-oriented programming language and object-based programming language?**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.  **4) Tomcat 8 Advantage?**  Support for Java Servlet 3.1, JavaServer Pages 2.3, Java Unified Expression Language 3.0 and Java WebSocket 1.0.  The default connector implementation is now the Java non-blocking implementation (NIO) for both HTTP and AJP.  A new resources implementation that replaces Aliases, VirtualLoader, VirtualDirContext, JAR resources and external repositories with a single, consistent approach for configuring additional web application resources. The new resources implementation can also be used to implement overlays (using a master WAR as the basis for multiple web applications that each have their own customizations).  Apache Tomcat 8.0.8 (beta) includes numerous fixes for issues identified in 8.0.5 as well as a number of other enhancements and changes. The notable changes since 8.0.5 include:  Further NIO2 fixes which are now considered BETA  Extend and improve memory leak protection and fix a few leaks that crept in during the various refactorings  Add additional protection against a failure to correctly recycle the request and response objects  Full details of these changes, and all the other changes, are available in the [Tomcat 8 changelog](http://tomcat.apache.org/tomcat-8.0-doc/changelog.html#Tomcat_8.0.8_(markt)).  The purpose of this beta release is to give users an opportunity to test Tomcat 8 and to provide feedback to the Tomcat community. Although it is a beta release, it is expected to run stably and is being used in production at the ASF for the ASF's Jira instance. The beta status is primarily because the Tomcat developers feel more feedback is required before a stable release and because there may be a small amount of further refactoring.  5) **Abstract Class can have Constructor? What is the use?**  **Yes**  A constructor in Java doesn't actually "build" the object, it is used to initialize fields.  Imagine that your abstract class has fields x and y, and that you always want them to be initialized in a certain way, no matter what actual concrete subclass is eventually created. So you create a constructor and initialize these fields.  Now, if you have two different subclasses of your abstract class, when you instantiate them their constructors will be called, and then the parent constructor will be called and the fields will be initialized.  If you don't do anything, the default constructor of the parent will be called. However, you can use the super keyword to invoke a specific constructor on the parent class.  **6) Static method can be overridden?**  **Can we Overload or Override static methods in java?**  Let us first define Overloading and Overriding.  [**Overriding**](http://en.wikipedia.org/wiki/Method_overriding) : Overriding is a feature of OOP languages like Java that is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class). The implementation to be executed is decided at run-time and decision is made according to the object used for call. Note that signatures of both methods must be same.  [**Overloading**](http://en.wikipedia.org/wiki/Function_overloading): Overloading is also a feature of OOP languages like Java that is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters. Note that in both C++ and Java, [methods cannot be overloaded according to return type.](http://www.geeksforgeeks.org/g-fact-75/)  **Can we overload static methods?** The answer is ‘Yes’. We can have two or more static methods with same name, but differences in input parameters. For example, consider the following Java program.   |  | | --- | | // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |   Output:  Test.foo() called  Test.foo(int) called  **Can we overload methods that differ only by static keyword?** We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example.   |  | | --- | | // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public void foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }      public static void main(String args[]) {          Test.foo();      }  } |   Output: Compiler Error, cannot redefine foo()  **Can we Override static methods in Java?** We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’. If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.   |  | | --- | | /\* Java program to show that if static method is redefined by     a derived class, then it is not overriding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // This method hides display() in Base      public static void display() {           System.out.println("Static or class method from Derived");      }        // This method overrides print() in Base      public void print() {           System.out.println("Non-static or Instance method from Derived");     }  }    // Driver class  public class Test {      public static void main(String args[ ])  {         Base obj1 = new Derived();           // As per overriding rules this should call to class Derive's static         // overridden method. Since static method can not be overridden, it         // calls Base's display()         obj1.display();           // Here overriding works and Derive's print() is called         obj1.print();      }  } |   Output:  Static or class method from Base  Non-static or Instance method from Derived  Following are some important points for method overriding and static methods in Java. **1)** For class (or static) methods, the method according to the type of reference is called, not according to the abject being referred, which means method call is decided at compile time.  **2)** For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.  **3)** An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors.   |  | | --- | | /\* Java program to show that if static methods are redefined by     a derived class, then it is not overriding but hidding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // Static is removed here (Causes Compiler Error)      public void display () {          System.out.println("Non-static method from Derived");      }        // Static is added here (Causes Compiler Error)      public static void print() {          System.out.println("Static method from Derived");     }  } |   In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass.  **7) What is Java Constructor Chaining?**  Calling another constructor in the same class from another constructor is called constructor chaining. By using this () we can call another constructor in the same class. In case we want to call another constructor, this () should be the first line in the constructor. Below example shows the code for constructor chaining.  **public** **class** ConstructorChaining {    **public** ConstructorChaining(){  System.*out*.println("In default constructor...");  }  **public** ConstructorChaining(**int** i){  **this**();  System.*out*.println("In single parameter constructor...");  }  **public** ConstructorChaining(**int** i,**int** j){  **this**(j);  System.*out*.println("In double parameter constructor...");  }    **public** **static** **void** main(String a[]){    ConstructorChaining ob = **new** ConstructorChaining(1,2);    }  }  **8) Serialization in java?**  **Serialization in java** is a mechanism of *writing the state of an object into a byte stream*.  It is mainly used in Hibernate, RMI, JPA, EJB, JMS technologies.  The reverse operation of serialization is called *deserialization*.  The String class and all the wrapper classes implements *java.io.Serializable* interface by default.  **Advantage of Java Serialization?**  It is mainly used to travel object's state on the network (known as marshaling).  Serializable is a marker interface (has no body). It is just used to "mark" java classes which support a certain capability.  It must be implemented by the class whose object you want to persist. Let's see the example given below:   import java.io.Serializable;   public class Student implements Serializable{    int id;    String name;    public Student(int id, String name) {     this.id = id;     this.name = name;    }   }  **9) How to protect the variable not to serialize?**  Using transient variables.  **8 ) What is Externalization?**  In case of Serializable, default serialization process is used. While in case of Externalizable custom Serialization process is used which is implemented by application.  JVM gives call back to readExternel() and writeExternal() of java.io.Externalizalbe interface for restoring and writing objects into persistence.  Externalizable interface provides complete control of serialization process to application.  readExternal() and writeExternal() supersede any specific implementation of writeObject and readObject methods.  Though Externalizable provides complete control, it also presents challenges to serialize super type state and take care of default values in case of transient variable and static variables in Java. If used correctly Externalizable interface can improve performance of serialization process.  **10) How to integrate Hibernate in Spring?**  **Step: 1** \src\main\webapp\WEB-INF\web.xml  1. **<servlet>** 2. **<servlet-name>**mvc-dispatcher**</servlet-name>** 3. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>** 4. **<init-param>** 5. **<param-name>**contextConfigLocation**</param-name>** 6. **<param-value>**/WEB-INF/**spring-config.xml</param-value>** 7. **</init-param>** 8. **<load-on-startup>**1**</load-on-startup>** 9. **</servlet>** 11. **<servlet-mapping>** 12. **<servlet-name>**mvc-dispatcher**</servlet-name>** 13. **<url-pattern>**/**</url-pattern>** 14. **</servlet-mapping>**   **spring-config.xml**   1. **<context:component-scan** base-package="com.beingjavaguys.controller" **/>** 2. **<context:property-placeholder** location="classpath:**database.properties**" **/>** 3. **<mvc:annotation-driven** **/>** 5. **<bean** id="dataSource" 6. class="org.springframework.jdbc.datasource.DriverManagerDataSource"**>** 7. **<property** name="driverClassName" value="${database.driver}" **/>** 8. **<property** name="url" value="${database.url}" **/>** 9. **<property** name="username" value="${database.user}" **/>** 10. **<property** name="password" value="${database.password}" **/>** 11. **</bean>** 13. **<bean** id="sessionFactory" 14. class="org.springframework.orm.hibernate4.LocalSessionFactoryBean"**>** 15. **<property** name="dataSource" ref="dataSource" **/>** 16. **<property** name="annotatedClasses"**>** 17. **<list>** 18. **<value>**com.beingjavaguys.domain.Employee**</value>** 19. **</list>** 20. **</property>** 21. **<property** name="hibernateProperties"**>** 22. **<props>** 23. **<prop** key="hibernate.dialect"**>**org.hibernate.dialect.MySQL5Dialect**</prop>** 24. **<prop** key="hibernate.show\_sql"**>**${hibernate.show\_sql}**</prop>** 25. **</props>** 26. **</property>** 27. **</bean>** 28. **<bean** id="txManager" 29. class="org.springframework.orm.hibernate4.HibernateTransactionManager"**>** 30. **<property** name="sessionFactory" ref="sessionFactory" **/>** 31. **</bean>** 33. **<bean** id="persistenceExceptionTranslationPostProcessor" 34. class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" **/>**   **database.properties**   database.driver=com.mysql.jdbc.Driver - database.url=jdbc:mysql://localhost:3306/springhibernate\_db - database.user=YOUR\_USERNAME - database.password=YOUR\_PASSWORD - hibernate.show\_sql=true  This is the configuration for spring with hibernate.  **11) What is Singleton design Pattern?**  The Singleton pattern is deceptively simple, even and especially for Java developers. In this classic JavaWorld article, David Geary demonstrates how Java developers implement singletons, with code examples for multithreading, classloaders, and serialization using the Singleton pattern. He concludes with a look at implementing singleton registries in order to specify singletons at runtime.  Sometimes it's appropriate to have exactly one instance of a class: window managers, print spoolers, and filesystems are prototypical examples. Typically, those types of objects—known as singletons—are accessed by disparate objects throughout a software system, and therefore require a global point of access. Of course, just when you're certain you will never need more than one instance, it's a good bet you'll change your mind.  The Singleton design pattern addresses all of these concerns. With the Singleton design pattern you can:   * Ensure that only one instance of a class is created * Provide a global point of access to the object * Allow multiple instances in the future without affecting a singleton class's clients   **12) How to achieve Singleton in Multithreaded Environment?**  If people are asked to name a design pattern, the most commonly uttered name is the Singleton. It is one of the most widely used patterns (and most abused, too). The idea behind the Singleton is simple. It ensures that the Singleton class has only one instance and provides a global access point for it. We can use a Singleton when the presence of multiple instances can potentially damage the system, and we need global access to the single instance. Let’s first see the classical implementation of Singleton:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13 | public class Singleton {      private static Singleton \_instance = null;        private Singleton() {}        public static Singleton getInstance() {          if (\_instance == null) {              \_instance = new Singleton();          }            return \_instance;      }  } |   The implementation is straight forward. The constructor of Singleton is private, so it is not possible to instantiate it from outside. The only way to get an instance is to call static getInstance() method. getInstance() first checks whether an instance is already created. If not, then it creates an instance, refers it via private static member \_instance and then returns that. And if already created then it returns the previously created \_instance. Thus only the first call to getInstance() instantiates a Singleton object and any further call returns the same object. Also note that the object is not instantiated until **getInstance**() is called, i.e. we only create that when actually required. This is called lazy initialization and becomes helpful if the object is resource hungry.  This looks very simple and straight forward implementation. But we have a slight problem with this. This classical implementation is not thread safe. Lets see what may happen in the presence of two threads.   1. Thread-1 enters getInstance() for the first time and sees that \_instance is null and thus the condition is true. 2. Before instantiating the object a thread switch occur. 3. Thread-2 enters getInstance() and it will see \_instance null too, as the instantiation by Thread-1 is not completed yet. 4. Thread-2 instantiate new object and then return. 5. Thread-1 knows nothing about Thread-2. So when it gets its turn again, it instantiates another object and returns that. At this point we have two instances of Singleton which violates the fundamental purpose of the pattern.   So what can we do solve this problem? The easiest solution comes up if we don’t want the lazy initialization. Something like this:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public class Singleton {      private static Singleton \_instance = new Singleton();        private Singleton() {}        public static Singleton getInstance() {          return \_instance;      }  } |   This is guaranteed to be thread safe. The only cost that we pay is that we loose the lazy initialization. If the singleton object is created at every run or the cost of the object is not so high then probably this is the best approach to implement a Singleton in multi-threaded environment.  But what if we want the lazy initialization? The general approach to write a thread safe code is to acquire a lock before accessing the shared resource. We can acquire a thread lock after entering getInstance(). Something like this:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public static Singleton getInstance() {      acquire\_lock();      if (\_instance == null) {          \_instance = new Singleton();      }      release\_lock();        return \_instance;  } |   Again, this is guaranteed to be thread safe. Thread-2 can not proceed until Thread-1 completes the instantiation and releases the lock. Obviously after acquiring the lock Thread-2 will see \_instance non-null and won’t create a separate instance. But unfortunately we have slight problem with this method, namely performance. Acquiring a thread lock is a very costly operation. Acquiring a lock at every call of getInstance() may affect the overall performance severely, specially when the calls are frequent. And to make matter worse, we only need the lock for the first time. Once \_instance is set up with a valid value, there is no need of the locking. But we are acquiring the lock every time and thus wasting our resource. The very idea of lazy initialization is to use resource efficiently, but this method of locking seems overkilling.  Is there any way to get around this problem? We have already realized that the lock is only needed for the first time. There is no need to acquire lock once \_instance is initialized. So why don’t we acquire the lock only if \_instance is null, like this:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | public static Singleton getInstance() {      if (\_instance == null) {          acquire\_lock();          \_instance = new Singleton();          release\_lock();      }        return \_instance;  } |   A clever thought. But this will not work. Why? Thread-1 may see the condition true and enter the condition, but before acquiring the lock thread switch may occur. Then Thread-2 will again find the condition true and thus we are in the same situation as before.  Can we fix this? Indeed we can. And with a slight modification. We only need to check \_instance against null again after acquiring the lock. Like this:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | public static Singleton getInstance() {      if (\_instance == null) {          acquire\_lock();          if (\_instance == null) {              \_instance = new Singleton();          }          release\_lock();      }        return \_instance;  } |   Lets see what happens with the previous situation:  Thread-1 enters the condition but before acquiring the lock thread switch occurs.  Thread-2 enters the condition, acquires lock, instantiates, releases lock and returns like before.  Thread-1 acquires the lock. But now it will see that \_instance is non-null and thus it will not create a new object. It will simply release the lock and return the object created by Thread-2.  Looks like our problem with multiple threads is finally finished. This approach of checking twice is a design pattern in its own right and is called Double Checked Locking Pattern.  Unfortunately, this is NOT guaranteed to work either. The reason is our modern compilers are too smart in optimizing things. An optimizing compiler (all modern compilers are) can reorder the instructions for various reasons. It can reorder read/write calls to improve cache performance, it may try to run as many instructions as possible in parallel when multiple execution units are present (all our modern CPUs has multiple execution units) and for many such reasons. A concrete example might clarify.   |  |  | | --- | --- | | 1  2  3 | int a = 10;  int b = 20;  int c = a + b; |   Here the compiler guarantees that the value of c will be 30, but it does not guarantee anything else about a and b. It can completely eliminate the first two instructions as constant expression, it can execute 1st one first and then 2nd one, or it can execute 2nd one first and then 1st one, or even it can execute them in parallel if multiple execution units are present. The summary is: we can not depend on it.  Lets back to our double checked locking code. When \_instance = new Singleton() is executed three things happen mainly. A portion of memory is allocated for Singleton object, that memory is initialized with the object’s data and finally \_instance points to that memory location. \_instance is valid only when the initialization is complete. Here the catch is that the compiler may change the order of these, i.e. it may first allocate the memory and make \_instance point to that memory and then go for the initialization of the object. It is also possible that the initialization is taking place in another execution unit. Why the compiler would do so is a matter of study in compiler optimization theory, but the fact is \_instance may point to an uninitialized memory. Lets see what may happen to our double checked locking code in this case:   1. Thread-1 enters getInstance(), acquires the lock and starts the instantiation process. It allocates the memory and makes \_instance point to it. 2. Before the initialization of created object is complete, a thread switch is occurred. 3. Thread-2 enters getInstance() and finds \_instance non-null, as it is already pointed to the allocated memory by Thread-1. 4. As Thread-2 finds \_instance non-null, it thinks that the instantiation is complete and returns that. It has no way to know that the initialization is yet to be completed. 5. As a result the caller of getInstance() in Thread-2 receives something that is not initialized properly. If it tries to use the object before the initialization is actually completed, it will create major problem and may even crash the program.   To make matters worse, if this occurs than it will be very difficult to figure out the exact reason of the unexpected behavior or crash, as this will happen in random. How can we deal with this? Well … this might be heart breaking, but there is NO single way which can solve this problem in ALL compilers/hardwares/platforms. We have few work around depending on the platform.   1. Marking a memory location volatile denotes to the compiler that this memory can be changed in ways not known to the compiler and thus prevents such optimization. Originally that was introduced to handle memory-mapped I/O and was not related to thread, but J2SE 5.0 and Visual C++ 5.0 ensures that volatile works correctly with multiple threads. So making \_instance volatile will solve the problem for them. But that is not guaranteed to work with versions of J2SE lower than 5.0 or lower than Visual C++ 5.0 or in other C++ compilers. And also using volatile may affect the performance too. 2. A memory barrier instruction forces that all read/write instructions before the barrier must be completed before any read/write operation is executed after the barrier. For systems that support memory barrier we can solve this problem by introducing a memory barrier after the instantiation of Singleton object and by introducing a flag as test condition. Something like this:  |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | public class Singleton {      private static Singleton \_instance = null;      private static bool flag = false;        private Singleton() {}        public static Singleton getInstance() {          if (!flag) {              acquire\_lock();              if (!flag) {                  \_instance = new Singleton();                  memory\_barrier();                  flag = true;              }              release\_lock();          }            return \_instance;      }  } |  1. Note that now a new flag is added as the test condition. \_instance may point to uninitialized memory, but the memory barrier instruction before flag = true will ensure that flag will be false until the object is fully initialized. The combination of memory barrier and new flag instead of \_instance as test condition solves the problem. But the downside of this solution is that if the compiler used does not support a memory barrier instruction natively (.NET has a memory barrier instruction) then it might be difficult to implement a barrier correctly, and may even require assembly coding. 2. In addition to double checked locking pattern there is an another approach to implement Singleton correctly known as Initialization on demand holder idiom. This method works on all versions of Java.   None of the solutions works in all platforms. All of them exploit platform/compiler specific features. May be the best solution is to ignore lazy initialization. And if we want to stick with lazy initialization, then performance of the full locking version of getInstance() (where lock is acquired after entering getInstance()) can be improved by minimizing calls to getInstance() like this:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | // requires three calls and thus acquires lock three times  Singleton.getInstance().method1();  Singleton.getInstance().method2();  Singleton.getInstance().method3();    // requires only one call and thus acquires lock only ones  Singleton instance = Singleton.getInstance();  instance.method1();  instance.method2();  instance.method3(); |   This is not a real solution to the problem, but it can improve performance significantly in practice.  **13) what is difference Between throws and throw keyword in Exception?**   public void shutdown() throws IOException{          throw new IOException("Unable to shutdown");   }  if you see the code above, throws keyword is used to declare Exception thrown by this method shutdwon(), by looking at method signature we know that its throwing IOException. while inside method code throw keyword is actually used to throw instance of IOException. here are couple of more differences between throw and throws keyword in java:  You can declare multiple exception thrown by method in throws keyword by separating them in common e.g. throws IOException, ArrayIndexBoundException etc, while you can only throw one instance of exception using throw keyword e.g. throw new IOException("not able to open connection").  throws keyword gives a method flexibility of throwing an Exception rather than handling it. with throws keyword in method  signature a method suggesting its caller to prepare for Exception declared in throws clause, especially in case of [checked Exception](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html) and provide sufficient handling of them. On the other hand throw keyword transfer control of execution to caller by throwing an instance of Exception. throw keyword can also be used in place of return as shown in below example:   1. Throws clause in used to declare an exception and thow keyword is used to throw an exception explicitly. 2. If we see syntax wise than throw is followed by an instance variable and throws is followed by exception class names. 3. The keyword throw is used inside method body to invoke an exception and throws clause is used in method declaration (signature).   for Example  **Throw**  throw new Exception("You have some exception")  throw new IOException("Connection failed!!")  **Throws**  throws IOException, ArithmeticException, NullPointerException.  **14) Can we add throws at class level?**  No. Only method signature.  **15) HashCode and Equals?**  Java.lang.Object has methods called hasCode() and equals(). These methods play a significant role in the real time application. However its use is not always common to all applications. In some case these methods are overridden to perform certain purpose. In this article I will explain you some concept of these methods and why it becomes necessary to override these methods.  **hashCode()**  As you know this method provides the has code of an object. Basically the default implementation of hashCode() provided by Object is derived by mapping the memory address to an integer value. If look into the source of Object class , you will find the following code for the hashCode. public native int hashCode(); It indicates that hashCode is the native implementation which provides the memory address to a certain extent. However it is possible to override the hashCode method in your implementation class.  **equals()**  This particular method is used to make equal comparison between two objects. There are two types of comparisons in Java. One is using “= =” operator and another is “equals()”. I hope that you know the difference between this two. More specifically the “.equals()” refers to equivalence relations. So in broad sense you say that two objects are equivalent they satisfy the “equals()” condition. If you look into the source code of Object class you will find the following code for the equals() method.  public boolean equals(Object obj)  {  return (this == obj);  }  Now I will explain you when to override the equals() and hashCode() methods and why it is necessary to override these methods. In this regard there is a rule of thumb that if you are going to override the one of the methods( ie equals() and hashCode() ) , you have to override the both otherwise it is a violation of contract made for equals() and hashCode(). Please refer to the Sun’s java docs for the method’s contract. I provide some test case scenario where you will find the significance of these methods. Case-1: You can override the hashCode method in your own way.  **package** hascodeequals;  **public** **class** Emp {    **private** **int** age;  **public** Emp(**int** age) {  **super**();  **this**.age = age;  }        **public** **int** hashCode(){    **return** age;  }      **public** **boolean** equals(Object obj) {  Emp emp = (Emp) obj;  **if** (emp.age == age) {  **return** **true**;  }  **return** **false**;  }  }  **package** hascodeequals;  **public** **class** DemoEmp {  **public** **static** **void** main(String[] args) {    Emp emp1 = **new** Emp(100);  Emp emp2 = **new** Emp(100);  Emp emp3 = **new** Emp(100);    System.*out*.println(emp1.equals(emp2));  System.*out*.println(emp2.equals(emp3));    }  }  **OutPut**:  true  true  In case we are not using hashcode() , equals() means will get the output is false,false.  **16) difference between the below ?**  String aStr = null;  1) if(aStr != null) --> false  2) if(aStr.equlas(null)) --> NullPointerException  3) if(null.equals(aStr)) --> Can not invoke equals on the primitive type null  **17) How to access the JNDI datasource using Spring ?**  JNDI (Java Naming and Directory Interface) enables [Java](http://searchsoa.techtarget.com/definition/Java)platform-based applications to access multiple naming and directory services. Part of the Java Enterprise application programming interface (API) set, JNDI makes it possible for developers to create portable applications that are enabled for a number of different naming and directory services, including: [file system](http://searchstorage.techtarget.com/definition/file-system)s; [directory](http://searchwinit.techtarget.com/definition/directory) services such as Lightweight Directory Access Protocol ([LDAP](http://searchmobilecomputing.techtarget.com/definition/LDAP)), [Novell Directory Services](http://searchnetworking.techtarget.com/definition/Novell-Directory-Services), and Network Information System ([NIS](http://searchnetworking.techtarget.com/definition/NIS)); and distributed object systems such as the Common Object Request Broker Architecture ([CORBA](http://searchsqlserver.techtarget.com/definition/CORBA)), Java Remote Method Invocation (RMI), and Enterprise JavaBeans (EJB).  servlet-context.xml  <!-- Create DataSource Bean -->        <beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">          <beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>      </beans:bean>  **apache-tomcat/conf/server.xml**  <Resource name="jdbc/TestDB"        global="jdbc/TestDB"        auth="Container"        type="javax.sql.DataSource"        driverClassName="com.mysql.jdbc.Driver"        url="jdbc:mysql://localhost:3306/TestDB"        username="pankaj"        password="pankaj123"          maxActive="100"        maxIdle="20"        minIdle="5"        maxWait="10000"/>  **apache-tomcat/conf/context.xml**  <ResourceLink name="jdbc/MyLocalDB"                      global="jdbc/TestDB"                      auth="Container"                      type="javax.sql.DataSource" />    <http://www.journaldev.com/2597/spring-datasource-jndi-with-tomcat-example>  **18) What is the behavior of the Spring Container on the following case?**  Class A {  }  Class B {  }    In spring configuration Class A having Class B as reference , Class B is having Class A reference what will happen?  A --> B  B --> A    Yes it will work.  **20) What are the modules you worked in Spring?**  C:\Users\Dharma\Desktop\spring-overview.png |

**21) What are the ways to create an object - scopes?**

Method 1

**Using new keyword**. This is the most common way to create an object in java. Almost 99% of objects are created in this way.

Method 2

**Using Class.forName().** Class.forName() gives you the class object, which is useful for reflection. The [methods](http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/) that this object has are defined by[Java](http://crunchify.com/category/spring-mvc/), not by the programmer writing the class. They are the same for every class. Calling newInstance() on that gives you an instance of that class (i.e. callingClass.forName("ExampleClass").newInstance() it is equivalent to calling new ExampleClass()), on which you can call the methods that the class defines, access the visible fields etc.

Method 3

**Using clone().** The clone() can be used to create a copy of an existing object.

Method 4

**Using newInstance()**method

Method 5

**Using Object Deserialization**. Object Deserialization is nothing but creating an object from its serialized form.

**Sample Program:**

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.Serializable;

/\*\*

\* @author Crunchify.com

\*/

public class CrunchifyObj implements Cloneable, Serializable {

private static final long serialVersionUID = 1L;

public CrunchifyObj() {

System.out.println("Hello! CrunchifyObj() just got created..");

}

@Override

protected Object clone() throws CloneNotSupportedException {

return (CrunchifyObj) super.clone();

}

@SuppressWarnings({ "unused", "resource" })

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

// Create Object1

CrunchifyObj object1 = new CrunchifyObj();

// Create Object2

CrunchifyObj object2 = (CrunchifyObj) Class.forName("crunchify.com.tutorial.CrunchifyObj").newInstance();

// Create Object3

CrunchifyObj secondObject = new CrunchifyObj();

CrunchifyObj object3 = (CrunchifyObj) secondObject.clone();

// Create Object4

Object object4 = CrunchifyObj.class.getClassLoader().loadClass("crunchify.com.tutorial.CrunchifyObj").newInstance();

// Create Object5

// create a new file with an ObjectOutputStream

FileOutputStream out = new FileOutputStream("crunchify.txt");

ObjectOutputStream oout = new ObjectOutputStream(out);

// write something in the file

oout.writeObject(object3);

oout.flush();

// create an ObjectInputStream for the file we created before

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("crunchify.txt"));

CrunchifyObj object5 = (CrunchifyObj) ois.readObject();

}

}

**22) Default scope of the bean?**

Spring framework supports five type of scopes and for bean instantiation as of Spring 3.0 and also we can create a custom scope.

[**singleton**](http://javapapers.com/spring/spring-bean-scopes/#L568)

[**prototype**](http://javapapers.com/spring/spring-bean-scopes/#L570)

[**request**](http://javapapers.com/spring/spring-bean-scopes/#L572)

[**session**](http://javapapers.com/spring/spring-bean-scopes/#L574)

[**global\_session**](http://javapapers.com/spring/spring-bean-scopes/#L609)

singleton scope

**singleton is the default scope**. [Singleton design pattern](http://javapapers.com/design-patterns/singleton-pattern/) requires no introduction as it is the easiest of all to understand. In the bean definition if the scope is not given, then by default singleton scope is assumed. In a given Spring container a singleton scoped bean  
will be instantiated only once and the same will be used for its lifetime.

**23) Want to create a Singleton bean but not at the startup, upon first request will be created for only one and its maintained? what to do?**

Lazy init parameter in bean configuration

**24) Any of the Data Structure worked?**

The data structures provided by the Java utility package are very powerful and perform a wide range of functions. These data structures consist of the following interface and classes:

Enumeration

BitSet

Vector

Stack

Dictionary

Hashtable

Properties

All these classes are now legacy and Java-2 has introduced a new framework called Collections Framework, which is discussed in next tutorial:

2**6) Difference b/w REST and SOAP Web Service?**

A Web service, in very broad terms, is a method of communication between two applications or electronic devices over the World Wide Web (WWW). Web services are of two kinds: Simple Object Access Protocol ([SOAP](http://searchsoa.techtarget.com/definition/SOAP)) and Representational State Transfer ([REST](http://searchsoa.techtarget.com/definition/REST)).

SOAP defines a standard communication protocol (set of rules) specification for [XML](http://searchsoa.techtarget.com/definition/XML)-based message exchange. SOAP uses different transport protocols, such as [HTTP](http://searchwindevelopment.techtarget.com/definition/HTTP) and [SMTP](http://searchexchange.techtarget.com/definition/SMTP). The standard protocol HTTP makes it easier for SOAP model to tunnel across [firewalls](http://searchsecurity.techtarget.com/definition/firewall) and [proxies](http://whatis.techtarget.com/definition/proxy-server)without any modifications to the SOAP protocol. SOAP can sometimes be slower than middleware technologies like [CORBA](http://searchsqlserver.techtarget.com/definition/CORBA) or [ICE](http://searchsoa.techtarget.com/definition/ice) due to its verbose XML format.

REST describes a set of architectural principles by which data can be transmitted over a standardized interface (such as HTTP). REST does not contain an additional messaging layer and focuses on design rules for creating stateless services. A client can access the resource using the unique [URI](http://searchsoa.techtarget.com/definition/URI) and a representation of the resource is returned. With each new resource representation, the client is said to transfer state. While accessing RESTful resources with HTTP protocol, the URL of the resource serves as the resource identifier and GET, PUT, DELETE, POST and HEAD are the standard HTTP operations to be performed on that resource.

REST

The RESTful Web services are completely stateless. This can be tested by restarting the server and checking if the interactions are able to survive.

Restful services provide a good caching infrastructure over HTTP GET method (for most servers). This can improve the performance, if the data the Web service returns is not altered frequently and not dynamic in nature.

The service producer and service consumer need to have a common understanding of the context as well as the content being passed along as there is no standard set of rules to describe the REST Web services interface.

REST is particularly useful for restricted-profile [devices such as mobile](http://searchsoa.techtarget.com/answer/REST-or-SOAP-Which-offers-the-most-benefits-for-mobile-applications) and PDAs for which the overhead of additional parameters like headers and other SOAP elements are less.

REST services are easy to integrate with the existing websites and are exposed with XML so the HTML pages can consume the same with ease. There is hardly any need to refactor the existing website architecture.

This makes developers more productive and comfortable as they will not have to rewrite everything from scratch and just need to add on the existing functionality.

REST-based implementation is simple compared to SOAP.

SOAP

The [Web Services Description Language](http://searchsoa.techtarget.com/tutorial/WSDL-Tutorial) (WSDL) contains and describes the common set of rules to define the messages, bindings, operations and location of the Web service.

WSDL is a sort of formal contract to define the interface that the Web service offers.

SOAP requires less plumbing code than REST services design, (i.e., transactions, security, coordination, addressing, trust, etc.) Most real-world applications are not simple and support complex operations, which require conversational state and contextual information to be maintained. With the [SOAP approach](http://searchsoa.techtarget.com/tutorial/Simple-Object-Access-Protocol-SOAP-Tutorial), developers need not worry about writing this plumbing code into the application layer themselves.

SOAP Web services (such as JAX-WS) are useful in handling asynchronous processing and invocation.

SOAP supports several protocols and technologies, including WSDL, XSDs, SOAP, WS-Addressing

In a nutshell, when you're publishing a complex application program interface (API) to the outside world, SOAP will be more useful. But when something with a lower learning curve, and with lightweight and faster results and simple transactions (i.e., [CRUD](http://searchdatamanagement.techtarget.com/definition/CRUD-cycle) operations) is needed, my vote goes to REST.

**27) How the request URL is mapped in REST?**

/api/posts"(controller: "postRest", parseRequest: true) {  
 action = [GET: "list", POST: "save", PUT: "unsupported", DELETE: "unsupported"]  
}

**28) How to deploy the application in WAS?**

Normal **WAS** setup.

**29) How to integrate Struts and Spring?**

**Web.xml**

<servlet>

<servlet-name>action</servlet-name>

<servlet-class>

org.apache.struts.action.ActionServlet

</servlet-class>

<init-param>

<param-name>config</param-name>

<param-value>

/WEB-INF/struts-config.xml

</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

**struts-config.xml**

<plug-in className=*"org.springframework.web.struts.ContextLoaderPlugIn"*>

<set-property property=*"contextConfigLocation"*

value=*"/WEB-INF/classes/SpringBeans.xml"* />

</plug-in>

**SpringBeans.xml**

*<import resource="com/mkyong/customer/spring/CustomerBean.xml"/>*

**CustomerBean.xml**

<bean id=*"customerDao"* class=*"com.mkyong.customer.dao.impl.CustomerDaoImpl"* >

<property name=*"sessionFactory"* ref=*"sessionFactory"*></property>

</bean>

**30) What are the Spring lifecycle methods? And the order in which executes?**

Spring Bean Life Cycle Methods – InitializingBean, DisposableBean, @PostConstruct, @PreDestroy and \*Aware interfaces

By implementing**InitializingBean** and **DisposableBean**interfaces – Both these interfaces declare a single method where we can initialize/close resources in the bean. For post-initialization, we can implement InitializingBean interface and provide implementation of afterPropertiesSet() method. For pre-destroy, we can implement DisposableBean interface and provide implementation of destroy()method. These methods are the callback methods and similar to servlet listener implementations.

This approach is simple to use but it’s not recommended because it will create tight coupling with the Spring framework in our bean implementations.

Providing**init-method** and**destroy-method** attribute values for the bean in the spring bean configuration file. This is the recommended approach because of no direct dependency to spring framework and we can create our own methods.

**31) What is Destroy method how it is configured in Spring?**

*<beans xmlns="*[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)*"*

*xmlns:xsi="*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"*

*xsi:schemaLocation="http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*

*default-init-method="init"*

*default-destroy-method="destroy">*

*<bean id="netManager" class="com.java2novice.beans.NetworkManager">*

*<property name="urlStr" value="*[*http://www.google.com/search?q=java2novice*](http://www.google.com/search?q=java2novice)*"/>*

*</bean>*

*</beans>*

**32) What is Component Mapping in hibernate?**

**<hibernate-mapping>**

**<class** name="com.mkyong.customer.Customer" table="customer"

catalog="mkyongdb"**>**

**<id** name="custId" type="java.lang.Integer"**>**

**<column** name="CUST\_ID" **/>**

**<generator** class="identity" **/> (or)**

<generator class="native" />

**</id>**

**<property** name="custName" type="string"**>**

**<column** name="CUST\_NAME" length="10" not-null="true" **/>**

**</property>**

**<property** name="age" type="int"**>**

**<column** name="AGE" not-null="true" **/>**

**</property>**

**<component** name="Address" class="com.mkyong.customer.Address"**>**

**<property** name="address1" type="string"**>**

**<column** name="ADDRESS1" not-null="true" **/>**

**</property>**

**<property** name="address2" type="string"**>**

**<column** name="ADDRESS2" not-null="true" **/>**

**</property>**

**<property** name="address3" type="string"**>**

**<column** name="ADDRESS3" not-null="true" **/>**

**</property>**

**</component>**

**<property** name="createdDate" type="date"**>**

**<column** name="CREATED\_DATE" length="10" not-null="true" **/>**

**</property>**

**<property** name="createdBy" type="string"**>**

**<column** name="CREATED\_BY" length="10" not-null="true" **/>**

**</property>**

**</class>**

**</hibernate-mapping>**

**33) How to create abstract instance in Spring?**

<bean id="abstractTest" class="AbstractTest" abstract="true">

<property name="strName" value="TestName"/>

</bean>

**34) How to create Parent child relation in Spring configuration?**

<bean id="abstratImpl" class="AbstractImpl" parent="abstractTest"/>

**35)** **How to create a Table in Hibernate?**

We can able to create two ways.

<property name="hibernate.hbm2ddl.auto" value="create"/>

**35)** **What is 2nd level Cache in Hibernate?**

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.

Any third-party cache can be used with Hibernate. An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

**Query Cache**

\*\*\*\*\*\*\*\*\*

hibernate.cache.use\_query\_cache="true"

Session session = SessionFactory.openSession();

Query query = session.createQuery("FROM EMPLOYEE");

query.setCacheable(true);

List users = query.list();

SessionFactory.closeSession();

**36) What is Lazy initialization? And how to?**

The simplest way that Hibernate can apply lazy load behavior upon your entities and associations is **by providing a proxy implementation** of them. Hibernate intercepts calls to the entity by substituting a proxy for it derived from the entity’s class. Where the requested information is missing, it will be loaded from the database before control is ceded to the parent entity’s implementation.

Please note that when the association is represented as a collection class, then a wrapper (essentially a proxy for the collection, rather than for the entities that it contains) is created and substituted for the original collection. When you access this collection proxy then what you get inside returned proxy collection are not proxy entities; rather they are actual entities. You need not to put much pressure on understanding this concept because on runtime it hardly matters.

**@OneToMany( mappedBy = "category", fetch = FetchType.LAZY )**

**private Set<ProductEntity> products;**

**37) What are all the design pattern used by Struts ?**

Struts is based on model 2 MVC (Model-View-Controller) architecture. Struts controller uses the command design pattern and the action classes use the adapter design pattern. The process() method of the RequestProcessor uses the template method design pattern. Struts also implement the following J2EE design patterns.

**38) What are the attributes of bean definition?**

| **Name** | **Description** |
| --- | --- |
| Bean ID | Name of the Spring bean.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.BeanId](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#BeanId) |
| Application Context | Display name of the application context that this Spring bean is defined in. The application context is the Spring Inversion of Control (IoC) container.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.ApplicationContextDisplayName](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#ApplicationContextDisplayName) |
| Bean Class Name | Class name of this Spring bean, as defined in the application context of the Spring application.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.BeanClassname](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#BeanClassname) |
| Resource | The name of the resource that this bean definition comes from. May be empty if the bean is implicitly registered.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.ResourceDescription](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#ResourceDescription) |
| Role | Role hint of this bean definition. The role is one of ROLE\_APPLICATION, ROLE\_SUPPORT, or ROLE\_INFRASTRUCTURE.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Role](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Role) |
| Scope | Scope of this bean. The scope is "singleton", "prototype", or other web specific or user defined values.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Scope](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Scope) |
| Parent Bean ID | Name (Id) of parent bean definition.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.ParentId](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#ParentId) |
| Singleton | Whether this is a singleton Spring bean. There is just one instance of a singleton bean per bean definition per application context.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Singleton](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Singleton) |
| Abstract | Whether this Spring bean is "abstract". An abstract bean definition can be used as a base for other definitions but cannot be instantiated.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Abstract](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Abstract) |
| Lazy Init | Whether this bean should be lazily initialized. A lazy initialized bean is not created until it is needed.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.LazyInit](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#LazyInit) |
| Autowire Candidate | Whether this bean is a candidate to be autowired to other beans.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.AutowireCandidate](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#AutowireCandidate) |
| Aliases | Get the aliases for this bean definition. Aliases are other names this bean definition is known by.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Aliases](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Aliases) |
| Dependencies | Get the names (ids) of other bean definitions that this bean definition depends on.  MBean Attribute:  [SpringBeanDefinitionRuntimeMBean.Dependencies](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e13951/mbeans/SpringBeanDefinitionRuntimeMBean.html#Dependencies) |

**39) Difference b/w Spring prototype,Singleton beans?**

if ClassB is prototype, a newly instance of ClassB is always created, it does not take care of other classes at creation.

So ClassB being prototype and ClassA singleton, you could have N instances of ClassB and only 1 of ClassA in your application at some point.

In your case, as only one instance of ClassA will exist on your application life-cycle, it will only have one instance ClassB which will be different to any other ClassB referenced by other beans in your application.

**40) Add 3 Consecutive Odd number to give value as 48?**

**public** **class** AddThreeNO {

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

**for** (**int** i = 1; i <= 48; i++) {

System.*out*.println(i = i + 2);

}

}

}

**41) 10 balls are there , in that one ball having more weight how to find that? How many iterations?**

**Only One time.**

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

**public** **class** BollWeight{

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

HashMap<String,Integer> obMap = **new** HashMap<String,Integer>();

obMap.put("ball 1", 69);

obMap.put("ball 2", 6);

obMap.put("ball 3", 2055);

obMap.put("ball 4", 54);

obMap.put("ball 5", 25);

obMap.put("ball 6", 1000);

obMap.put("ball 7", 9897);

obMap.put("ball 8", 2090);

obMap.put("ball 9", 545);

obMap.put("ball 10", 276);

Set<Entry<String,Integer>> obSet = obMap.entrySet();

List<Entry<String,Integer>> obList = **new** ArrayList<Entry<String,Integer>>(obSet);

Collections.*sort*(obList,**new** Comparator<Map.Entry<String, Integer>>() {

@Override

**public** **int** compare(Entry<String, Integer> o1,

Entry<String, Integer> o2) {

**return** o2.getValue().compareTo(o1.getValue());

}

});

**for**(Map.Entry<String, Integer> entry : obList){

System.*out*.println(entry.getKey() +" "+entry.getValue());

**break**;

}

}

}

**42) what will happen if we try to increase the size of a collection inside iterator?**

Exception will occur:

**Example:**

**import** java.util.ArrayList;

**import** java.util.concurrent.CopyOnWriteArrayList;

**public** **class** ExampleIterator {

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

ArrayList<String> obList = **new** ArrayList<String>(); //exception coming

CopyOnWriteArrayList<String> obList = **new** CopyOnWriteArrayList<String>(); //It will work

obList.add("bus");

obList.add("car");

obList.add("bike");

**for** (String obStr : obList) {

obList.add("bycycle");

System.*out*.println(obStr);

}

}

}

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.ArrayList$Itr.checkForComodification(Unknown Source)

at java.util.ArrayList$Itr.next(Unknown Source)

out put:

bus

car

bike

**43) What is Dialect in Hibernate?**

For connecting any hibernate application with the database, you must specify the SQL dialects. There are many Dialects classes defined for RDBMS in the org.hibernate.dialect package.

**44) If we want to restrict the List to only for you how to do that? So that other developer will not modify?**

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

/\*\*

\* **@author** Dharma

\*

\*/

**public** **class** UnModifiedList {

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

List<String> obList = **new** ArrayList<String>();

obList.add("dharma");

obList.add("male");

System.*out*.print(obList);

List<String> unmodifyList = Collections.*unmodifiableList*(obList); //we can't modify the list value using collections.

unmodifyList.add("female"); //exception will occur

}

}

**//We can use this also:**

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** UnmodifiableArrayList<E> **extends** ArrayList<E> {

**private** **static** **final** **long** *serialVersionUID* = 898989898L;

**public** UnmodifiableArrayList(Collection<? **extends** E> c) {

**super**(c);

}

**public** **boolean** add(**int** index) {

**return** **false**;//Returning false as the element cannot be added

}

**public** **boolean** addAll(Collection<? **extends** E> c) {

**return** **false**;//Returning false as the element cannot be added

}

**public** E remove(**int** index) {

**return** **null**;//Returning null as the element cannot be removed

}

}

**45) How to JVM loads the class? How the memory allocated?**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent).

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, and instance of JVM is created.

**The JVM performs following operation:**

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**VM provides definitions for the:**

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

|  |
| --- |
| It contains: |
| **1) A virtual processor** |
| **2) Interpreter:** Read byte code stream then execute the instructions. |
| **3) Just-In-Time(JIT) compiler:** It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.  **46) Output System.out.println(null);?**  The method println(char[]) is ambiguous for the type  PrintStream ?    Answer - println(String), println(char[]) both matches this pattern so compiler error. |

**47) how will get the current url address in java script ?**

window.location.href;

**48)Have you worked in any of the JavaScript Object Oriented Technology?**

<!DOCTYPE html>

<html>

<head>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

<script>

$(document).ready(function(){

$("button").click(function(){

$.ajax({url: "demo\_test.txt", success: function(result){

$("#div1").html(result);

}});

});

});

</script>

</head>

<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>

</body>

</html>

**49) Difference b/w Annotation and XML Configuration? Which is best?**

We use annotations for classes in our own project and resort to XML when

We need to inject beans in 3rd party libraries because XML is the only way we can do that.

XML definition will override the annotation

**50) Narrow Conversion? Widening conversion ?**

For a primitive data types, a value narrower data type can be converted to a value of a broader data type without loss of information. This is called Widening conversion. For example an int is directly converted to a double without first having to convert it to a long and a float.

Converting from a broader data type to a narrower data type is called narrowing conversion, which can result in loss of information. For example conversion between char and byte and short are narrowing conversions. For example byte takes 8 bits, short takes 16 bits, int takes 32 bits, long takes 64 bits. Conversion of short to int or int to long is widening conversion and conversion of int to short is a narrowing conversion.

**Widening conversions**   
  
From a **byte**to a short, an int, a long, a float, or a **double**  
From a **short**to an int, a long, a float, or a **double**  
From a **char**to an int, a long, a float, or a **double**  
From an **int**to a long, a float, or a **double**  
From a **long**to a **float**or a **double**  
From a **float**to a **double**  
  
**Narrow conversions**  
  
From a **byte**to a **char**  
From a **short**to a **byte**or a **char**  
From a **char**to a **byte**or a **short**  
From an **int**to a byte, a short, or a **char**  
From a **long**to a byte, a short, a char, or an **int**  
From a **float**to a byte, a short, a char, an int, or a **long**  
From a **double**to a byte, a short, a char, an int, a long, or a **float**

**51) Difference b/w String and StringBuffer and StrinBuilder?**

String immutable (once created can not be changed ),threadsafe(no tow threads can change the value), stores the values in Constant String Pool

StringBuffer mutable( one can change the value of the object ), threadsafe, The object created through StringBuffer is stored in the heap

StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified,

StringBuilder is also not thread safe. StringBuilder is fast as it is not thread safe.

StringBuffer methods -> append, delete, reverse, insert,replace.

52) **Difference b/w String ="" and String s = new String("");**

**Example:**

String s = "Work";

String t = "Work";

Work is a string is in String pool, then s and t actually refer to the

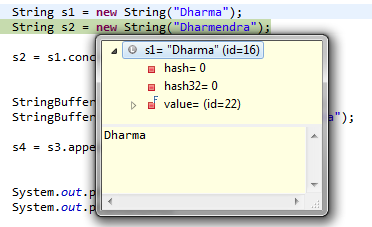
same object, and s==t will return true, for "==" for objects read "is the same object"

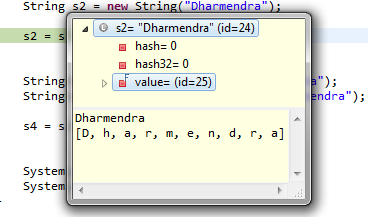
String s = new String("Work");

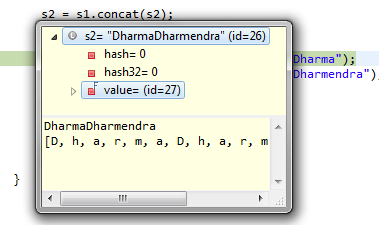
String t = new String("Work");

created in heap, both are different object. s!=t and s.equals(t) will return true

STRING Immutable example:

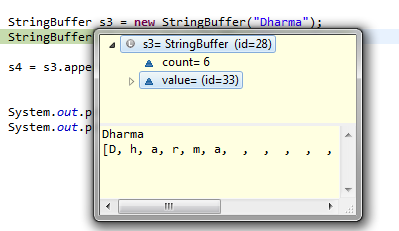
****

****

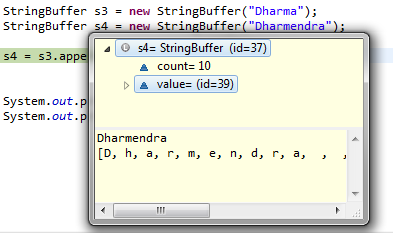
****

**String have each object memory pool id, objects will be change for different values.**

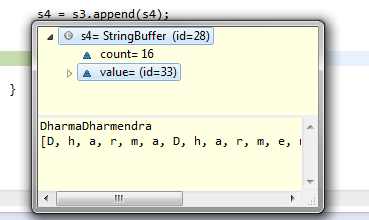
String Buffer mutable examble : Pic 1

****

**Pic 2:**

****

**Pic: 3**

****

**We have reassign the memory id , object in StringBuffer ex: Picture: 1,3 have same memory id,values id.**

**53) How sessions are handled in your project?**

## Maintaining Client State

Many applications require that a series of requests from a client be associated with one another. For example, a web application can save the state of a user’s shopping cart across requests. Web-based applications are responsible for maintaining such state, called a **session**, because HTTP is stateless. To support applications that need to maintain state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions.

### Accessing a Session

Sessions are represented by an HttpSession object. You access a session by calling the getSession method of a request object. This method returns the current session associated with this request; or, if the request does not have a session, this method creates one.

### Associating Objects with a Session

You can associate object-valued attributes with a session by name. Such attributes are accessible by any web component that belongs to the same web context **and** is handling a request that is part of the same session.

Recall that your application can notify web context and session listener objects of servlet lifecycle events ([Handling Servlet Lifecycle Events](http://docs.oracle.com/javaee/6/tutorial/doc/bnafi.html#bnafj)). You can also notify objects of certain events related to their association with a session such as the following:

* When the object is added to or removed from a session. To receive this notification, your object must implement the javax.servlet.http.HttpSessionBindingListener interface.
* When the session to which the object is attached will be passivated or activated. A session will be passivated or activated when it is moved between virtual machines or saved to and restored from persistent storage. To receive this notification, your object must implement the javax.servlet.http.HttpSessionActivationListener interface.

### Session Management

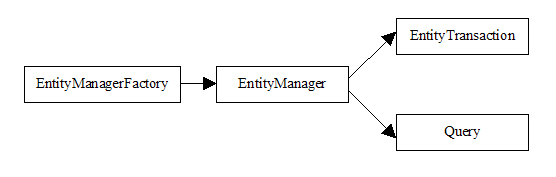
Because an HTTP client has no way to signal that it no longer needs a session, each session has an associated timeout so that its resources can be reclaimed. The timeout period can be accessed by using a session’s getMaxInactiveInterval and setMaxInactiveInterval methods.

* To ensure that an active session is not timed out, you should periodically access the session by using service methods because this resets the session’s time-to-live counter.
* When a particular client interaction is finished, you use the session’s invalidate method to invalidate a session on the server side and remove any session data.

## Session tracking methods:

1. User authorization
2. Hidden fields
3. URL rewriting
4. Cookies
5. Session tracking API

**54) How to connect to DB using JPA?**



[EntityManagerFactory](http://www.objectdb.com/api/java/jpa/EntityManagerFactory) emf =

[Persistence](http://www.objectdb.com/api/java/jpa/Persistence).[createEntityManagerFactory](http://www.objectdb.com/api/java/jpa/Persistence/createEntityManagerFactory_String)("myDbFile.odb");

Another form of the [createEntityManagerFactory](http://www.objectdb.com/api/java/jpa/Persistence/createEntityManagerFactory_String_Map) method takes a map of [persistence unit properties](http://www.objectdb.com/java/jpa/entity/persistence-unit#persistence.xml) as a second parameter:

Map<String, String> properties = new HashMap<String, String>();

properties.put("javax.persistence.jdbc.user", "admin");

properties.put("javax.persistence.jdbc.password", "admin");

[EntityManagerFactory](http://www.objectdb.com/api/java/jpa/EntityManagerFactory) emf = [Persistence](http://www.objectdb.com/api/java/jpa/Persistence).[createEntityManagerFactory](http://www.objectdb.com/api/java/jpa/Persistence/createEntityManagerFactory_String_Map)(

"objectdb://localhost:6136/myDbFile.odb", properties);

When the application is finished using the EntityManagerFactory it has to be closed:

emf.[close](http://www.objectdb.com/api/java/jpa/EntityManagerFactory/close)();

To connect to an ObjectDB server [registered username and password](http://www.objectdb.com/java/jpa/setting/users#The_user_elements) have to be specified:

[EntityManagerFactory](http://www.objectdb.com/api/java/jpa/EntityManagerFactory) emf = [Persistence](http://www.objectdb.com/api/java/jpa/Persistence).[createEntityManagerFactory](http://www.objectdb.com/api/java/jpa/Persistence/createEntityManagerFactory_String)(

"objectdb://localhost/myDbFile.odb;user=admin;password=admin");

o obtain a connection to an empty database (discarding existing content if any) the drop parameter has to be specified:

[EntityManagerFactory](http://www.objectdb.com/api/java/jpa/EntityManagerFactory) emf =

[Persistence](http://www.objectdb.com/api/java/jpa/Persistence).[createEntityManagerFactory](http://www.objectdb.com/api/java/jpa/Persistence/createEntityManagerFactory_String)("objectdb:myDbFile.tmp;drop");

## EntityManager

An [EntityManager](http://www.objectdb.com/api/java/jpa/EntityManager) instance may represent either a remote connection to a remote database server (in client-server mode) or a local connection to a local database file (in embedded mode). The functionality in both cases is the same. Given an [EntityManagerFactory](http://www.objectdb.com/api/java/jpa/EntityManagerFactory) emf, a short term connection to the database might have the following form:

[EntityManager](http://www.objectdb.com/api/java/jpa/EntityManager) em = emf.[createEntityManager](http://www.objectdb.com/api/java/jpa/EntityManagerFactory/createEntityManager)();

try {

// TODO: Use the EntityManager to access the database

}

finally {

em.[close](http://www.objectdb.com/api/java/jpa/EntityManager/close)();

}

## EntityTransaction

Operations that affect the content of the database (store, update, delete) must be performed within an active transaction. The [EntityTransaction](http://www.objectdb.com/api/java/jpa/EntityTransaction) interface represents and manages database transactions. Every [EntityManager](http://www.objectdb.com/api/java/jpa/EntityManager) holds a single attached EntityTransaction instance that is available via the [getTransaction](http://www.objectdb.com/api/java/jpa/EntityManager/getTransaction) method:

try {

em.[getTransaction](http://www.objectdb.com/api/java/jpa/EntityManager/getTransaction)().[begin](http://www.objectdb.com/api/java/jpa/EntityTransaction/begin)();

// Operations that modify the database should come here.

em.[getTransaction](http://www.objectdb.com/api/java/jpa/EntityManager/getTransaction)().[commit](http://www.objectdb.com/api/java/jpa/EntityTransaction/commit)();

}

finally {

if (em.[getTransaction](http://www.objectdb.com/api/java/jpa/EntityManager/getTransaction)().[isActive](http://www.objectdb.com/api/java/jpa/EntityTransaction/isActive)())

em.[getTransaction](http://www.objectdb.com/api/java/jpa/EntityManager/getTransaction)().[rollback](http://www.objectdb.com/api/java/jpa/EntityTransaction/rollback)();

}

**55) What is NamedQuery and Native Query in JPA?**

**@NamedNativeQuery** lets you write a named SQL query, while **@NamedQuery** lets you write a named HQL query (or JPQL).

In general, you should prefer to write HQL queries because then you can let Hibernate handle the intricacies of converting the HQL into the various SQL dialects. This will make your job much simpler when you choose to switch DBMS providers.

**56) Difference b/w join and subquery in Database?**

**Subquery:**

SELECT last\_name

FROM employee

WHERE employee\_number IN

-- this is also a subquery:

(SELECT manager\_employee\_number

FROM department)

subquery can be used for simple logical queries , to get  
required output/condition  
  
but if you want to compare two or more tables and get output  
in the way you want it would be lot difficult to manage all  
those with subqueries - so we can use the concept of joins

**57) What is JQuery? usage ?**

jQuery is a popular open-source JavaScript framework that Web developers use to solve basic app development problems across all browsers while opening the code to the public for community-driven development and support. jQuery's design lets developers extend the core libraries with new projects and plugins to add new features and functionality.

For Force.com developers, jQuery simplifies user interface development by simplifying basic functions like DOM manipulations and giving access to a large library of UI elements and techniques for technologies including AJAX and mobile devices. jQuery can be used with any Visualforce or Site.com page.

[jQuery](http://jquery.com) itself is the core jQuery library which handles basic functionality like DOM manipulation and finding specific nodes within a page. Using jQuery, it becomes simple to find and control elements on the page. For instance, jQuery (in this example, defined as "j$ by noConlflict()") can find an element or array of elements by looking for specific CSS selectors, and then hide all the elements found by the selector in a single method call. Here, we'll hide all elements with the class "accountDiv":

|  |  |
| --- | --- |
| 1 | j$ = jQuery.noConflict(); |
| 2 | j$('.accountDiv').hide(); | |

**58) Features in Java 5?**



**59) Features in java 6?**

 Pluggable Annotation Processing API (JSR 269)

 Common Annotations (JSR 250)

 Java API for XML Based Web Services – 2.0 (JSR 224)

 JAXB 2.0 (JSR 222)

 Web Services Metadata (JSR 181)

 Streaming API for XML (JSR 173)

 XML Digital Signature (JSR 105)

 Java Class File Specification Update (JSR 202)

 Java Compiler API (JSR 199)

 JDBC 4.0 (JSR 221)

 Scripting in the Java Platform (JSR 223)

**59) Features in java 7?**

The important features of JavaSE 7 are try with resource, catching multiple exceptions etc.

String in switch statement (Java 7)

Binary Literals (Java 7)

The try-with-resources (Java 7)

Caching Multiple Exceptions by single catch (Java 7)

Underscores in Numeric Literals (Java 7)

**60) Features in java 8?**

[Lambda Expressions](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-lambda-expressions)

[Default Methods](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-default-methods)

[Streams](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-streams)

[Nashorn](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-nashorn) JavaScript

[New Date and Time API](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-new-date-and-time-api)

[No More Permanent Generation](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-no-more-permanent-generation)

[Base64](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-base64)

[Annotations on Java Types](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-annotations-on-java-types)

[Repeating Annotations](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-repeating-annotations)

[Functional Programming in Java 8](https://leanpub.com/whatsnewinjava8/read#leanpub-auto-functional-programming-in-java-8).

**61) Difference between JDK, JRE, JVM?**

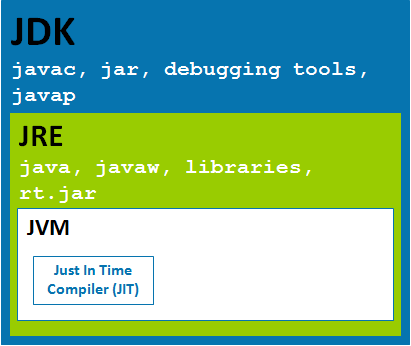
**JDK**

Java Developer Kit contains tools needed to develop the Java programs, and **JRE** to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Appletviewer, etc… Compiler converts java code into byte code. Java application launcher opens a **JRE**, loads the class, and invokes its main method.

You need **JDK**, if at all you want to write your own programs, and to compile the m. For running java programs, JRE is sufficient. JRE is targeted for execution of Java files i.e**. JRE = JVM +** **Java Packages Classes(**like util, math, lang, awt,swing etc**)+**runtime libraries. JDK is mainly targeted for java development. I.e. You can create a Java file (with the help of Java packages), compile a Java file and run a java file.

**JRE**

Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc. Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any java program, you need to have JRE installed in the system



The Java Virtual Machine provides a platform-independent way of executing code; programmers can concentrate on writing software, without having to be concerned with how or where it will run. But, note that JVM itself not a platform independent. It only helps Java to be executed on the platform-independent way. When JVM has to interpret the byte codes to machine language, then it has to use some native or operating system specific language to interact with the system. One has to be very clear on platform independent concept. Even there are many JVMs written on Java, however hey too have little bit of code specific to the operating systems.

**JVM**

As we all aware when we compile a **Java file**, output is not an ‘exe’ but it’s a ‘.class’ file. **‘.class’** file consists of **Java byte codes** which are understandable by **JVM**. Java Virtual Machine interprets the byte code into the machine code depending upon the underlying operating system and hardware combination. It is responsible for all the things like garbage collection, array bounds checking, etc… JVM is platform dependent.

The **JVM** is called “virtual” because it provides a machine interface that does not depend on the underlying operating system and machine hardware architecture. This independence from hardware and operating system is a cornerstone of the write-once run-anywhere value of Java programs.

There are different JVM implementations are there. These may differ in things like performance, reliability, speed, etc. These implementations will differ in those areas where Java specification doesn’t mention how to implement the features, like how the garbage collection process works is JVM dependent, Java spec doesn’t define any specific way to do this.

**62) Difference between Heap , Stack memory?**

1) Main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call, while heap memory is used to store objects in Java. No matter, where object is created in code e.g. as member variable, local variable or class variable,  they are always created inside heap space in Java.

2) Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own stack which can be specified using**-Xss**JVM parameter, similarly you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of heap and -Xmx is maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).

3) If there is no memory left in stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError  in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).

**63) Types of out of memory space and how to resolve?**

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and JVM throws java.lang.OutOfMemoryError when it ran out of memory in heap. OutOfMemoryError in Java can come any time in heap mostly while you try to create an object and there is not enough space in heap to allocate that object. [javavdoc of OutOfMemoryError](http://download.oracle.com/javase/6/docs/api/) is not very informative about this though.

**Types of OutOfMemoryError in Java**

 Java.lang.OutOfMemoryError: Java heap space  
Java.lang.OutOfMemoryError: PermGen space  
  
**Difference between "java.lang.OutOfMemoryError: Java heap space" and "java.lang.OutOfMemoryError: PermGen space"**  
  
If you are familiar with different generations on heap and [How garbage collection works in java](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html) and aware of new, old and permanent generation of heap space then you would have easily figured out this **OutOfMemoryError** in Java. Permanent generation of heap is used to store String pool and various Meta data required by JVM related to Class, method and other java primitives. Since in most of JVM default size of Perm Space is around **"64MB"** you can easily ran out of memory if you have too many classes or huge number of Strings in your project. Important point to remember is that it doesn't depends on –Xmx value so no matter how big your total heap size you can ran OutOfMemory in perm space. Good think is you can specify size of permanent generationusing JVM options **"-XX:PermSize"** and  **"-XX:MaxPermSize"** based on your project need.  
  
One small thing to remember is that **"="** is used to separate parameter and value while specifying size of perm space in heap while **"="** is not required while [setting maximum heap size in java](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html), as shown in below example.  
  
export **JVM\_ARGS="-Xmx1024m -XX:MaxPermSize=256m"**

Another reason **of "java.lang.OutOfMemoryError: PermGen"** is memory leak through Classloaders and it’s very often surfaced in WebServer and application server like tomcat, webshere, glassfish or weblogic. In Application server different classloaders are used to load different web application so that you can deploy and undeploy one application without affecting other application on same server, but while undeploying if container some how keeps reference of any class loaded by application class loader than that class and all other related class will not be garbage collected and can quickly fill the PermGen space if you deploy and undeploy your application many times**. "java.lang.OutOfMemoryError: PermGen”** has been observed many times in tomcat in our last project but solution of this problem are really tricky because first you need to know which class is causing memory leak and then you need to fix that. Another reason of OutOfMemoryError in PermGen space is if any thread started by application doesn't exit when you undeploy your application.

These are just some example of infamous classloader leaks, anybody who is writing code for loading and unloading classes have to be very careful to avoid this. You can also use visualgc for monitoring PermGen space, this tool will show graph of PermGen space and you can see how and when Permanent space getting increased. I suggest using this tool before reaching to any conclusion.

Another rather unknown but interesting cause **of "java.lang.OutOfMemoryError: PermGen"** we found is introduction of JVM options **"-Xnoclassgc".** This option sometime used to avoid loading and unloading of classes when there is no further live references of it just to avoid performance hit due to frequent loading and unloading, but using this option is J2EE environment can be very dangerous because many framework e.g. Struts, spring etc uses reflection to create classes and with frequent deployment and undeployment you can easily ran out of space in PermGen if earlier references was not cleaned up. This instance also points out that some time bad JVM arguments or configuration can cause **OutOfMemoryError** in Java.

So conclusion is avoid using **"-Xnoclassgc"** in J2EE environment especially with AppServer.

**64) What is Heap Dump how to analysis?**

Memory dumps are usually something that you don’t want to deal with. In a perfect Java application, normally everything should run fine and it will never run out of memory or misbehave. Unfortunately, such a perfect application does not exist, and chances are likely that you will run into “OutOfMemory” exceptions at some point or another. Memory dumps are a very useful feature of the JVM to analyze the contents of the memory at any given time, but their usage requires some experience, and in this post I will share with you some tips and tricks that I’ve learned over the years, so hopefully they will be useful to you too.

In a [previous blog post](http://www.jahia.com/home/download/jahia-academy/jahias-blog/blog-posts/analyzing-system-performance-wit.html), I illustrated how you could use the YourKit Profiler to analyze overall system performance, but I didn’t go into much detail concerning the generation and analysis of memory dumps. Now we will try to explain how you can generate these memory dumps and especially what pitfalls to avoid when transferring and analyzing them.

The dreaded “OutOfMemory” exception is generated when the application reaches the maximum allowed heap configured on the command line using the “-Xmx” option (or when the default size is reached if there is no such command line option).

# Generating memory dumps

There are different ways of generating memory dumps, and we will quickly list them.

Have them automatically generated when an OutOfMemory exception occurs. This is usually the most useful way of generating memory dumps, although you should be made aware that since memory dumps are a disk intensive task, it will pause the JVM for quite a long time while dumping memory contents, and if the application is under heavy load, it might even crash after (or sometimes even during) the generation of the memory dump file. You can activate the automatic generation on the JVM command line, by using the following option :  
-XX:+HeapDumpOnOutOfMemoryError  
There are also other options that you can use to control automatic memory dump generation, such as -XX:HeapDumpPath=path\_to\_file which will allow you to override the default name (usually something like java\_pid.hprof). Depending on the JVM version there might be even some newer options, so you can check the [official command line documentation](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#DebuggingOptions) for more information. It is usually acceptable to use this setting on a production server since normally it shouldn’t reach out of memory scenarios, and if it does, you will want to understand what happened.

Using the jmap command line tool, as in the following example:  
jmap -dump:file=path\_to\_file java\_process\_id

Use a JVM profiler such as YourKit and use it’s own built-in memory dump feature

(There is actually a [fourth method](http://blog.codecentric.de/en/2008/07/memory-analysis-part-1-obtaining-a-java-heapdump/) but it is a bit more tricky to do using jconsole).

Once you have generated the dump, depending on the method used, you will find the dump file either at the specified path location or if you didn’t specify a location it should be in the JVM’s working directory. If, for example, you launched Tomcat from the bin/startup.sh script, it will probably be located in the bin/ subdirectory.

**65) Iterator and for loop difference ?**

**for loop**

for (Iterator<DataType> it = list.iterator(); list.hasNext(); )

In this case, you are declaring the Iterator object locally and it will be **eligible fot GC after the for loop.**

**In while loop,**

while (it.hasNext()) since you have declared the object Iterator outside a loop. So its **scope is may be** the entire program or the method it is in. Or incase if it is referenced anywhere, so it**wont be eligible for GC**.

**66) What is young generation and old generation in JVM?**

Sizing the young generation is the most important tuning step in a generational GC setup. First, it must be large enough to handle all threads of the concurrent working set without tenuring short-lived objects to the old generation. Second, we want long-lived objects tenured as quickly as possible. Remember, the more objects remain live, the more expensive the GC cycle!

## Young Generation Sizing

We're trying to achieve an efficient balance between two extremes, and we'll start by defining a large enough young generation to prevent the old generation growing too fast under load (at least for the 90% of the time that matters most).

For the Oracle Hotspot JVM, try using a throughput collector with adaptive sizing:

-XX:+UseAdaptiveSizePolicy

Pause Goal: -XX:MaxGCPauseMillis

Throughput Goal: -XX:GCTimeRatio

You need to optimize for the desired survivor size, so keep an eye on the survivor spaces. After a young-generation GC, one of the two survivor spaces should not be filled more than 75%. At the same time, the old generation should not grow. If adaptive sizing isn't working, adjust the young generation’s size manually.

The IBM WebSphere JVM and Oracle JRockit need to be sized manually. The IBM WebSphere JVM has two areas in the young generation instead of three, but the same target is still valid. The survivor area should not be filled more than 75% after a young-generation GC and the old generation should not grow.

Next, we need to assure that there are not so many live objects in the young generation that a minor GC becomes expensive. For the generational GC, middle-lived objects can be problematic. If they stick around too long in the young generation, they lead to long GC cycles. If we tenure them, the old generation grows, eventually bringing about its own excessive GCs.

By monitoring garbage collections under full load we can analyze the situation and find out what's going on. If young-generation GCs become too expensive while the old generation does not grow, we have too many live objects in the young generation. In the HotSpot JVM we can tune the tenure threshold, which defines how often an object must survive a GC cycle before it gets tenured. This enables us to tenure objects sooner while making sure that temporary objects die in the young generation. The effect should be a faster young-generation GC and a slightly growing old generation. In the IBM WebSphere and JRockit JVMs, all we can do is shrink the young generation by resizing.

If we have to accept a spillover of middle-lived objects to the old generation, we should use the CMS old-generation GC to deal with it. However if that spillover becomes too great we will experience very expensive major GCs. In such a case GC tuning alone does not suffice anymore and we need to optimize the transactional memory usage of the application with an allocation analysis.

## Old-Generation Tuning

Once you've achieved near perfection in the young-generation size, you know the old generation isn't going to grow under load. This makes it relatively easy to discover the optimal size of the old generation. Check your application's utilization of the old generation after the initial warm-up phase and note the value. Reconfigure your JVM so that the old generation’s size is 25% bigger than this observed value to serve as a buffer. For the Oracle HotSpot JVM, you must size the buffer to be at least as big as Eden plus one survivor to accommodate[HotSpot's young-generation guarantee](http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html#0.0.0.%2520The%2520Young%2520Generation%257Coutline) (check the HotSpot documentation for further details).

If you were unable to achieve optimal young-generation sizing—sometimes it simply is not possible—things become more complicated.

If your application is response time–oriented, you will want to use a concurrent GC in the old generation. In this case you need to tune both the concurrent GC thresholds and the size of the old generation so that the average fill state is never higher than 75%. Again, the 25% extra represents headroom needed for the concurrent GC. If the old generation fills up too much, the CMS will not be able to free up memory fast enough. In addition, fragmentation can easily lead to allocation errors. Both cases will trigger a real major GC, stopping the entire JVM.

If your application is throughput-oriented, things are not as bad. Use a parallel GC with compaction and make the old generation big enough to accommodate all concurrent running transactions. Without this, the GC will not free sufficient memory to support this level of application concurrency. If necessary, it's okay to expand the headroom a bit, keeping in mind that more memory only delays a GC—it cannot prevent it.

**67) Any Continuous Integration tool?**

In modern enterprise development, a complex project scope is very common. In this article, I will talk about the latest continuous instigation tools such as Apache's Continuum, Cruise Control (CC), and Hudson; all of them are designed to deal with project complexity, streamline build process, and report issues with the code as soon as they occur. They are perfect for use in an enterprise environment with large to medium size teams, may require a dedicated machine, and provide visual dashboards. These tools also enforce the process of continuous integration and are tightly coupled with a source control system, such as CVS, Subversion, and so forth. They may also simplify code maintenance and reduce the number of the issues that need to be fixed during the QA cycles.

**The Continuous Integration Process Defined**

The term "continuous integration" refers to a process that builds and tests code on a frequent basis. It was coined by Martin Fowler and Kent Beck, who first wrote about this process near the turn of the millennium.

The continuous integration servers constantly monitor source code repositories and as soon as new changes/commits are detected, they initiate a new build cycle. The build cycle actually involves code compilation and, in addition, may involve various tests and code analysis. If the process encounters errors, it may notify the build master or the culprit who checked in broken/invalid code.

**The process can be summarized in these four steps:**

Team members check in code artifacts into the source control repository.

The automated build server constantly monitors the repository.

New code is continuously checked out [by this server].

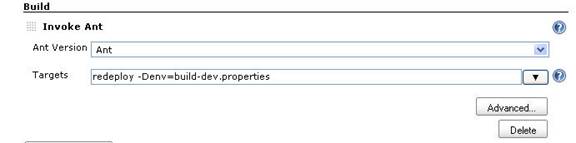
A new project build is continuously integrated and any issues are reported in real time.

There is much more to this concept and to the actual implementation of these automated build servers&mdahs;or continuous integration tools—such as definitions of best practices for the code structure, check-in locations, and so on. For instance, the process specifies that all working code should be in the main branch (trunk) of the code repository, and even though various tools can work with multiple branches, it is not recommended to do that.

**Continuous Integration Tools**

All continuous integration tools consist of a core engine that is designed to monitor and check out various artifacts that are needed to build the project from the source repository. On top of that, some tools can run automated scripts that can be extended to do much more robust functions then just code compilation.

For example, if the tool can run ANT scripts, developers may write their own script to FTP successful builds to a specific location, at the end of the build cycle.

[[](http://www.developer.com/img/2009/02/Continuous01.jpg)](http://www.developer.com/img/2009/02/Continuous01.jpg)

**Figure 1: Hudson Ant Configuration**

The core of any continuous integration server is usually exposed via a UI control panel or dashboard (web-based or stand alone). Depending on the vendor, the tools also can offer other features, such as analysis of the code, execution of ANT scripts, configuration via command line or XML, statistics on the code quality, RSS feeds to see various processes and their outcome, and email functionality on the status of the build to the build master or specific people who broke the code.

As I mentioned, most of these tools support and tightly integrate with the most popular source control repositories, such as CVS, Subversion, SourceSafe, and the like to continuously monitor for the new code changes.

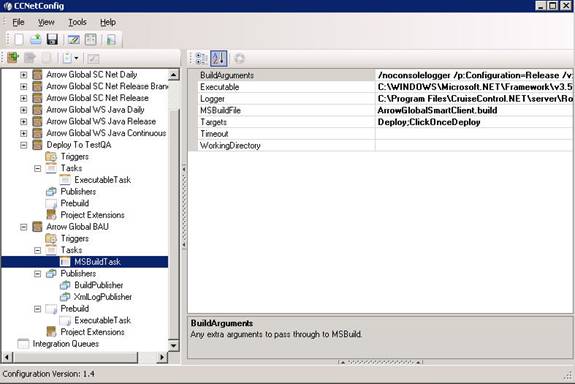
**Cruise Control, Continuum, and Hudson**

The most popular tools are Apache's Continuum, Cruise Control (CC), and Hudson. They are all free open source tools and have varying developer base support. All three tools have the ability to build and test code per a configurable schedule.

Hudson and Apache's Continuum are primarily designed for Java, and Cruise Control (CC) supports both Java and .Net (with the CC.Net version). Because these tools are designed to be very flexible, it's possible to extend them to support build processes in other development languages. For example, Hudson can be extended via plug-ins to work with C#, Python, Maven, Ruby, and others.

**Cruise Control (CC)**

Cruise Control is a free continuous integration tool designed for Java, but there is a version designed for .NET as well (Cruise Control .Net). All configurations are done via an XML file and there is a web dashboard component that can be installed to view the build progress. Compared to the other tools, CC does not have a configuration UI. This means that you have to look for special editors that can work with CC's config file if you don't want to edit XML by hand. (You can install CCNetConfig program, which helps somewhat with the XML writing). CC also has the most primitive web UI, and high learning curve; however, once installed and properly configured, it proves to be very powerful.



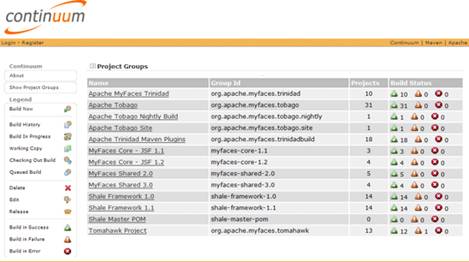
**Apache's Continuum**

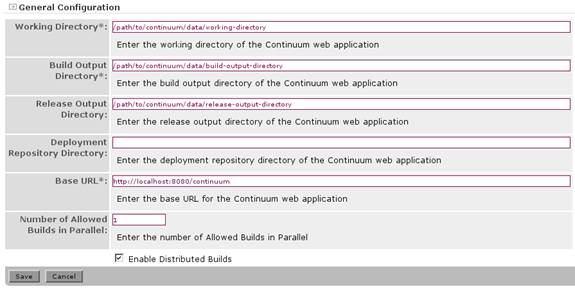
Apache Continuum is an open source continuous integration server that is designed primarily for Java builds. One feature that sets it apart from the other tools is the role-based security. This feature gives you control on what aspects of the build can be seen by specific users. All other main features are similar to Hudson and CC, such as release management, email notification, and integration with popular build tools, such as Maven, and source control management systems, like SVN or CVS.

To install this server, you need to download the tar file and extract its contents to a directory on the machine that will be a build server. Java Runtime also has to be installed on the same box. The initial configuration of Apache Continuum is similar to Cruise Control's, and you need to get your hands dirty with some XML editing. The main file, continuum.xml, is in the conf/ directory.

The operation is fairly straightforward: "continuum start" to start, "continuum status" to check status, and so forth.

The tool's UI is web based, and after the server is up the dashboard can be accessed via a local host link (http://localhost:8080/continuum).





**68) How to configure multiple database in hibernate?**

The sample we discuss here shows how an Employee object can be configured to store in both Oracle Data base and Derby Database. Here we create a POJO class called Employee and store and retrieve its objects from both Oracle and Derby database

|  |  |  |
| --- | --- | --- |
| Employee.java  package hibernatepack.samples; | | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | public class Employee { | |
| 4 | private int empid; |

|  |  |  |
| --- | --- | --- |
| 5 | private String empname; | |
| 6 | private double salary; |

|  |  |  |
| --- | --- | --- |
| 7 | public int getEmpid() { | |
| 8 | return empid; |

|  |  |  |
| --- | --- | --- |
| 9 | } | |
| 10 | | public void setEmpid(int empid) { | |

|  |  |  |
| --- | --- | --- |
| 11 | this.empid = empid; | |
| 12 | } |

|  |  |
| --- | --- |
| 13 |  |
| 14 | public String getEmpname() { | |

|  |  |  |
| --- | --- | --- |
| 15 | return empname; | |
| 16 | } |

|  |  |
| --- | --- |
| 17 |  |
| 18 | public void setEmpname(String empname) { | |

|  |  |  |
| --- | --- | --- |
| 19 | this.empname = empname; | |
| 20 | } |

|  |  |
| --- | --- |
| 21 |  |
| 22 | public double getSalary() { | |

|  |  |  |
| --- | --- | --- |
| 23 | return salary; | |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | public void setSalary(double salary) { | |

|  |  |  |
| --- | --- | --- |
| 27 | this.salary = salary; | |
| 28 | } |

|  |  |
| --- | --- |
| 29 | } |
|  |  |

**Employee.hbm.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="hibernatepack.samples.Employee" table="HBEmployeeDetails" >

<id name= "empid" column="EmpNo" />

<property name= "empname" column = "EmpName" />

<property name="salary" column="Salary" />

</class>

</hibernate-mapping>

**oracleconfig.cfg.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>

<property name="hibernate.connection.driver\_class">oracle.jdbc.OracleDriver</property>

<property name="hibernate.connection.url">jdbc:oracle:thin:@10.154.117.76:1521:oracle</property>

<property name="hibernate.connection.username">user</property>

<property name="hibernate.connection.password">password</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<property name="hibernate.show\_sql">true</property>

<mapping resource="Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>

**derbiconfig.cfg.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.DerbyDialect</property>

<property name="hibernate.connection.driver\_class">org.apache.derby.jdbc.ClientDriver</property>

<property name="hibernate.connection.url">jdbc:derby://localhost:1527/HibernateDB</property>

<property name="hibernate.connection.username">user</property>

<property name="hibernate.connection.password">pwd</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<mapping resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

package hibernatepack.samples;

import java.util.List;

public interface IEmployeeDAO {

public void findAllEmployees();

public void insertEmployee(Employee e);

}

**EmloyeeDaoImpl.java**

public class EmployeeDaoImpl implements IEmployeeDAO {

**SessionFactory sessionFactory1 = new Configuration().configure("oracleconfig.cfg.xml").buildSessionFactory();**

**SessionFactory sessionFactory2 = new Configuration().configure("derbyconfig.cfg.xml").buildSessionFactory();**

Session session = null;

Transaction transaction = null;

public void findAllEmployees() {

ArrayList empList = new ArrayList();

try {

session **= sessionFactory1.openSession();**

transaction = session.beginTransaction();

transaction.begin();

Criteria crit = session.createCriteria(Employee.class);

empList = (ArrayList) crit.list();

System.out.println("Records from Oracle Database");

for (Employee emp : empList) {

System.out.println(emp.getEmpid() + " " + emp.getEmpname() + " " + emp.getSalary());

}

session.close();

session = **sessionFactory2.openSession();**

Criteria crit1 = session.createCriteria(Employee.class);

empList = (ArrayList) crit1.list();

System.out.println("Records from Derby Database");

for (Employee emp : empList) {

System.out.println(emp.getEmpid() + " " + emp.getEmpname() + " " + emp.getSalary());

}

session.close();

} catch (Exception he) {

he.printStackTrace();

}

}

public void insertEmployee(Employee e) {

try {

session = sessionFactory1.openSession();

transaction = session.beginTransaction();

transaction.begin();

session.save(e);

transaction.commit();

session.close();

session = sessionFactory2.openSession();

transaction = session.beginTransaction();

transaction.begin();

session.save(e);

transaction.commit();

session.close();

} catch (HibernateException he) {

he.printStackTrace();

}

}

}

EmployeeTest.java

package hibernatepack.samples;

import java.awt.Choice;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class EmployeeTest {

private static int choice;

public static void main(String[] args) {

EmployeeDaoImpl empOperations = new EmployeeDaoImpl();

Employee e1 = new Employee();

do {

System.out.println("1. Insert ");

System.out.println("2. List ");

System.out.println("3. Exit ");

System.out.println("Enter your choice ");

Scanner sc = new Scanner(System.in);

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("Enter the employee Number ");

Scanner sc1 = new Scanner(System.in);

int empid = sc1.nextInt();

System.out.println("Enter the employee Name ");

Scanner sc2 = new Scanner(System.in);

String empname = sc2.nextLine();

System.out.println("Enter the Salary ");

Scanner sc3 = new Scanner(System.in);

double empsal = sc3.nextDouble();

e1.setEmpid(empid);

e1.setEmpname(empname);

e1.setSalary(empsal);

empOperations.insertEmployee(e1);

break;

case 2:

empOperations.findAllEmployees();

break;

}

} while (choice != 3);

}

}

**We can use annotation also: Below the link**

[**http://java.dzone.com/articles/connecting-multiple-databases**](http://java.dzone.com/articles/connecting-multiple-databases)

**69) What is projection in Hibernate?** In criteria, we are able to load complete object right….! let us see how to load the partial objects while working with criteria.  The projections concept is introduced in hibernate 3.0 and mainly we can do the following 2 operations using the projection

**We can load partial object from the database**

**We can find the Result of Aggregate functions**

Projection is an Interface given in “org.hibernate.criterion” package, Projections is an class given in same package,  actually Projection is an interface, and Projections is an class and is a factory for producing projection objects.

In Projections class, we have all static methods and each method of this class returns Projection interface object.

If we want to add a Projection object to Criteria then we need to call a method **setProjection()**

While adding projection object to criteria, it is possible to add one object at a time.  It means if we add 2nd projection object then this 2nd one will overrides the first one (first one wont be work), so at a time we can only one projection object to criteria object.

Using criteria, if we want to load partial object from the database, then we need to create a projection object for property that is to be loaded from the database

Criteria crit = session.createCriteria(Products.class);

crit.setProjection(Projections.proparty("proName"));

List l=crit.list();

Iterator it=l.iterator();

while(it.hasNext())

{

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

// ---- print -----

}

If we add multiple projections to criteria then the last projection added will be considered to execute see…

Criteria crit = session.createCriteria(Products.class);

Projection p1 = Projection.property("proName");

Projection p2 = Projection.property("price");

crit.setProjection(p1):

crit.setProjection(p2):

List l=crit.list();

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

------- - ---  ----

Here collections list l, is going to contain the price in the form of Double objects, but product names are over ridden,  second projection over rides the first one, i mean p2 only will works p1 will not works, actually there is a way to add multiple projections to criteria.

**70) Reflection access method, attributes of a class?**

**Accessing private methods,variables:**

PrivateMethodClass.java

**package** privatemethodaccess;

**public** **class** PrivateMethodClass {

**private** String obString = **null**;

**public** PrivateMethodClass(String obString) {

**this**.obString = obString;

}

@SuppressWarnings("unused")

**private** String getPrivateMethodCheck(){

System.*out*.println(obString);

**return** obString;

}

}

AccessPrivateMethodsClass.java

**package** privatemethodaccess;

**import** java.lang.reflect.InvocationTargetException;

**import** java.lang.reflect.Method;

**public** **class** AccessPrivateMethodsClass {

@SuppressWarnings("unused")

**public** **static** **void** main(String ...s) **throws** NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException{

PrivateMethodClass obPrivateMethodClass = **new** PrivateMethodClass("RAMA");

Method obMethod = PrivateMethodClass.**class**.getDeclaredMethod("getPrivateMethodCheck", **null**);

obMethod.setAccessible(**true**);

String obString = (String)obMethod.invoke(obPrivateMethodClass, **null**);

System.*out*.println(obString);

}

}

**71) Query for getting 2nd highest salary of an employee using Oracle, MYSQL?**

**Oracle:**

SELECT \* FROM Employee t1 WHERE ( 2 ) = (SELECT COUNT( t2. salary) FROM Employee t2 WHERE t2. salary >= t1. salary);

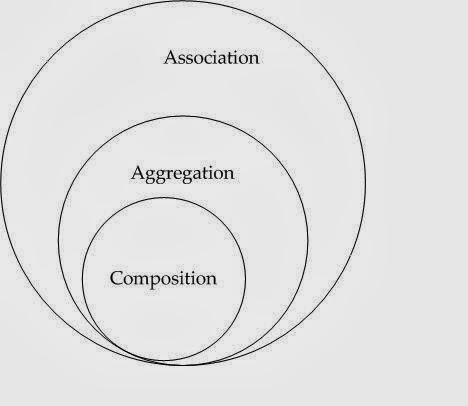
**MYSQL:**

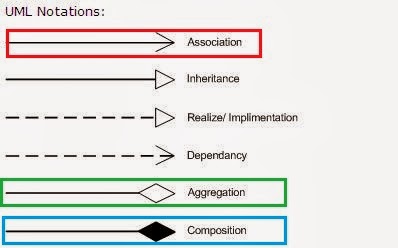
SELECT max(salary) FROM Employee WHERE salary NOT IN (SELECT max(salary) FROM Employee);

SELECT max(salary) FROM Employee WHERE salary < (SELECT max(salary) FROM Employee);

**72)** [**Difference between Association, Composition and Aggregation in Java, UML and Object Oriented Programming**](http://javarevisited.blogspot.in/2014/02/ifference-between-association-vs-composition-vs-aggregation.html) **?**

In Object-oriented programming, one object is related to other to use functionality and service provided by that object. This relationship between two object is known as association in  object oriented general software design, and depicted by an arrow in Unified Modelling language or UML. Both Composition and Aggregation are form of association between two objects, but there is subtle difference between composition and aggregation, which is also reflected by their UML notation. We refer association between two objects as composition, when one class owns other class and other class cannot meaningfully exist, when it's owner destroyed, for example Human class is composition of several body parts including Hand, Leg and Heart. When human object dies, all it's body part ceased to exist meaningfully, this is one example of Composition. Programmers often confuse between Association, Composition and Aggregation in Object oriented design discussions, this confusion also makes difference between Association, Composition and Aggregation one of the popular questions in Java Interviews, only after [difference between abstract class and interface](http://javarevisited.blogspot.com/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html) . Another example of Composition is Car and it's part e.g. engines, wheels etc. Individual parts of car can not function, when car is destroyed.  While in case of Aggregation, including object can exists without being part of main object e.g. a Player which is part of a Team, can exists without team and can become part of other teams as well. Another example of Aggregation is Student in School class, when School closed, Student still exist and then can join another School or so.  In UML notation, composition is denoted by a filled diamond, while aggregation is denoted by an empty diamond, which shows their obvious difference in terms of strength of relationship. Composition is more stronger than Aggregation.  In Short, relationship between two objects is referred as association, and an association is known as composition when one object owns other, while an association is known as aggregation when one object uses other object. In this OOPS tutorial, we will see couple of more examples to understand difference between Association, Composition and Aggregation better.





## Association

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

http://javapapers.com/wp-content/uploads/2010/06/association.jpg

## Aggregation

Aggregation is a special case of association. A directional association between objects. When an object ‘has-a’ another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a “Has-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/aggregation.jpg

## Composition

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

http://javapapers.com/wp-content/uploads/2010/06/composition.jpg

## 73) Explain IS-A and HAS-A relationship?

## IS-A Relationship:

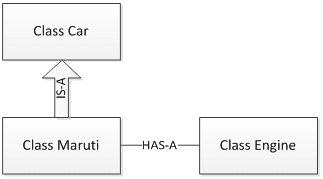
In object oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is uni-directional. For example House is a Building. But Building is not a House.

It is key point to note that you can easily identify the IS-A relationship. Wherever you see an extends keyword or implements keyword in a class declaration, then this class is said to have IS-A relationship.

## HAS-A Relationship:

Composition(HAS-A) simply mean use of instance variables that are references to other objects. For example: Maruti has Engine, or House has Bathroom.

Let’s understand these concepts with example of Car class.



**74) Difference between Comparator and comparable ?**

**omparator vs Comparable**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Comparable** | **Comparator** |
| Sorting logic | Sorting logic must be in same class whose objects are being sorted. Hence this is called natural ordering of objects | Sorting logic is in separate class. Hence we can write different sorting based on different attributes of objects to be sorted. E.g. Sorting using id,name etc. |
| Implementation | Class whose objects to be sorted must implement this interface.e.g Country class needs to implement comparable to collection of country object by id | Class whose objects to be sorted do not need to implement this interface.Some other class can implement this interface. E.g.-CountrySortByIdComparator class can implement Comparator interface to sort collection of country object by id |
| Sorting method | int compareTo(Object o1) This method compares this object with o1 object and returns a integer.Its value has following meaning 1. positive – this object is greater than o1 2. zero – this object equals to o1 3. negative – this object is less than o1 | int compare(Object o1,Object o2) This method compares o1 and o2 objects. and returns a integer.Its value has following meaning. 1. positive – o1 is greater than o2 2. zero – o1 equals to o2 3. negative – o1 is less than o1 |
| Calling method | Collections.sort(List) Here objects will be sorted on the basis of CompareTo method | Collections.sort(List, Comparator) Here objects will be sorted on the basis of Compare method in Comparator |
| Package | Java.lang.Comparable | Java.util.Comparator |

**75) Explain Database Normalization?**

**Normalization of Database**

Database Normalisation is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

**Eliminating reduntant(useless) data.**

**Ensuring data dependencies make sense i.e data is logically stored.**

**Problem Without Normalization**

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_id** | **S\_Name** | **S\_Address** | **Subject\_opted** |
| 401 | Adam | Noida | Bio |
| 402 | Alex | Panipat | Maths |
| 403 | Stuart | Jammu | Maths |
| 404 | Adam | Noida | Physics |

**Updation Anamoly :** To update address of a student who occurs twice or more than twice in a table, we will have to update **S\_Address** column in all the rows, else data will become inconsistent.

**Insertion Anamoly :** Suppose for a new admission, we have a Student id(S\_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.

**Deletion Anamoly :**If (S\_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

**Normalization Rule**

Normalization rule are divided into following normal form.

* 1. First Normal Form
  2. Second Normal Form
  3. Third Normal Form
  4. BCNF

**First Normal Form (1NF)**

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary** key is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

**Student Table :**

|  |  |  |
| --- | --- | --- |
| **Student** | **Age** | **Subject** |
| Adam | 15 | Biology, Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

**Student Table following 1NF will be :**

|  |  |  |
| --- | --- | --- |
| **Student** | **Age** | **Subject** |
| Adam | 15 | Biology |
| Adam | 15 | Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

**Second Normal Form (2NF)**

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form**.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is {**Student**, **Subject**}, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

**New Student Table following 2NF will be :**

|  |  |
| --- | --- |
| **Student** | **Age** |
| Adam | 15 |
| Alex | 14 |
| Stuart | 17 |

In Student Table the candidate key will be **Student** column, because all other column i.e **Age** is dependent on it.

**New Subject Table introduced for 2NF will be :**

|  |  |
| --- | --- |
| **Student** | **Subject** |
| Adam | Biology |
| Adam | Maths |
| Alex | Maths |
| Stuart | Maths |

In Subject Table the candidate key will be **{Student, Subject}** column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

**Third Normal Form (3NF)**

Third Normal form applies that every non-prime attribute of table must be dependent on primary key. Thetransitive functional dependency should be removed from the table. The table must be in Second Normal form. For example, consider a table with following fields.

**Student\_Detail Table :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student\_id** | **Student\_name** | **DOB** | **Street** | **city** | **State** | **Zip** |

In this table Student\_id is Primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called **transitive dependency**. Hence to apply **3NF**, we need to move the street, city and state to new table, with **Zip** as primary key.

**New Student\_Detail Table :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_id** | **Student\_name** | **DOB** | **Zip** |

**Address Table :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Zip** | **Street** | **city** | **state** |

The advantage of removing transitive dependency is,

Amount of data duplication is reduced.

Data integrity achieved.

**Boyce and Codd Normal Form (BCNF)**

**Boyce and Codd Normal Form** is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

### 76) How to Implementing Inheritance in Hibernate?

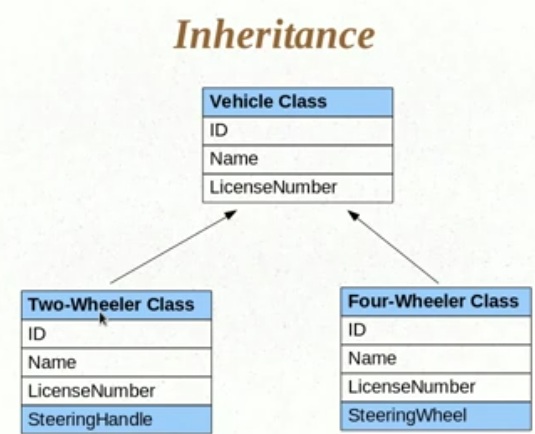
**1. Vertical Inheritance**

**2. Horizontal Inheritance**

**There are three types of inheritance mapping in hibernate**1. Table per concrete class with unions   
2. Table per class hierarchy(Single Table Strategy)   
3. Table per subclass

Example:   
Let us take the simple example of 3 java classes.   
Class **TwoWheelerVehicle** and **FourWheelerVehicle** are inherited from **Vehicle** Abstract class.

* 1. Table per concrete class with unions   
     In this case there will be 2 tables   
     Tables: **TwoWheelerVehicle**, **FourWheelerVehicle**[all common attributes will be duplicated]
  2. Table per class hierarchy   
     Single Table can be mapped to a class hierarchy   
     There will be only one table in database called **'Vehicle'** that will represent all the attributes required for all 3 classes.   
     But it needs some **discriminating** column to differentiate between **TwoWheelerVehicle** and  **FourWheelerVehicle**;
  3. Table per subclass   
     In this case there will be 3 tables represent **TwoWheelerVehicle** , **FourWheelerVehicle** and **Vehicle.**

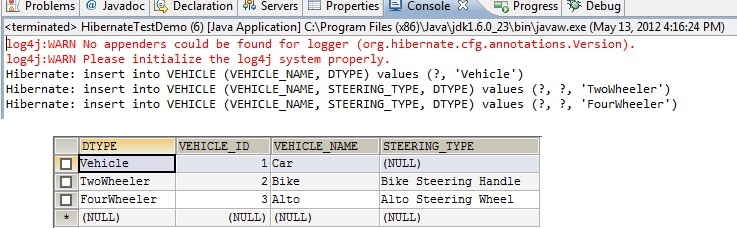


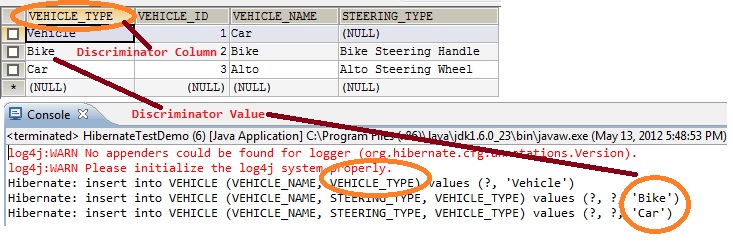
**Single Table Strategy,**

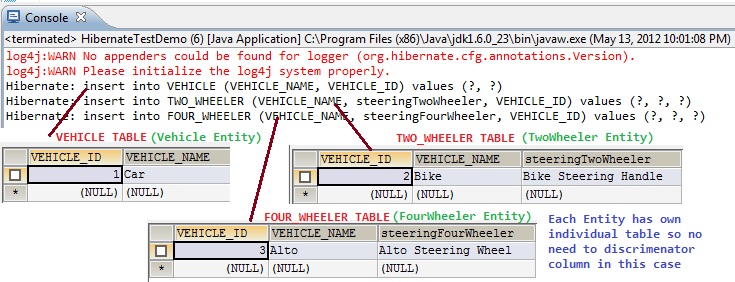
**With Table Per Class Strategy,**

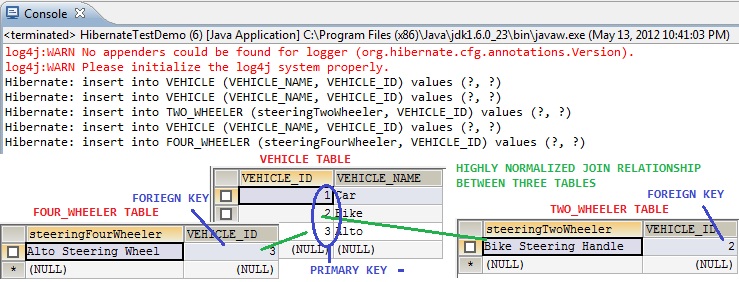
**With Joined Strategy**

<http://www.dineshonjava.com/p/implementing-inheritance-in-hibernate.html>









**77) What is JMS ?**

The **Java Management Extensions** (JMX) technology is a standard part of the Java Platform, Standard Edition (Java SE platform). The JMX technology was added to the platform in the Java 2 Platform, Standard Edition (J2SE) 5.0 release.

The JMX technology provides a simple, standard way of managing resources such as applications, devices, and services. Because the JMX technology is dynamic, you can use it to monitor and manage resources as they are created, installed and implemented. You can also use the JMX technology to monitor and manage the Java Virtual Machine (Java VM).

The JMX specification defines the architecture, design patterns, APIs, and services in the Java programming language for management and monitoring of applications and networks.

Using the JMX technology, a given resource is instrumented by one or more Java objects known as Managed Beans, or MBeans. These MBeans are registered in a core-managed object server, known as an MBean server. The MBean server acts as a management agent and can run on most devices that have been enabled for the Java programming language.

The specifications define JMX agents that you use to manage any resources that have been correctly configured for management. A JMX agent consists of an MBean server, in which MBeans are registered, and a set of services for handling the MBeans. In this way, JMX agents directly control resources and make them available to remote management applications.

The way in which resources are instrumented is completely independent from the management infrastructure. Resources can therefore be rendered manageable regardless of how their management applications are implemented.

The JMX technology defines standard connectors (known as JMX connectors) that enable you to access JMX agents from remote management applications. JMX connectors using different protocols provide the same management interface. Consequently, a management application can manage resources transparently, regardless of the communication protocol used. JMX agents can also be used by systems or applications that are not compliant with the JMX specification, as long as those systems or applications support JMX agents.

**78) What is JMS? Difference between Queue and Topic?**

**The Java Message Service (JMS)** [API](http://en.wikipedia.org/wiki/Application_Programming_Interface) is a [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [Message Oriented Middleware](http://en.wikipedia.org/wiki/Message_Oriented_Middleware) (MOM) API for sending messages between two or more [clients](http://en.wikipedia.org/wiki/Client_(computing)). JMS is a part of the [Java Platform, Enterprise Edition](http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition), and is defined by a specification developed under the [Java Community Process](http://en.wikipedia.org/wiki/Java_Community_Process) as JSR 914. It is a messaging standard that allows application components based on the Java Enterprise Edition (Java EE) to create, send, receive, and read messages. It allows the communication between different components of a [distributed application](http://en.wikipedia.org/wiki/Distributed_computing) to be [loosely coupled](http://en.wikipedia.org/wiki/Loose_coupling), reliable, and asynchronous.

**Queue VS Topic**  
**Queue:**

Point-to-point model.

Only one consumer gets the message.

Messages have to be delivered in the order sent.

A JMS queue only guarantees that each message is processed only once.

The Queue knows who the consumer or the JMS client is. The destination is known

.

The JMS client (the consumer) does not have to be  active or connected to the queue all the time to receive or read the message.

Every message successfully processed is acknowledged by the consumer.  
  
Descriptive example: A JMS queue is a channel through which users "pull" messages they want to receive using the p2p model, instead of automatically receiving messages on a particular topic. The producer submits messages to the queue, and recipients can browse the queue and decide which messages they wish to receive. In the p2p model, users can see the contents of the messages held in the queue before deciding whether or not to accept their delivery.

**Topic:**

Publish/subscribe model.

Multiple clients subscribe to the message.

There is no guarantee messages have to be delivered in the order sent.

There is no guarantees that each message is processed only once. -- As this can be sensed from the model .

The Topic, have multiple subscribers and there is a chance that the topic does not know all the subscribers. The destination is unknown.

The subscriber / JMS client needs to the active when the messages are produced by the producer, unless the subscription was a durable subscription.

No, Every message successfully processed is not acknowledged by the consumer/subscriber.

Descriptive Example: A JMS topic is the channel through which users subscribe to receive specific messages from a producer in the publish-and-subscribe model of JMS messaging. The model can be compared to subscribing to a newspaper; for example, if John Doe subscribed to "The New York Times," he would receive the paper every day from the newspaper producer. Similarly, if John Doe used JMS messaging to subscribe to a particular topic, he would receive all sent messages from a producer regarding that topic.

|  |  |
| --- | --- |
| |  | | --- | | **79) JDBC Driver Types?** | |

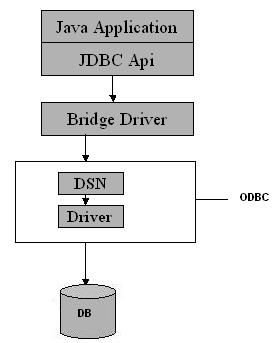
**JDBC drivers** are divided into four types or levels. The **different types of jdbc drivers** are:

**Type 1:** JDBC-ODBC Bridge driver (Bridge)  
**Type 2:** Native-API/partly Java driver (Native)  
**Type 3:** AllJava/Net-protocol driver (Middleware)  
**Type 4:** All Java/Native-protocol driver (Pure)

## Type 1 JDBC Driver

**JDBC-ODBC Bridge driver**

The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge driver is recommended only for experimental use or when no other alternative is available.



Type 1: JDBC-ODBC Bridge

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

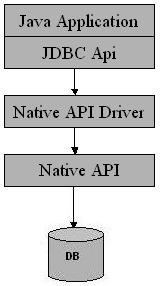
Disadvantages

1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.  
2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.  
3. The client system requires the ODBC Installation to use the driver.  
4. Not good for the Web.

## Type 2 JDBC Driver

**Native-API/partly Java driver**

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.



Type 2: Native api/ Partly Java Driver

Advantage

The distinctive characteristic of type 2 jdbc drivers are that they are typically offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type  
1 and also it uses Native api which is Database specific.

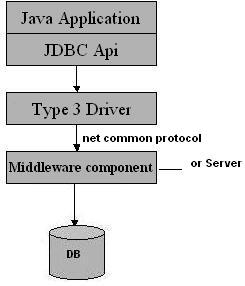
Disadvantage

1. Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.   
2. Like Type 1 drivers, it’s not written in Java Language which forms a portability issue.   
3. If we change the Database we have to change the native api as it is specific to a database  
4. Mostly obsolete now  
5. Usually not thread safe.

### Type 3 JDBC Driver

**All Java/Net-protocol driver**

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.



Type 3: All Java/ Net-Protocol Driver

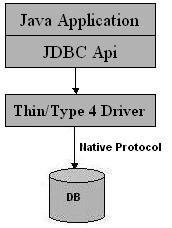
Advantage

1. This driver is server-based, so there is no need for any vendor database library to be present on client machines.  
2. This driver is fully written in Java and hence Portable. It is suitable for the web.  
3. There are many opportunities to optimize portability, performance, and scalability.   
4. The net protocol can be designed to make the client JDBC driver very small and fast to load.   
5. The type 3 driver typically provides support for features such as caching (connections, query results, and so on), load balancing, and advanced   
system administration such as logging and auditing.  
6. This driver is very flexible allows access to multiple databases using one driver.  
7. They are the most efficient amongst all driver types.  
  
Disadvantage

It requires another server application to install and maintain. Traversing the recordset may take longer, since the data comes through the backend server.

### Type 4 JDBC Driver

**Native-protocol/all-Java driver**

The Type 4 uses java networking libraries to communicate directly with the database server.  
  


Type 4: Native-protocol/all-Java driver

Advantage

1. The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.   
2. Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.   
3. You don’t need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantage  
  
With type 4 drivers, the user needs a different driver for each database.

**79) Explain garbage collection?**

**Java Memory Management**

Java Memory Management, with its built-in garbage collection, is one of the language’s finest achievements. It allows developers to create new objects without worrying explicitly about memory allocation and deallocation, because the garbage collector automatically reclaims memory for reuse. This enables faster development with less boilerplate code, while eliminating memory leaks and other memory-related problems. At least in theory.

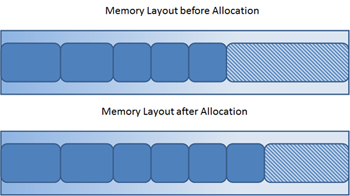
Ironically, Java garbage collection seems to work too well, creating and removing too many objects. Most memory-management issues are solved, but often at the cost of creating serious performance problems. Making garbage collection adaptable to all kinds of situations has led to a complex and hard-to-optimize system. In order to wrap your head around garbage collection, you need first to understand how memory management works in a Java Virtual Machine (JVM).

## How Garbage Collection Really Works

Many people think garbage collection collects and discards dead objects. In reality, Java garbage collection is doing the opposite! Live objects are tracked and everything else designated garbage. As you’ll see, this fundamental misunderstanding can lead to many performance problems.

Let's start with the heap, which is the area of memory used for dynamic allocation. In most configurations the operating system allocates the heap in advance to be managed by the JVM while the program is running. This has a couple of important ramifications:

* Object creation is faster because global synchronization with the operating system is not needed for every single object. An allocation simply claims some portion of a memory array and moves the offset pointer forward (see Figure 2.1). The next allocation starts at this offset and claims the next portion of the array.
* When an object is no longer used, the garbage collector reclaims the underlying memory and reuses it for future object allocation. This means there is no explicit deletion and no memory is given back to the operating system.

Figure 2.1: New objects are simply allocated at the end of the used heap.

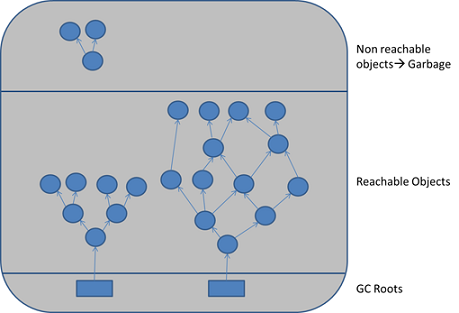
All objects are allocated on the heap area managed by the JVM. Every item that the developer uses is treated this way, including class objects, static variables, and even the code itself. As long as an object is being referenced, the JVM considers it alive. Once an object is no longer referenced and therefore is not reachable by the application code, the garbage collector removes it and reclaims the unused memory. As simple as this sounds, it raises a question: what is the first reference in the tree?

## Garbage-Collection Roots — The Source of All Object Trees

Every object tree must have one or more root objects. As long as the application can reach those roots, the whole tree is reachable. But when are those root objects considered reachable? Special objects called garbage-collection roots (GC roots; see Figure 2.2) are always reachable and so is any object that has a garbage-collection root at its own root.

There are four kinds of GC roots in Java:

1. **Local variables** are kept alive by the stack of a thread. This is not a real object virtual reference and thus is not visible. For all intents and purposes, local variables are GC roots.
2. **Active Java threads** are always considered live objects and are therefore GC roots. This is especially important for thread local variables.
3. **Static variables** are referenced by their classes. This fact makes them de facto GC roots. Classes themselves can be garbage-collected, which would remove all referenced static variables. This is of special importance when we use application servers, [OSGi containers](http://www.wikipedia.com/osgi) or class loaders in general. We will discuss the related problems in the Problem Patterns section.
4. **JNI References** are Java objects that the native code has created as part of a JNI call. Objects thus created are treated specially because the JVM does not know if it is being referenced by the native code or not. Such objects represent a very special form of GC root, which we will examine in more detail in the Problem Patterns section below.

Figure 2.2: GC roots are objects that are themselves referenced by the JVM and thus keep every other object from being garbage-collected.

Therefore, a simple Java application has the following GC roots:

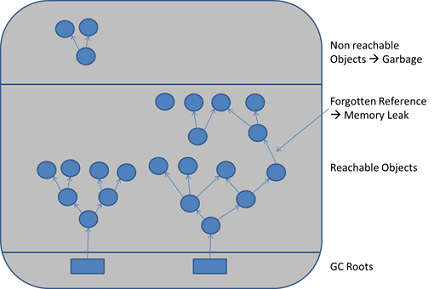
* Local variables in the main method
* The main thread
* Static variables of the main class

## Marking and Sweeping Away Garbage

To determine which objects are no longer in use, the JVM intermittently runs what is very aptly called a [mark-and-sweep algorithm](http://en.wikipedia.org/wiki/Garbage_collection_%28computer_science%29%2522%2520%255Cl%2520%2522Na.C3.AFve_mark-and-sweep%2522%2520%255Ct%2520%2522_blank). As you might intuit, it’s a straightforward, two-step process:

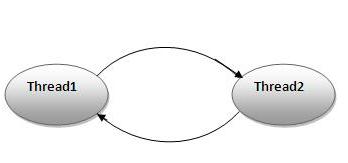
1. The algorithm traverses all object references, starting with the GC roots, and marks every object found as alive.
2. All of the heap memory that is not occupied by marked objects is reclaimed. It is simply marked as free, essentially swept free of unused objects.

Garbage collection is intended to remove the cause for classic memory leaks: unreachable-but-not-deleted objects in memory. However, this works only for memory leaks in the original sense. It’s possible to have unused objects that are still reachable by an application because the developer simply forgot to dereference them. Such objects cannot be garbage-collected. Even worse, such a logical memory leak cannot be detected by any software (see Figure 2.3). Even the best analysis software can only highlight suspicious objects. We will examine memory leak analysis in the Analyzing the Performance Impact of Memory Utilization and Garbage Collection section, below.

Figure 2.3: When objects are no longer referenced directly or indirectly by a GC root, they will be removed. There are no classic memory leaks. Analysis cannot really identify memory leaks; it can only point out suspicious objects.

**80) Explain the Dead Lock thread with sample code?**

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



**public** **class** CheckDeadLock {

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

**final** String resource1 = "FIRST USER";

**final** String resource2 = "SECOND USER";

// t1 tries to lock resource1 then resource2

Thread t1 = **new** Thread() {

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

**synchronized** (resource1) {

System.*out*.println("Thread 1: locked resource 1");

**try** { Thread.*sleep*(100);} **catch** (Exception e) {}

**synchronized** (resource2) {

System.*out*.println("Thread 1: locked resource 2");

}

}

}

};

// t2 tries to lock resource2 then resource1

Thread t2 = **new** Thread() {

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

**synchronized** (resource2) {

System.*out*.println("Thread 2: locked resource 2");

**try** { Thread.*sleep*(100);} **catch** (Exception e) {}

**synchronized** (resource1) {

System.*out*.println("Thread 2: locked resource 1");

}

}

}

};

t1.start();

t2.start();

}

}

**Output:**

Thread 1: locked resource 1

Thread 1: locked resource 2

Thread 2: locked resource 2

Thread 2: locked resource 1

**81) JavaScript Prototype?**

Syntax : object.prototype.name=value

Every JavaScript object has a prototype. The prototype is also an object.

All JavaScript objects inherit their properties and methods from their prototype.

## JavaScript Prototypes

All JavaScript objects inherit the properties and methods from their prototype.

Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype.

Objects created with new Date() inherit the Date.prototype.

The Object.prototype is on the top of the prototype chain.

All JavaScript objects (Date, Array, RegExp, Function, ....) inherit from the Object.prototype.

## Creating a Prototype

The standard way to create an object prototype is to use an object constructor function:

## Example

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}