============================================================================**When and how do you optimize your code**

1. Design level: The architectural design of a system affects the overall performance of the system. Specify efficient algorithms to carry out the task. Specify efficient design patterns like below.
   1. Algorithm Strategy Pattern.
   2. Execution Pattern
   3. Implementation Strategy Pattern like program organization and data structures to use.
   4. Structural Design Patterns which explains the structure of application being developed.
      1. Creational Pattern
      2. Structural Pattern
      3. Behavioral Pattern
   5. Choose Hibernate
2. Source Code level:
   1. Use VisualVM to test the performance of system, for analysing the threads, memory and cpu.
   2. Use Java Coding java.util.concurrent, caching,
   3. From database tune sql, apply index. Refere Database Performance Tuning document.
   4. Tools like PMD, CheckStyle can be used to improve the efficiency of code.
3. Assembly level: Use Native Coding (JNI) in case of hardware level interaction is needed, which will boost the speed of program.
4. After Del

============================================================================**How to Create User Defined Immutable Objects in java?**

1. Make fields private and final.
2. Force callers to construct an object completely in a single step, instead of using a no-argument constructor combined with subsequent calls to setXXX methods. No more setter for attribute, because final attribute needs to be initialized while creating the object, and no more it can be changed.
3. Ensure the class cannot be overridden - make the class final.
4. If the fields in class are mutable (Ex:Date, Employee Object). Create a copy of object and send new object every time on getter.

============================================================================**What are the 4types of memory in java**

**1) Stack**

**2) Heap – Garbage Collector takes control of this area. –XMS(initial Heap space) –XMX(max Heap space)**

**3) PermGen(Permanent Generation) Space – memory area for primitives, String pool, classes, stacks(methods) and various metadata. –XX:PermSize(initial PermGen) –XX:MaxPermSize(maximum PermGen)**

**4) Native Space**

============================================================================**How many types of OutOfMemoryError can occur in Java**

1) **Java.lang.OutOfMemoryError: Java heap space ->** Because of object leak in user program, object garbage collection problem. To avoid increase –XMX  
2) **Java.lang.OutOfMemoryError: PermGen space ->** Because of memory leak in class loaders, String pool size increased, stack recursive calls. To avoid increase –XX:MaxPermSize

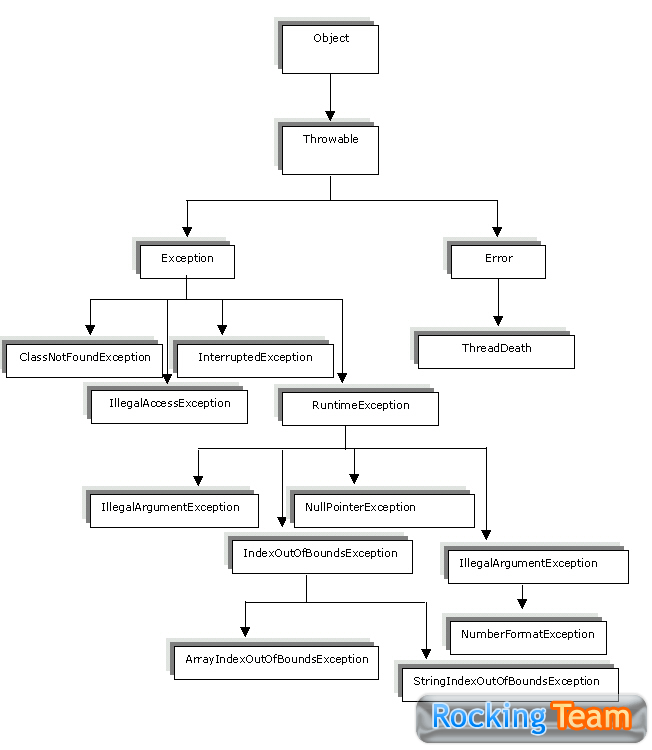
============================================================================

**How to Convert Checked Exception into UnChecked Exception?**

Catch the checked exception throw custom unchecked exception.

============================================================================

**How to Create Custom Exception with Checked Exception and UnChecked Exception**



If the Class extends Exception then it is Checked Exception.

If the Class extends RunTimeException then it is UnChecked Exception.

============================================================================

**try has a return and finally has a return which will work.**

only finally return will work. if finally dont have a return then try return will work. Note: if finally dont have a return, then there should be a return after try finally block even if try has or dont have return.

============================================================================

**How to put hetorogeneous objects inside collection?**

1) Create collection without generics

2) In use wild card “?”.

EX:

Collection<?> col = new ArrayList<String>();

col.add(new Object()); // Compile time error

or

Collection<? extends Object> c = new ArrayList<String>();

c.add(“test”); // Compile time error

Above makes col as readable no add or delete operation is allowed.

============================================================================

package certification;

public class Parent {

protected int x = 9; // protected access

}

package other;

import certification.Parent;

class Child extends Parent {

public void testIt() {

System.out.println("x /is " + x); // No problem; Child

// inherits x

Parent p = new Parent(); // Can we access x using the p reference? No

System.out.println("X in parent is " + p.x); // Compiler error!

}

}

============================================================================**Static method cannot be overridden in java**

class SuperClass{

static void method(){

System.out.println("superClass method");

}

}

class SubClass extends SuperClass{

**// If static is removed in child class method then compilation error. This instance method cannot //override the static method from SuperClass**

static void method(){

System.out.println("SubClass method");

}

}

============================================================================

**Key Rule in constructor : The first line in a constructor must be a call to super() or a call to this().**

No exceptions. If you have neither of those calls in your constructor, the compiler will insert the no-arg call to super(). The compiler won't put in a default constructor if you've already got one or more constructors in your class. But, it still inserts a call to super() in any constructor that doesn't explicitly have a call to the super constructor—unless, that is, the constructor already has a call to this().

============================================================================

**Polymorphism:**Ability of an object to take more than one form.

**Coupling**

Let's start by making an attempt at a definition of coupling. Coupling is the degree

to which one class knows about another class. If the only knowledge that class A

has about class B, is what class B has exposed through its interface, then class A and

class B are said to be **loosely coupled**…that's a good thing. If, on the other hand,

class A relies on parts of class B that are not part of class B's interface, then the

coupling between the classes is **tighter…*not* a good thing**. In other words, if A knows

more than it should about the way in which B was implemented, then A and B are

tightly coupled.

Using this second scenario, imagine what happens when class B is enhanced. It's

quite possible that the developer enhancing class B has no knowledge of class A,

why would he? Class B's developer ought to feel that any enhancements that don't

break the class's interface should be safe, so he might change some non-interface

part of the class, which then causes class A to break.

===============================================================

**Cohesion should be more**

Instead of one class that does everything, we have broken the system into four main classes, each with a very specific, or *cohesive*, role.

===============================================================

**Overloading with Boxing and Var-args**

class AddBoxing {

static void go(Integer x) { System.out.println("Integer"); }

static void go(long x) { System.out.println("long"); }

public static void main(String [] args) {

int i = 5;

go(i); // which go() will be invoked?

}

}

the output will be : long

**Rules**

1 Widening beats boxing

2 Widening beats var-args

===============================================================

class BoxOrVararg {

static void go(Byte x, Byte y) { System.out.println("Byte, Byte"); }

static void go(byte... x) { System.out.println("byte... "); }

public static void main(String [] args) {

byte b = 5;

go(b,b); // which go() will be invoked?

}

}

As it turns out, the output is

Byte, Byte

===============================================================

**Instance init block and Static init block**

class SmallInit {

static int x;

int y;

static { x = 7 ; } **// static init block**

{ y = 8; } **// instance init block**

}

1) Init blocks execute in the order they appear.

2) Static init blocks run once, when the class is first loaded.

3) Instance init blocks run every time a class instance is created.

4) Instance init blocks run after the constructor's call to super().

===============================================================

**Is it possible to iterate Map using object for loop**

Yes.. Because Map extends Iterable Interface where List and Set does

===============================================================

**Major difference between arrays and arraylist is :**

Arrays can't shrink and can't grow at runtime, whereas arraylist can.

**Points:**

1. goto and const are reserved keywords in java, but should not be in code. If used compiler throws error.
2. default value for integer 0, object reference null.
3. Use == to compare two primitives, or to see if two references refer to the same object.
4. Use the equals() method to see if two objects are equal.
5. Arraylist without generics can hold heterogeneous type of data.
6. java.lang is imported by default in java EX: System, String, Integer etc.......
7. StringBuilder from 1.5 similar to StringBuffer, but not synchronized, so fast.
8. A reference variable marked final can't ever be reassigned to refer to a different object. So there are no final objects, only final references.
9. Arrays are considered as object. Because they can’t be initialized without new EX: int[] a = new int[100];
10. final reference variables (instance) must be initialized before the constructor completes.
11. Prior to java1.4 java doesnt have file for constant definition, which we will be doing through public static final int MALE = 0.Enum is Like constant definer.
    1. At runtime ENUM is converted into java class like below final class Foo extends java.lang.Enum<Foo> {}
12. 20 80 priniciple: 20% will go for creation and testing of code whereas 80% will go for maintenance and enhancement of code.
13. Classes can accept, only public, abstract, final and strictfp. if nothing given then, it is default scope
14. interface can accept, only public. if nothing given then, it is default scope, interface variable-- public,static, final by default, interface method-- public abstract by default… Inside a class, for a method or attribute if no scope is given, then it is default, whereas it is public for interface.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the [equals(Object)](file:///C:\Users\GoldPrabhu\Documents\Study\Java%20Materials\APIs\jdk-1_5_0-doc\docs\api\java\lang\String.html) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

EX1:

String s = "test";

String s1 = "test";

String s2 = new String("test").intern();

String s3 = new String ("test").intern();

Total objects in the above example is 1

**Adding or modifying or removing data to list while retrieving through**

In case of iterator it will throw runtime exception .java.util.ConCurrentModificationException. Because iterators are fail-fast based ….

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

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

System.*out*.println(it.next());

list.add("4");

}

The same happens to Object For loop.

List<String> l = Arrays.*asList*("a", "b", "c");

**for**(String s : l) {

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

l.remove(0);// throws ConcurrentModificationException

}

**Difference between and Iterator--> Both are used for Collection object iteration. Now java 1.5 has object for loop.**

Enumeration is older one. Iterator is newer one.

Both are used for iterating an object.

Both has methods hasMore() and next() methods. Additionally iterator has remove() method.

ArrayList is latest and Vector is old

We cannot iterate ArrayList through enumeration whereas

we can iterate Vector through an iterator as well as enumeration.

**Argument values and return values are implicitely promoted to higher value if necessary.**

public void test2(int a, long b) {

}

public static void main(String… args) {

int x = 10;

test2(x,x);// passes fine

}

ArrayList<String> test = null;

for(String t : test) { //NullPointerException

System.out.println(t);

}

ArrayList<String> test = new ArrayList<String>();

for(String t : test) { // Prints Nothing

System.out.println(t);

}

int x=0;

int z = ++x; //(the value of the overall expression is the value after the increment)

o/p: x=1, z=1

int x=0;

int z = x++; //(the value of the overall expression is the value after the increment)

o/p: x=1, z=0

long y == 40002;

40002 exceeds the 16-bit limit of a short

short x = (short) y; x now equals -25534!

float f = 3.14f;

int x = (int) f; x will equal 3

**How to sort the collection in reverse for default string and primitives**

ArrayList arrayList = new ArrayList();

arrayList.add("D");

arrayList.add("A");

arrayList.add("E");

Comparator comparator = Collections.reverseOrder();

System.out.println("Before sorting ArrayList in descending order : "+ arrayList );

Collections.sort(arrayList,comparator);

System.out.println("After sorting ArrayList in descending order : "+ arrayList);

**Explain Diff Between JVM(don’t need Physical machine just Virtual Machine i.e software is enough to run compiled .javac code) and JRE?**

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

**JRE** = **JVM** + Java Packages Classes (like util, math, lang, awt, swing etc) +runtime libraries.

**How String concatenation works?**

From Java 1.5 String concatenation is carried out through StringBuilder internally.

String h = "hello" + "world";

====

String h ="helloworld";

====

String i = new StringBuilder().append("hello").append("world").toString();

All 3 above statements are same.

Thus no temporary objects will be created for this type of concatenation.

**Still 1 point to note: Always** use StringBuilder inside loop to be on safer side.

StringBuilder result = new StringBuilder(10000);

for(int i=0; i<=1000;i++) {

result.append("Hello"+i);

}

return result.toString();

**When LinkedHashSet is better compared to HashSet**

1. To maintain order
2. If the operation is mostly iteration of element by element, then LinkedHashSet provides O(n) for all element getting, whereas HashSet it needs to get element from bucket, do hash logic, iterate list during collision which is costlier.

**Why Choose Generics?**

1. Generic provides Compile Time Safety. Type is Erased at RunTime i.e Type Eraser.
2. I can design my class with Run Time Element like Element E,Type T, Key K, Value V

**When not to choose hibernate? Or Problem with Hibernate**

1. Don't use ORM (or Java) for processing data in bulk, if you can possibly avoid it. A stored procedure will be much faster. If your application queries large volume of data and then processes it, Hibernate could slow down processing and could lead to memory issues. Because internally it binds each db record to an object and loads the entire resultset into a collection in memory.

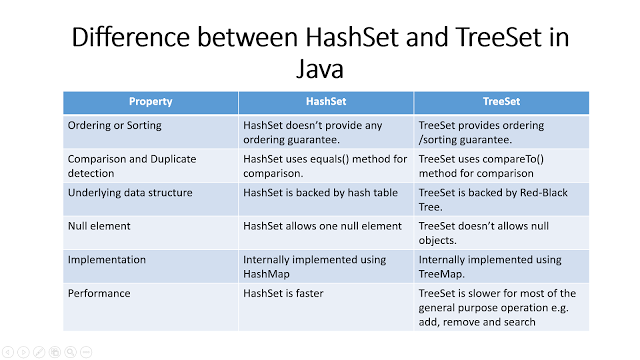
Ex: processing approx 200K records i.e. querying from database and creating

a proprietary file. Using Spring JDBC gave a much better performance.

1. Hibernate by default is lazy. So if employee has address. It will fetch only employee and address has to be fetched again which creates N+1 problem.
2. The main example of where ORM - and indeed Java - is not suitable is the case of processing data in bulk. It is simply never going to be efficient to fetch millions of rows from the database, into your JVM, and then update them one at a time. Don't use Java for this. Use a stored procedure.
3. Hibernate saves productivity, in an application which is having 100s of table. Building JDBC query is very tedious.
4. You should use Hibernate in an object-oriented domain model. Not every application needs a domain model, so not every application needs ORM. If your application does a lot of business logic - rather than just displaying tabular data on a webpage - then a domain model is usually a good thing.

**Do I need to override object's equals() and hashcode() method to place a custom class in TreeMap ?**

1. hashcode() method is not required.
2. Treemap is an Red-Black tree based NavigableMap implementation. This implementation provides guaranteed log(n) time cost for the containsKey, get, put and remove operations.
3. Only hashing data structures require hashcode() method for achieving Big O(1) for data retrieval. HashMap, hashtable, ConcurrentHashMap, LinkedHashMap are few of the hashing data structures which require both hashcode() and equals() method.
4. TreeMap, on the other hand uses Comparator/Comparable equals() method for maintaining unique key and sorting of elements.
5. TreeMap Searching is done through tree navigation. So log(n).
6. It always a good practice to keep equals() method in sync with the Comparator and hashCode to have consistency in your code for **both sorting and hashing actions**



**What is the advantage of static inner class or nested class?**

The client code can directly call Inner Class with out instantiation.

Ex: Map.Entry, LinkedList.Entry

**Lambda vs Anonymous Inner Class**

interface Employee {

abstract void address();

}

public static void main(String[] args) {

Employee anonymousPerson = new Employee() {

public void address() {

System.out.println("Anonymous Employee Address");

}

};

anonymousPerson.address();

Employee lambdaPerson = () -> System.out.println("Lambda Employee Address");

lambdaPerson.address();

}

**Why method local inner class can access only final variable?**

Local variables always live on the stack, the moment method is over all local variables are gone. Inner class objects might be on heap even after the method is over, so in that case it would not be able to access the local variable, since they are gone. There is also a possibility that the variable could change before the inner class accesses it. Making the local variable final prevents these scenarios.

class Outer{

private String x = "instance variable";

void doStuff(){

final String z = "local variable";

class Inner{

public void seeOuter(){

System.out.println("Outer x is : "+ x);

//below line will not compile if z is not final

System.out.println("Local variable z is : " + z);

}

}

}

}

**Can the keys in Hashing data structure be made Mutable?**

No. Because if the key is changed, then retrieval by get is not possible. But still map can be iterated and we can still get value.

**How to print country specific currency informations, Ex: INR, RS 10.50?**

1) Set the locale... or get default from system.

2) Get the Currency from Locale

3) Get the precision from Currency and set into NumberFormat.

**How ConcurrentHashMap solves the problem with Collections.synchronizedMap?**

For the issue1 with iterator, CHM copies the current map and iterates

For issue2 CHM uses putIfAbsent. This is equivalent to below code, except that the action is performed atomically.

**if (!map.containsKey(key))**

**return map.put(key, value);**

**else**

**return map.get(key);**

**How to use Map syncMap =**

**Collections.synchronizedMap(existingMap)**

The map needs to synchronized with the map reference itself on 2 conditions Issue1)During iteration of map, because Iterations over collections in Java is not thread safe, even if you are using one of the synchronized wrappers.

Map m = Collections.synchronizedMap(new HashMap());

Set s = m.keySet(); // Needn't be in synchronized block

synchronized (m) { // Synchronizing on m, not s!

Iterator i = s.iterator(); // Must be in synchronized block

while (i.hasNext())

foo(i.next());

}

Issue2) When doing 2 or more operations over the collection sequentially. Map methods like put, get, size, containsKey, add…. all are synchronized but if thread T1 tries to put element E1 by verifying, if element E1 is present. This scenario is an sequential operation. After T1 verifying for element E1 if T2 puts element E1, T1 again tries to put element E1, which will cause malfunction. To avoid this below code is necessary.

synchronized (synchronizedMap) {

// test for a key in the map

if (synchronizedMap.containsKey(key)) {

synchronizedMap.get(key).add(value);

} else {

List<String> valuesList = new ArrayList<String>();

valuesList.add(value);

// store a value into the map

synchronizedMap.put(key, valuesList);

}

}

Failure to follow this advice may result in non-deterministic behaviour.

**What is the difference between Process and Thread?**

1) A Java.exe program in execution is often referred as process. A thread is a subset(part) of the process.  
2) A process consists of multiple threads. A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.  
3) A process is sometime referred as task. A thread is often referred as lightweight process.  
4) A process has its own address space. A thread uses the process’s address space and share it with the other threads of that process.  
5) Every program which is started with java.exe has its own JVM. I can configure heap space, logging, agents etc…. You can have any number of JVMs as each java.exe starts a JVM. The JVMs are independent processes. Garbage collection operates on each JVM independently.  
6) A thread can communicate with other thread (of the same process) directly by using methods like wait(), notify(), notifyAll(). A process can communicate with other process by using Inter Process communications.

Communication between 2 JVM can happen through RMI, Socket Programming, Web-Service.. etc

7) A process can have child process (Ex: How server handles request, security, session… each can be thought of as child-process, and each child process will have threads) or child threads. A process does not have control over the sibling process(i.e another JVM), but it has control over its child processes.

8) The process is heavy weight. Thread is Light Weight.

Ex: In Servlet for each and every request 1 instance is created and multiple threads are created. Here the main instance can be called as Child-Process.

**What is unmodifiable list?**

Integer arr[] = { 10, 20, 30 };

List list = Arrays.*asList*(arr); // This is an unmodifiable list

list.remove(1);// Will throw error

The Arrays.asList(..) is collection that can't be expanded or shrunk (because it is backed by the original array, and it can't be resized).

**How to avoid this?**

Integer arr[] = { 10, 20, 30 };

List list = **new** ArrayList(Arrays.*asList*(arr));//create list and initialize

list.remove(1);

**How to Create ArrarList object inside stack memory?**

public void foo(){

ArrayList<String> myList = new ArrayList<>();

}

All Java Objects are created in Heap memory section, so the ArrayList will be created on the heap. But the local reference (myList) will be created in the Stack section of memory. Once the method call is finished and if myList variable is not escaped from this method then GC will collect the ArrayList object from heap.

As of JDK 1.6\_14, escape analysis can be enabled by setting the appropriate JVM flag (java -XX:+DoEscapeAnalysis) which hints the compiler to convert heap allocations to stack allocations if the method local objects do not escape the method scope.

-----DoEscapeAnalysis is analysis default in java8

**Difference between** [**NoClassDefFoundError**](http://download.oracle.com/javase/6/docs/api/java/lang/NoClassDefFoundError.html) **and** [**ClassNotFoundException**](http://download.oracle.com/javase/6/docs/api/java/lang/ClassNotFoundException.html)

[NoClassDefFoundError](http://download.oracle.com/javase/6/docs/api/java/lang/NoClassDefFoundError.html) happens when the JVM cannot load a class that is required, almost always because you forgot to include it in the classpath.

[ClassNotFoundException](http://download.oracle.com/javase/6/docs/api/java/lang/ClassNotFoundException.html) happens when you try to load a class via reflection (for example, when doing Class.forName("...")) and the class cannot be found. Usually this also means that the JAR or directory that contains the class file is missing from the classpath. This is an exception and can be handled i.e I can load alternate class or load from remote location instead of loading from jar which is cached.

**Explain 4ways of creating object**

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

<http://stackoverflow.com/questions/1036754/difference-between-wait-and-sleep>

Abstract class can have 0 abstract methods. But if a single method is abstract the class must be abstract.

**Disadvantage of Reflection Class Loading**

1. Exception Handling is difficult. You lose all the compile-check feature. Because the method that we load at runtime may throw exception and handling make reflection code unreadable.
2. Reflection is slow.

**What are updatable views?**

Views are not only read-only but also updateable. However in order to create an updateable view, the SELECT statement which defines View has to follow several following rules:

* SELECT statement must not reference to more than one table. It means it must not contain more than one table in FROM clause, other tables in JOIN statement, or UNION with other tables.
* SELECT statement must not use GROUP BY or HAVING clause.
* SELECT statement must not use DISTINCT in the selection list.
* SELECT statement must not reference to the view that is not updateable
* SELECT statement must not contain any expression (aggregates, functions, computed columns…)

**Why Strings are immutable in java?**

To make Java more memory efficient, the JVM sets aside a special area of memory called the "String constant pool". When the compiler encounters a String literal, it checks the pool to see if an identical String already exists. If a match is found, the reference to the new literal is directed to the existing String, and no new String literal object is created. (The existing String simply has an additional reference.).

If several reference variables refer to the same String without even knowing it, it would be very bad if any of them could change the String's value.

Also, but what if someone overrides the String class functionality; couldn't that cause problems in the pool? That's one of the main reasons that the String class is marked final.

80% of java objects are Strings. So by making string as immutable all the string value that are present in multiple place will refer the same value which saves huge object space, for memory efficiency, for Security purpose it is made final Class

1. String Pool concept is possible only because of immutability.
2. Mutilple threads access String without data corruption, So Thread Safety is possible only because of immutability.
3. No Synchronization is needed for concurrent threads to work.
4. No one can change the logic of String class.
5. String pool cannot be reached by garbage collector. Wrappers (primitive to Object and Object to primitive) are also immutable.

Strings are immutable (meaning read-only)

Example:

String s = "test";

String s1 = "test";

String s2 = new String("test");

String s3 = new String ("test");

Total objects in the above example is 3

**How to display Japanese character in JSP:**

Note: Japanese character are double byte.

<%@ page contentType="text/html; charset=SHIFT\_JIS "pageEncoding="SHIFT\_JIS"%>

**What is MBean and Message Driven Bean**

MDB != MBean

MBean ---> Management Bean

These beans are used to manage an application. For more information (mlet, notification, relation, ...) consult the specification of JMX.

**Is it possible to extend singleton class?**

No because Singleton has private Constructor. Below class will throw compile error.

**public** **class** SingletonTest **extends** MySingleton {}

**String vs StringBuilder**

StringBuilder is mutable. So if lots of String concat operation is needed StringBuilder is good, because it overrides its value.

**Can I use StringBuilder as key in HashMap**

No, because StringBuilder and StringBuffer doesn't override equals and hashcode

**If there are multiple nested streams opened, do we need to close all streams?**

If multiple streams are chained together, then closing the one which was the last constructed, will automatically closes all the underlying streams.

Additionally close() also calls flush(). Use try with resources which is auto close.

**How to handle System.exit() from web-application or any standard java?**

* Change the SecurityManager .policy file to ignore the exit/halt
* Provide shutdownhook for exit. But it will fail for halt.

[**http://blog.joda.org/2014/02/exiting-jvm.html**](http://blog.joda.org/2014/02/exiting-jvm.html)

**Difference between System.exit() and Runtime.getRunTime.halt()?**

System.exit is sequenced exit. So it will execute runFinalizersOnExit and shutdownHook

System.exit internally calls Runtime.getRunTime.exit.

Runtime.getRunTime.halt forcefully closes JVM. So runFinalizersOnExit and shutdownHook will not run.

**How to redirect System.out to FileWrite instead of Console?**

// create file

FileOutputStream f = new FileOutputStream("file.txt");

System.setOut(new PrintStream(f));

// this text will get redirected to file

System.out.println("This is System class!!!");

**How do we implement deep cloning?**

Deep cloning can be done by two ways:-

**All objects implement a clone method** Every object is responsible for cloning itself via its clone () method. So when the parent is called it makes calls to all the referenced objects inside the class and calls its clone method.

**Serialization** This is the best way to deep cloning and not to mention best answer in interview of how to implement deep cloning. There are three steps to do deep cloning using serialization:- √ Ensure that all classes in the object are serializable √ Create output stream for writing the new object and input stream for reading the same. √ Pass the object you want to copy to the output stream. √ And finally read the object using the input stream. Below is the code snippet which implements all the above steps.

**What is Mutex?**

Mutex stands for mutually exclusive, only one kind of operation (READ or WRITE) is allowed at a given timeframe.

**What is Synchronizer?**

A synchronizer is any object that coordinates the control of flow of threads based on its state. For example, semaphore, CountDownLatch, FutureTask, Exchanger, CyclicBarrier, etc.

**What are Upper and Lower bounds in Generics ? Where to choose one**



**Upper Bounded Wildcards**

Upper bounded wildcard restricts the unknown type to be a specific type or subtype of that type.

For example, If we want to write a method that accepts List<Number> and its subtypes i.e. List<Double> and List<Integer>, etc then we can use Upper bounded wildcard.

public static void process(**List<? extends Number>** list) { /\* ... \*/ }

**Lower Bounded Wildcards**

Lower bounded wildcard restricts the unknown type to be a specific type or super type of that type.

For Example, If you want to write a method that puts Integer objects into a list. To maximize flexibility, you may like the method to work on List<Integer>, List<Number>, but not List<Double> - anything that can hold Integer values.

public static void addNumbers(**List<? super Integer>** list) {/\*.....\*/}

**What will happen when an exception occurs from within a synchronized code block ?**

When an exception occurs from within a synchronized code block, then JVM smartly releases all the locks acquired by the current thread and will start unwinding the execution stack, till the exception is handled using catch block, otherwise killing the thread.

But the same does not happen when we write explicit locking code using Lock interface. In that case we need to release the lock manually in the finally block.

**What happens when a exception occurs in run() method of thread..?** Because run cannot throw any exception. So checked exception mandatorily needs to be captured in run() method

**What is difference between intrinsic synchronization and explicit lock­ing using Lock ?**

## **Intrinsic Synchronization:** JVM provides through monitor locks. Each object, class in Java owns a monitor on which the threads can be synchronized.

## **Extrinsic Synchronization:** JDK 1.5 introduced concept of using Lock and Condi­tion classes which offers advanced features over intrinsic synchronization

public interface Lock {

void lock();

void lockInterruptibly() throws InterruptedException;

boolean tryLock();

boolean tryLock(long time, TimeUnit unit) throws InterruptedException;

void unlock();

Condition newCondition();

}

## 

## **How does Arraylist internally works. How it manages size?** ArrayList internally uses arrays to store data. The default size is 10.

## **From the javaDoc**

\* Resizable-array implementation of the <tt>List</tt> interface. Implements

\* all optional list operations, and permits all elements, including

## **\* <tt>null</tt>.**

## **When size increase above 10 below is the logic Java1.5**

**int** newCapacity = (oldCapacity \* 3)/2 + 1; ====16

elementData = (E[])**new** Object[newCapacity];

System.*arraycopy*(oldData, 0, elementData, 0, size);

**Java1.6**elementData = Arrays.copyOf(elementData, newCapacity);

If it needs to shift the elements in order to add something over the existing index, then it displaces the elements using following System method -

## **System.arraycopy(elementData, index, elementData, index + 1, size - index);**

## **Setters are called mutators and getters are calles accessors**

## **Why immutable objects are threadsafe?** Immutable objects are Thread safe, because there are no mutators present and values cannot be changed after constructor, So multiple threads will go for accessor only, which will not create concurrency problems.

**How would you detect a DeadLock in a running program ?**

1. Using Jconsole - JDK installation ships with jconsole tool which can connect to a running java process using JMX protocol. Jconsole can tell us whether there is a dead lock in the program or not. JConsole is a graphical monitoring tool to monitor Java Virtual Machine (JVM) and Java applications both on a local or remote machine
2. Using JMX Management package as shown below

**If hashcode() method of an object always returns 0 then what will be the impact ?**

All value will lands in same bucket. So search will become O(n) rather than O(1)

If hashcode() method returns currentTimeInMilliseconds then we will never get back that value…. Need to verify this

How to add element from collection without concurrentModificationException  
**1) Iterate using ListIterator and add element to iterator**public void addIntoCollection(LinkedList<Integer> marks) {

for (ListIterator<Integer> iterator = marks.listIterator(); iterator.hasNext(); ) {

Integer mark = iterator.next();

if (mark < 40)

iterator.add(mark);

}

System.out.println("marks = " + marks);

## **}**

## **Interface ListIterator<E>(add can be used only with LinkedList)**

An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration.

**How to Use wait?** wait() should always be used inside while loop, because of spurious(false) wakeup. So that it will wait again till it gets notified.

**How to delete data from Collection without concurrentModificationException**

1. Use Enumeration, But is supported only for limited collections
2. During iteration, remove item from iterator rather than removing item from collection

public void removeFromCollection(List<Integer> marks) {

for (Integer mark : marks) {

if (mark < 40)

marks.remove(mark); **==> Will throw java.util.ConcurrentModificationException**

}

}

public void removeFromCollection(List<Integer> marks) {

for (Iterator<Integer> iterator = marks.iterator(); iterator.hasNext(); ) {

Integer mark = iterator.next();

if (mark < 40)

iterator.remove(); **==> Safe to call remove() on Iterator**

}

}

**In your application what you put generally in finally**

1) session.close()--> for hibernate session per request

**Is it possible to have 2 methods with different return type in an class.:** No,, Function Name and Arguments are important..return Types are not considered

**Is it possible to have 2 methods with different return type in an class and subClass:** Yes But in subclass, the return type can be sub type, but cannot be supertype.

**Is it possible for the subclass to be more restrictive?** No. If parent method is protected, then child can be protected or public cannot be private.

**In case of overriding, parent method dont throws any Exception, is it possible for the sub class method to throw any Exception?** Yes and only of type RunTimeException i.e unchecked exception.

**If parent method is throwing IOException, will the child method can throw Exception?** No(Super Type is not allowed). whereas(Sub Type or even without throwing anything) reverse is possible.

**If child method implements Serializable, will the parent method is also Serializable without implementing Serializable?** No. The parents are considered Transient. whereas reverse is true (i.e) if parent implements Serializable all its subclasses are auto serializable.

**If a class named main implements serializable and it has property class Person which is not serializable**? On Serializing main what will happen. ? RunTimeException java.io.NotSerializableException will be thrown.

**Java is "pass by value" or "pass by reference" ?** Java is always "pass by value" even in case of passing Object. There is no concept of Address in Java.

Assume int x = 10;--> if you are asked what is the value of x it is 10 and it is passed to method.

Person p = new Person();--> Here p has 2 value.

1) p value

2) p reference value..>which points new Person().

call fun(Person p1) { }

Now java passes p value and not p reference value. Hence call by value.

Note : But Still new Person() can be changed through p1. Because now p1 points p and p points to new Person(). It is not that since new Person() is changed through p1 it is pass by reference.... as java passed p value and p reference value. Check the swap example.

<http://www.javaworld.com/javaworld/javaqa/2000-05/03-qa-0526-pass.html>

**What is a java marker interface? Is there any alternate way that can serve the same purpose as marker interface?**

From java 1.5, the need for marker interface is eliminated by the introduction of the java annotation feature.

**When 2 or more resource (Database, JMS, Web Service) transaction needs to be committed in a single method. what is the strategy to be used?**

Two-phase commit protocol

1) Commit request phase

2) Commit phase

 [javax.transaction](https://docs.oracle.com/javaee/7/api/javax/transaction/package-summary.html)—Normal Transation

 [javax.transaction.xa](https://docs.oracle.com/javaee/7/api/javax/transaction/xa/package-summary.html)—XA Transaction

 [javax.transaction.xa.XAResource](https://docs.oracle.com/javaee/7/api/javax/transaction/xa/XAResource.html)—XA Resource

**What is the difference between the “FindBugs” and the “PMD”. Means when to use FindBugs and when to use PMD Tool?**

PMD analyzes the source code while Findbugs analyzes the compiled class files. Both look for quite different types of mistakes, so it's good to use both of them in your project.

**How to run some code b4 JVM quits.Ex on below conditions**

Because all of its threads have completed execution

Because of call to System.exit()

Because user hit CNTRL-C

System level shutdown or User Log-Off

**Solution:**

Shutdown Hook comes to rescue in all such scenarios. Application attach a shutdown hook to thereself, that JVM runs when application goes down.

Runtime.getRuntime().addShutdownHook

link : http://hellotojavaworld.blogspot.com/2010/11/runtimeaddshutdownhook.html

**What are the default interface to implement or methods to override**

inerface : serializable, comparable, equals(), hashcode(), Clonabale, clone(), compareTo(), toString()

**Why negative numbers are stored in 2’s compliment**

What is 2’s complement : inverting bits and adding 1.

Signed numbers are stored in 2’s compliment form. Because if make arithmetic operations correct.. EX: To add 5 and -5

5 will be represented as 0000 0101

-5 will be represented as 1111 1010

+1

========

1111 1011

========

When we add -5 and +5 = 0;

+5 = 0000 0101

-5 = 1111 1011

===========

0000 0000

===========

Java doesn’t have unsigned integer or any of unsigned.

**How to add duplicate keys in Map**

Map<String, List<String>> bigMap = new TreeMap<>();

Note: Map will not duplicate keys. Only option is make value as List and if put is not null, then update the list with new value and put again.

**TreeMap in reverse Order?**

Map<BigDecimal, List<String>> bigMap = **new** TreeMap<>(Collections.*reverseOrder*());

**How to sort a collection in reverse order?**

Collections.sort(list, collections.reverseOrder());

**Note: Collections.sort is only applicable for list**

**Format a Double Value with 2 fractions:**

1. String.format( "%.2f", myDouble );
2. DecimalFormat df = new DecimalFormat("#.00");
   1. df.format(myDouble);

**How to sort the Map by value**

**package** com.sample.basics;

**import** java.util.Comparator;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.TreeMap;

**class** ValueComparator **implements** Comparator<String> {

Map<String, Double> base;

**public** ValueComparator(Map<String, Double> base) {

**this**.base = base;

}

// Note: this comparator imposes orderings that are inconsistent with

// equals.

**public** **int** compare(String a, String b) {

**if** (base.get(a) >= base.get(b)) {

**return** -1;

} **else** {

**return** 1;

}

}

}

**public** **class** Dummy<K, V> **extends** HashMap<K, V> {

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

HashMap<String, Double> map = **new** HashMap<String, Double>();

ValueComparator bvc = **new** ValueComparator(map);

TreeMap<String, Double> sorted\_map = **new** TreeMap<String, Double>(bvc);

map.put("A", 99.5);

map.put("B", 67.4);

map.put("C", 111.4);

map.put("D", 90.3);

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

sorted\_map.putAll(map);

System.***out***.println("results: " + sorted\_map);

}

}

Output:

unsorted map: {A=99.5, B=67.4, C=111.4, D=90.3}

results: {C=111.4, A=99.5, D=90.3, B=67.4}

What will happen if I give load factor value greater than 1?

How will you limit the size of hashmap? Extend hashMap override put

**public** **class** MyHashMap<K, V> **extends** HashMap<K, V> {

**private** Map<K, V> map;

**private** **static** **final** **int** ***MAX\_ELEMENT*** = 16;

**public** MyHashMap() {

map = **new** HashMap<K, V>(***MAX\_ELEMENT***,1);

}

**public** V put(K key, V value) {

**if** (map.size() >= ***MAX\_ELEMENT*** && !map.containsKey(key)) {

**return** **null**;

} **else** {

**return** map.put(key, value);

}

}

}

interface A {

}

public class B {

public static void main(String a[]) {

Class c = Class.forName("A");

System.out.println(c.isInterface());//true

Class c1 = A.class;

System.out.println(c1.isInterface());//true

}

}

A class can be load by 2 ways at Runtime using Class.forName at compile time using ClassName.class

So Reflection at compile time and runtime both are possible.

**What is strictfp?**

The strictfp keyword is used to force the precision of floating point calculations (float or double) in Java conform to IEEE’s 754 standard, explicitly. Without using strictfp keyword, the floating point precision depends on target platform’s hardware, i.e. CPU’s floating point processing capability. In other words, using strictfp ensures result of floating point computations is always same on all platforms.

The strictfp keyword can be applied for classes, interfaces and methods.

**Rules**

1. strictfp cannot be applied for constructors.
2. If an interface or class is declared with strictfp, then all methods and nested types within that interface or class are implicitly strictfp.
3. strictfp cannot be applied for interface methods.

**instanceof**

"At run time, the result of the instanceof operator is true if the value of the RelationalExpression is **not nul**l and the reference could be cast to the ReferenceType (parent or current class) without raising a ClassCastException. Otherwise the result is false." So below code will not throw NullPointerException

String s = **null**;

**if** (s **instanceof** String) {

System.***out***.println("null is string");

} **else** {

System.***out***.println("null is not String");

}

Output: “null is not String”

Each Servlet should have a empty constructor. Otherwise during 1st request container throws error.

ServletConfig is not available inside constructor. It is available only after constructor is called.

Servlet Init config and Application Init config has only getter and no setter. Only container can set the values read from the web.xml.

User request <http://localhost:8080/myApp/JMS/login.do(Note>: myApp is web app root context so it will be taken all time)

Response.sendRedirect ("foo/bar.html") <http://localhost:8080/myApp/JMS/foo/bar.html>

Response.sendRedirect ("/foo/bar.html")http://localhost:8080/myApp/foo/bar.html

Before or After sendRedirect nothing should be written to PrintWriter stream, otherwise illegelStateException thrown. Note: But if you are not flushing the stream and doing the sendRedirect still work. But that is lame.

<load-on-startup>

If the value is a negative integer, or the element is not present, the container is free to load the servlet whenever it chooses. If the value is a positive integer or 0, the container must load and initialize the servlet as the application is deployed.

polymorphism

1) Static->overloading->1) through different number of argument

2) through different types of argument

2) Runtime-> overriding through Inheritance EX: Parent class has a method fun and Child class(which extends Parent) is also having a method fun. Now putting Parent parent = new Child(); The compiler will assume it is calling method at Parent Class(it will check whether method is present at parent and checks if parent class throws any exception). But at Runtime only child class fun method is called.

**Dynamic Dispatch:** Runtime polymorphism can also be called as dynamic dispatch. Because when a compiler compiles this java file into .class, it will have multiple if else statements, if parent object then call parent method, if child object then call child method, like that dispatch is decided based on how we initializes the object at runtime.

===========================================================================

**subType:short can be sent instead of int. or Horse can be sent instead of Animal**

narrower or broader in inheritance overriding

1) function return type --> same or subtype (narrow..This facility is from 1.5)

2) function access level --> same or broader

3) function throws --> same or subtype (narrow) or none... also unchecked exception if base class is none

4) funtion arguments -->same....if subtype or any different then it is overloading

===========================================================================

**is it possible to override private methods of parent class:** No. Because the projection of method is within class only. Still we can specify exact function name(with same return type, access level and argument), but it is not considered override. Some new method in child class.if i try to give this new method @Override annotation, then compiler will throw error, the method must exist in parent.

===========================================================================

**is it possible to override static methods of parent class: No. Overriding concept is only for object.** Because static methods are per class it cant be overridden. Still we can specify exact function name(with same return type, access level and argument), but it is not considered override. Some new method in child class.if we try to give this new method @Override annotation, then compiler will throw error, the method must exist in parent(technically this is wrong information which eclipse is giving).

*static methods can't be overridden*! This doesn't mean they

can't be redefined in a subclass, but redefining and overriding aren't the same thing.

===========================================================================

A method marked private cannot be overridden in child class, since the projection of private method is m a parent class. NOTE: But if I define a same method in child class it is accepted. Try to include @Overridden annotation inside child class, then it will throw compiler error stating there must be a method of same in parent class.

===========================================================================

interface--> interface declares behaviour, a class which implements this class should have the behaviour of interface, but behaviour is independent of sub class..

===========================================================================

abstract--> it can say a behaviour or leave the behaviour to its subclass to implement.

===========================================================================

**Base Class method doesn’t have throws any Exception, is it possible for the child class overridden method to throw any exception?** Yes, the subclass can throw any unchecked Exception.

===========================================================================

**Base Class method throws some Exception, is it possible for the child class overridden method to neglect exception?** Yes, the subclass throws can be smaller than base type exception and can also be nothing

===========================================================================