Note: Programming challenge 2 codes are inside Challenge_WW folder.

Part 2: Representative Questions

1) **How do you design an application with JMS messaging?**
   Ans: JMS stands for Java Messaging System which is an API that allows user to create, send, read and receive the messages between software components (loosely coupled) in a system. JMS is also asynchronous and there is no retransmission of messages causing duplication. The Java EE platform enhances the JMS API by providing support for distributed transactions and allowing for the concurrent consumption of messages.
   To design an enterprise level application with asynchronous JMS messaging following steps should be followed:

a) Use of the references to the interfaces defined in the JMS package. JMS defines a generic view of messaging that maps onto the transport or connectors. The application that uses JMS should makes use of following interfaces:
   Connection: Provides access to the underlying transport, is used to create Sessions.
   Session: Provides a context for producing and consuming the messages, using:
   MessageProducers (used to send messages)
   MessageConsumers (used to receive messages).
   Destination and ConnectionFactory are for administrative purpose.
   There are more interfaces specific to type of messaging service:
   point-to-point or publisher /consumer.

Application Server are also bound to JNDI namespace. We can directly use them in the application. This allows the required encapsulation with all the benefits of JMS API.

b) To improve performance, the application server pools connections and sessions with the JMS provider. We have to configure the connection and session pool properties appropriately for our applications, otherwise we might not get the connection and session behavior that we want. Applications must not cache JMS connections, sessions, producers or consumers.
c) A typical connection to message sending and closing the connection involves following steps:

1. Get the initial context.
2. Look up the queue.
3. Look up the queue connection factory(for point to point message domain, depends on message domain)
4. Create a queue connection.
5. Create a queue session
6. Create a queue sender
7. Create the message to be send to Subscriber.
8. Send the message
9. Close the queue connection.

Most of the steps for the receiver side are same as for the sender application – except it will listen the message rather than sending the JMS message.

➔There are basic and advance reliability mechanism provided by JMS. The basic mechanism allows to :->
Control message acknowledgement: to specify various levels of control over message acknowledgment.
Specify message persistence: either persistent or non-persistent i.e. they must not be lost in the event of a provider failure or otherwise.
Set message priority levels: level of priority that affect the order in which messages are delivered.
Allow message to expire: we can define the time for message to expire so they will not be delivered if they are obsolete.
Create temporary destinations: to last only during connection are present.

And advance mechanism allows application developer to :->
Create durable subscription: to receive messages published while subscriber was not active. It offers the reliability of queues to the P/S message domain.
Use local transaction: We can pool connections and session together in a transaction. It also provides the roll back in case of exception handling.

2) **How do you handle exception in JMS consumers and how to you recover?**
   JMS consumers can handle different situation like if the message received is corrupted then it should receive the message and put the notification back to the consumer/client. And if the message is correct but the processing at server side cannot be finished then the message should be left in the queue as JMS provides the feature of storing the message element in the queue even when the server or destination is not present.

   Apart from that JMS provides exception handling class JMSException. It's the most generic way of catching any exception related to JMS API. Several types of exceptions occur when client being reconnected after a failover. Catching of exception depends on whether it's a transacted session, type of exception and client is producer or consumer.

   - In case of transacted session, JMS consumer can fail either during processing transaction statements which is called open transaction or during session.commit() which is called commit transaction.
   - In the case of a failover during an open transaction, when the client application calls Session.commit(), the client runtime will throw TransactionRolledBackException and roll back the transaction causing the messages produced are discarded which were not committed successfully before failover and message consumed are redelivered to the consumer if not committed. A new connection is automatically started. When its failover during a call to session.commit() , either it can be transaction

is committed successfully and the call to Session.Commit doesn't return exception,JMS consumer don't have to do anything.

In case of Non-transacted session, JMS consumer can face various exception:
1. In consumer acknowledgement mode, calling Message.acknowledgement or MessageConsumer.receive can raise a JMS Exception. The consumer should call Session.recover to recover or redeliver the unacknowledged messages and then call Message.acknowledge or MessageConsumer.receive
2. Also in auto acknowledgement mode, on getting JMS Exception, the synchronous consumer should pause for some seconds and then resume by calling MessageConsumer.receive to continue receiving message. Any messages failed to be acknowledged will be redelivered using redelivered flag set to True.
3. Even in duplicate Acknowledgement mode, the synchronous consumer should wait few seconds after getting exception and call receive. It's possible that messages delivered and acknowledged could be redelivered.

3) How do you implement LRU or MRU cache?

LRU stands for Least Recently used and MRU for Most Recently Used. In Java, we can create a stack data structure to store the objects which are recently used in Last In first Out fashion. Even the objects references by JVM are stored in stack as function calls are finished and returned to the old order of processing in program.

We can also implement it by using Doubly LinkedList along with HashTable which are synchronized implicitly to have concurrent behavior.
get(key)= returns the value of the node of doubly linked list else return -1;
set(key,value)= insert the value if key is not present. When capacity is full then discard the least recently used item and store a new one.

Note: example in Cache.java class

4) How would you implement Executor Service?

Executor Service is an interface in Java which allows user to execute task in parallel. It allows user to run tasks with asynchronous execution mechanism to execute task in the background just like tread pool execution in Java. Its present in util.concurrent package and have implementation similar to thread pool.
We can implement Executor Service for Java Thread where each task in thread can be provided to executor service which runs this task in parallel with the native thread processing.
In Concurrent package, there are two implementation of Executor Service: ThreadPoolExecutor and ScheduleThreadPoolExecutor.

Note: Example in ExecutorService.java class

5) **Describe singleton pattern - how do you implement ?**
   Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.

The singleton class must provide a global access point to get the instance of the class. It is used for logging, driver objects, caching and thread pool. It is also used in other design patterns like AbstractFactory, Builder, Prototype, Façade etc. It is also used in core java classes, for example java.lang.Runtime, java.awt.desktop.

Advantage: Saves memory because object is not created at each request. Only single instance is reused again and again.

Creating singleton design pattern:

i. Private Constructor to restrict instantiation of the class from other classes.

ii. Private static variable of the same class that is the only instance of the class.

iii. Static factory method: This provides the global point of access to the singleton object and returns instance to the caller.

Note: example in SingletonDemo.java class

6) Describe the properties of string?
   In java, String is basically an object that represents sequence of char values.
   Every immutable objects are thread safe so string is also thread safe. The java.lang.String class implements Serializable, Comparable, and CharSequence interface.

   String Object can be created by two ways:

   a) By string literal
   Each time we create a string literal, the JVM checks the string constant pool first. If the string already exists in a pool, a reference to the pooled instanced is returned. If the string doesn't exist in the pool, new string instance is created and placed in the pool.

   For Example: String s1 = "welcome";
   String s2= "welcome" ; //will not create new instance

   b) By new keyword:
   String s=new String ("Welcome"); //creates two objects and one reference variable
   In such a case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

References:

1) https://www.javatpoint.com/jms-tutorial
2) https://dzone.com/articles/java-message-service-jms
3) http://www.coderpanda.com/jms-tutorial/