Part 2

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

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

JMS is an interface to send messages. You can design your application with Temporary Queues

 Basic tasks a JMS application must perform:

* Creating a connection and a session
* Creating message producers and consumers
* Sending and receiving messages

In a Java EE application, some of these tasks are performed, in whole or in part, by the container.

Each example uses two clients: one that sends messages and one that receives them. You can run the clients in NetBeans IDE or in two terminal windows.

When you write a JMS client to run in an enterprise bean application, you use many of the same methods in much the same sequence as for an application client. However, there are some significant differences.

- **How do you handle exception in JMS consumers and how to you recover?**

If a connection is failed over for a message consumer, the consequences vary with the sessions acknowledge mode:

* In client-acknowledge mode, calling Message.acknowledge or MessageConsumer.receive during a failover will raise a JMSException. The consumer should call Session.recover to recover or re-deliver the unacknowledged messages and then call Message.acknowledge or MessageConsumer.receive.
* In auto-acknowledge mode, after getting a JMSException, the synchronous consumer should pause a few seconds and then call MessageConsumer.receive to continue receiving messages. Any message that failed to be acknowledged (due to the failover) will be redelivered with the redelivered flags set to true.
* In dups-OK-acknowledge mode, the synchronous consumer should pause a few seconds after getting an exception and then call MessageConsumer.receive to continue receiving messages. In this case, it's possible that messages delivered and acknowledged (before the failover) could be redelivered.

JMS message has been acknowledged. While creating Queue session, make AUTO\_ACKNOWLEDGE as false. That means consumer has to acknowledge. When the consumer sends acknowledgement of message, then the message will be deleted from the queue, otherwise it will remain in the queue.

On the consumer side you have to do the same thing, create a queue session with AUTO\_ACKNOWLEDGE as false.

After working on your message, you can send acknowledge to delete the message from the queue or the message will remain in the queue.

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

Typically, LRU cache is implemented using a doubly linked list and a hash map. Doubly Linked List is used to store list of pages with most recently used page at the start of the list. So, as more pages are added to the list, least recently used pages are moved to the end of the list with page at tail being the least recently used page in the list. Hash Map (key: page number, value: page) is used for O(1) access to pages in cache  
  
When a page is accessed, there can be 2 cases:  
1. Page is present in the cache - If the page is already present in the cache, we move the page to the start of the list.  
2. Page is not present in the cache - If the page is not present in the cache, we add the page to the list.   
How to add a page to the list:  
   a. If the cache is not full, add the new page to the start of the list.  
   b. If the cache is full, remove the last node of the linked list and move the new page to the start of the list.

**public** **class** LRUCache extends LinkedHashMap<Integer, String> {

**private** **int** cacheSize;

**public** LRUCache(**int** size) {

**super**(size, 0.75f, **true**);

**this**.cacheSize = size;

}

@Override

**protected** boolean removeEldestEntry(

java.util.Map.Entry<Integer,String> eldest) {

// remove the oldest element when size limit is reached

**return** size() > cacheSize;

}

}

- **How would you implement Executor Service?**

ExecutorService is an interface that extends Executor class and represents an asynchronous execution. It provides us mechanisms to manage the end and detect progress of the asynchronous tasks.

How you create an ExecutorService depends on the implementation you use. However, you can use the Executors factory class to create ExecutorService instances too. Here are a few examples of creating an ExecutorService:

ExecutorService executorService1 = Executors.newSingleThreadExecutor();

ExecutorService executorService2 = Executors.newFixedThreadPool(10);

ExecutorService executorService3 = Executors.newScheduledThreadPool(10);

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

It involves only one class which is responsible to instantiate itself, ensures that only one object has need to be created for entire main stack (per application).

The implementation involves a static member in the "Singleton" class, a private constructor and a static public method that returns a reference to the static member.

E.g.: Logger Class – The singleton pattern is used in the design of logger class. This classes are usually implemented as a singleton, and provides a global logging access point in all the application components without being necessary to create an object each time a logging operations is performed.

- **Describe properties of Java String.**

Java String is an immutable object; you can’t modify any of it’s attribute’s values. JVM maintains a memory pool for String. When you create a String, first this memory pool is scanned. If the instance already exists, then this new instance is mapped to the already existing instance. If not, a new java String instance is created in the memory pool. When you use ‘new’ to instantiate a String, you will force JVM to store this new instance is fresh memory location thus bypassing the memory map scan.

**Methods of String class**  
  
**Accessor Methods**  
  
1. length() – To find the number of characters of a string  
2. charAt(index) – Returns the character a the given index  
3. split(string,delimiter) – Splits the at every delimiter and returns a String array  
4. substring(start[,endindex]) – Returns all the characters from start upto but not including endindex  
  
**Modifier Methods**  
  
1. concat(string) : Returns a new string after concatenating str to the original string. The original string remain unchanged.  
2. replace(charwith, charreplacement) : Replaces charwith with charreplacement and returns a new string.  
  
**Boolean test methods**  
  
1. endsWith(strend) : Returns true, if the string ends with strend  
2. equals(str) : Returns true, if the string and str are equal  
3. equalsIgnoreCase(str) : Returns true, if the string and the str are equal irrespective of case sensitivity  
4. startsWith(strbeg) : Returns true, if the string starts with strbeg