**Java**

**Object-oriented programming**

**Inheritance**

Inheritance used in OOP means that a derived class can use common functions defined in a super class. It is used for code sharing.

Modularity

**Polymorphism**

Polymorphism used in OOP means that a client program can use a function defined in an interface without knowing the real behaviors at development time. The real behaviors are defined at run-time.

**Encapsulation**

Encapsulation or information hiding means to hide how function is implemented.

**Design Patterns**

<http://www.tutorialspoint.com/design_pattern/>

1. Factory Method
2. Abstract Factory
3. Adapter – converts one class to another class

InputStream is = System.in;

Reader aReader = new InputStreamReader(is);

1. Proxy
2. Decorate – add new functionalities to an existing object without changing its structure.

BufferedReader bfReader = new BufferedReader(new InputStreamReader(System.in));

1. Singleton
2. Composite
3. Iterator – provides a consistent way to sequentially access items in different types of collections.

1. Builder vs factory

**Enterprise Integration Design Pattern**

1. Messaging, JMS (Java Message Service)
2. Shared Database
3. FTP
4. Remote Procedure Call – socket connection, Restful API, RMI

**Java related Technologies**

**JMS and Messaging**

JMS - Java Message Service

Point-to-point: Using Queue. A message is guaranteed consumed by one and only once by a consumer. Messages are kept in the queue until consumed or expired.

Publish/Subscribe: Using Topic. A message can be consumed by 0, if no subscribers are active, or many consumers if there are multiple consumers are active. A durable subscription consumer will receive messages which sent before it subscribes to the topic.

**Message Selector**

Using message selector, a message consumer can filter messages. Message selectors assign the work of filtering message to the JMS provider rather than to the application.  Message selectors only filter message based on message headers and properties.  A message selector cannot select message on the basis of the content of the message body.

**Queue Browser**

In JMS queue browser represents a static snapshot of the queue.  The order of messages returned reflects the order of messages a regular message receiver would see.

<Sample Code>

@Resource(lookup = "jms/ConnectionFactory")

private static ConnectionFactory connectionFactory;

@Resource(lookup = "jms/Queue")

private static Queue queue;

@Resource(lookup = "jms/Topic")

private static Topic topic;

Connection connection = connectionFactory.createConnection();

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

//Producers

MessageProducer producer = session.createProducer(dest);

MessageProducer producer = session.createProducer(queue);

MessageProducer producer = session.createProducer(topic);

producer.send(message);

MessageProducer anon\_prod = session.createProducer(null);

anon\_prod.send(dest, message);

//Consumers

MessageConsumer consumer = session.createConsumer(dest);

MessageConsumer consumer = session.createConsumer(queue);

MessageConsumer consumer = session.createConsumer(topic);

//Consumer receive messages – blocked/synchronously

connection.start();

Message m = consumer.receive();

connection.start();

Message m = consumer.receive(1000); // time out after a second

//Consumer listens for messages – non-blocked/asynchronously

Listener myListener = new Listener();

consumer.setMessageListener(myListener);

**Sun Java Real-Time System (Java RTS)**

1. Sun's commercial implementation of the Real-Time Specification for Java
2. Real-time -> the ability to reliably and predictably respond to a real-world event.
3. Features and Benefits
   1. New Real-Time Threads, Scheduling and Synchronization
   2. New Memory Management Schemes
   3. Asynchronous Events Handling & Asynchronous Transfer of Control
   4. Time & Timers
   5. Direct Access to Physical Memory
4. Programming in Real-Time Specification for Java (RTSJ): A Conversation with Distinguished Engineer Greg Bollella ~ <http://java.sun.com/developer/technicalArticles/Interviews/Bollella_qa2.html>

**Apache Storm** ~ <http://storm.apache.org/>

1. Start Storm

>cd /apps/storm

>./nyttistrom.sh nimbus &

>./nyttistorm.sh supervisor

2. Question:

1. declareOutputFields on ISpout is called at submitTopology time. You cannot change the fields after ISpout has been submitted
2. If a topology uses a config file which is not loaded form resource directories, how we are going to package it for the submission?
3. How to config a topology to log into a separate log file other than worker's log file?
4. How to version a topology and rollback to an older version?
5. A more powerful UI, shows
6. tuple of each spout and bolt
7. connection between spout and bold and data flow
8. Can you change number of workers at run-time?
9. Submit topology from development box?

**ZeroMQ/jzmq**

1. Jzmq - Java language binding for libzmq (ZeroMQ)

**ZooKeeper**

ZooKeeper is a distributed coordination service for distributed applications. It provides a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

1. Quick Started ~ <http://zookeeper.apache.org/doc/r3.3.3/zookeeperStarted.html>, https://zookeeper.apache.org/doc/current/zookeeperOver.html
2. Need uuid
3. Start Zookeeper in Single Server Mode
4. >bin/zkServer.sh start
5. >bin/zkCli.sh -server [127.0.0.1:2181](http://127.0.0.1:2181/)

**uuid-1.6.0**

<http://www.ossp.org/pkg/lib/uuid/>

Get uuid-1.6.0.tar.gz from [ftp.ossp.org/pck/lib/uuid](http://ftp.ossp.org/pck/lib/uuid)

./shtool install -c -m 755 -s .libs/uuid /usr/local/bin/uuid

./shtool install -c -m 644 ./uuid.1 /usr/local/share/man/man1/

Install uuid-devel

<http://rpm.pbone.net/index.php3/stat/4/idpl/6914383/dir/opensuse/com/uuid-devel-1.6.0-3.55.i586.rpm.html>

**Database JDBC, Spring SimpleJdbcTemplate, Hibernate**

* Raw JDBC steps to query

Class.forName (\" driver class for that specific database\");

Connection con = DriverManager.getConnection (url, username, password);

Statement stmt = Connection.Statement ("select \* from TABLE NAME");

stmt.exceuteUpdate();

//close connection

* How does JDBC know which driver it should use if you register more than one driver classes?

Each JDBC driver has an acceptsURL method to test if the driver can open a connection based on URL parameter. DriverManager goes though all registered drivers to find one which accepts the URL. Every driver also tries to register itself when ClassLoad first load driver class.

**Hibernate ORM (Hibernate in short)**

ORM – Object-relational mapping.

Session vs SessionFactory

A SessionFactory is an expensive-to-create, thread-safe object, intended to be shared by all application threads. It is created once, usually on application startup, from a Configuration instance. A Session is an inexpensive, non-thread-safe object that should be used once and then discarded for: a single request, a conversation or a single unit of work. A Session will not obtain a JDBC Connection, or a DataSource, unless it is needed. It will not consume any resources until used.

[Source: <http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/transactions.html#]>

**Spring framework**

Inversion of Control (IoC) and Dependence Control (DI) – It is a process whereby objects define their dependencies (that is, the other objects they work with) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes or a mechanism such as the Service Locator pattern.

1. Benefits to use Spring

* Industry standard
* Easy to test
* Object decoupled

1. Spring security and integration with Active Directory

Web application + active directory authentication + remember me

<http://static.springsource.org/spring-security/site/docs/3.0.x/reference/appendix-namespace.html#nsa-form-login>

<http://thinkinginsoftware.blogspot.com/2011/01/remember-me-with-ldap-spring-security.html>

<http://www.tikalk.com/java/spring-security-3-remember-me-ldap-authentication>

<remember-me>

- Remember-Me has two concrete implementations

    PersistentTokenBasedRememberMeServices

    TokenBasedRememberMeServices

- Both Remember-Me implementations need a UserDetailsService

Create a RememberMeService based attributes

  - TokenBasedRememberMeService

  - PersistentTokenBasedRememberMeService

  - Custom implementation of RememberMeService

Attributes

  - data-source-ref

    Use PersistentTokenBasedRememberMeService + JdbcTokenREpositoryImpl

  - token-repository-ref

    Use PersistentTokenBasedRememberMeService + cutome PersistentTokenRepository

  - services-ref

    Use a custom implementation of RememberMeServies

<ldap-user-service> to create an LdapUserDetailsService

1. Boost – MFC a simple sample Restful service

<http://localhost:8080/submit/id/ID123432?logout=true>

<http://localhost:8080/submit/id/ID123432?name=sam&password=543432>

@RestController

public class GreetingController {

@RequestMapping (value = "/submit/id/{id}", method = RequestMethod.GET, produces="text/xml")

public String showLoginWindow(@PathVariable("id") String id,

@RequestParam(value = "logout", required = false) String logout,

@RequestParam("name", required = false) String username,

@RequestParam("password", required = false) String password,

@ModelAttribute("submitModel") SubmitModel model,

BindingResult errors) throws LoginException {...}

}

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**Java Application challenges**

* Performance Issues
* Transaction Issues
* Threading Issues
* Memory Issues

**Java EE/Web Application technologies, libraries and frameworks**

Application Server: IBM WebSphere eCommerce Platform, WebLogic

Servlet Engine: Apache Tomcat, Jetty

JSP/Servlet

JAXP - Java API for XML Processing, there basic parsing interfaces Document Object Model(DOM), Simple API for XML(SAX) and Streaming API for XML(StAX)

JAXB - Java Architecture for XML Binding

JAX-WS - Java API for Web Service. It is included in Java SE 6

XSLT - XML Stylesheet Language for Transformations allows conversion of an XML document into other forms of dada.

JMS - Java Message Service

JNDI - Java Naming and Directory Interface

JMX - Java Management Extension (JMX) API is a standard API for management and monitoring of resources such as applications, devices, services and the java virtual machine. It is included J2SE5 and J2SE 6

EJB

**JMX Consoles**

jminix - It is a simple embeddable restful JMX console jmanage - come with a servlet container (web server). It can manage multiple JVM

JGroups - A toolkit for reliable multicast communication

SPNEGO - Integrated Windows Authentication in Java <http://spnego.sourceforge.net/>

**Java 1.5 New Features** <http://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html>

* Language Features
* Generics
* Enhanced for loop
* Autoboxing/Unboxing
* Typesafe Enums
* Varargs
* Static Imports
* Metadata (Annotations)
* Base Libraries
* Concurrency Utilities
* JAXP- Java API for XML Processing

**Java 1.6 New Features**

**Java 1.7 New Features -** <http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html>

* Small Language Enhancements (Project Coin)
* Diamond Operation

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

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

* Use String in swatch statements

switch(string){

case ”NEW”:

case “OLD”:

..

}

* Automatic resource management

Numeric literal with underscores

int thousand = 1\_000;

int million = 1\_000\_000;

* Improved exception handling

try{

}

catch(ExceptionOne | ExceptionTwo | ExceptionThree){

}

**Java 1.8 New Features**

* Lambda Expressions
* Stream on Collections
* Functional Interface
  1. @FunctionalInterface annotation – an informative annotation to indicate an interface type declaration is intended to be a functional interface
  2. An interface is considered as a functional interface if
     + The type is an interface type and not an annotation type, enum or class
     + The annotated type satisfies the requirements of a functional interface, only has extract one abstract method. An interface declares an abstract method overriding one of the public methods of java.lang.Object is not a functional interface
  3. You don’t have to use @FuntionalInterface annotation to make an interface as a functional interface. Any interface satisfies the functional interface requirement is a functional interface
* Nashorn - JavaScript engine
* Date/Time change - new java.time package. Most classes are thread safe and immutable

**Bitwise and Bit Shift Operators**

~ Unary bitwise complement

<< Signed left shift

>> Signed right shift

>>> Unsigned right shift

&     Biwise AND

^     Bitwise exclusive OR

|      Bitwise inclusive OR

- the leading bit in an integer indicate the sign. It is a positive number if the leading bit is 0. It is a negative number is the leading bit is 1.

- 3>>> 1,    is a unsigned right shift.,

**Java Generics**

* Type parameter are removed at run-time
* Bounded Type Parameters

public <U extends Number> void inspect(U u){

       System.out.println("T: " +

           t.getClass().getName());

       System.out.println("U: " +

           u.getClass().getName());

}

For multiple interfaces

<U extends Number & MyInterface>

* Wildcards

**Lower Bounded wildcard**

Cage<? extends Animal> cage = new Cage<Lion>();

Cage<Lion> is a subclass of Cage<? extends Animal>

Cage<Animal> is a subclass of Cage<> extends Animal>

**Upper bounded wildcard**

Cage<? super Animal> cage

**Unbounded wildcard**

Cage<?>

**Java Annotations**

1. Information for the compiler - helps the compiler to detect errors and suppress warnings
2. Compiler-time and deployment-time processing - Software tools can process annotation information to generate code, XML file and so forth.
3. Runtime processing - Some annotations are available to be examined at run time.

**Thread**

1. JConsole to detect deadlocked threads which uses ThreadMBean.findDeadlockedThreads in JMX. You could use method in your code to watch if there is deadlock occurred.
2. kill -3 and Ctrl+C will get thread stack which may show you the deadlocked threads
3. Thread, Executor, ExecutorService, Executors (factory to create Executor or ExecutorService)

**Java Garbage Collection**

* <http://www.oracle.com/technetwork/java/gc-tuning-5-138395.html>
* <http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#available_collectors>
* Available Collectors
  1. Serial Collector - single thread to perform all garbage collections work. Efficient since no communication between threads. Good for small data set applications and single processor machine
  2. Parallel Collector (throughput collector) performs minor collections in parallel. Good for large data set applications on a multiprocessor machine
  3. Concurrent Collector - performs most collection work concurrently while the application is running to keep garbage collection pause short. Good for large data set applications which response time is more important that overall throughput
  4. Generational Collection - all available collectors are generational collection.
* java command line options

-verbose:gc prints every garbage collections

[GC 325407K->83000K(776768K), 0.2300771 secs]

[GC 325816K->83372K(776768K), 0.2454258 secs]

[Full GC 267628K->83769K(776768K), 1.8479984 secs]

-XX:MaxGCPauseMillis=<N> - max pause time in milliseconds

-XX:GCTimeRatio=<N> - total ratio of time spending on garbage collection vs time spending application. N=19 means 1/20 of total run time sends on garbage collection

* Selecting a Collector
  + Let JVM selecting first
  + Small data set, single processor machine or no pause time requirement limit, use Serial Collector

**Java Threading**

* Thread creation
* new Thread()
* using utilities: Executor and ExecutorService
* Executor - An object that executes submitted Runnable tasks. This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of threads use. It only has one method of void execute (Runnable)
* ExecutorService - an interface implements Executor interface.  It provides methods to manage termination and methods that can produce a Future for tracking progress of one or more asynchronous tasks.
* Callable/Future

A task returns a result and may throw an exception.  The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception. Call submit(Callable) on ExecutService to run a callable with returns a Future.

interface Future { boolean cancel (boolean mayInterruptIfRunning);

                         V get();

                         V get(long timeout, TimeUnit unit);

                         boolean isCancelled();

                         boolean isDone();}

You can the exception thrown by Callable from the ExcutionException which thrown by get().

* Methods on Object class : notify(), notifyAll(), wait(), wait(long timeout, int nanos)

Since notify() only wakes up one thread, it is possible that the waken up thread cannot continue its process because it is waiting for another condition. If the condition only can be changed by the thread which is also waiting for this object, all threads cannot be continued. In the other side, notifyAll wakes up all threads, so all threads will have a chance to compete the object and continue its process.

Here is example, suppose there are three threads, t1, t2 and t3. t1 is having the monitor of a object A and t2 and t3 are waiting to get the monitor of A. t2 will change a status of A which t3 needs it to continue its process. Lets say, t1 finished its task and calls notify() and t3 gets the monitor. Since the status of A has not been changed by t2 yet, so t3 can not continue its work and calls wait() to wait. Now t2 and t3 both are in waiting status and con not continue since one to wake up them again.

In the same situation, lets say t1 calls notifyAll which wakes up both t2 and t3. Assume t3 gets the monitor, because the status of A has not changed yet, so t3 goes to waiting status and release the monitor. Then t2 can get the monitor and changes the status of A and it calls notify() or notifyAll() to wake up t3 and t3 can continue its work.

* Methods on Thread class : join(), sleep(long mills), yield(), interrupt()
* Key words: volatile and synchronized
* ThreadGroup - do we really need a ThreadGroup? Seems a not very useful class any more after stop, resume and suspend methods were deprecated.
* Synchronization aid classes - CountDownLatch, CyclicBarrier
* Callable vs Runnable - Callable.call() returns a value and might throw an Exception. Runnable.run does not allow either.
* BlockingQueue (ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue)
* CocurrentHashMap, Hashtable, HashMap, Collections.synchronized(map)

CocurrentHashMap - retrieving doesn’t lock. Updating locks

* CopyOnWriteArrayList
* Future - get(long timeout, TimeUnit unit), isDone(), isCancelled()
* Execute - execute(Runnable command)
* ScheduledExecutorService - submit(Callable)
* ThreadPoolExecutor and ScheduledThreadPoolExecutor
* **Java.util.concurrent.**atomic package ~ support lock-free thread-safe programming on single variable. Provides an atomic conditional update operation of the formboolean compareAndSet (expectedValue, updateValue)

Some classes: AtomicBoolean, AtomicInteger, AtomicIntergerArray

The memory effects for accesses and updates of atomics generally follow the rules for volatiles:

* get has the memory effects of reading a volatile variable.
* set has the memory effects of writing (assigning) a volatile variable.
* ***weakCompareAndSet*** atomically reads and conditionally writes a variable, is ordered with respect to other memory operations on that variable, but otherwise acts as an ordinary non-volatile memory operation.
* compareAndSet and all other read-and-update operations such as getAndIncrement have the memory effects of both reading and writingvolatile variables.
* **Simple strategy for creating immutable objects**

1. Don’t provide “setter” methods
2. Make all fields final and private
3. Don’t allow subclasses to override methods.
4. If the instance fields include reference to mutable objects, don’t allow those objects to be changed:

∙        Don’t provide methods that modify the mutable objects

∙        Don’t share reference to the mutable objects. Never store references to external, mutable objects passed to the constructor, if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

**Multithreading in general**

* Multiple threads run concurrently in a single process. They share the process’s resources but are able to execute independently
* Multithreaded program is run faster on computer systems that have multiple CPUs, CPUs with multiple cores, or across a cluster of machines.
* Race conditions and atomic operations – **explain it in more detail**
* Even on a single-CPU system, a multithreaded program may be user friendlier than a non-multithreaded program since a multithreaded program runs more responsible to user input.
* Multithreading scheduler on Operating System – **explain it in more detail**
  + Preemptive multithreading
  + Cooperative multithreading
* **Liveness, deadlock, starvation and livelock**
* **Liveness** ~ a concurrent application’s ability to execute in a timely manner is known as its liveness.
* **Deadlock** ~ describes a situation where two or more threads are blocked forever, waiting for each other.
* **Starvation** ~ describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.
* **Livelock** ~ a thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then livelock may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor: Alphonse moves to his left to let Gaston pass, while Gaston moves to his right to let Alphonse pass. Seeing that they are still blocking each other, Alphone moves to his right, while Gaston moves to his left. They're still blocking each other, so...

Multithreading Reading List

<http://en.wikipedia.org/wiki/Thread_(computer_science>)

http://www.serpentine.com/blog/threads-faq/

**Core Java**

**Collection Framework**

All general-purpose implementations have no restrictions on the elements they may contain.

All general-purpose implementations are unsynchronized.  Use Collections.synchronizedCollection() method to get synchronized collection.

* Queue~ Queues typically order elements in FIFO. It also can order elements according to supplied comparator. Whatever the ordering is used, the remove() and pull() move/return the head element.

ArrayBlockingQueue, LinkedBlockingQueue

ConcurrentLinkedQueue

PriorityQueue/PriorityBlockingQueue - ordered according to their natural ordering or by a Comparator

ConcurrentLinkedQueue vs. LinkedBlockingQueue

* Deque ~ “double ended queue” pronounced as “deck”.

addFirst/offerFirst, addLast/offerLast

removeFirst/pollFirst, removeLast/pollLast

getFirst/peekFirst, getLast/peekLast

Queue interface (FIFO)

add <-> addLast

offer <-> offerLast

remove/poll <-> removeFirst/pollFirst

element <-> getFirst

peek <-> peekFirst

Works as Stack

push <-> addFirst

pop <-> removeFirst

peek <-> peekFirst

* Stack ~ LIFO. Operations: empty(), peek(), pop(), push(E). Java Stack class is derived from Vector. It is preferred to use Deque instead of Stack
* List ~ an ordered collection. You can access element by the index. Typically List allows duplicated elements and multiple nulls.
* Set ~ no duplicate elements. It can only have one null element.
* HashMap and Hashtable. HashMap is unsynchronized and permits null value and key. Hashtable is synchronized and doesn’t allow either null value or null key.
* LinkedHashMap - maintains the insertion order
* TreeMap: sort the entries in ascending order of keys
* Vector and ArrayList. Vector is synchronized whereas ArrayList is not. Both Vector and ArrayList permit null values. The size of Vector/ArrayList can grow or shrink its size. An array is fixed size.
* LinkedList and ArrayList

**Enum Types**

1. All enum types implicitly extend java.lang.Enum. enum type cannot extend anything else since Java does not support multiple inheritance.
2. enum type only can have package-private and private constructors. It automatically creates constants defined at the beginning of the enum body.
3. public enum Planet {
4. MERCURY (3.303e+23, 2.4397e6),
5. VENUS   (4.869e+24, 6.0518e6),
6. EARTH   (5.976e+24, 6.37814e6),
7. MARS    (6.421e+23, 3.3972e6),
8. JUPITER (1.9e+27,   7.1492e7),
9. SATURN  (5.688e+26, 6.0268e7),
10. URANUS  (8.686e+25, 2.5559e7),
11. NEPTUNE (1.024e+26, 2.4746e7);
12. private final double mass;   // in kilograms
13. private final double radius; // in meters
14. Planet(double mass, double radius) {
15. this.mass = mass;
16. this.radius = radius;
17. }
18. private double mass()   { return mass; }
19. private double radius() { return radius; }
20. // universal gravitational constant  (m3 kg-1 s-2)
21. public static final double G = 6.67300E-11;
22. double surfaceGravity() {
23. return G \* mass / (radius \* radius);
24. }
25. double surfaceWeight(double otherMass) {
26. return otherMass \* surfaceGravity();
27. }
28. public static void main(String[] args) {
29. if (args.length != 1) {
30. System.err.println("Usage:  java Planet <earth\_weight>");
31. System.exit(-1);
32. }
33. double earthWeight = Double.parseDouble(args[0]);
34. double mass = earthWeight/EARTH.surfaceGravity();
35. for (Planet p : Planet.values())
36. System.out.printf("Your weight on %s is %f%n",
37. p, p.surfaceWeight(mass));
38. }
39. }

**Nested classes**

1. A nested class can be declared as private, public, protected or package private. A top-level can only be declared as public or package private.
2. A static nested class is behaviorally a top-level class that has been nested in another top-level class for packaging convenience
3. The instance of a static nested class can be created as OuterClass.StaticNestedClass instance = new OuterClass.StaticNestedClass();
4. A static nested class only can directly access static members and methods of the outer class.
5. A non-static nested class is also called as inner class. It can directly access instance members of outer class.
6. Local inner class is an inner class declared inside of a method
7. Anonymous inner class is an inner class declared inside of a method without naming it.

**final used with method parameters**

public void method(final String s, final int i){

s = “abc”;

System.out.prinltn(s);

Runnable r = new Runnable(){

public void run(){

}

}

}

**StringBuffer and StringBuilder**

~StringBuffer: thread safe

~StringBuilder: not thread safe

**What are those methods defined on Collections class?**

Java.util.Collections contains a set of **static** methods:

* Creates empty immutable Set, List and Map
* Creates single element immutable Set, List and Map
* Creates synchronized collection type based on given Collection, List, Map, Set, SortedMap and SortedSet.
* Creates unmodifiable collection type based on given Collection, List, Map, Set, SortedMap and SortedSet

**Transient**

It declares that an instance field is not part of the default serialized form of an object. When an object is serialized, only the values of its non-transient instance fields are included in the default serial representation. When an object is de-serialized, transient fields are initialized only to their default value.

**Anonymous classes**

|  |
| --- |
| public class Y extends R {  public static void main(String[] args) {    Y y = new Y()  {    private R create() {  System.out.println(“RR”);  return null;  };  };  y.create();  }    private R create() {  System.out.println(“Y”);  return null;  }  } |

* An anonymous class must always extends a super class or implements an interface but is cannot have an explicit extends or implements clause
* An anonymous class must implement all the abstract methods in the super class or the interface
* An anonymous class always uses the default constructor from the super class to create an instance
* The class file of an anonymous class will be named as ClassName$number.class, for above class, we will have two class files, Y.class and Y$1.class

**What is an abstract class?**

An abstract class must be extended (subclassed). It serves as a template. It can not be instantiated. An abstract class may contain static data. Any class with an abstract method is automatically abstract itself, and must be declared as such.

* If you cannot instantiate an abstract class, do you need to vide a constructor?  Why do people still provide a constructor? It is used to provide a way to initialize “abstract class” part.
* A class may be declared abstract even if it has no abstract methods to prevent it from being instantiated.

**What is the difference between an Interface and an Abstract class?**

An abstract class can have instance methods that implement a default behavior. An abstract class is a class which may have the usual flavors of class members (private, protected, etc.), but has some abstract methods.

An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation.

Interface

* Use interface for “multiple inheritance”
* All methods are public. All member data are implicit to final and static.

Abstract

* Use abstract class as a base class so it has some default behaviors that a derived class can reused.

**What are pass by reference and pass by value?**

Pass by Reference means the passing the address itself rather than passing the value. Pass by Value means passing a copy of the value to be passed.

* Which data type is passed by reference and which are passed by value?

**Difference between Swing and Awt?**

AWT are heavy-weight components. Swings are light-weight components. Hence swing works faster than AWT.

**State the significance of public, private, protected, default modifiers both singly and in combination and state the effect of package relationships on declared items qualified by these modifiers.**

public : Public class is visible in other packages, field is visible everywhere (class must be public too)

private : Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that owns the feature.

protected : Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.

default :What you get by default ie, without any access modifier (ie, public private or protected).It means that it is visible to all within a particular package.

Q: Does importing a package imports the subpackages as well? e.g. Does importing com.MyTest.\* also import com.MyTest.UnitTests.\*?

A: No you will have to import the subpackages explicitly. Importing com.MyTest.\* will import classes in the package MyTest only. It will not import any class in any of it's subpackage.

 [ Received from Sandesh Sadhale]  TOP

**How can I customize the seralization process? i.e. how can one have a control over the serialization process?**

Yes, it is possible to have control over serialization process. The class should implement Externalizable interface. This interface contains two methods namely readExternal and writeExternal. You should implement these methods and write the logic for customizing the serialization process.

**What is Externalizable interface?**

Externalizable is an interface which contains two methods readExternal and writeExternal. These methods give you a control over the serialization mechanism. Thus if your class implements this interface, you can customize the serialization process by implementing these methods.

**When you serialize an object, what happens to the object references included in the object?**

The serialization mechanism generates an object graph for serialization. Thus it determines whether the included object references are serializable or not. This is a recursive process. Thus when an object is serialized, all the included objects are also serialized along with the original object.

**What one should take care of while serializing the object?**

One should make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

**What happens to the static fields of a class during serialization?**

A: There are three exceptions in which serialization doesnot necessarily read and write to the stream. These are

1. Serialization ignores static fields, because they are not part of ay particular state state.

2. Base class fields are only hendled if the base class itself is serializable.

3. Transient fields.

**What are Checked and UnChecked Exception?**

A checked exception is some subclass of Exception (or Exception itself), excluding class RuntimeException and its subclasses.

Making an exception checked forces client programmers to deal with the possibility that the exception will be thrown. eg, IOException thrown by java.io.FileInputStream's read() method·

Unchecked exceptions are RuntimeException and any of its subclasses. Class Error and its subclasses also are unchecked. With an unchecked exception, however, the compiler doesn't force client programmers either to catch the

exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. eg, StringIndexOutOfBoundsException thrown by String's charAt() method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be.

**What are RuntimeExceptions?**

RuntimeException and its subclasses are unchecked exceptions. A method is not required to declare in its throws clause.

**What is the difference between error and an exception?**

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors and you cannot repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.).

**If I write System.exit (0); at the end of the try block, will the finally block still execute?**

No in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

**How are Observer and Observable used?**

A: Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.

 [Received from Venkateswara Manam]  TOP

Q: How does Java handle integer overflows and underflows?

A: It uses those low order bytes of the result that can fit into the size of the type allowed by the operation.

 [ Received from Venkateswara Manam] TOP

**Does garbage collection guarantee that a program will not run out of memory?**

Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection

**What is the difference between preemptive scheduling and time slicing?**

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Q: When a thread is created and started, what is its initial state?

A: A thread is in the ready state after it has been created and started.

 [ Received from Venkateswara Manam] TOP

Q: What is the purpose of finalization?

A: The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.

 [ Received from Venkateswara Manam] TOP

Q: What is the Locale class?

A: The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region.

 [ Received from Venkateswara Manam] TOP

Q: What is the difference between a while statement and a do statement?

A: A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.

 [ Received from Venkateswara Manam] TOP

Q: What is the difference between static and non-static variables?

A: A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

 [ Received from Venkateswara Manam] TOP

**How are this() and super() used with constructors?**

this() is used to invoke a constructor of the same class. super() is used to invoke a super class constructor.

Q: What are synchronized methods and synchronized statements?

A: Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

 [ Received from Venkateswara Manam] TOP

**What is a daemon thread and which method is used to create the daemon thread?**

A daemon thread only runs when at least there is a non-daemon thread runs. In other words, JVM exits when the only threads running are all daemon threads. Use setDaemon method to make thread as daemon thread.

Q: Can applets communicate with each other?

A: At this point in time applets may communicate with other applets running in the same virtual machine. If the applets are of the same class, they can communicate via shared static variables. If the applets are of different classes, then each will need a reference to the same class with static variables. In any case the basic idea is to pass the information back and forth through a static variable.

An applet can also get references to all other applets on the same page using the getApplets() method of java.applet.AppletContext. Once you get the reference to an applet, you can communicate with it by using its public members.

It is conceivable to have applets in different virtual machines that talk to a server somewhere on the Internet and store any data that needs to be serialized there. Then, when another applet needs this data, it could connect to this same server. Implementing this is non-trivial.

 [ Received from Krishna Kumar ] TOP

**How does a try statement determine which catch clause should be used to handle an exception?**

When an exception is thrown within the body of a try statement, the catch clauses of the try statement are examined in the order in which they appear. The first catch clause that is capable of handling the exceptions executed. The remaining catch clauses are ignored.

**Can an unreachable object become reachable again?**

An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects.

**What method must be implemented by all threads?**

All tasks must implement the run() method, whether they are a subclass of Thread or implement the Runnable interface.

**What are synchronized methods and synchronized statements?**

A: Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. Synchronized statements are similar to synchronized methods. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

 [ Received from P Rajesh] TOP

Q: What modifiers are allowed for methods in an Interface?

A: Only public and abstract modifiers are allowed for methods in interfaces.

**What are some alternatives to inheritance?**

Delegation is an alternative to inheritance. Delegation means that you include an instance of another class as an instance variable, and forward messages to the instance. It is often safer than inheritance because it forces you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense. On the other hand, it makes you write more code, and it is harder to re-use (because it is not a subclass).

What does it mean that a method or field is "static"?

A: Static variables and methods are instantiated only once per class. In other words they are class variables, not instance variables. If you change the value of a static variable in a particular object, the value of that variable changes for all instances of that class.

Static methods can be referenced with the name of the class rather than the name of a particular object of the class (though that works too). That's how library methods like System.out.println() work out is a static field in the java.lang.System class.

Q: What is the catch or declare rule for method declarations?

A: If a checked exception may be thrown within the body of a method, the method must either catch the exception or declare it in its throws clause.

 [ Received from P Rajesh] TOP

Q: Can a .java file contain more than one java classes?

A: Yes, a .java file contain more than one java classes, provided at the most one of them is a public class.

 [ Received from Sandesh Sadhale] TOP

**What will be the initial value of an object reference which is defined as an instance variable?**

The object references are all initialized to null in Java. However in order to do anything useful with these references, you must set them to a valid object, else you will get NullPointerExceptions everywhere you try to use such default initialized references.

**What are the different scopes for Java variables?**

The scope of a Java variable is determined by the context in which the variable is declared. Thus a java variable can have one of the three scopes at any given point in time.

* Instance : - These are typical object level variables, they are initialized to default values at the time of creation of object, and remain accessible as long as the object accessible.
* Local : - These are the variables that are defined within a method. They remain accessible only during the course of method excecution. When the method finishes execution, these variables fall out of scope.
* Static: - These are the class level variables. They are initialized when the class is loaded in JVM for the first time and remain there as long as the class remains loaded. They are not tied to any particular object instance.

**What is the default value of the local variables?**

The local variables are not initialized to any default value, neither primitives nor object references. If you try to use these variables without initializing them explicitly, the java compiler will not compile the code. It will complain about the local variable not being initialized.

**How many objects are created in the following piece of code?**

MyClass c1, c2, c3;

c1 = new MyClass ();

c3 = new MyClass ();

Only 2 objects are created, c1 and c3. The reference c2 is only declared and not initialized.

**Can a public class MyClass be defined in a source file named YourClass.java?**

No the source file name, if it contains a public class, must be the same as the public class name itself with a .java extension.

**Can main method be declared final?**

Yes, the main method can be declared final, in addition to being public static.  Main method is defined as static. Static methods can not be overridden that means they are final methods.

**What will be the output of the following statement?**

**System.out.println ("1" + 3);**

It will print 13.

**What will be the default values of all the elements of an array defined as an instance variable?**

If the array is an array of primitive types, then all the elements of the array will be initialized to the default value corresponding to that primitive type. e.g. All the elements of an array of int will be initialized to 0, while that of boolean type will be initialized to false. Whereas if the array is an array of references (of any type), all the elements will be initialized to null.

**Monitoring and Management Using JMX**

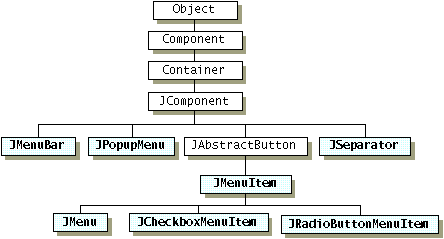
The JVM has build-in instrumentation that enables you to monitor and mange it using JMX.

* To enable to use JMX, you need to set some system properties.

com.sun.management.jmxremote.port=*portNum*

**Menu**

Classes



**Equals and hashCode**

-Always override hashCode when you override equals

HashCode

|  |
| --- |
| Returns a hash code value for the object. This method is supported for the benefit of Hashtable such as those provided by java.util.Hashtable.  The general contract of hashCode is:  Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.  If two objects are equal according to the equals (Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.  It is *not* required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.  As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the JavaTM programming language.) |

1. final class, final method and final variable
2. mutable class
3. system exception
4. Assertion
5. volatile and synchronized
6. lock and wait
7. producer and consumer problem
8. hashCode
9. How to implement a sortable table in Swing?
10. finalize and force garbage collection

3. How to make a child class to "hide" a public method on its parent class.

4. Explain keyword volatile

5. synchornized drawback and dangrous

6. lock and wait

7. Producer/Consumer problem

8. Explain hashCode method and how to overwrite it

9. How to implement a swing sortable table?

10. Finallize and force to garbige collection

SQL questions

1. outer join

2. cluster Index

3. Sybase dialogaster tools

4. Using transaction in a store procedure and using transaction in java

**Development Tools**

**Eclipse**

1.    Find which version of Ant Eclipse is using?

Windows>Preferences>Ant>Runtime, then open Ant Home Entries which will show all jars related Ant.

2.    Use different version of Ant

Windows>Preferences>Ant>Runtime, click on the Ant Home button to select installed Ant you want to use.

3.    Excludes a file from the project

Project->Properties->Resource->Resource Filters->Add

4.    Exclude XML files from Validation

•    Project->Properties->Validation\Enable Project specifics settings\XML Validation->Click Settings Button->Add Rule to Exclude Group

•    After add exclude rule, clean the project

5.

Maven

1.    Maven Integration (m2e) – Maven Eclipse plug-in

    To install it from Eclipse

2.    command line examples

    a) mvn clean install

    b) mvn -X

      -- you will find which javac is used

      -- you will find which repositories is used

    c) -s path/to/user/settings.xml

    d) -gs path/to/global/settings.xml

UEclipse + Maven creating web application

1.    File\New\Other\Maven\Maven Project

2.    Next-> Select maven-archetype-webapp -> Next

3.    Input Group Id and Artifact Id

4.    Finish

5.    Fix Java Build Path Problems

Build path specifies execution environment J2SE-1.5. There are no JREs installed in the workspace that are strictly compatible with this environment.

•    Window\Preferences\Java\Installed JRE change it to jdk1.7\_03

•    Project\Properties\Java Build Path, remove JRE System Library [J2SE-1.5]. Click Add Library, select JRE System Library, then select workspace default JRE(jdk1.7.0\_03)

This also fixes missing src/main/java directory

6.    Exclude JavaScript files from the validation

•    Project\Properties\JavaScript\Include Path->Source->expend TradeMonitor/WebContent

•    Highlight Exclude item then click Add button

7.    If you see an error of Validation Message in Eclipse Markers tab and its description is empty, it probably is because you have include a jar in your local lib directory which is already included in your maven dependences. You can verify this by looking into Maven Dependencies and Web App Libraries under Java Sources/Libraries in Project Explorer

8.    Ideally if you use Maven, you shouldn’t have any jars in your lib directory. In fact, you don’t need a traditional lib directory in your Build Path at all.

**Source Controls**

* MKS Integrity
* SVN
* Git

∙

**What is the purpose of garbage collection in Java, and when is it used?**

The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

∙        Java doesn’t guarantee when an object get collected by garbage collector.

∙        You can use System.gc()? to ask java runtime to start a garbage collection, but that still does guarantee when an object get collected.

∙        Finalize method will be called when an object got collected. Again it is not guarantee the finalize method will be called and when be called.

∙

**Describe synchronization in respect to multithreading.**

A: synchronized provides basic object-level monitor locking. Each object has an associated lock which is used by synchronized statement and synchronized method.

Synchronized does two things: mutual-exclusion lock and visibility. Synchronized prevents two threads get the lock of a object at same time. It does not prevent other threads to calling non-synchronized methods or to access object’s fields.

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared variable while another thread is in the process of using or updating same shared variable. This usually leads to significant errors.

**Explain different way of using thread?**

The thread could be implemented by using runnable interface or by inheriting from the Thread class. The former is more advantageous, 'cause when you are going for multiple inheritance..the only interface can help.

**What is the difference between a constructor and a method?**

A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the new operator.

A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the dot operator.

∙        Another difference is a constructor can not be inherenced

**What is an Iterator?**

Some of the collection classes provide traversal of their contents via a java.util.Iterator interface. This interface allows you to walk through a collection of objects, operating on each object in turn. Remember when using Iterators that they contain a snapshot of the collection at the time the Iterator was obtained; generally it is not advisable to modify the collection itself while traversing an Iterator.

**What is static in java?**

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class. Static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object. A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

**What is final?**

A final class can't be extended ie., final class may not be subclassed. A final method can't be overridden when its class is inherited. You can't change value of a final variable (is a constant).

Q: Are the imports checked for validity at compile time? e.g. will the code containing an import such as java.lang.ABCD compile?

A: Yes the imports are checked for the semantic validity at compile time. The code containing above line of import will not compile. It will throw an error saying,can not resolve symbol

symbol : class ABCD

location: package io

import java.io.ABCD;

  [ Received from Sandesh Sadhale]  TOP

**Can a top level class be private or protected?**

A: No. A top level class can not be private or protected. It can have either "public" or no modifier. If it does not have a modifier it is supposed to have a default access.If a top level class is declared as private the compiler will complain that the "modifier private is not allowed here". This means that a top level class can not be private. Same is the case with protected.

**How do I serialize an object to a file?**

A: The class whose instances are to be serialized should implement an interface Serializable. Then you pass the instance to the ObjectOutputStream which is connected to a fileoutputstream. This will save the object to a file.

**What is Externalizable?**

A: Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, writeExternal(ObjectOuput out) and readExternal(ObjectInput in)

  [ Received from Venkateswara Manam] TOP

HashCode

|  |
| --- |
| Returns a hash code value for the object. This method is supported for the benefit of Hashtable such as those provided by java.util.Hashtable.   * Return same value on same object during the execution * “equal” objects must have same hashcode. * “non-equal” objects can have same hashcode. But it might improve the HashTable performance if we can make a distinct hashcode for non-equal objects |

**Java Interview Questions**

1. Can you use enum for Singleton?
2. List some new language feature introduced in Java 1.5, 1.6, 1.7(?)
3. Why doesn’t make Map derived from Collection interface?
4. Explain race condition
5. How many ways can create a List which only has one element?
6. Tell me how RNDS works?
7. How do you use MQ Series, facilities let you know queue status?
8. What are benefits we can get from producer/consumer pattern?
9. How do update a table where it constantly updates the data?
10. How do you manage release system to different environments?
11. ION ticketing, how do you popular some data from your internal data source on the ticket?
12. How many ways to create a thread? Runnable and Callable
13. Class.fromName
14. Write code to using JDBC to query a table?
15. Spring
16. Query returns huge amount data, how can retrieve them page by page?
17. Write a security program, SQL injection
18. Select the row which has largest number on a column?
19. Select the row which has second largest number on a column?
20. What kinds of new features do you want for the next Java release?
21. HashMap vs. HashTable vs. ConcurrentHashMap
22. initialCapacity – number of buckets
23. loadFactor – measure of how full the hash table is allowed to get before its capacity is automatically increased
24. ConcurrentHashMap: It only locks on bracket level, not on entire table.  Retrieval method does not lock. Update method does not lock entire table. Iteration does throw ConcurrentModificationException. The constructor parameter, concurrencyLevel, defines how many threads can update table same time without contention. The default level is 16. Max is 2^16.
25. It implements three new atomic methods on CocurrentMap interface, putIfAbsent(key, value), remove(key, value), replace(key, value) and replace(key, oldValue, newValue)
26. CopyOnWriteArrayList
27. ArrayList vs. LinkedList vs. Vector
28. Collection class when iterator loop, call remove method?
29. Why doesn’t Map implements Collection interface?
30. Why did/do/lose you leave your last/current/last job?
31. What are you currently earning?
32. What are your salary expectations?
33. On a scale of 1-10, how would you rank your skill level of…?
34. What are your career goals?
35. What are your greatest strengths?
36. What were your 3 greatest accomplishments?
37. ETL layer ~ extract, transform, load
38. An introduction to AOP/AspectJ ~ <http://www.ibm.com/developerworks/java/tutorials/j-aopintro/section2.html>
39. Finalize and force to garbige collection
40. How to implement a swing sortable table
41. Explain hashCode method and how to overwrite it
42. Producer/Consumer problem
43. lock and wait
44. Explain keyword volatile
45. synchronized drawback and dangerous
46. How to make a child class to "hide" a public method on its parent class.
47. final class, final method, final variable

Throughput vs Performance

JBOSS - application server

OOD

* inheritance(is-a relationship, code sharing, override behaviours), encapsulation (Data Hiding)  and Polymorphism (Dynamic Binding, behaviours defined at run-time)
* Composition over Inheritance

JavasSript/AJAX/JQuery/JSON

**Why Java String is designed as immutable?**