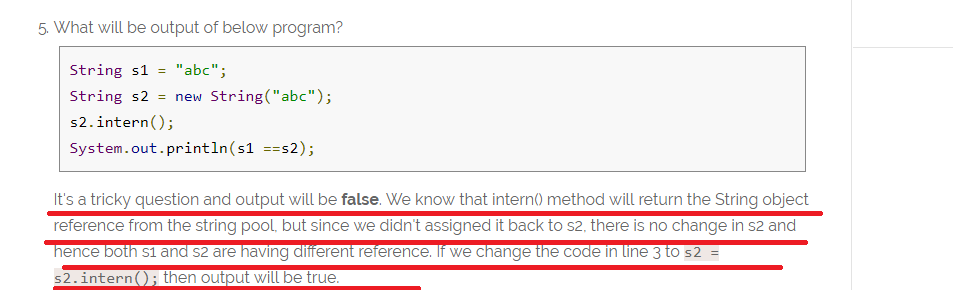
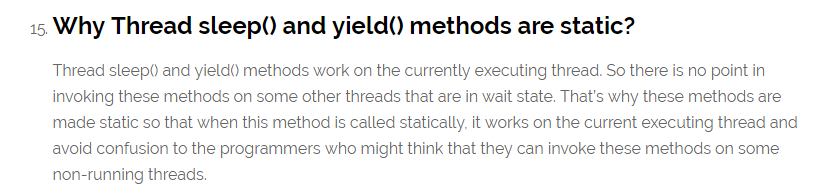
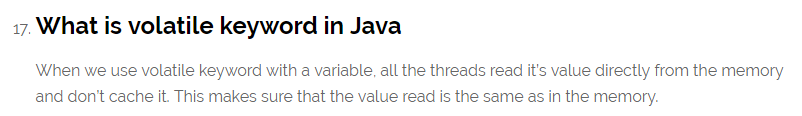
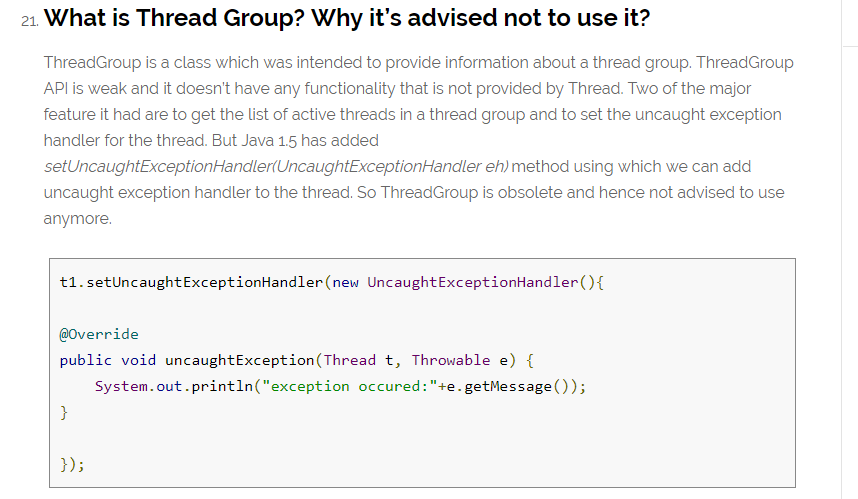
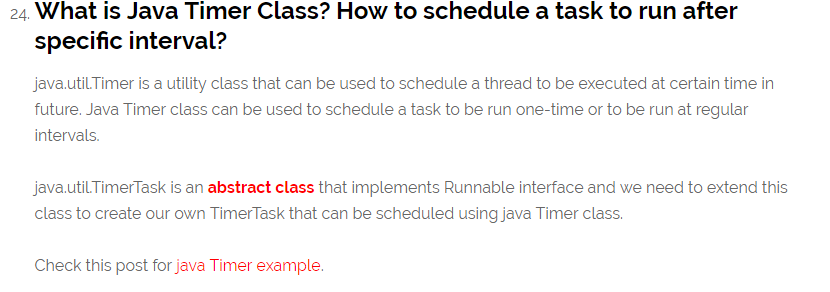
Journal Dev- String

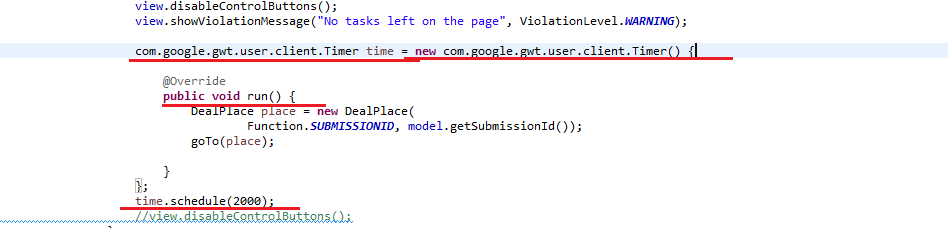
1. 
2. System.out.println("substring equals subSequence ? " + (str.substring(4, 14).equals(str.subSequence(4, 14))));

For the point 2 output is “**true”**

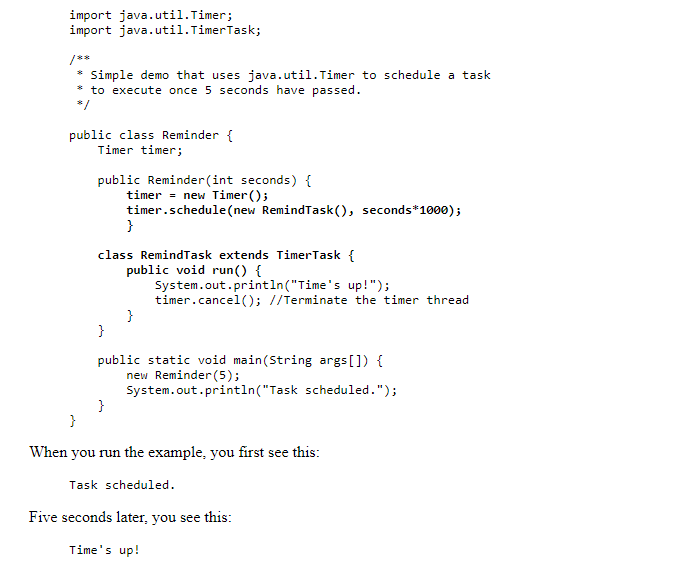
Journal Dev- Thread

Part -1 🡺 Basic

1. 
2. 
3. Java ThreadLocal is used to create thread-local variables.
4. 
5. Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation, read this post to learn how to [analyze deadlock in java](https://www.journaldev.com/1058/deadlock-in-java-example) with sample program. 🡺 **needed practical example on how deadlock created and how it can be avoid**
6. 

An Example on GWT Timer 🡺 

An Example on Timer and TimerTAsk 🡺



**java.util.TimerTask** is an [**abstract class**](https://www.journaldev.com/1582/abstract-class-in-java) that implements Runnable interface and we need to extend this class to create our own **TimerTask** that can be scheduled using java Timer class.

package com.journaldev.threads;

import java.util.Date;

import java.util.Timer;

import java.util.TimerTask;

public class MyTimerTask extends TimerTask {

@Override

public void run() {

System.out.println("Timer task started at:"+new Date());

completeTask();

System.out.println("Timer task finished at:"+new Date());

}

private void completeTask() {

try {

//assuming it takes 20 secs to complete the task

Thread.sleep(20000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void main(String args[]){

TimerTask timerTask = new MyTimerTask();

//running timer task as daemon thread

Timer timer = new Timer(true);

timer.scheduleAtFixedRate(timerTask, 0, 10\*1000);

System.out.println("TimerTask started");

//cancel after sometime

try {

Thread.sleep(120000);

} catch (InterruptedException e) {

e.printStackTrace();

}

timer.cancel();

System.out.println("TimerTask cancelled");

try {

Thread.sleep(30000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

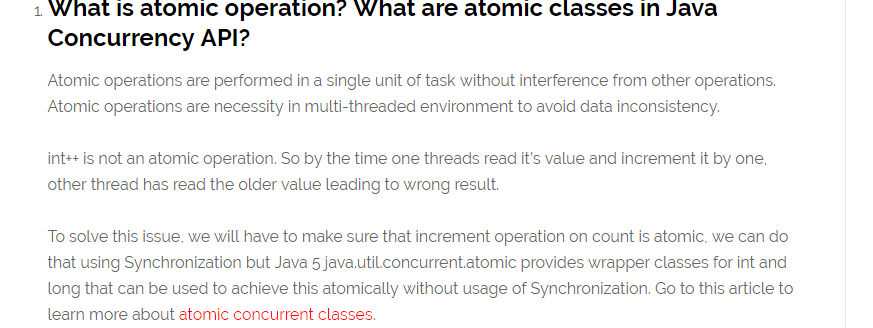
Notice that one thread execution will take 20 seconds but Java Timer object is scheduled to run the task every 10 seconds. Here is the output of the program:

Timer cancel() method is used to terminate the timer and discard any scheduled tasks, however it doesn’t interfere with the currently executing task and let it finish. If the timer is run as [daemon thread](https://www.journaldev.com/1072/daemon-thread-in-java), whether we cancel it or not, it will terminate as soon as all the user threads are finished executing.

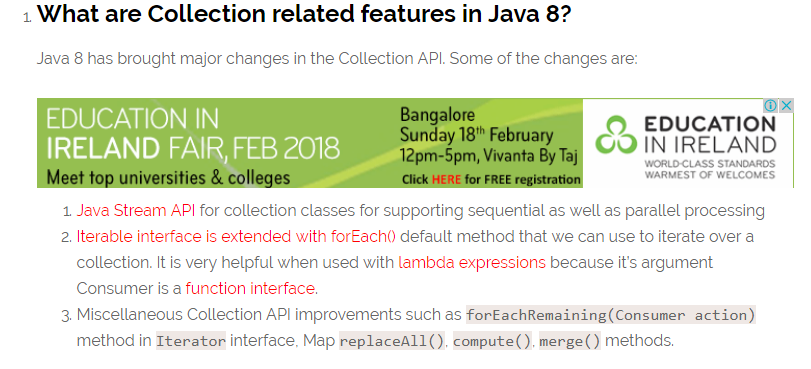
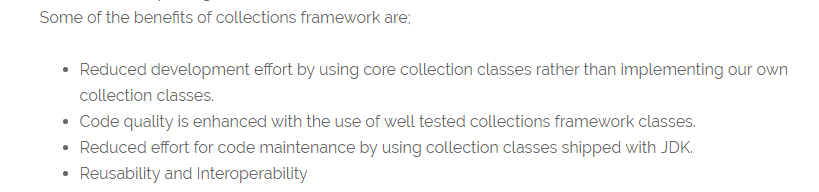
Timer class contains several **schedule**() methods to schedule a task to run once at given date or after some delay. There are several **scheduleAtFixedRate**() methods to run a task periodically with certain interval

While scheduling tasks using Timer, you should make sure that time interval is more than normal thread execution, otherwise tasks queue size will keep growing and eventually task will be executing always. That’s all for a quick roundup on Java Timer and Java TimerTask.

## Part 2: Java Concurrency Interview Questions and Answers

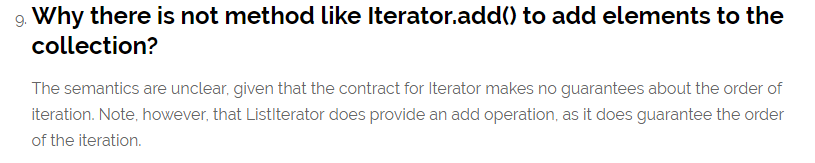
1. 
2. **NOTE FEW MORE QUESTIONS ARE PRESENT ON CONCURRENCY BUT ITS QUITE HIGH LEVEL**

JOURNALdEV-Collection

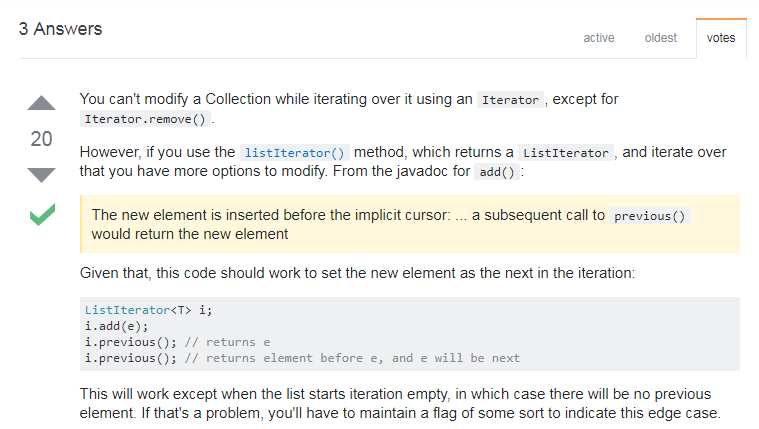
1. 
2. 

### What is the purpose of  Cloneable and Serializable interfaces?

That said, the one thing it does is "enable" the default implementation of clone() in Object . If you don't implement Cloneable then invoking **super.clone**() **will** throw a**CloneNotSupportedException** . Purpose of **clone() method** is create a new instance (**copy**) of object on which it is called

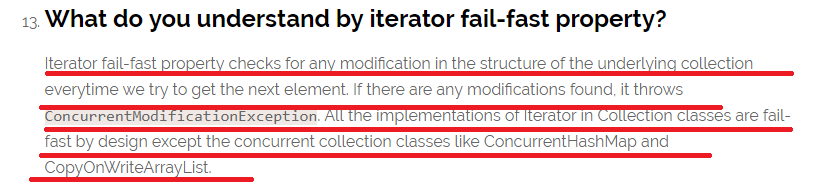
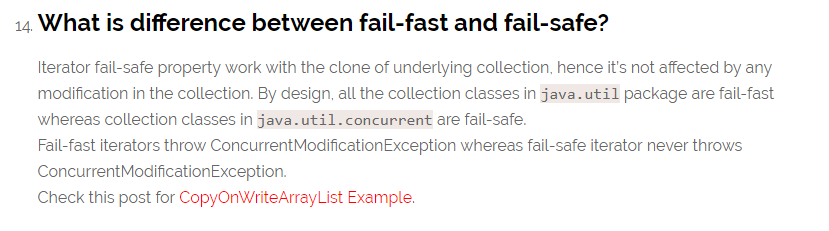
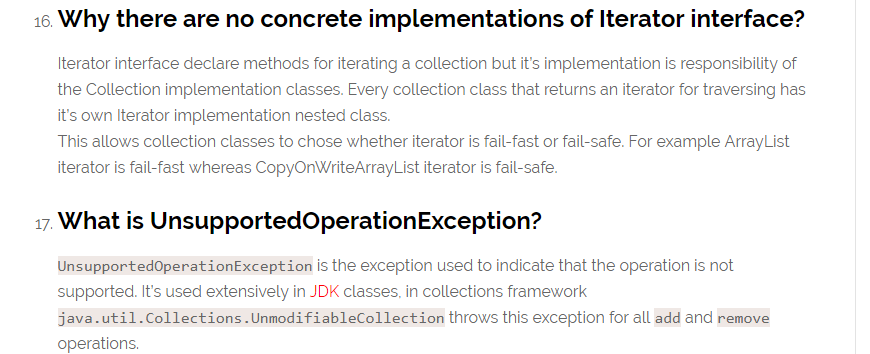
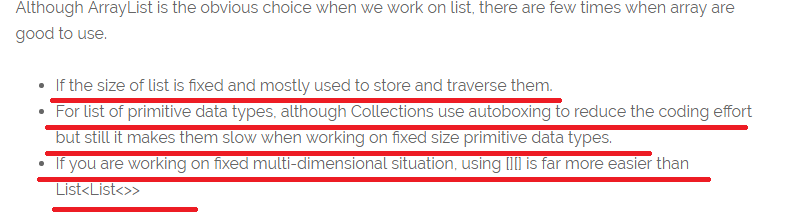
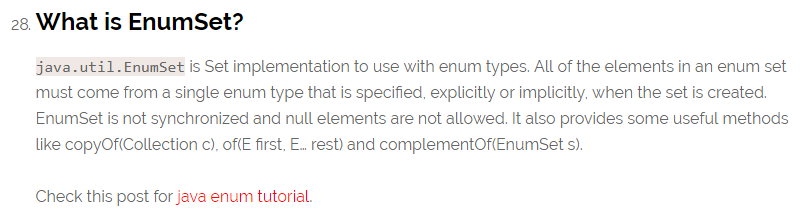
1. 

Note 🡺 from Stack Overflow



### What are different ways to iterate over a list?

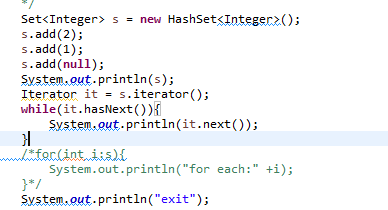
We can iterate over a list in two different ways – using iterator and using for-each loop.

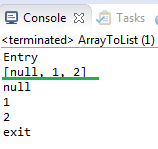
1. 
2. 
3. 
4. 
5. 

WILL THE SET ALLOWS NULL?

We have two scenarios 🡺 normatlly set will allow one null(in case of more null then overriding will happen)

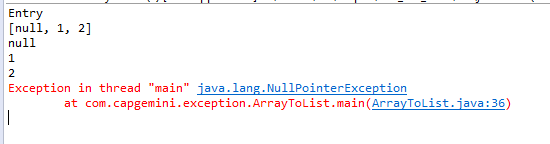
Case 1:



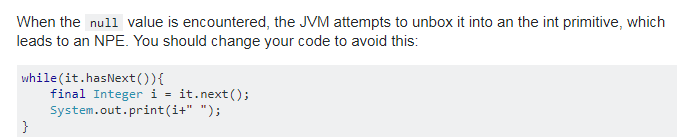
OUTPUT 🡺 NO EXCEPTION 🡺

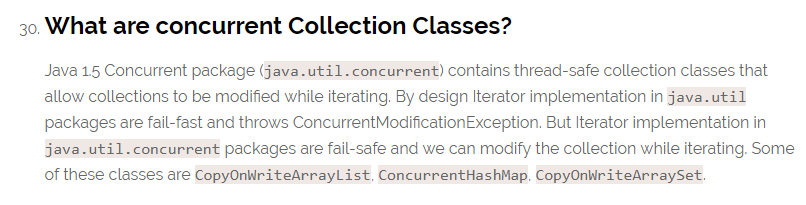
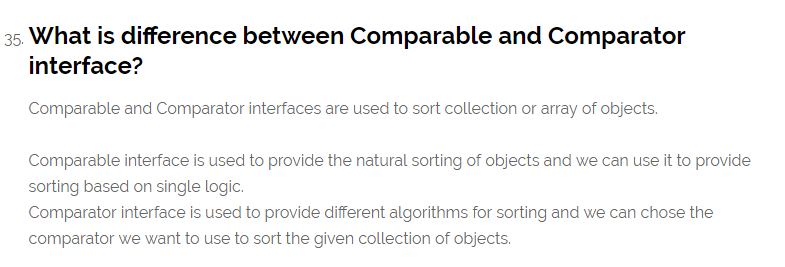
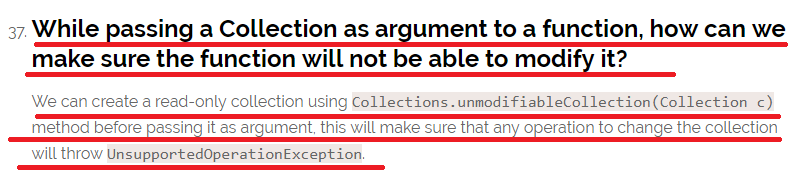
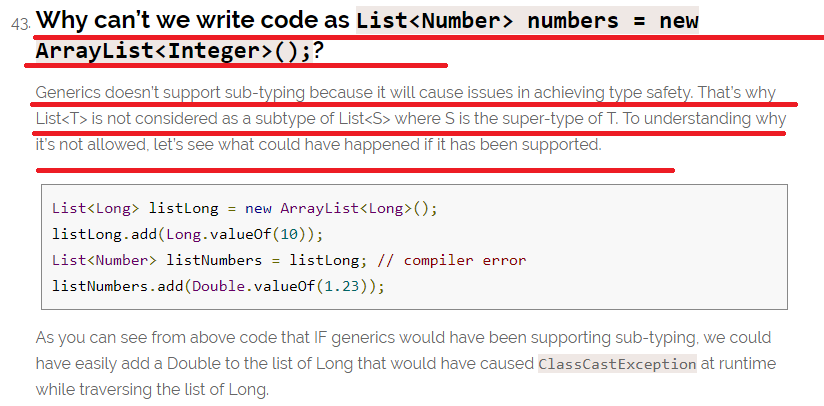
CASE 2: NOW LETS SEE THE ERROR SCENARIO BY UNCOMMING THE ABOVE SCREEN SHOT CODE

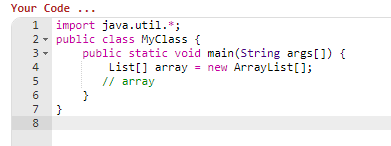


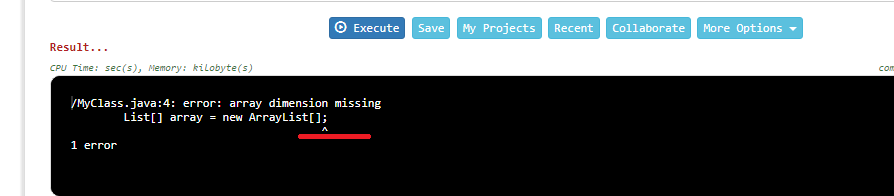
OUTPUT 🡺

THE REASON FOR THE EXCEPTION IS 🡺FROM STACK OVERFLOW IS 🡺

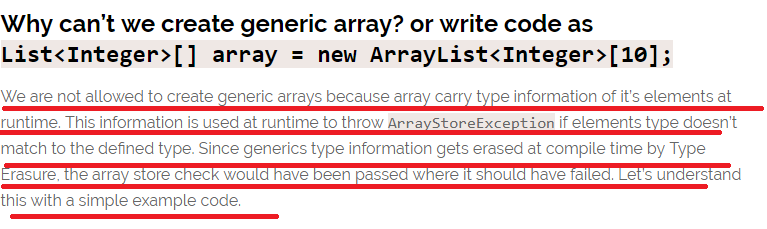


1. 
2. 
3. 
4. 
5. What is the Output of the following code?



Output 🡺

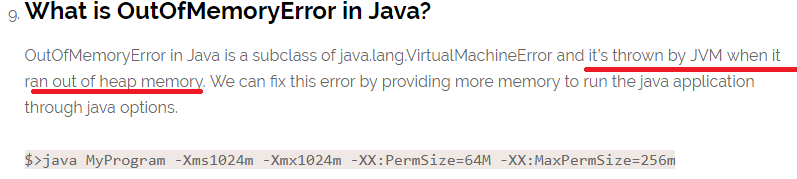
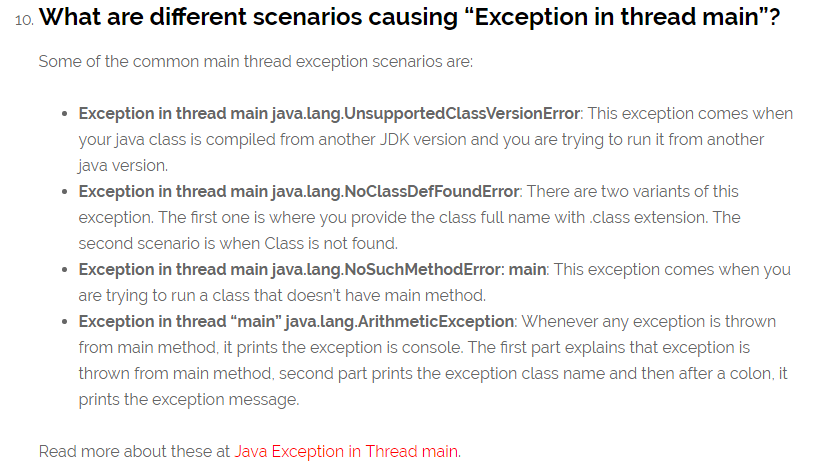
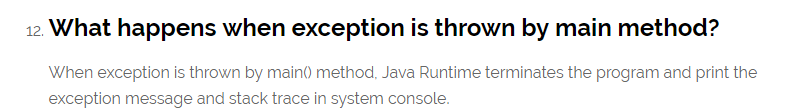
**See here though array created is of Collection type it requires fixed Size.**

1. 

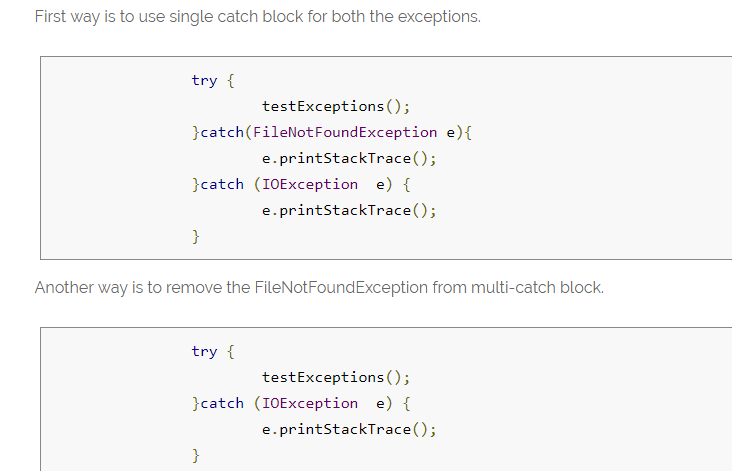
Journal Dev- Exception Interview Questions

1. Exception is an error event that can happen during the execution of a program and disrupts it’s normal flow.

### Explain Java 7 ARM Feature and multi-catch block? 🡺

1. 
2. What about StackOverFlow?
3. 
4. final keyword can be used with class variables so that they can’t be reassigned
5. 
6. 

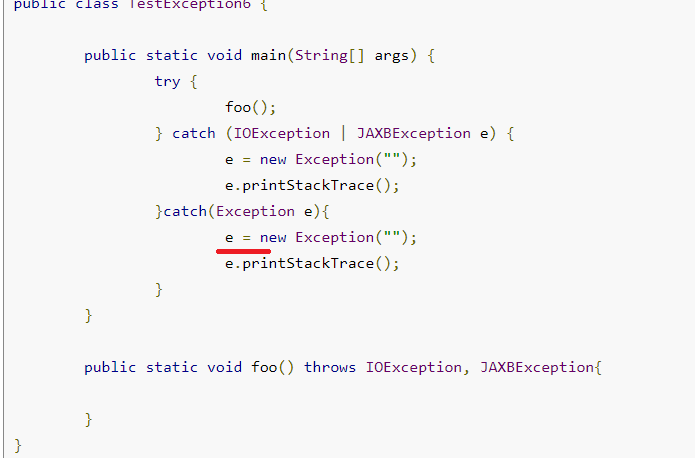
Above program won’t compile and you will get error message as “The exception FileNotFoundException is already caught by the alternative IOException”. This is because FileNotFoundException is subclass of IOException, there are two ways to solve this problem.



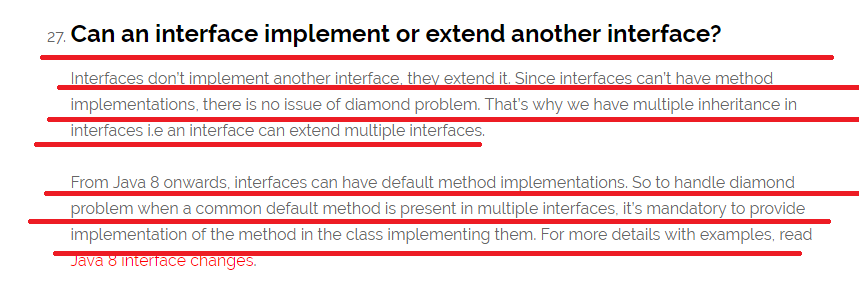
1. What is the output for the below program?



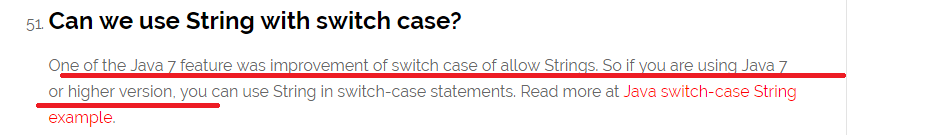
**The program won’t compile because JAXBException is a checked exception and foo() method should throw this exception to catch in the calling method. You will get error message as “Unreachable catch block for JAXBException. This exception is never thrown from the try statement body”. 🡺 To solve this issue, you will have to remove the catch block of JAXBException.**

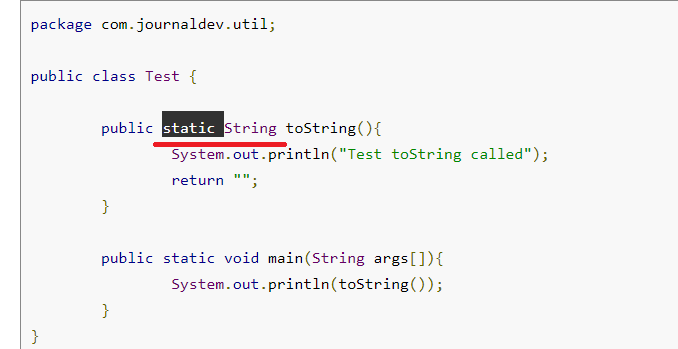
1. **exception object in multi-catch block is final and we can’t change it’s value.  🡺 is from Java 7**
2. What is the output for the below code? 🡺the red colour underlined line throws an exception

Now Let’s see the Journal Dev general Code Java Questions

1. 
2. Java Reflection API provides ability to inspect and modify the runtime behavior of java application

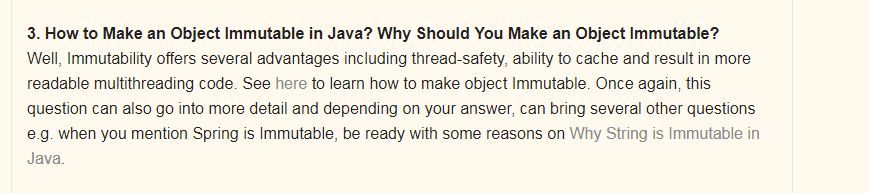
### How to sort a collection of custom Objects in Java? 🡺 without using the build function

1. 
2. What is the output for the following code🡺

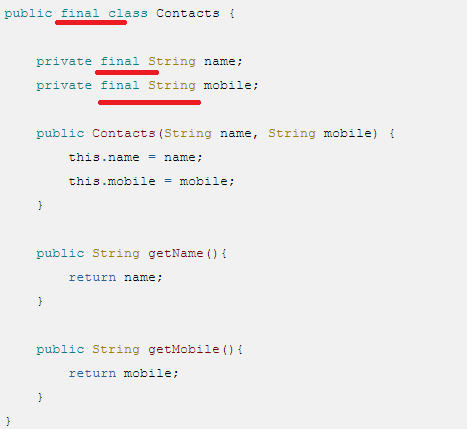


**Answer: The code won’t compile because we can’t have an Object class method with static keyword. Note that Object class has toString() method. You will get compile time error as “This static method cannot hide the instance method from Object”. The reason is that static method belongs to class and since every class base is Object, we can’t have same method in instance as well as in class. You won’t get this error if you change the method name from toString() to something else that is not present in super class Object.**

Interview Questions from 🡺 <http://www.java67.com/2013/07/15-advanced-core-java-interview-questions-answers-senior-experienced-5-6-years-programmers-developers.html>

1. 

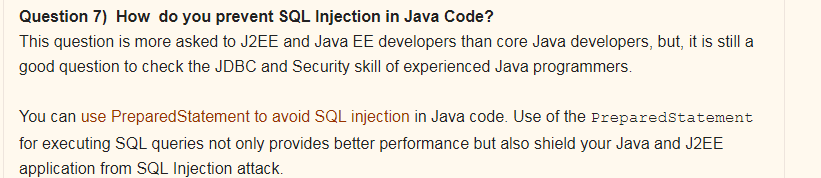
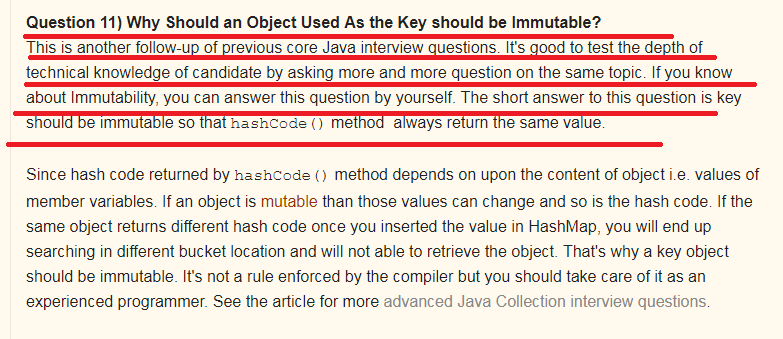
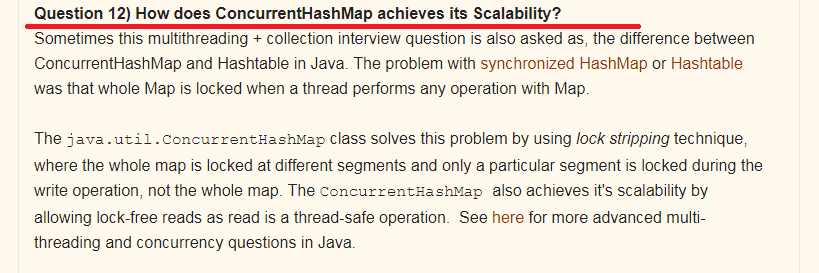
NOW LETS SEE HOW TO MAKE AN OBJECT IMMUTABLE🡺 WHAT IS THE DIFFERENCE BETWEEN MAKING A CLASS AS IMMUTABLE OR MAKING AN OBJECT AS IMMUTABLE

1. Syntax to create a immutable class

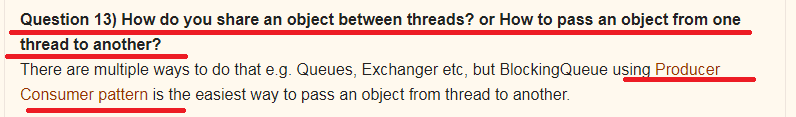
Explanation 🡺

1. This Java class is immutable, because its state can not be changed once created.
2. You can see that all of its fields are final. 🡺 NOTE WE HAD MARKED IN THE CODE AS FINAL BUT I DID HANDS ON THAT IT WAS NOT FINAL BY DEFAULT,   
     
   Read more: <http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html#ixzz53s9qcmAf>

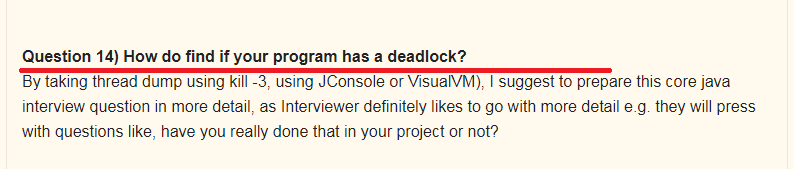
Doubt 🡺

1. Which Design Patterns have You Used in Your Java Project?  
     
   Always expect some design patterns related question for Core Java Interview of senior developer position. It's a better strategy to mention any GOF design pattern rather than Singleton or MVC, which almost every other Java developer use it.   
     
   Your best bet can be [Decorator pattern](http://java67.blogspot.sg/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html) or may be [**Dependency Injection Pattern**](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html)**, 🡺 NOTE WE CAN TALK ABOUT THE Dependency Injection Design Pattern.**
2. 
3. 
4. 

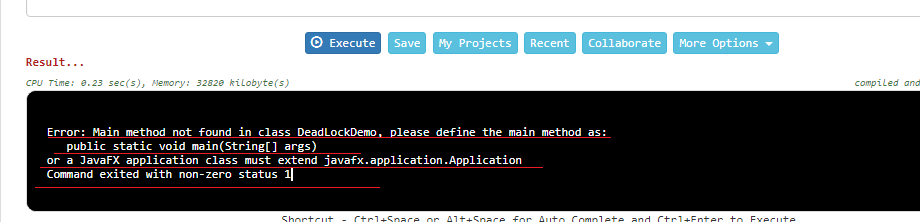
Explanation🡺

1. 

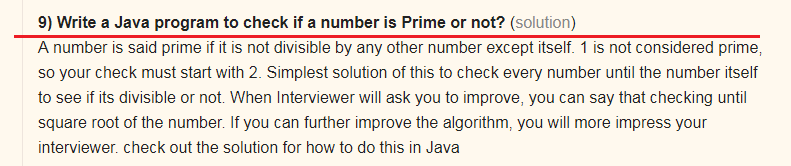
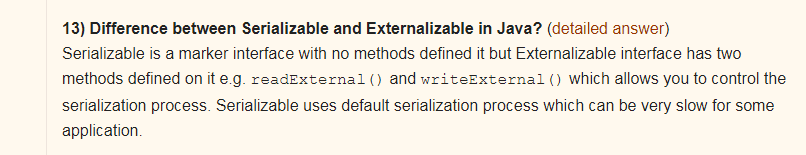
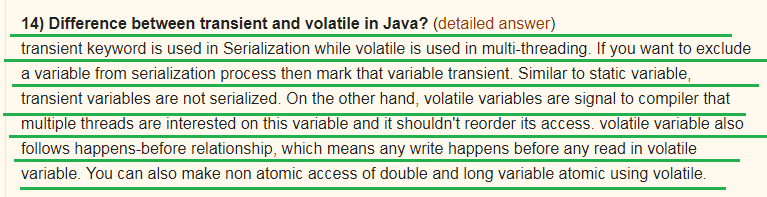
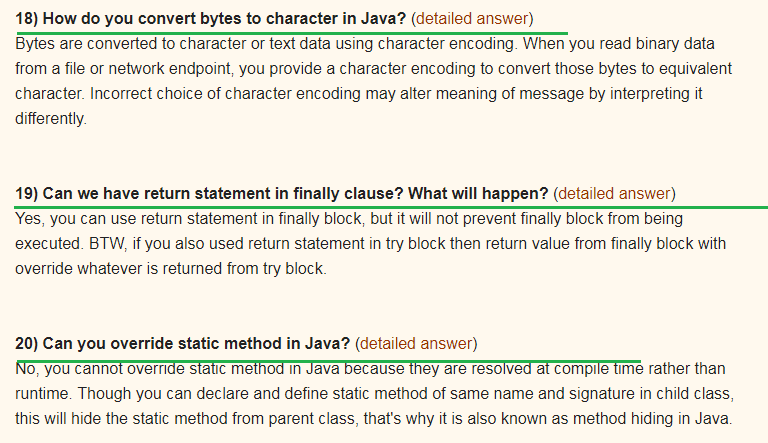
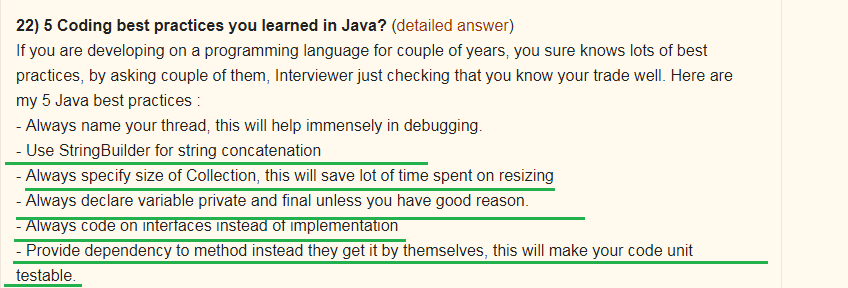
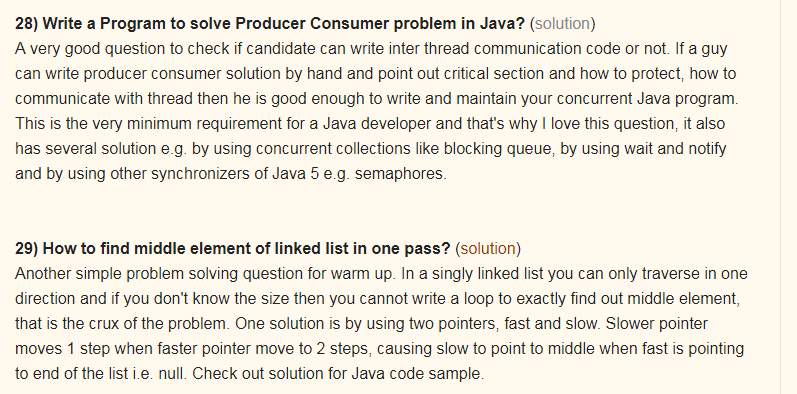
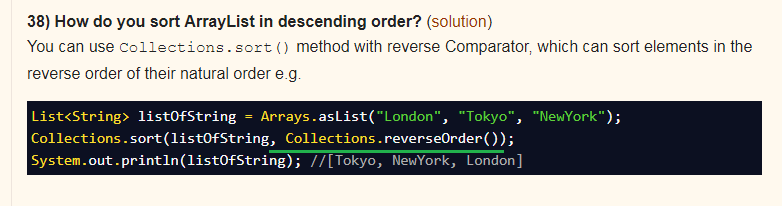
Explanation🡺

1.  Explanation:
2. Write a Java program which will result in deadlock?  
   

**See here usually we have seen the syntax of methodName(Datatype variable) but here it is in the format methodName(ClassNAme.class)**

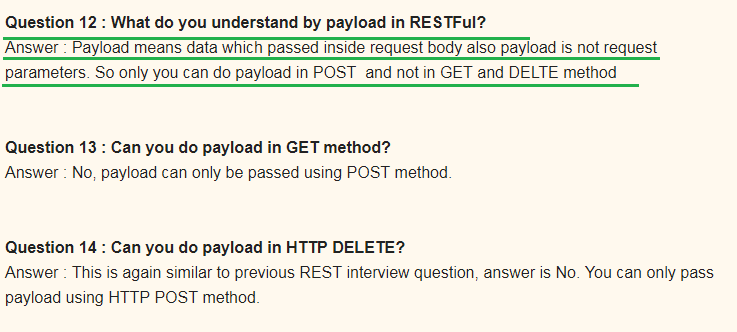
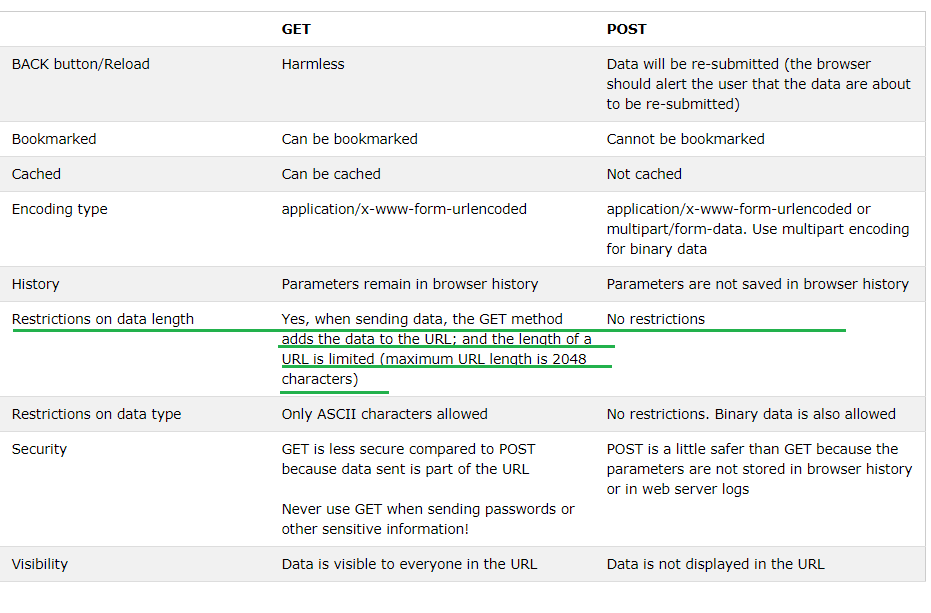
1. What will be the Output if we run the Application Without Main Method🡺

Core Java Interview Questions 🡺 from 🡺 <http://www.java67.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html>

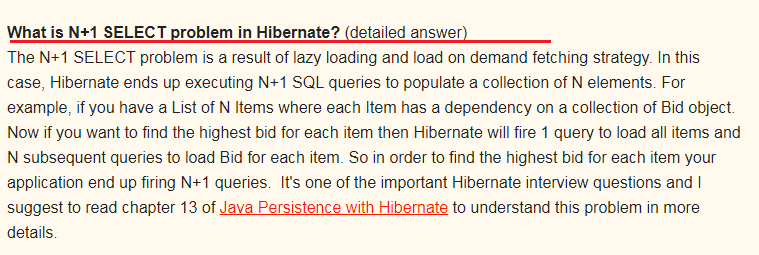
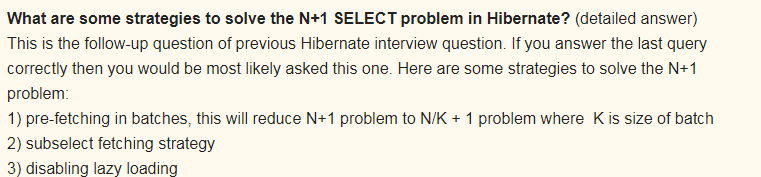
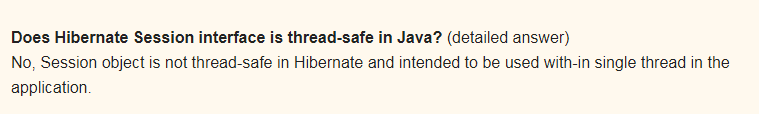
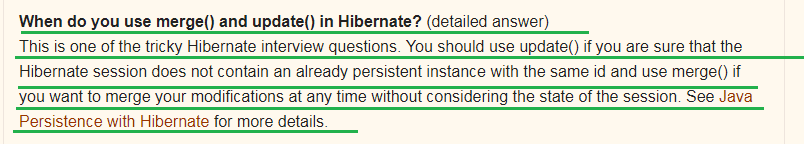
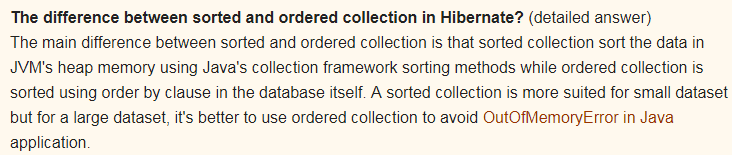
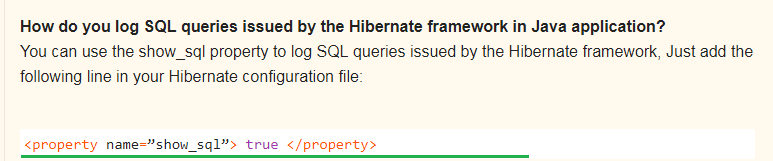
1. Logic for Fibonacci series 🡺 sum of previous two numbers i.e. f(n) = f(n-1) + f(n-2).
2. 
3. **11) How to check if linked list contains loop in Java?** ([solution](http://javarevisited.blogspot.sg/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html))
4. 
5. 
6. 
7. 
8. 
9. **28) Write a Program to solve Producer Consumer problem in Java?** ([solution](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))
10. **29) How to find middle element of linked list in one pass?** ([solution](http://javarevisited.blogspot.sg/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html))
11. 

Rest Based Interview Question from 🡺 <http://www.java67.com/2015/09/top-10-restful-web-service-interview-questions-answers.html>

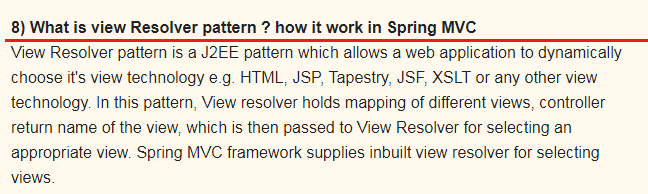
Rest Service Questions

1. Question 3: What is HTTP Basic Authentication and how it works?
2. Question 4: Can you tell me which API can be used to develop RESTFul web service in Java? 🡺 There are many framework and libraries out there which helps to develop RESTful web services in Java including JAX-RS which is standard way to develop REST web services. Jersey is one of the popular implementation of JAX-RS which also offers more than specification recommends. Then you also have RESTEasy, RESTlet and Apache CFX. If you like Scala then you can also use Play framework to develop RESTful web services
3. Question 5 : How do you configure RESTFul web service?
4. Question 8 : How you maintain session in RESTful services?
5. 
6. 

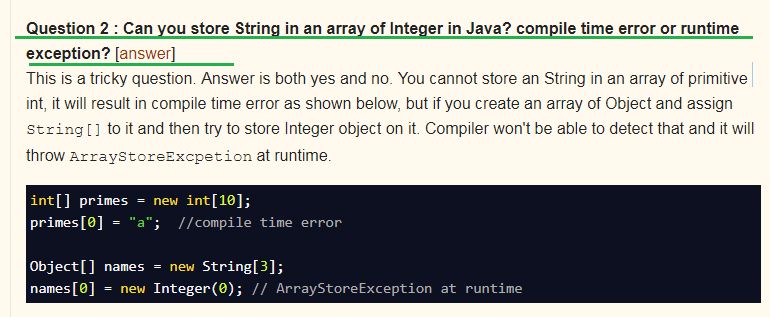
Top 20 Hibernate Interview Questions for Java J2EE Programmers 🡺 <http://www.java67.com/2016/02/top-20-hibernate-interview-questions.html>

1. 
2. 
3. 
4. 
5. 
6. 

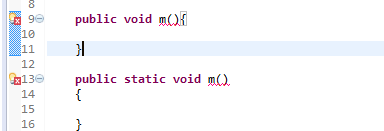
Top 23 Spring MVC Framework Interview Questions Answers - Java JEE 🡺 <http://www.java67.com/2012/08/spring-interview-questions-answers.html>

1. 
2. 15) Can we use more than one configuration file for our Spring project?
3. 18) Can you use Spring MVC framework along with Struts ? I have an existing Java MVC application which is based in Struts, Can I migrate that to use Spring MVC ? **How ?**
4. 19) What is the advantage of Spring MVC framework over Struts 1.0 or Struts 2.0 ? is it worth to convert an existing Struts application to Spring MVC ?

22 Array Concepts Interview Questions Answers in Java 🡺 <http://www.java67.com/2015/07/array-concepts-interview-questions-answers-java.html>

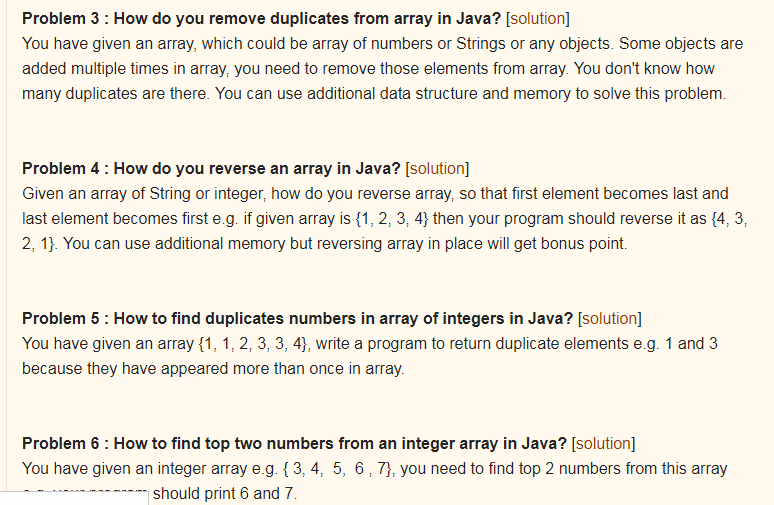
1. 

Above Question is Very Important

 🡺 reason for CE 🡺 is duplicate method

1. **Question 4 : Can you use Generics with array?** [[answer](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html)]  
   No, you cannot use Generic with array, that's why sometime List is better choice over array in Java.

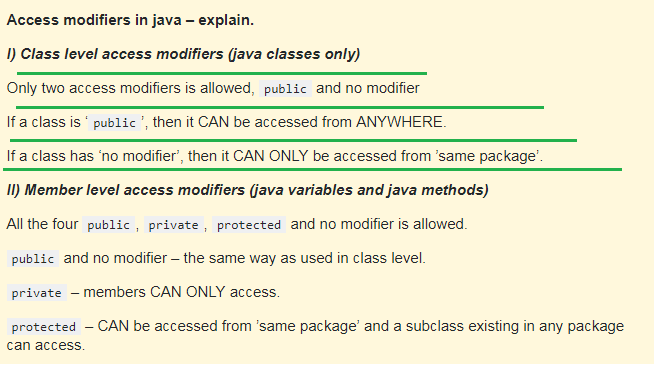
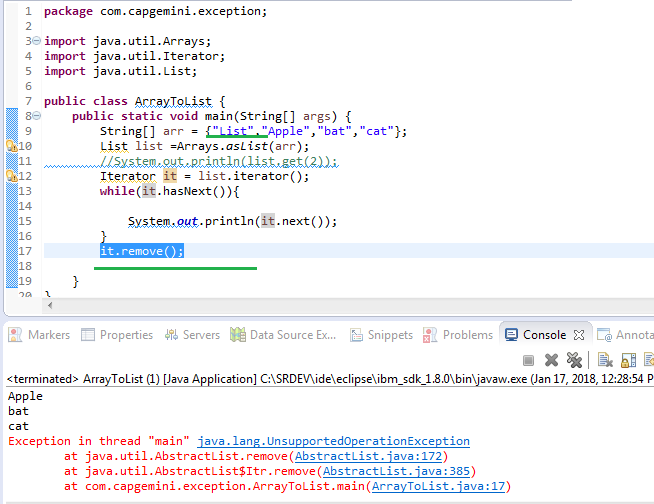
Array Based Programs

1. Problem 2 : How do you find all pair whose sum is equal to given number from integer array in Java?
2. Problem 2 : **How do you find all pair whose sum is equal to given number from integer array in Java?** [[solution](http://javarevisited.blogspot.sg/2014/08/how-to-find-all-pairs-in-array-of-integers-whose-sum-equal-given-number-java.html)]
3.   
     
     
   16/1/2018
4. Can we have more than one class @ted with @Configuration? 🡺 YES
5. Not only the add but also remove is not possible with the array converted into list

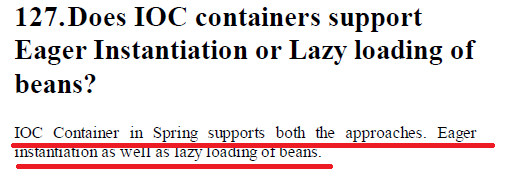
# Stack overflow 🡺[**Why are interface variables static and final by default?**](https://stackoverflow.com/questions/2430756/why-are-interface-variables-static-and-final-by-default)

Since interface doesn't have a direct object, the only way to access them is by using a class/interface and hence that is why if interface variable exists, it should be static otherwise it wont be accessible at all to outside world. Now since it is static, it can hold only one value and any classes that implements it can change it and hence it will be all mess.

Hence if at all there is an interface variable, it will be implicitly static, final and obviously public!!!

1. **A constructor can have any of the Access specifiers like public, protected, private , none but unlike methods constructor can take up only the access specifier, therefore constructor cannot be abstract, final, static, synchronized, native**
2. **Ms 🡺 but outer class can have final and inner class can be static**
3. 
4. **SEE THE BELOW IMAGE ITERATOR HAS REMOVED THE Object but after removing it had thrown the “UnSupportedOperation” Exception**;

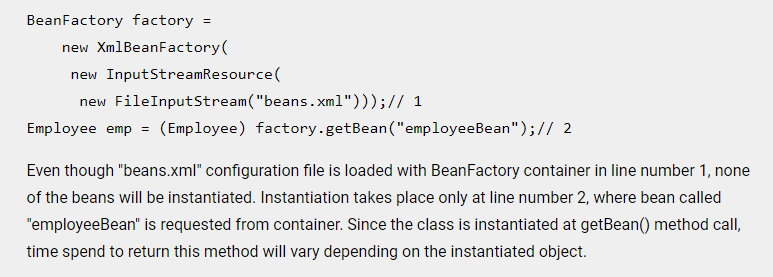
**ARRAY CONVERTED TO LIST DOES NOT ALLOW BOTH ADDITION OR REMOVAL OF OBJECTS**

1. Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app. SPAs use AJAX and HTML5 to create fluid and responsive Web apps, without constant page reloads. However, this means much of the work happens on the client side, in JavaScript.
2. REGEXP\_SUBSTR()
3. 

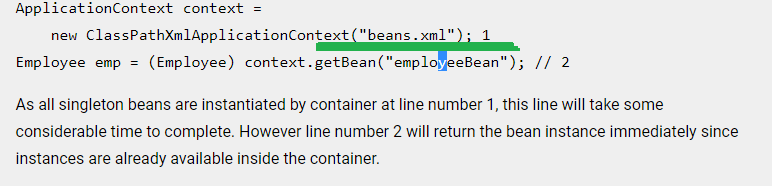
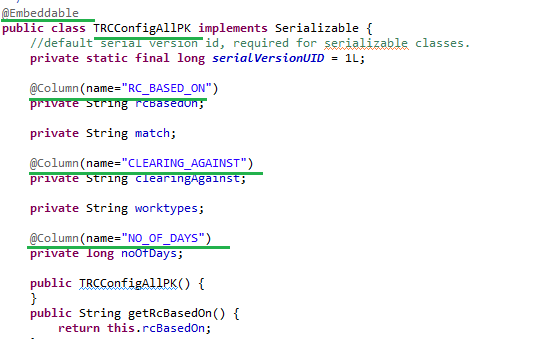
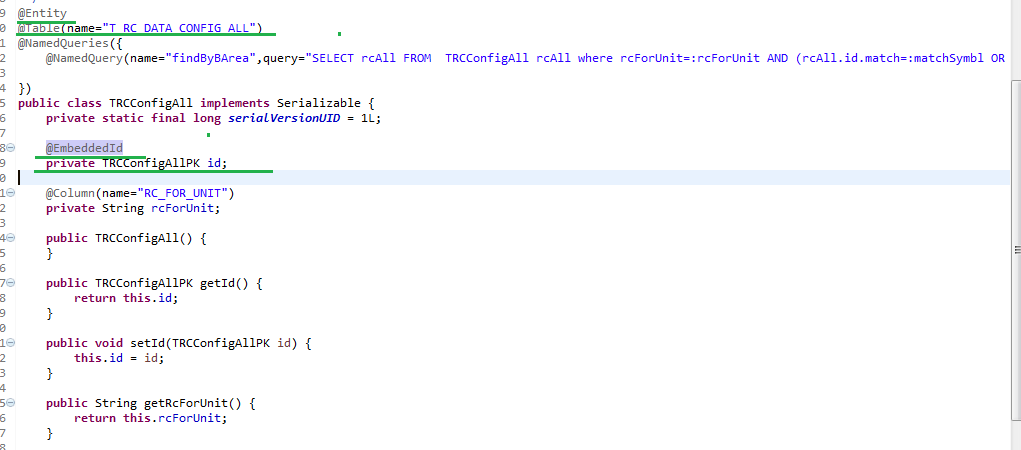
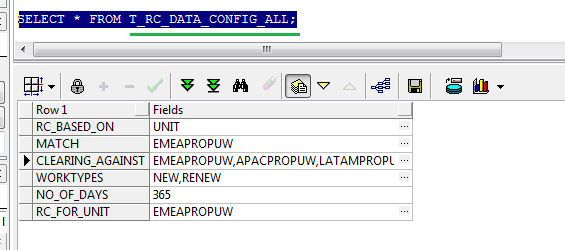
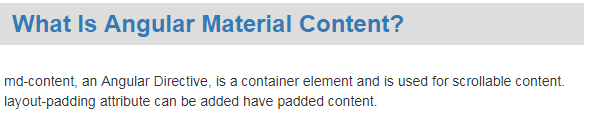
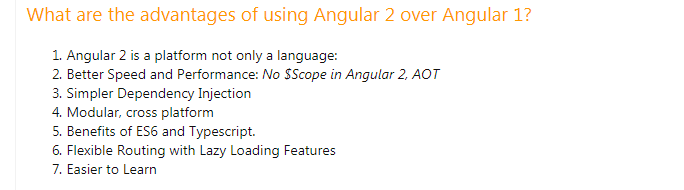
Explanation 🡺

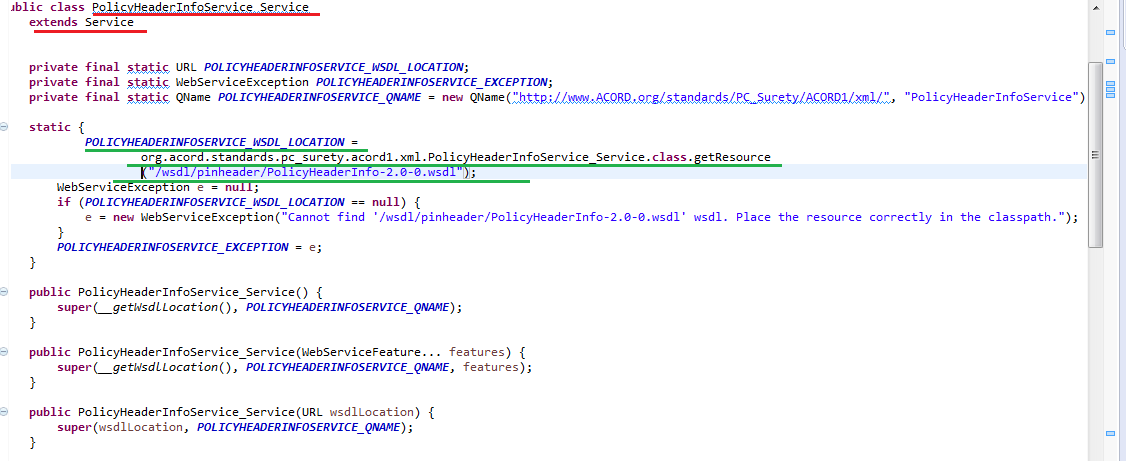
1. Spring framework can instantiate and bind (called loading) related Java objects (called beans) according to a given configuration
2. An XML file can easily be used to define these bindings.
3. Spring framework supports two different types of loading methods; lazy loading and pre-loading respectively managed by BeanFactory and ApplicationContext containers.

## Lazy Loading

1. A bean is loaded only when an instance of that Java class is requested by any other method or a class. org.springframework.beans.factory.BeanFactory (and subclasses) container loads beans lazily.
2. 

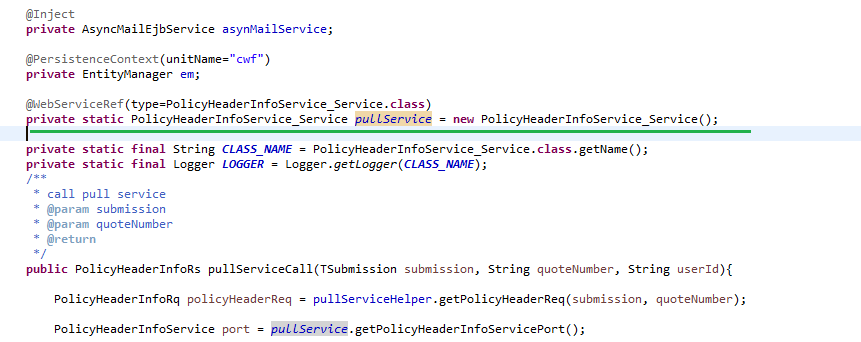
## Pre-loading

1. All beans are instantiated as soon as the spring configuration is loaded by a container. org.springframework.context.ApplicationContext container follows pre-loading methodology.
2. 
3. **How will you call a stored procedure in Hibernate? 🡺 needed information on this 🡺 and also check how we had called in CNA**
4. Let’s see an example for how EmbeddedId works 🡺
5. A class which is going to act as a @EmbeddedId field
6. A class which has @EmbeddedId and one of its own field called “ **RC\_FOR\_UNIT** ” 🡺
7. NOW LETS SEE THE TABLE STRUCTURE 🡺
8. See here an interface is having abstract keyword🡺
9.  🡺 needed more information on this
10. **how will you handle errors in Angular2 application**
11. Explain the life cycle hooks of Angular 2 application
12. 
13. Pending work in angular🡺 Custom filters/pipe, Auto completion, Event Emitter, input and output, Life cycle hooks, pagination, modal, and others see the notes.
14. Few Information on Soap Service 🡺
15. Auto generated Service class which is used to get the reference/port of out WS interface



1. Now lets see the flow for getting the Service Object first and then through service object how to get the Port or WS reference

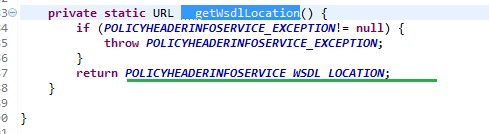
* First lets see how to get the object of the auto generated Service class that extends Service



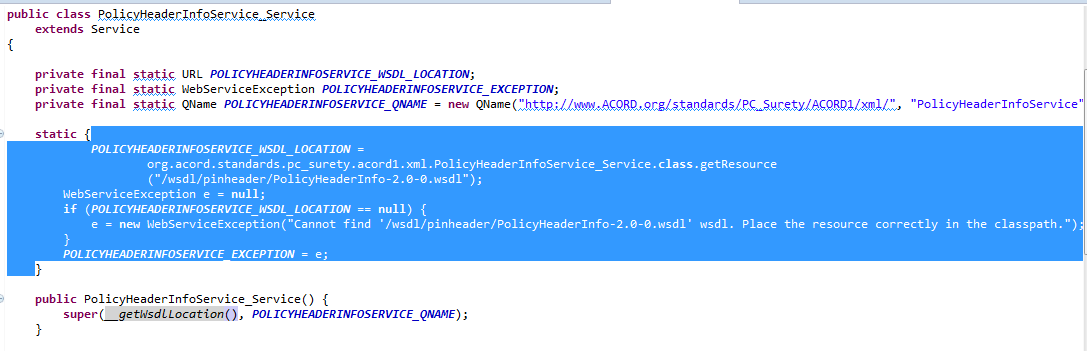
Now let’s see what happens in the PolicyHeaderInfoService\_Service() constructor🡺



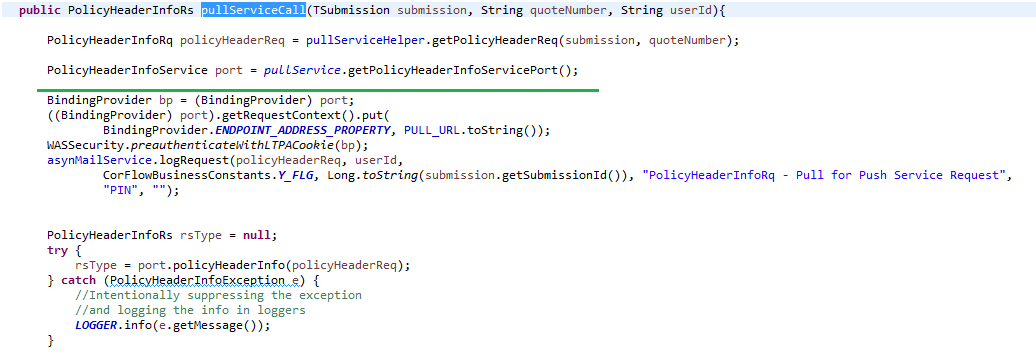
Now Let’s see what is this getWsdlLocation\_() method is doing



Now lets see Where and how POLICYHEADERINFO\_SERVICE\_EXCEPTION and POLICYHEADERINFOSERVICE\_WSDL\_LOCATION is set🡺



* Now let’s see how is this port or WS reference is obtained using the auto generated service stub object

1. 

22/1/2018

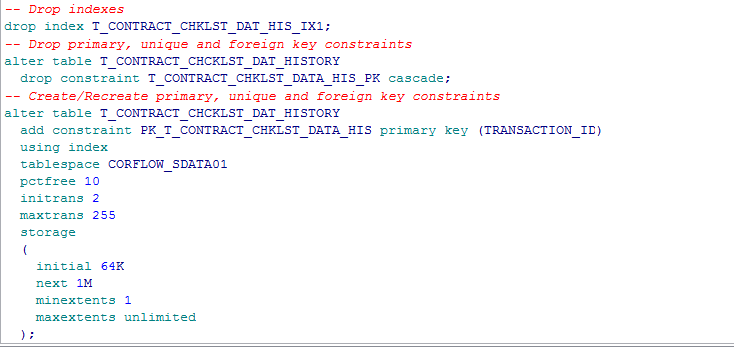
1. **evict()** To detach the object from **session** cache,
2. what is the difference between evict and delete methods in Hibernate Session 🡺

# [clear(), evict() and close() methods in Hibernate](https://www.connect2java.com/tutorials/hibernate/clear-evict-and-close-methods-in-hibernate/)

1. public void clear()🡺Completely clear the session and is used to dissociate/disconnect all the objects from the session. 🡺 ms clears entire session
2. **evict():**  🡺 Removes the object from the session. This method is used to dissociate/disconnect the specified object from the session 🡺**public void evict(Object object)** throws HibernateException 🡺ms removes only particular object

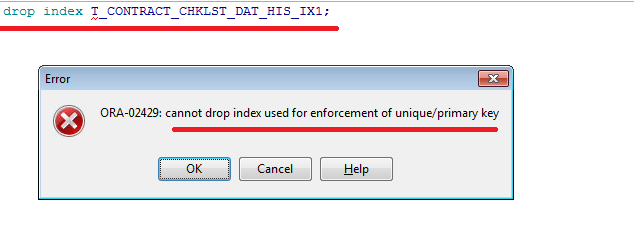
1/2/2018

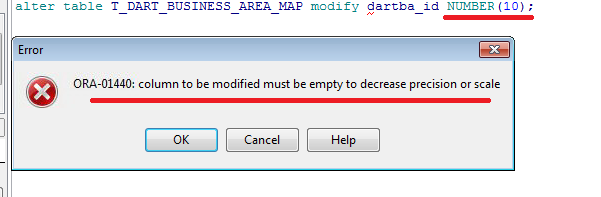
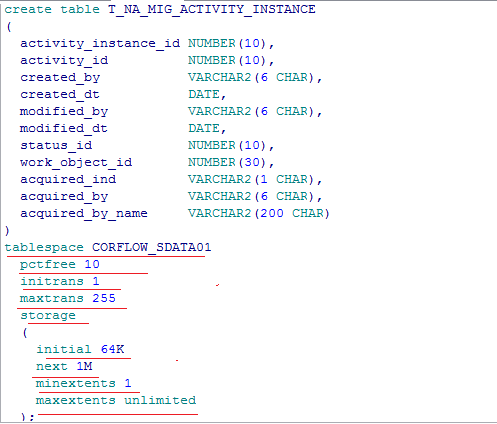
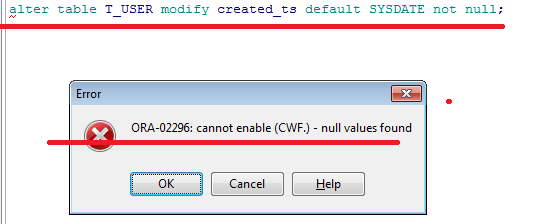
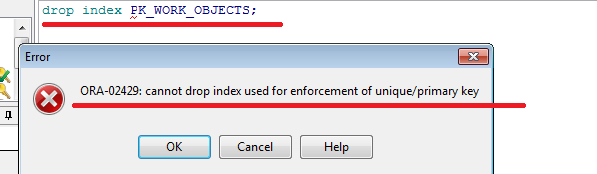
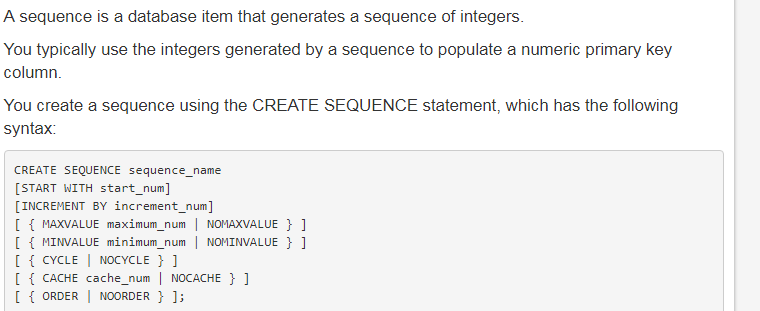
1. While Comparing the two schemas I found that



There are dropping aa index and a primary key constraint and after that they again creating a primary key with index as shown above 🡺 note actually they are doing all this circus just to rename the PKey constraint and index

Output🡺 on executing the drop index command I got the following error



1. While trying to reduce the column size the error was 🡺
2. Ms Difference between Normal and Unique index 🡺 normal index allows duplicates(not unique values) and Unique index does not allows unique index
3. How to apply Unique constraint on 2 columns 🡺 that is combination of two columns should be a unique value🡺
4. Ms 🡺 to create a unique index then I need to add Unique keyword in the “Create Index” query otherwise it will create a normal index except for the primary key column that is if I create a index for a primary key column then even without adding the the Unique Keyword to “Create index” by default Unique index gets created
5. 
6. 
7. NOTE WHEN WE TRY TO DROP AN INDEX WHICH IS CRESTED DUE TO THE PRIMARY KEY COLUMN (or) created on primary key column then we get the following error message🡺
8. Syntax to create a sequence🡺
9. The default start\_num is 1.
10. The default increment number is 1.
11. NOMINVALUE is the default.
12. NOMAXVALUE is the default.
13. CYCLE specifies the sequence generates integers even after reaching its maximum or minimum value.
14. NOCYCLE specifies the sequence cannot generate any more integers after reaching its maximum or minimum value. 🡺NOCYCLE is the default.
15. CACHE cache\_num specifies the number of integers to keep in memory🡺The default number of integers to cache is 20
16. NOCACHE specifies no integers are to be stored.
17. ORDER guarantees the integers are generated in the order of the request.
18. NOORDER doesn't guarantee the integers are generated in the orde🡺NOORDER is the default

# **ALTER SEQUENCE**

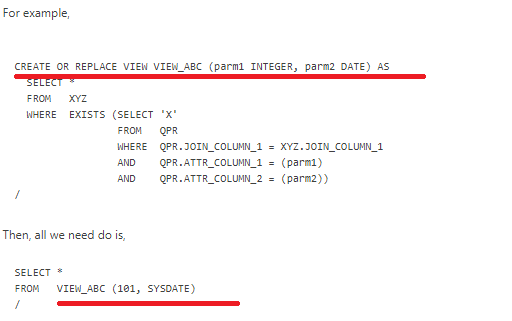
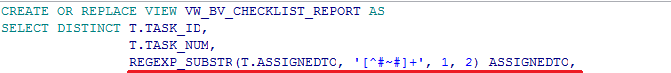
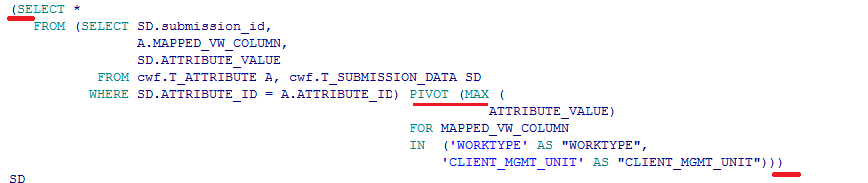
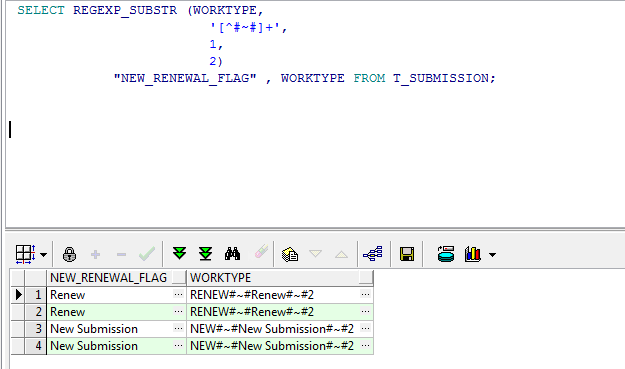
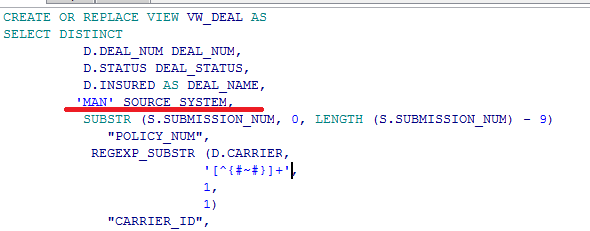
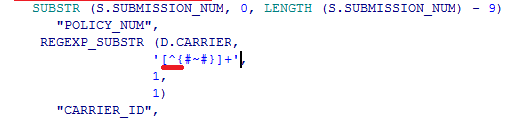
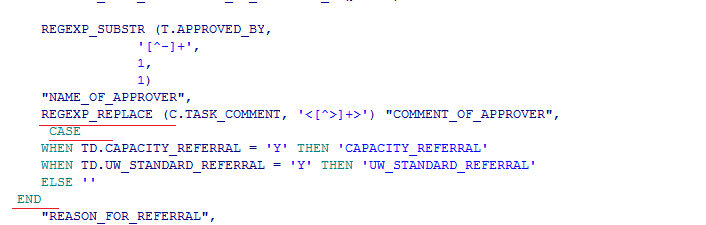
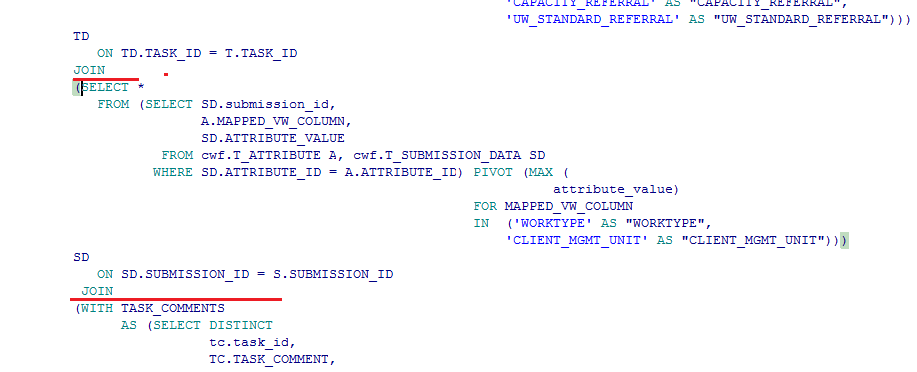
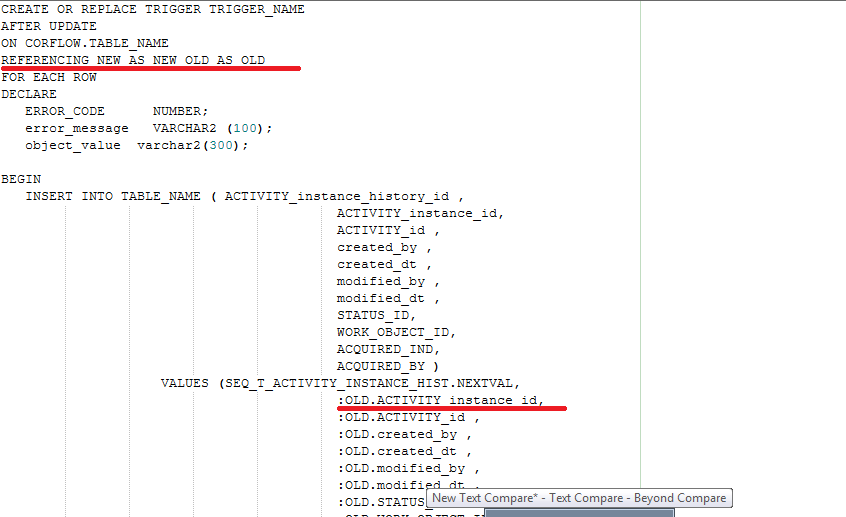
1. Use the ALTER SEQUENCE statement to change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence. This statement affects only future sequence numbers.
2. ALTER SEQUENCE customers\_seq MAXVALUE 1500;

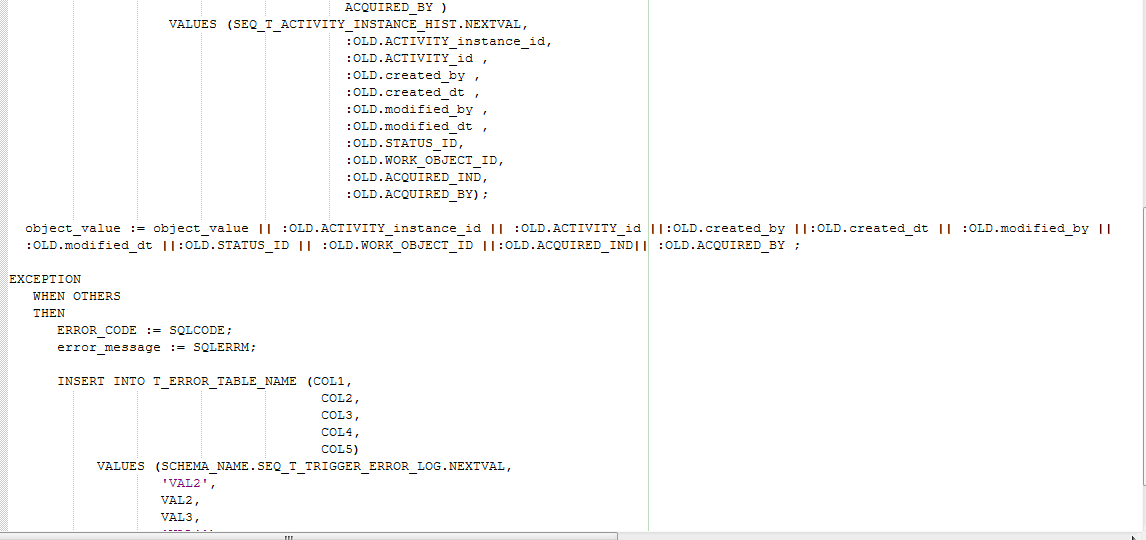
This statement sets a new maximum value for the customers\_seq

1. ALTER SEQUENCE customers\_seq CYCLE CACHE 5;

This statement turns on CYCLE and CACHE for the customers\_seq sequence:

2/2/2018

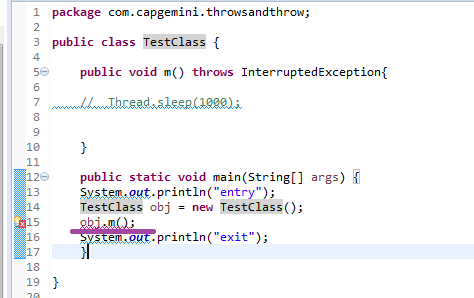
1. Syntax to create a parameterized view in oracle
2. Syntax for Substring in view🡺 
3. Needed more information on the PIVOT🡺 
4. I SHOULD LEARN SOME OF THE PATTERN USED IN REQEXP\_SUBSTR to understand it completely 🡺
5. View analysis
6. While doing any changes in a view run it once so that my changes will get saved otherwise my changes will be lost
7. In this case a column with name SOURCE\_SYSTEM will be created and it will have only “MAN” as the value
8. Needed more information on this pattern used here for REGEXP\_SUBSTR🡺 
9. 
10. Example for Rregexp\_REPLACE , CASE 🡺
11. WHAT KIND OF JOIN IS THIS 🡺 
12. NLSSORT?
13. TRIM FUNCTION
14. 



8/2/2018

Exception Coding

1. See her we will not get CE here if we don’t call m() which is throwing the Checked exception



1. Core java Interview Questions from 🡺 <http://www.baeldung.com/java-exceptions-interview-questions>

### **Q2. What is the purpose of the**throw**and**throws**keywords?**

The throws keyword is used to specify that a method may raise an exception during its execution. It enforces explicit exception handling when calling a method:

|  |  |
| --- | --- |
| 1  2  3 | public void simpleMethod() throws Exception {      // ...  } |

The throw keyword allows us to throw an exception object to interrupt the normal flow of the program. This is most commonly used when a program fails to satisfy a given condition:

|  |  |
| --- | --- |
| 1  2  3 | if (task.isTooComplicated()) {      throw new TooComplicatedException("The task is too complicated");  } |

### **Q3. How can you handle an exception?**

By using a try-catch-finally statement:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try {      // ...  } catch (ExceptionType1 ex) {      // ...  } catch (ExceptionType2 ex) {      // ...  } finally {      // ...  } |

The block of code in which an exception may occur is enclosed in a try block. This block is also called “protected” or “guarded” code.

If an exception occurs, the catch block that matches the exception being thrown is executed, if not, all catch blocks are ignored.

The finally block is always executed after the try block exits, whether an exception was thrown or not inside it.

### **Q4. How can you catch multiple exceptions?**

There are three ways of handling multiple exceptions in a block of code.

The first is to use a catch block that can handle all exception types being thrown:

|  |  |
| --- | --- |
| 1  2  3  4  5 | try {      // ...  } catch (Exception ex) {      // ...  } |

You should keep in mind that the recommended practice is to use exception handlers that are as accurate as possible.

Exception handlers that are too broad can make your code more error-prone, catch exceptions that weren’t anticipated, and cause unexpected behavior in your program.

The second way is implementing multiple catch blocks:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | try {      // ...  } catch (FileNotFoundException ex) {      // ...  } catch (EOFException ex) {      // ...  } |

Note that, if the exceptions have an inheritance relationship; the child type must come first and the parent type later. If we fail to do this, it will result in a compilation error.

The third is to use a multi-catch block:

|  |  |
| --- | --- |
| 1  2  3  4  5 | try {      // ...  } catch (FileNotFoundException | EOFException ex) {      // ...  } |

This feature, first introduced in Java 7; reduces code duplication and makes it easier to maintain.

### **Q5. What is the difference between a checked and an unchecked exception?**

A checked exception must be handled within a try-catch block or declared in a throws clause; whereas an unchecked exception is not required to be handled nor declared.

Checked and unchecked exceptions are also known as compile-time and runtime exceptions respectively.

All exceptions are checked exceptions, except those indicated by Error, RuntimeException, and their subclasses.

### **Q6. What is the difference between an exception and error?**

An exception is an event that represents a condition from which is possible to recover, whereas error represents an external situation usually impossible to recover from.

All errors thrown by the JVM are instances of Error or one of its subclasses, the more common ones include but are not limited to:

* OutOfMemoryError – thrown when the JVM cannot allocate more objects because it is out memory, and the garbage collector was unable to make more available
* StackOverflowError – occurs when the stack space for a thread has run out, typically because an application recurses too deeply
* ExceptionInInitializerError – signals that an unexpected exception occurred during the evaluation of a static initializer
* NoClassDefFoundError – is thrown when the classloader tries to load the definition of a class and couldn’t find it, usually because the required classfiles were not found in the classpath
* UnsupportedClassVersionError – occurs when the JVM attempts to read a class file and determines that the version in the file is not supported, normally because the file was generated with a newer version of Java

Although an error can be handled with a try statement, this is not a recommended practice since there is no guarantee that the program will be able to do anything reliably after the error was thrown.

### **Q7. What exception will be thrown executing the following code block?**

|  |  |
| --- | --- |
| 1  2 | Integer[][] ints = { { 1, 2, 3 }, { null }, { 7, 8, 9 } };  System.out.println("value = " + ints[1][1].intValue()); |

It throws an ArrayIndexOutOfBoundsException since we’re trying to access a position greater than the length of the array.

### **Q8. What is exception chaining?**

Occurs when an exception is thrown in response to another exception. This allows us to discover the complete history of our raised problem:

|  |  |
| --- | --- |
| 1  2  3  4  5 | try {      task.readConfigFile();  } catch (FileNotFoundException ex) {      throw new TaskException("Could not perform task", ex);  } |

### **Q9. What is a stacktrace and how does it relate to an exception?**

A stack trace provides the names of the classes and methods that were called, from the start of the application to the point an exception occurred.

It’s a very useful debugging tool since it enables us to determine exactly where the exception was thrown in the application and the original causes that led to it.

### **Q10. Why would you want to subclass an exception?**

If the exception type isn’t represented by those that already exist in the Java platform, or if you need to provide more information to client code to treat it in a more precise manner, then you should create a custom exception.

Deciding whether a custom exception should be checked or unchecked depends entirely on the business case. However, as a rule of thumb; if the code using your exception can be expected to recover from it, then create a checked exception otherwise make it unchecked.

Also, you should inherit from the most specific Exception subclass that closely relates to the one you want to throw. If there is no such class, then choose Exception as the parent.

### **Q11. What are some advantages of exceptions?**

Traditional error detection and handling techniques often lead to spaghetti code hard to maintain and difficult to read. However, exceptions enable us to separate the core logic of our application from the details of what to do when something unexpected happens.

Also, since the JVM searches backward through the call stack to find any methods interested in handling a particular exception; we gain the ability to propagate an error up in the call stack without writing additional code.

Also, because all exceptions thrown in a program are objects, they can be grouped or categorized based on its class hierarchy. This allows us to catch a group exceptions in a single exception handler by specifying the exception’s superclass in the catch block.

### **Q12. Can you throw any exception inside a lambda expression’s body?**

When using a standard functional interface already provided by Java, you can only throw unchecked exceptions because standard functional interfaces do not have a “throws” clause in method signatures:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | List<Integer> integers = Arrays.asList(3, 9, 7, 0, 10, 20);  integers.forEach(i -> {      if (i == 0) {          throw new IllegalArgumentException("Zero not allowed");      }      System.out.println(Math.PI / i);  }); |

However, if you are using a custom functional interface, throwing checked exceptions is possible:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4 | @FunctionalInterface  public static interface CheckedFunction<T> {      void apply(T t) throws Exception;  } | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | | public void processTasks(    List<Task> taks, CheckedFunction<Task> checkedFunction) {      for (Task task : taks) {          try {              checkedFunction.apply(task);          } catch (Exception e) {              // ...          }      }  }    processTasks(taskList, t -> {      // ...      throw new Exception("Something happened");  }); |

### **Q13. What are the rules we need to follow when overriding a method that throws an exception?**

Several rules dictate how exceptions must be declared in the context of inheritance.

When the parent class method doesn’t throw any exceptions, the child class method can’t throw any checked exception, but it may throw any unchecked.

Here’s an example code to demonstrate this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() {          // ...      }  }    class Child extends Parent {      void doSomething() throws IllegalArgumentException {          // ...      }  } |

The next example will fail to compile since the overriding method throws a checked exception not declared in the overridden method:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // Compilation error      }  } |

When the parent class method throws one or more checked exceptions, the child class method can throw any unchecked exception; all, none or a subset of the declared checked exceptions, and even a greater number of these as long as they have the same scope or narrower.

Here’s an example code that successfully follows the previous rule:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class Parent {      void doSomething() throws IOException, ParseException {          // ...      }        void doSomethingElse() throws IOException {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // ...      }        void doSomethingElse() throws FileNotFoundException, EOFException {          // ...      }  } |

Note that both methods respect the rule. The first throws fewer exceptions than the overridden method, and the second, even though it throws more; they’re narrower in scope.

However, if we try to throw a checked exception that the parent class method doesn’t declare or we throw one with a broader scope; we’ll get a compilation error:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Parent {      void doSomething() throws FileNotFoundException {          // ...      }  }    class Child extends Parent {      void doSomething() throws IOException {          // Compilation error      }  } |

When the parent class method has a throws clause with an unchecked exception, the child class method can throw none or any number of unchecked exceptions, even though they are not related.

Here’s an example that honors the rule:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Parent {      void doSomething() throws IllegalArgumentException {          // ...      }  }    class Child extends Parent {      void doSomething()        throws ArithmeticException, BufferOverflowException {          // ...      }  } |

### **Q14. Will the following code compile?**

|  |  |
| --- | --- |
| 1  2  3  4 | void doSomething() {      // ...      throw new RuntimeException(new Exception("Chained Exception"));  } |

Yes. When chaining exceptions, the compiler only cares about the first one in the chain and, because it detects an unchecked exception, we don’t need to add a throws clause.

### **Q15. Is there any way of throwing a checked exception from a method that does not have a**throws**clause?**

Yes. We can take advantage of the type erasure performed by the compiler and make it think we are throwing an unchecked exception, when, in fact; we’re throwing a checked exception:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public <T extends Throwable> T sneakyThrow(Throwable ex) throws T {      throw (T) ex;  }    public void methodWithoutThrows() {      this.<RuntimeException>sneakyThrow(new Exception("Checked Exception"));  } |

## ****3. Conclusion****

1. Core Java Exception Interview Question from 🡺 <http://javaconceptoftheday.com/java-exception-handling-interview-questions-and-answers/>

# [**30 Java Exception Handling Interview Questions And Answers**](http://javaconceptoftheday.com/java-exception-handling-interview-questions-and-answers/)

1. The url for the above 30 Questions I s🡺 <http://javaconceptoftheday.com/java-exception-handling-interview-questions-and-answers/>
2. Interview Questions from Tpoint