# Part 2

## Representative Questions – Please write your answers with an example for each questions.

**How do you design an application with JMS messaging?**

For messaging operations, you should write application programs that use only references to the interfaces defined in Sun's javax.jms package.

JMS defines a generic view of messaging that maps onto the underlying transport. An enterprise application that uses JMS makes use of the following interfaces that are defined in Sun's javax.jms package:

Connection

Provides access to the underlying transport, and is used to create Sessions.

Session

Provides a context for producing and consuming messages, including the methods used to create MessageProducers and MessageConsumers.

MessageProducer

Used to send messages.

MessageConsumer

Used to receive messages.

The generic JMS interfaces are subclassed into the following more specific versions for point-to-point and publish/subscribe behavior.

* How do you handle exception in JMS consumers and how to you recover?
* public OurMessageListener(TopicSession topicSession) {
* this.topicSession = topicSession;
* }
* public void onMessage(Message m) {
* try {
* process(m);
* //if we get here, we've processed the message
* topicSession.commit();
* } catch (Throwable t) {
* log(t);
* }
* }

**How do you implement LRU or MRU cache?**

We use two data structures to implement an LRU Cache.

Queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size).The most recently used pages will be near front end and least recently pages will be near rear end.

A Hash with page number as key and address of the corresponding queue node as value.

**How would you implement Executor Service?**

The easiest way to create ExecutorService is to use one of the factory methods of the Executors class.

For example, the following line of code will create a thread-pool with 10 threads:

ExecutorService executor = Executors.newFixedThreadPool(10);

**Describe singleton design pattern – how would you implement?**

The singleton pattern is a design pattern that restricts the instantiation of a class to one object.

Method 1: Classic Implementation

// Classical Java implementation of singleton

// design pattern

class Singleton

{

private static Singleton obj;

// private constructor to force use of

// getInstance() to create Singleton object

private Singleton() {}

public static Singleton getInstance()

{

if (obj==null)

obj = new Singleton();

return obj;

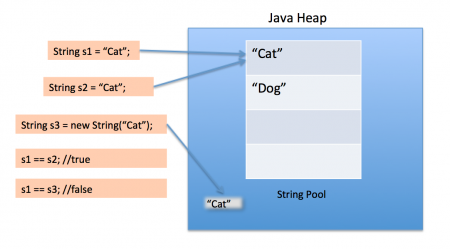
}

}

Here we have declared getInstance() static so that we can call it without instantiating the class. The first time getInstance() is called it creates a new singleton object and after that it just returns the same object. Note that Singleton obj is not created until we need it and call getInstance() method.

**Describe properties of Java String.**

String is Immutable in Java because String objects are cached in String pool. Since cached String literal is shared between multiple client there is always a risk, where one client's action would affect all other client. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable.  
  
Another reason of why String class is immutable could de due to HashMap. Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap.  
Since Strings are immutable in Java if you store password as plain text it will be available in memory until Garbage collector clears it and since String are used in String pool for reusability there is pretty high chance that it will be remain in memory for long duration, which pose a security threat.



When we use double quotes to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference.

However using new operator, we force String class to create a new String object and then we can use intern() method to put it into the pool or refer to other String object from pool having same value.

String subsequence method is nothing but a substring. Substring equals() subsequence but not==. To compare strings implement comparable interface and use compareTo(String anotherString) method. We can use charAt() method to get character at given index.

StringBuffer operations are thread-safe and synchronized. when multiple threads are working on same String, we should use StringBuffer but in single threaded environment we should use StringBuilder. StringBuilder performance is fast than StringBuffer because of no overhead of synchronization.

‘==’ will check whether the two objects have the same reference or same memory location but not the same content or value. Equals method meant for content comparison. Equals method result depends on overridden implementation. Since java.lang.String class override equals method, it returns true if two String objects has the same content.

If we want to compare two objects, it will call equals method in object class where ‘==’ is used to compare the object references but not the content, so equals method return false. To avoid this we have to override equals method so that comparing the ids or some features which uniquely identifies. When 2 instances are added as keys to hash map, using this equals method hash map can return back the appropriate values. Equals method must be reflexive, symmetric, transitive, consistent and if two objects are true for equals method then both objects hashcode return must be same. Hash code must be overridden to add objects as keys to hashmap. Each Hashcode return value will contain 1 hash bucket and same object hashcode values will go to same hash bucket. So that will search for hashbucket.