**Part 2 – Representative questions**

* How do you design an application with JMS messaging?

The following should be configured for every JMS application:

The following properties should be set for jndiTemplate bean

java.naming.provider.url

java.naming.factory.initial

java.naming.factory.url.pkgs

java.naming.security.principal

java.naming.security.credentials

The following properties be configured for each JMS listener

autoStartup

connectionFactory

destination

messageListener

cacheLevelName

maxMessagesPerTask

concurrentConsumers

maxConcurrentConsumers

transactionManager

sessionTransacted

transactionTimeout

acceptMessagesWhileStopping

The following properties should be configured on the JMS sender side

connectionFactory

defaultDestination

sessionTransacted

deliveryMode

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

The client code should handle all possible exceptions caused by applications business logic that are thrown from a message listener.

Use 'CLIENT\_ACKNOWLEDGE' mode which provides redelivery in case of an exception.

* How do you implement LRU or MRU cache?

The LRU caching scheme is to remove the least recently used item when the cache is full and a new item is referenced which is not in cache. Queue implemented with doubly linked list is the ideal data structure to represent LRU cache. The maximum size of the queue will be the number of items available. The most recently used items will be at the front end of the queue and the least recently used items will be at the rear end of the queue. We maintain a Hashtable with keys and double linked nodes as values.

Here is the pseudocode for the implementation of get and set operations on LRU cache:

GET:

1. If the item exists in the cache, move it to the front of the queue as it is referenced recently

and return the value.

public int get(int key) {

if(map.containsKey(key)){

Node n = map.get(key);

remove(n);

setHead(n);

return n.value;

}

return -1;

}

SET:

1. If the item exists in cache, set this value to the existing item and move it to the front of the queue.
2. If the item does not exist in cache, create a new item.
3. If the size of the map has reached its capacity, then remove the item at the end of the queue, and add the new item to the front of the queue.
4. Else add the new item at the front of the queue.

public void set(int key, int value) {

if(map.containsKey(key)){

Node old = map.get(key);

old.value = value;

remove(old);

setHead(old);

}else{

Node created = new Node(key, value);

if(map.size()>=capacity){

map.remove(end.key);

remove(end);

setHead(created);

}else{

setHead(created);

}

map.put(key, created);

}

}

* How would you implement Executor Service?

ExecutorService present in the java.util.concurrent package is a thread pool implementation. An ExecutorService is created from one of the factory methods of Executors class.

ExecutorService executorService = Executors.newFixedThreadPool(5);

This creates an ExecutorService with fixed thread pool size 5.

We can create n tasks and hand them to the executor service. It can handle both Runnable and callable tasks.

Simple implementation of ExecutorService with Runnable tasks is as follows:

ExecutorService executorService = Executors.newFixedThreadPool(5);

try{

for (int i = 0; i < 10; i++) {

Runnable task = new TaskThread("" + i);

executorService.execute(task);

}

} catch(Exception e) {

e.printStackTRace();

} finally {

executorService.shutdown();

}

Simple implementation of ExecutorService with Callable tasks is as follows:

ExecutorService executorService = Executors.newFixedThreadPool(1);

ExecutorCompletionService taskCompletionService = new ExecutorCompletionService<Object>(executorService);

Callable c = new Callable() {

public Object call() {

try {

System.out.println("From callable");

} catch (Exception e) {

throw new AppRuntimeException(e);

}

return null;

}

};

taskCompletionService.submit(c);

* Describe singleton design pattern – how would you implement?

The following is the definition of Singleton pattern as given in GoF patterns.

Definition: Ensure a class has only one instance, and provide a global point of access to it.

1. Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine. A private static inner class restricts the instantiation of the class by other classes. A private static variable of the class ensures single instance.
2. The singleton class must provide a global access point to get the instance of the class. A public static method that returns the instance of the class ensures the global access point.
3. Singleton pattern is used for [logging](https://www.journaldev.com/977/logger-in-java-logging-example), drivers objects, caching and [thread pool](https://www.journaldev.com/1069/threadpoolexecutor-java-thread-pool-example-executorservice).

public class MySingleton {

private MySingleton(){}

private static class SingletonHelper{

private static final MySingleton INSTANCE = new MySingleton();

}

public static MySingleton getInstance(){

return SingletonHelper.INSTANCE;

}

}

* Describe properties of Java String.

1. String is immutable. The content cannot be modified once created. Any operations on the String object results in creating a new String.
2. String is associated with String literal. One can assign a String literal directly to a String variable instead of calling the constructor to create it. String literals are stored in a common pool. String literals with the same content share the storage. String objects created via new operator are stored in heap.
3. String is designed to be in between primitive and class for performance reasons. The “+” operator is overloaded to concatenate two String objects.
4. Java’s String class privately maintains a pool of strings, where String literals are automatically interned. When the intern() method is invoked on a String object it looks the string contained by this String object in the pool, if the string is found there then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.