# Part 2

* How do you design an application with JMS messaging?

Answer:-

Process: Designing an application with JMS, we 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 map 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: which provides access to the underlying transport, and is used to create Sessions.

Session: which provides a context for producing and consuming messages, including the methods used to create Message Producers and Message Consumers.

Message Producer: Used to send messages.

Message Consumer: Used to receive messages.

We need to decide whether we are implementing Queue or Topic, based on that we should use respective interfaces.

Queue: Point-to-point messaging.

Interfaces: QueueConnectionFactory, Queue, QueueSender, QueueReceiver, QueueSession.

Topic: Publish/Subscribe messaging.

Interfaces: TopicConnectionFactory, Topic, TopicSession, TopicPublisher, TopicSubscriber.

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

Answer:

When we are working on a JMS- application, which is being used as an integration point between several systems.

We need to handle exceptions. If checked exceptions, those errors that can be handled by the system, If unchecked exceptions, those errors that cannot be handled.

“What happens if processing the message fails?”

In this scenario we should throwing an unchecked exception.

This being our first scenario, we’ve assumed that any runtime exception is a transient error possibly down to one of the systems we’re integrating with being down.

The simplest way to re-try a message is to use a transactional JMS session and rollback the session.

Which returns the message back on to the topic/queue, and then we specify a maximum number of times a message can be retried.

When using a transactional session you need to commit the session if the message was successfully processed.

 We do have to expose the session inside the listener.

Try{

topisSession.commit();

}catch(Throwable t)

{

--;

topicSession.rollback();

}

(Or)

public void onMessage(Message m) {

try {

Object command = new MessageHandler(m);

process(command);

topicSession.commit();

} catch (MessageInvalidException e) {

log(t);

//we don't want to retry the message

topicSession.commit();

} catch (Throwable t) {

log(t);

topicSession.rollback();

}

}

* How do you implement LRU or MRU cache?

Answer: We can implement LRU cache two ways-

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.

Hash: Page number as key and address of the corresponding queue node as value.

We add a new node to the front of the queue and update the corresponding node address in the hash. If the queue is full, that means all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.

* How would you implement Executor Service?

Answer:

Executor Service 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.

Initially an Executor Service is created using the newFixedThreadPool() factory method.

Which will creates a thread pool with number of threads executing tasks.

Second, an anonymous implementation of the Runnable interface is passed to the execute() method.

Which causes the Runnable to be executed by one of the threads in the Executor Service.

We have to do task delegation to the Executor Service, then threads will performs their tasks.

When you are implementing Executor Service:

This Executor Service has to implementation based on the java.util.concurrent package.

1.ThreadPoolExecutor

2.Sch eduledThreadPoolExecutor.

Note: You can use the Executor Factory class to generate Executor Instances.

Task Delegation:

There few different ways to delegate tasks for execution to an Executor Service.

1.execute(Runnable)

2.submit(Runnable)

3.submit(callable)

4.invokeAny()

5.invokeAll()

Executor Service Shutdown:

When we are done using the Executor Service you should shut it down, so the threads do not keep running.

Assume, if your application is started via a main() method and main thread exits your application, the application will keep running if you have an active Executor Service in your application.

To terminate the threads inside the Executor Service you call its shutdown () method.

If you want to shut down the Executor Service immediately, you can call the shutdownNow() method.

* Describe singleton design pattern – how would you implement?

Answer:

Java Singleton pattern is one of the Gangs of Four Design patterns and comes in the Creational Design Pattern category.

From the definition, it seems to be a very simple design pattern but when it comes to implementation, it comes with a lot of implementation concerns.

The implementation of Java Singleton pattern has always been a controversial topic among developers.

Here we will learn about Singleton design pattern principles, different ways to implement Singleton design pattern and some of the best practices for it’s usage.

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.

Singleton pattern is used for logging, driver’s objects, and caching and thread pool.

Pattern:

Private constructor to restrict instantiation of the class from other classes.

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

Public static method that returns the instance of the class, this is the global access point for outer world to get the instance of the singleton class.

Steps:

1. Eager initialization

2. Static block initialization

3. Lazy Initialization

4. Thread Safe Singleton

5. Bill Pugh Singleton Implementation

6. Using Reflection to destroy Singleton Pattern

7. Enum Singleton

8. Serialization and Singleton

* Describe properties of Java String.

Answer:

String is a sequence of characters. But in java, string is an object that represents a sequence of characters.

The java.lang.String class is used to create string object.

Creating String two ways:

1. By string literal

2. By new keyword

Java String literal is created by using double quotes.

Example:

String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first.

If the string already exists in the pool, a reference to the pooled instance is returned.

If string doesn't exist in the pool, a new string instance is created and placed in the pool.

Example:

String s1="Welcome";

String s2="Welcome";//will not create new instance

Uses:

Memory efficient, because no new objects are created if it exists already in string constant pool.

Java New Keyword:

String s=new String("Welcome");

In such 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).

1.String is immutable:

String objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once we create object we cannot modify it.

Example:

class Testimmutablestring{

public static void main(String args[]){

String s="Gopi";

s.concat(" Krishna");

System.out.println(s);

} }

Output: Gopi

String methods:

CompareTo()

Concat()

Contains()

Equals()

Length()

Split()

Trim()