1) JMS enables loosely coupled design, so a big software can be broken into small parts and those parts can communicate with each other through JMS or it also enables on application to talk to another external application.

A JMS centric application has the following parts:

1. JMS provider: It is a messaging system that implements JMS specification.
2. JMS Client: Applications that send and receive messages.
3. Messages: These are the objects that communicate information between clients.
4. Administered Objects:

* Connection Factories: Used by client to create a connection to a provider.
* Destination: Specifies producer and consumer of the message.

2) An exception listener should be implemented with the connection.

* Client receives messages in a bulk of 5 and acknowledges them all.
* A session in CLIENT\_ACKNOWLEDGE mode is used to do so if space error occurs, the backup space takes its place as the primary space. Transaction is aborted and unacknowledged messages are automatically recovered.
* An exception is thrown, TransactionRolledBackException for synchronous consumer and SpaceLostException for asynchronous consumer.
* In the CLIENT\_ACKNOWLEDGE mode, client reacts as it would to a session recovery.
* The next message is the first unacknowledged message.
* The JMSRedelivered header of recovered messages is set.

3) An LRU cache can be implemented by extending a LinkedHashMap with access order set to true.

**private** LRUCache(**int** size) {

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

**this**.size = size;

}

**protected** **boolean** removeEldestEntry(Map.Entry<K, V> eldest) {

**return** size() > size;

}

4) Java Executor Service has two implementations:

* Thread Pool Executor
* Scheduled Thread Pool Executor

Java Executor service is used to create a thread pool, those threads are then used to do multiple tasks.

Here a thread pool is being created using a factory method:

ExecutorService executorService = Executors.*newFixedThreadPool*(10);

executorService.execute(**new** Runnable() {

@Override

**public** **void** run() {

System.***out***.println("abc");

}

});

5) A singleton class has just one instance at a time, to create a singleton class we have to make the constructor private and keep a static method which creates the instance of the class for the first time and the returns the instance whenever called.

**class** Singleton

{

**private** **static** Singleton *instance* = **null**;

**public** String s;

**private** Singleton()

{

//code

}

**public** **static** Singleton getInstance()

{

**if** (*instance* == **null**)

*instance* = **new** Singleton();

**return** *instance*;

}

}

6) Properties of String:

* A Java String is an object and not just an array of characters.
* String is a sequence of Unicode characters.
* String is immutable.

String a ="xyz";

a= "abc";//The contents of a were not modified, instead a new String was created.

* String can be assigned value directly by double quoted text instead of calling the constructor.

References:

<https://docs.gigaspaces.com/latest/dev-java/jms-failover.html>

<http://tutorials.jenkov.com/java-util-concurrent/executorservice.html#java-executor-service-video-tutorial>

<https://www.javatpoint.com/jms-tutorial>