http://javarevisited.blogspot.in/2011/11/collection-interview-questions-answers.html

1) Exceptions in java

Checked vs Unchecked Exceptions in Java In Java, there two types of exceptions:

i) Checked Exceptions : are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

For example, consider the following Java program that opens file at locatiobn “C:\test\a.txt” and prints first three lines of it. The program doesn’t compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

```
import java.io.*;

class Main {
    public static void main(String[] args) {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        fileInput.close();
    }
}
```

Output:

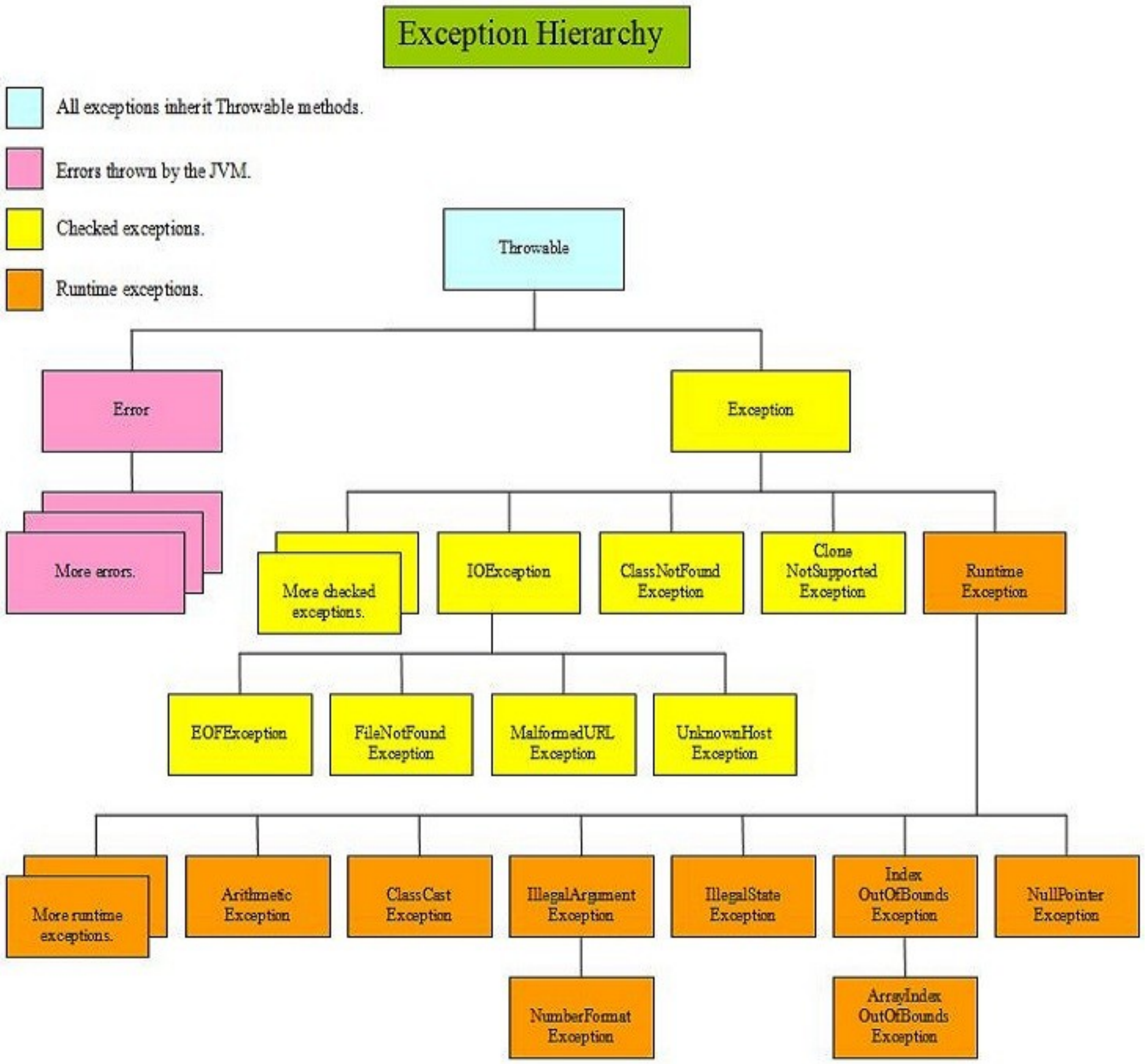
```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source
code -

unreported exception java.io.FileNotFoundException; must be caught or declared
to be
thrown
            at Main.main(Main.java:5)
```

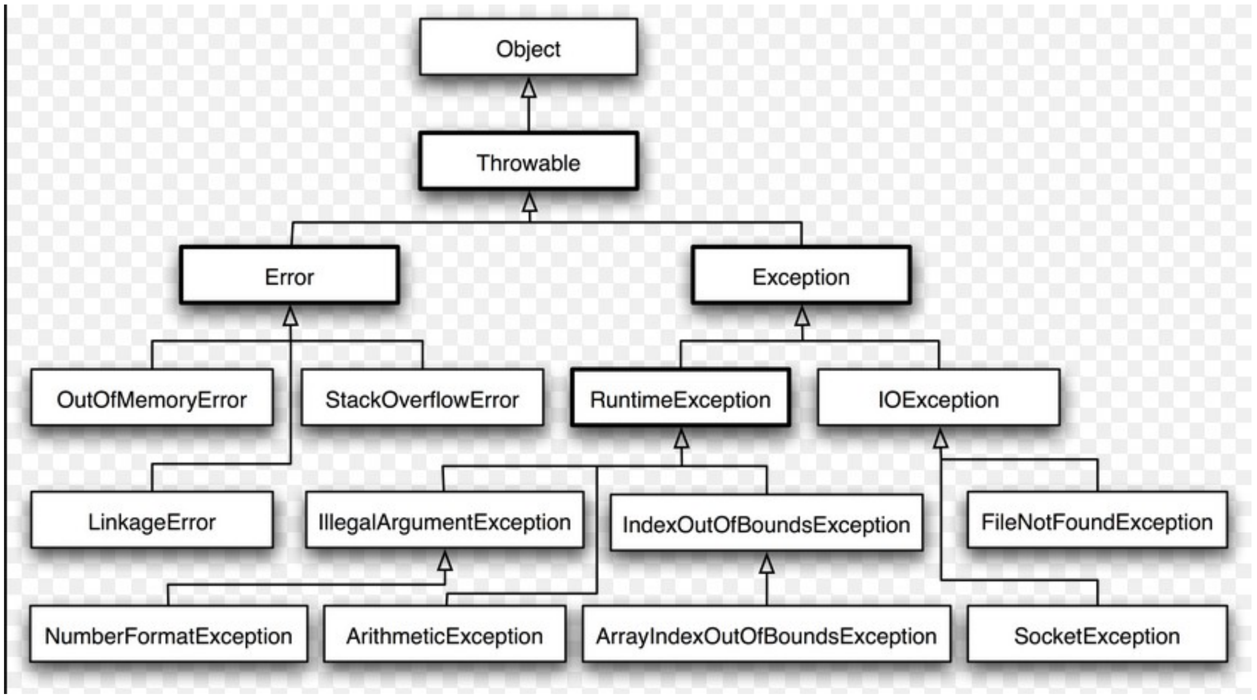
To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

```
import java.io.*;

class Main {
    public static void main(String[] args) throws IOException {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        fileInput.close();
    }
}
```

Output: First three lines of file “C:\test\a.txt”

ii) Unchecked are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.
In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.


Exception Hierarchy
All exceptions inherit Throwable methods.
Errors thrown by the JVM.
Checked exceptions.
Runtime exceptions.
Throwable
Error
Exception
More errors.
More checked exceptions.
IOException
ClassNotFound Exception
Clone NotSupported Exception
Runtime Exception
EOFException
FileNotFound Exception
MalformedURL Exception
UnknownHost Exception
More runtime exceptions.
Arithmetic Exception
ClassCast Exception
IllegalArgument Exception
IllegalState Exception
Index OutOfBounds Exception
NullPointer Exception
NumberFormat Exception
ArrayIndex OutOfBounds Exception


Object
Throwable
Error
Exception
OutOfMemoryError
StackOverflowError
RuntimeException
IOException
LinkageError
IllegalArgumentException
IndexOutOfBoundsException
FileNotFoundException
NumberFormatException
ArithmeticException
ArrayIndexOutOfBoundsException
SocketException

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile, because *ArithmeticException* is an unchecked exception.

```
class Main {
   public static void main(String args[]) {
      int x = 0;
      int y = 10;
      int z = y/x;
  }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at Main.main(Main.java:5)
Java Result: 1
```

2) Different ways to create objects in Java?

It’s a bit tricky question and people often get confused. There are four different ways (I really don’t know is there a fifth way to do this) to create objects in java:

1. Using new keyword

this is the most common way to create an object in java. I read somewhere that almost 99% of objects are created in this way.

```
MyObject object = new MyObject();
```

2. Using Class.forName()

If we know the name of the class & if it has a public default constructor we can create an object in this way.

```
MyObject object = (MyObject)Class.forName(“subin.rnd.MyObject”).newInstance();
```

3. Using clone()

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

```
MyObject anotherObject = new MyObject();
MyObject object = anotherObject.clone();
```

4. Using object deserialization

Object deserialization is nothing but creating an object from its serialized form.

```
ObjectInputStream inStream = new ObjectInputStream(anInputStream );
MyObject object = (MyObject) inStream.readObject();
```

3) Difference between java version 1.6 and 1.5. and 1.8 (4 major difference)

Java 1.5

- **Generics:** provides compile-time (static) type safety for collections and eliminates the need for most typecasts (type conversion).

- Metadata: also called annotations; allows language constructs such as classes and methods to be tagged with additional data, which can then be processed by metadata-aware utilities.

- **Autoboxing/unboxing**: automatic conversions between primitive types (such as int) and primitive wrapper classes (such as integer).

- **Enumerations**: the enum keyword creates a typesafe, ordered list of values (such as day.monday, day.tuesday, etc.). Previously this could only be achieved by non-typesafe constant integers or manually constructed classes (typesafe enum pattern).

- Swing: new skinnable look and feel, called synth.

- **Var args**: the last parameter of a method can now be declared using a type name followed by three dots (e.g. Void drawtext(string... Lines)). In the calling code any number of parameters of that type can be used and they are then placed in an array to be passed to the method, or alternatively the calling code can pass an array of that type.

- Enhanced for each loop: the for loop syntax is extended with special syntax for iterating over each member of either an array or any iterable, such as the standard collection classesfix the previously broken semantics of the java memory model, which defines how threads interact through memory.

- Automatic stub generation for rmi objects.

- Static imports concurrency utilities in package java.util.concurrent.

- Scanner class for parsing data from various input streams and buffers.

- **Assertions**

- **StringBuilder class (in java.lang package)**

- **Annotations**

-Concurrent collection classes added

(ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue)

Java 1.6

Features added:

- Support for older win9x versions dropped.

- Scripting lang support: Generic API for integration with scripting languages, & built-in mozilla javascript rhino integration

- Dramatic performance improvements for the core platform, and swing.

- Improved web service support through JAX-WS JDBC 4.0 support

- Java compiler API: an API allowing a java program to select and invoke a java compiler programmatically.

- Upgrade of JAXB to version 2.0: including integration of a stax parser.

- Support for pluggable annotations

- Many GUI improvements, such as integration of swingworker in the API, table sorting and filtering, and true swing double-buffering (eliminating the gray-area effect).

Java 8

- **Lambda Expression and Virtual Extension Methods**
  Highlighting feature of Java SE 8 is the implementation of Lambda expressions and supporting features to the Java programming language and platform.
- **Date and Time API**
  This new API will allow developers to handle date and time in a more natural, cleaner and easier to understand way.
- **Nashhorn JavaScript Engine**
  A new lightweight, high performance implementation of JavaScript engine is integrated to JDk and is available to Java applications via existing APIs.
- **Improved Security**
  Replacing the existing hand-maintained list of caller sensitive methods with a mechanism that accurately identifies such methods and allows their callers to be discovered reliably.

4) What is javaFX? Tell me in short.

JavaFX is a software platform for creating and delivering rich internet applications (RIAs) that can run across a wide variety of devices. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future.[3]
JavaFX has support for desktop computers and web browsers on Microsoft, Linux, and Mac OS X.

5) What will happen if you put return statement or System.exit () on try or catch block ? Will finally block execute?

This is a very popular tricky Java question and its tricky because many programmer think that no matter what, but finally block will always execute. This question challenge that misconcept by putting return statement in try or catch block or calling System.exit from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put returnstatement in try block or catch block but finally block won't run if you call System.exit form try or catch.

6) Can you override private or static method in Java ?

Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, you can not override private or static method in Java, if you create similar method with same return type and same method arguments in child class then it will hide the super class method, this is known as method hiding. Similarly you cannot override private method in sub class because it's not accessible there, what you do is create another private method with same name in child class. See Can you override private method in Java or more details.

7) What will happen if we put a key object in a HashMap which is already there ?

This tricky Java questions is part of another frequently asked question, How HashMap works in Java. HashMap is also a popular topic to create confusing and tricky question in Java. Answer of this question is, if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys. Same key will result in same hashcode and will end up at same position in bucket. Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take Key object form each entry and compare with this new key using equals() method, if that return true then value object in that entry will be replaced by new value. See How HashMap works in Java for more tricky Java questions from HashMap.

8) What is difference between CyclicBarrier and CountDownLatch in Java?

Relatively newer Java tricky question, only been introduced form Java 5. Main difference between both of them is that you can reuse CyclicBarrier even if Barrier is broken but you can not reuse CountDownLatch in Java.
See CyclicBarrier vs CountDownLatch in Java for more differences.

9) What is difference between StringBuffer and StringBuilder in Java ?

Classic Java questions which some people think tricky and some consider very easy.StringBuilder in Java was introduced in JDK 1.5 and only difference between both of them is that StringBuffer methods
e.g. length(), capacity() or append() are synchronized while corresponding

methods in StringBuilder are not-synchronized. Because of this fundamental difference, concatenation of String using StringBuilder is faster than StringBuffer. Actually its considered bad practice to use StringBuffer any more, because in almost 99% scenario, you perform string concatenation on same thread. See StringBuilder vs StringBuffer for more differences.

10) Can you access non static variable in static context?

Another tricky Java question from Java fundamentals. No you can not access non-static variable from static context in Java. If you try, it will give compile time error. This is actually a common problem beginners in Java face, when they try to access instance variable inside main method. Because main is static in Java, and instance variables are non-static, you can not access instance variable inside main. Read why you can not access non-static variable from static method to learn more about this tricky Java questions.

11) What is difference between fail-fast and fail-safe Iterators?

This is relatively new collection interview questions and can become trick if you hear the term fail-fast and fail-safe first time. Fail-fast Iterators throws ConcurrentModificationException when one Thread is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure. On the other hand fail-safe Iterators works on copy of collection instead of original collection.

12) What is difference between poll() and remove() method of Queue interface?

Though both poll() and remove() method from Queue is used to remove object and returns head of the queue, there is subtle difference between them. If Queue is empty() then a call to remove() method will throw Exception, while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon queue's ordering policy and varies between different implementation, for example PriorityQueue keeps lowest element as per Comparator or Comparable at head position.

13) How do you remove an entry from a Collection? and subsequently what is difference between remove() method of Collection and remove() method of Iterator, which one you will use, while removing elements during iteration?

Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove object at specific index. You can use any of these method to remove an entry from Collection, while not iterating.

Things change, when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes current element from Iterator's perspective.

If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use Iterator remove() method to remove objects from Collection.

14) . What is difference between Synchronized Collection and Concurrent Collection?

Java 5 has added several new Concurrent Collection classes e.g. ConcurrentHashMap, CopyOnWriteArrayList, BlockingQueue etc, which has made Interview questions on Java Collection even trickier. Java Also provided way to get Synchronized copy of collection e.g. ArrayList, HashMap by using Collections.synchronizedMap() Utility function.One Significant difference is that Concurrent Collections has better performance than synchronized Collection because they lock only a portion of Map to achieve concurrency and Synchronization.

15) ConcurrentHashMap vs Hashtable vs Synchronized Map

Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself, which uses synchronized methods to achieve thread-safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. Only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using Collections.synchronizedMap() method. On the other hand, ConcurrentHashMap is especially designed for concurrent use i.e. more than one thread. By default it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in internal implementation of ConcurrentHashMap class. Unlike Hashtable and Synchronized Map, it never locks whole Map, instead it divides the map in segments and locking is done on those. Though it perform better if number of reader threads is greater than number of writer threads.

To be frank, Collections classes are heart of Java API though I feel using them judiciously is an art. Its my personal experience where I have improved performance of Java application by using ArrayList where legacy codes were unnecessarily using Vector etc. Prior Java 5, One of the major drawback of Java Collection framework was lack of scalability. In multi-threaded Java application synchronized collection classes like Hashtable and Vector quickly becomes bottleneck; to address scalability JDK 1.5 introduces some good concurrent collections which is highly efficient for high volume, low latency system electronic trading systems In general those are backbone for Concurrent fast access of stored data. In this tutorial we will look on ConcurrentHashMap, Hashtable, HashMap and synchronized Map .

How HashMap works in Java

16) Why String, Integer and other wrapper classes are considered good keys ?

String, Integer and other wrapper classes are natural candidates of HashMap key, and String is most frequently used key as well because String is immutable and final,and overrides equals and hashcode() method. Other wrapper class also shares similar property. Immutabiility is required, in order to prevent changes on fields used to calculate hashCode() because if key object return different hashCode during insertion and retrieval than it won't be possible to get object from HashMap. Immutability is best as it offers other advantages as well like thread-safety, If you can keep your hashCode same by only making certain fields final, then you go for that as well.
Since equals() and hashCode() method is used during reterival of value object from HashMap, its important that key object correctly override these methods and follow contact. If unequal object return different hashcode than chances of collision will be less which subsequently improve performance of HashMap.

17. What is difference between Iterator and Enumeration?

1) Iterator duplicate functionality of Enumeration with one addition of remove() method and both provide navigation functionally on objects of Collection.

Another difference is that Iterator is more safe than Enumeration and doesn't allow another thread to modify collection object during iteration except remove() method and throws ConcurrentModificaitonException.

Vector supports enumeration but arraylist not support it.

Read more: http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html#ixzz3Wu2YmZ83

18. How does HashSet is implemented in Java, How does it uses Hashing ?

This is a tricky question in Java, because for hashing you need both key and value and there is no key for store it in a bucket, then how exactly HashSet store element internally. Well, HashSet is built on top of HashMap. If you look at source code of java.util.HashSet class, you will find that that it uses a HashMap with same values for all keys, as shown below :

```
public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, java.io.Serializable

{
   private transient HashMap<E,Object> map;
   // Dummy value to associate with an Object in the backing Map
   private static final Object PRESENT = new Object();

   public HashSet() {
      map = new HashMap<>();
   }
      // SOME CODE ,i.e Other methods in Hash Set

   public boolean add(E e) {
      return map.put(e, PRESENT)==null;
   }

   // SOME CODE ,i.e Other methods in Hash Set
}
```

Since keys are unique in a HashMap, it provides uniqueness guarantee of Set interface.

19. What do you need to do to use a custom object as key in Collection classes like Map or Set?

Answer is : If you are using any custom object in Map as key, you need to override equals() and hashCode() method, and make sure they follow there contract. On the other hand if you are storing a custom object in Sorted Collection e.g. SortedSet or SortedMap, you also need to make sure that your equals() method is consistent to compareTo() method, otherwise those collection will not follow there contacts e.g. Set may allow duplicates.

20) Difference between HashMap and Hashtable?

1. Synchronization or Thread Safe : This is the most important difference between two . HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized.
When to use HashMap ? answer is if your application do not require any multi-threading task, in other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.

2. Null keys and null values : Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.

21) When do you use ConcurrentHashMap in Java?

This is another advanced level collection interview questions in Java which normally asked to check whether interviewer is familiar with optimization done on ConcurrentHashMap or not. ConcurrentHashMap is better suited for situation where you have multiple readers and one

Writer or fewer writers since Map gets locked only during write operation. If you have equal number of reader and writer than ConcurrentHashMap will perform in line of Hashtable or synchronized HashMap.

22) What is NavigableMap in Java ? What is benefit over Map?

NavigableMap Map was added in Java 1.6, it adds navigation capability to Map data structure. It provides methods like lowerKey() to get keys which is less than specified key, floorKey() to return keys which is less than or equal to specified key, ceilingKey() to get keys which is greater than or equal to specified key and higherKey() to return keys which is greater specified key from a Map. It also provide similar methods to get entries e.g. lowerEntry(), floorEntry(), ceilingEntry() and higherEntry(). Apart from navigation methods, it also provides utilities to create sub-Map e.g. creating a Map from entries of an exsiting Map like tailMap, headMap and subMap. headMap() method returns a NavigableMap whose keys are less than specified, tailMap() returns a NavigableMap whose keys are greater than the specified and subMap() gives a NavigableMap between a range, specified by toKey to fromKey.

23) Which one you will prefer between Array and ArrayList for Storing object and why?

Though ArrayList is also backed up by array, it offers some usability advantage over array in Java. Array is fixed length data structure, once created you can not change it's length. On the other hand, ArrayList is dynamic, it automatically allocate a new array and copies content of old array, when it resize. Another reason of using ArrayList over Array is support of Generics. Array doesn't support Generics, and if you store an Integer object on a String array, you will only going to know about it at runtime, when it throws ArrayStoreException. On the other hand, if you use ArrayList, compiler and IDE will catch those error on the spot. So if you know size in advance and you don't need re-sizing than use array, otherwise use ArrayList.

24) Can we replace Hashtable with ConcurrentHashMap?

Answer 3 : Yes we can replace Hashtable with ConcurrentHashMap and that's what suggested in Java documentation of ConcurrentHashMap. but you need to be careful with code which relies on locking behavior of Hashtable. Since Hashtable locks whole Map instead of portion of Map, compound operations like if(Hashtable.get(key) == null) put(key, value) works in Hashtable but not in concurrentHashMap. instead of this use putIfAbsent() method of ConcurrentHashMap

25) Differences between String, StringBuffer and StringBuilder in Java

1) String is immutable while StringBuffer and StringBuilder is mutable object.

2) StringBuffer is synchronized while StringBuilder is not which makes StringBuilder faster than StringBuffer.

3) Concatenation operator "+" is internal implemented using either StringBuffer or StringBuilder.

4) Use String if you require immutability, use Stringbuffer in java if you need mutable + thread-safety and use StringBuilder in Java if you require mutable + without thread-safety.

JSP Questions:

1) What is the difference between page and page Context?

1. The page implicit object is of type Object and it is assigned a reference to the servlet that executing the _jspService() method. Page is the instance of the JSP page's servlet processing the current request. Not typically used by JSP page authors. Thus in the Servlet generated by tomcat the page object is created as
2. Object page = this;
3. Since page is a variable of type Object, it cannot be used to directly call the servlet methods. To access any of the methods of the servlet through page it must be first cast to type Servlet.
4. <%= this.getServletInfo(); %>
   <%= ((Servlet)page).getServletInfo(); %>
5. But the following code will generate error, because can not use page directly without casting:
   <%= page.getServletInfo(); %>

PageContext:

1) In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set, get or remove attribute from one of the following scopes:

Page
request
session
application

Note: In JSP, page scope is the default scope.

Example: setting attribute value in session level:
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

Getting value from session level:
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);

2) Used for storing and retrieving page-related information and sharing objects within the same translation unit and same request.

Also used as a convenience class that maintains a table of all the other implicit objects. For example,

```
     public void _jspService (
                    HttpServletRequest request,  HttpServletResponse response
) throws java.io.IOException,
ServletException {

 ...
   try {

     ...
   application = pageContext.getServletContext();

   config = pageContext.getServletConfig();

  session = pageContext.getSession();

   out = pageContext.getOut();
     ...

    } catch (Throwable t) {
     ...
    } finally {
     ...
    }
}
```

2) Which JSP life cycle method can be overridden?

- You cannot override the _jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page.
- jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page.
- It is good programming practice to free any allocated resources within jspDestroy().

3) How can avoid direct access of jsp pages from client interaction?

For a quick solution, just put your JSP pages to the WEB-INF folder (then they will not be directly accessible) and define them like this:

```
   <servlet>
       <description>
       </description>
       <display-name>hidden</display-name>
       <servlet-name>hidden</servlet-name>
       <jsp-file>/WEB-INF/hidden.jsp</jsp-file>
    </servlet>
    <servlet-mapping>
       <servlet-name>hidden</servlet-name>
       <url-pattern>/hidden</url-pattern>
    </servlet-mapping>
```

4) **Can we use JSP implicit objects in a method defined in JSP Declaration?**
No we can’t because JSP implicit objects are local to service method and added by JSP Container while translating JSP page to servlet source code. JSP Declarations code goes outside the service method and used to create class level variables and methods and hence can’t use JSP implicit objects.

5) **Which implicit object is not available in normal JSP pages?**
JSP **exception** implicit object is not available in normal JSP pages and it’s used in JSP error pages only to catch the exception thrown by the JSP pages and provide useful message to the client.

6) What are the benefits of PageContext implicit object?
JSP pageContext implicit object is instance of **javax.servlet.jsp.PageContext abstract** class implementation. We can use **pageContext** to get and set attributes with different scopes and to forward request to other resources.
pageContext object also hold reference to other implicit object.

7) **Can we define method in scriplet?**
Not possible: we have to use You need to use declaration syntax (<%! ... %>):

```
<%!
  public String doSomething(String param) {
  }
%>
```

All Scriplet code goes inside _jspService() method of jsp. If we write any method here then resultant we creating method inside other method which is violation of java rules.

```
<%
  String test = doSomething("test");
%>
```

8) **How can we configure initParam for JSP?**
web.xml

```
<servlet>
    <servlet-name>GetInitParam</servlet-name>
    <jsp-file>/GetInitParam.jsp</jsp-file>
    <init-param>
        <param-name>url</param-name>
        <param-value>hello</param-value>
    </init-param>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>GetInitParam</servlet-name>
    <url-pattern>/GetInitParam.jsp</url-pattern>
</servlet-mapping>
```

GetInitParam.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Example of getting init param</title>
</head>
<body>
<%!
    String url= null;
    public void jspInit() {
        ServletConfig config = getServletConfig();
        url= config.getInitParameter("url");
    }
%>
<%
    System.out.println(url);
%>
</body>
</html>
```

9) Can we define a class on JSP page?

Yes we can create like this:

```
<%@page contentType="text/html" pageEncoding="MacRoman"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<%!
private static class NdBadIdea {
  private final int foo = 42;

  public int getFoo() {
    return foo;
  }
}
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=MacRoman">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <%=new NdBadIdea().getFoo()%>
    </body>
</html>
```

10) When container initialize multiple JSP/Servlet Objects?

If we have multiple servlet and servlet-mapping elements in deployment descriptor for a single servlet or JSP page, then container will initialize an object for each of the element and all of these instances will have their own ServletConfig object and init params.For example, if we configure a single JSP page in web.xml like below.

```
<servlet>
  <servlet-name>Test</servlet-name>
  <jsp-file>/WEB-INF/test.jsp</jsp-file>
  <init-param>
    <param-name>test</param-name>
    <param-value>Test Value</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>Test</servlet-name>
  <url-pattern>/Test.do</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>Test1</servlet-name>
  <jsp-file>/WEB-INF/test.jsp</jsp-file>
</servlet>

<servlet-mapping>
  <servlet-name>Test1</servlet-name>
  <url-pattern>/Test1.do</url-pattern>
</servlet-mapping>
```

11) What is difference between include directive and jsp:include action?

1) <%@ include file="filename" %> is the JSP include directive.
   At JSP page translation time, the content of the file given in the include directive is 'pasted' as it is, in the place where the JSP include directive is used. Then the source JSP page is converted into a java servlet class. The included file can be a static resource or a JSP page. Generally JSP include directive is used to include header banners and footers.
2) The JSP compilation procedure is that, the source JSP page gets compiled only if that page has changed. If there is a change in the

included JSP file, the source JSP file will not be compiled and therefore the modification will not get reflected in the output.

3) <jsp:include page="relativeURL" /> is the JSP include action element. The jsp:include action element is like a function call. At runtime, the included file will be 'executed' and the result content will be included with the soure JSP page. When the included JSP page is called, both the request and response objects are passed as parameters.
4) If there is a need to pass additional parameters**, then jsp:param** element can be used. If the resource is static, its content is inserted into the calling JSP file, since there is no processing needed.

12) Difference between JSPWriter and servlet printWriter?

- **PrintWriter** is the actual object responsible for writing the content in response.
- JspWriter uses the PrintWriter object behind the scene and provide buffer support.
- When the buffer is full or flushed, JspWriter uses the PrintWriter object to write the content into response.

public class PrintWriter extends Writer {
public abstract class JspWriter extends Writer {

28) Ques: How you make servlet thread safe.

There are two different ways of making a servlet thread safe namely

1. By implementing SingleThreadModel. By implementing a SingleThreadModel it will be possible to create a Thread safe servlet. There can only be one user at a given point of time.
2. Synchornize the part of sensitive code.We can allow a single user at a given point of time by making that part of the code which is sensitive as synchronized.

There are situations where we want to protect your servlet member variables from being modified by different clients.In this case you can have your servlet by implementing the marker interface SigleThreadModel.
Everytime a client makes request to a servlet by implementing this interface,servlet engine will create a new instance of servlet.
For performance reason,servlet engine can also maintain a instance pool,handing out instances as they are needed.Or it could also serialize client request executing one after another.

29) If we put destroy method inside init method in servlet what happen.

The meaning of destroy() in java servlet is, the content gets executed just before when the container decides to destroy the servlet. But if you invoke the destroy() method yourself, the content just gets executed and then the respective process continues. With respective to this question, the destroy() gets executed and then the servlet initialization gets completed.:

30) Difference between wait and sleep method.

- Thread.sleep() will wait go to sleep for amount of time that is specified in it.After the times finishes it resumes automatically.

  wait() will also wait for some I/O and will NOT resume automatically but will resume only when notify() or notifyall() is called.
  - The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.
  - Thread.sleep() sends the current thread into the “Not Runnable” state for some amount of time. The thread keeps the monitors it has acquired — i.e. if the thread is currently in a synchronized block or method no other thread can enter this block or method. If another thread calls t.interrupt() it will wake up the sleeping thread. Note that sleep is a static method, which means that it always affects the current thread (the one that is executing the sleep method). A common mistake is to call t.sleep() where t is a different thread; even then, it is the current thread that will sleep, not the t thread.
  - object.wait() sends the current thread into the “Not Runnable” state, like sleep(), but with a twist. Wait is called on an object, not a thread; we call this object the “lock object.” Before lock.wait() is called, the current thread must synchronize on the lock object; wait() then releases this lock, and adds the thread to the “wait list” associated with the lock. Later, another thread can synchronize on the same lock object and call lock.notify(). This wakes up the original, waiting thread. Basically, wait()/notify() is like sleep()/interrupt(), only the active thread does not need a direct pointer to the sleeping thread, but only to the shared lock object.

31) Singleton design pattern with synchronization and clone() method not to be able to create object of singleton class.

Singleton is one of the most widely used creational design pattern to restrict the object creation by applications. In real world applications, resources like Database connections or Enterprise Information Systems (EIS) are limited and should be used wisely to avoid any resource crunch. To achieve this, we can implement Singleton design pattern to create a wrapper class around the resource and limit the number of object created at runtime to one.

In general we follow below steps to create a singleton class:

1) Override the private constructor to avoid any new object creation with new operator.
2) Declare a private static instance of the same class
3) Provide a public static method that will return the singleton class instance variable. If the variable is not initialized then initialize it or else simply return the instance variable.

ASingleton.java

```
package com.journaldev.designpatterns;

public class ASingleton {

    private static  ASingleton instance = null;

    private ASingleton() {
    }

    public static ASingleton getInstance() {
        if (instance == null) {
            instance = new ASingleton();
        }
        return instance;
    }

}
```

In the above code, getInstance() method is not thread safe i.e multiple threads can access it at the same time and for the first few threads when the instance variable is not initialized, multiple threads can enters the if loop and create multiple instances and break our singleton implementation.

There are three ways through which we can achieve thread safety.

1. **Create the instance variable at the time of class loading:**

Pros:

- Thread safety without synchronization
- Easy to implement

Cons:

- Early creation of resource that might not be used in the application.
- The client application can’t pass any argument, so we can’t reuse it. For example, having a generic singleton class for database connection where client application supplies database server properties.
-

2. **Synchronize the getInstance() method:**

Pros:

- Thread safety is guaranteed.
- Client application can pass parameters
- Lazy initialization achieved
-

Cons:

- Slow performance because of locking overhead.
- Unnecessary synchronization that is not required once the instance variable is initialized.

3. Use synchronized block inside the if loop:

Pros:

- Thread safety is guaranteed
- Client application can pass arguments
- Lazy initialization achieved
- Synchronization overhead is minimal and applicable only for first few threads when the variable is null.

ASingleton.java

```
package com.journaldev.designpatterns;

public class ASingleton{

    private static ASingleton instance= null;
    private static Object mutex= new Object();
    private ASingleton(){
    }

    public static ASingleton getInstance(){
        if(instance==null){
            synchronized (mutex){
                if(instance==null) instance= new ASingleton();
            }
        }
        return instance;
    }


    }

```

Draconian synchronization:

```
private static YourObject instance;

public static synchronized YourObject getInstance() {
    if(instance == null) {
        instance = new YourObject();
    }
    return instance;
}
```

This solution requires that *every* thread be synchronized when in reality only the first few need to be.

Double check synchronization:

```
private static volatile YourObject instance;

public static YourObject getInstance() {
    YourObject r = instance;
    if(r == null) {
        synchronized(lock) {    // while we were waiting for the lock, another
            r = instance;       // thread may have instantiated the object
            if(r == null) {
                r = new YourObject();
                instance = r;
            }
        }
    }
    return r;
}
```

This solution ensures that only the first few threads that try to acquire your singleton have to go through the process of acquiring the lock.

Initialization on Demand:

```
private static class InstanceHolder {
    private static final YourObject instance = new YourObject();
}

public static YourObject getInstance() {
    return InstanceHolder.instance;
}
```

This solution takes advantage of the Java memory model's guarantees about class initialization to ensure thread safety. Each class can only be loaded once, and it will only be loaded when it is needed. That means that the first time `getInstance` is called, `InstanceHolder` will be loaded and `instance` will be created, and since this is controlled by `ClassLoader`s, no additional synchronization is necessary.

Singleton and Thread Safety

Thread-safe code is particularly important in Singleton. Two instances of Singleton class will be created if the getInstance() called simultaneously by two threads. To avoid such issues, we'll make the getInstance() method synchronized. This way we force every thread to wait until its turn before it executes. I.e. no two threads can be entered into getInstance() method at the same time.

```
public static synchronized Singleton getInstance() {

		/* Lazy initialization, creating object on first use */
		if (instance == null) {
			instance = new Singleton();
		}
		return instance;
}
```

The above implementation will answer to thread safety problem, however synchronized methods are expensive and will have serious performance hit. We will change the getInstance() method not to check for synchronization, if instance is already created.

```
public static synchronized Singleton getInstance() {

		/* Lazy initialization, creating object on first use */
		if (instance == null) {
			synchronized (Singleton.class) {
				if (instance == null) {
					instance = new Singleton();
				}
			}
		}

	return instance;

}
```

Singleton and Early Initialization

Using early initialization we will initialize upfront before your class is being loaded. This way you don't need to check for synchronization as it is initialized before being used ever.

```
package com.javatechig.creational.singleton;

class Singleton implements Cloneable {
	private static Singleton instance;

	/* Private Constructor prevents any other class from instantiating */
	private Singleton() {
	}

	public static synchronized Singleton getInstance() {

		/* Lazy initialization, creating object on first use */
		if (instance == null) {
			synchronized (Singleton.class) {
				if (instance == null) {
					instance = new Singleton();
				}
			}
		}

		return instance;
	}

	/* Prevent cloning */
	@Override
	public Object clone() throws CloneNotSupportedException {
		throw new CloneNotSupportedException();
	}

	public void display() {
		System.out.println("Hurray! I am display from Singleton!");
	}
}
```

Singleton and Object Cloning

Java has the ability to create a copy of object with similar attributes and state form original object. This concept in java is called cloning. To implement cloning, we have to implement java.lang.Cloneableinterface and override clone() method from Object class. It is a good idea to prevent cloning in a singleton class. To prevent cloning on singleton object, let us explicitly throw CloneNotSupportedExceptionexception in clone() method.

```
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable {

	private static Singleton instance;
```

```
        private int value;

        /* Private Constructor prevents any other class from instantiating */
        private Singleton() {
        }

        public static synchronized Singleton getInstance() {

                /* Lazy initialization, creating object on first use */
                if (instance == null) {
                        synchronized (Singleton.class) {
                                if (instance == null) {
                                        instance = new Singleton();
                                }
                        }
                }

                return instance;
        }

        /* Restrict cloning of object */
        @Override
        public Object clone() throws CloneNotSupportedException {
                throw new CloneNotSupportedException();
        }

        public void display() {
                System.out.println("Hurray! I am display from Singleton!");
        }

        public int getValue() {
                return value;
        }

        public void setValue(int value) {
                this.value = value;
        }
}
```

Singleton and Serialization

Java Serialization allows to convert the state of an object into stream of bytes so that it can easily stored or transferred. Once object is serialized, you can deserialize it, back to object from byte stream. If a singleton class is meant to be serialized, it will end up creating duplicate objects. Let us have a look into the example below, explaining the problem,

Singleton.java

```java
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable {

	private static Singleton instance;

	private int value;

	/* Private Constructor prevents any other class from instantiating */
	private Singleton() {
	}

	public static synchronized Singleton getInstance() {

		/* Lazy initialization, creating object on first use */
		if (instance == null) {
			synchronized (Singleton.class) {
				if (instance == null) {
					instance = new Singleton();
				}
			}
		}

		return instance;
	}

	/* Restrict cloning of object */
	@Override
	public Object clone() throws CloneNotSupportedException {
		throw new CloneNotSupportedException();
	}

	public void display() {
		System.out.println("Hurray! I am display from Singleton!");
	}

	public int getValue() {
		return value;
	}

	public void setValue(int value) {
		this.value = value;
	}
}
```

SerializationTest.java

```
package com.javatechig.creational.singleton;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

public class SerializationTest {

        public static void main(String[] args) {

                //getting singleton instance
                Singleton instanceOne = Singleton.getInstance();
                instanceOne.setValue(10);

                try {
        // Serialize to a file
                        ObjectOutput out = new ObjectOutputStream(new
FileOutputStream("filename.txt"));
        out.writeObject(instanceOne);
        out.close();

        instanceOne.setValue(20);

        // Serialize to a file
        ObjectInput in = new ObjectInputStream(new
FileInputStream("filename.txt"));
        Singleton instanceTwo = (Singleton) in.readObject();
        in.close();

        System.out.println("Instance One Value= " + instanceOne.getValue());
        System.out.println("Instance Two Value= " + instanceTwo.getValue());

      } catch (IOException e) {
        e.printStackTrace();
      } catch (ClassNotFoundException e) {
        e.printStackTrace();
      }
  }
}
```

In the above example, the Singleton class is implementing Serializable interface, which means the state of its object can be persisted. SerializableTest is the test class, used to test Singleton class. Inside main() method we are persisting the state of Singleton instance into a file and retrieve it later. Now compile and run the program, you will notice that the state of both instances (instanceOne and instanceTwo) are different, which means that they are two different objects. Here we are violating the rules of singleton by allowing it to create two different objects of same class.

To solve this issue, we need to include readResolve() method in our DemoSingleton class. This method will be invoked before the object is deserialized. Inside this method, we will call getInstance() method to ensure single instance of Singleton class is exist application wide.

Singleton.java

```
package com.javatechig.creational.singleton;

import java.io.Serializable;

class Singleton implements Cloneable, Serializable{

	private static final long serialVersionUID = 1L;
	private static Singleton instance;
	private int value;

	/* Private Constructor prevents any other class from instantiating */
	private Singleton() {
	}

	public static synchronized Singleton getInstance() {

		/* Lazy initialization, creating object on first use */
		if (instance == null) {
			synchronized (Singleton.class) {
				if (instance == null) {
					instance = new Singleton();
				}
			}
		}

		return instance;
	}

	protected Object readResolve() {
      return getInstance();
   }

	/* Restrict cloning of object */
	@Override
	public Object clone() throws CloneNotSupportedException {
		throw new CloneNotSupportedException();
	}

	public void display() {
```

```
                System.out.println("Hurray! I am display from Singleton!");
        }

        public int getValue() {
                return value;
        }

        public void setValue(int value) {
                this.value = value;
        }
}
```

Behind the scene

1. Serializable is a marker interface. A marker interface contains no fields or method declaration. Serializable interface is used as a marker to specify that the class can be serialized.
2. writeObject() and readObject() methods are called while the object is serialized or deserialized. writeObject() method is used to write the state of the object for its particular class so that its corresponding readObject() method can read it.
3. readObject() method is responsible for reading the object from stream and to restore the class fields.

Rules of Thumb

1. Singleton classes are used sparingly. Do not think of this pattern, unless you know what you are doing. As the object is created in global scope, this is riskier in resource constrained platforms
2. Beware of object cloning. Double check and block object's clone method
3. Careful when multiple threads accessing the singleton class
4. Careful of multiple class loaders they can break your singleton
5. Implement strict type if your singleton class is serialized

Exception handing in JSP:

Process.jsp

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

Error.jsp:

```
<%@ page isErrorPage="true" %>
```

```

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

web.xml file if you want to handle any exception

```
<web-app>

 <error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```

1) web.xml file if you want to handle the exception for a specific error code

```
<web-app>

 <error-page>
  <error-code>500</error-code>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```