**1.How do you design an application with JMS messaging?**

To develop an application with JMS messaging we require

1. Imported Required Packages
2. Setting up a JMS application
3. Sending Messages
4. Receiving Messages
5. Acknowledging Receiving messages
6. Releasing Object Resources

The following steps are involved during design development:

* Manage connection and session processing
* Create destinations dynamically
* Create durable subscriptions
* Manage message processing by setting and browsing message header and property fields, filtering messages, and/or processing messages concurrently.
* Use JMS within transactions.

**2.How do you handle exceptions in JMS consumers and how to you recover?**

In the application thread that receives the exception should pause for a few seconds and then resend the messages. The client application may want to set a flag on the resent messages to indicate that they could be duplicates.

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 MesssageConsumer.receive during a failover will raise a JMS exception. The consumer should call Session.recover to recover 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.

**3.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 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.

**Example:**

java.util.HashMap;

import java.util.Map;

public class LRUCache {

private DoublyLinkedList pageList;

private Map<Integer, Node> pageMap;

private final int cacheSize;

public LRUCache(int cacheSize) {

this.cacheSize = cacheSize;

pageList = new DoublyLinkedList(cacheSize);

pageMap = new HashMap<Integer, Node> ();

}

public void accessPage(int pageNumber) {

Node pageNode = null;

if(pageMap.containsKey(pageNumber)) {

// If page is present in the cache, move the page to the start of list

pageNode = pageMap.get(pageNumber);

pageList.movePageToHead(pageNode);

} else {

// If the page is not present in the cache, add the page to the cache

if (pageList.getCurrSize() == pageList.getSize()) {

// If cache is full, we will remove the tail from the cache pageList

// Remove it from map too

pageMap.remove(pageList.getTail(). getPageNumber());

}

pageNode = pageList.addPageToList(pageNumber);

pageMap.put(pageNumber, pageNode);

}

} public void printCacheState() {

pageList.printList();

System.out.println();

}

public static void main (String [] args) {

int cacheSize = 4;

LRUCache cache = new LRUCache(cacheSize);

cache.accessPage(4);

cache.printCacheState();

cache.accessPage(2);

cache.printCacheState();

cache.accessPage(1);

cache.printCacheState();

cache.accessPage(1);

cache.printCacheState();

cache.accessPage(4);

cache.printCacheState();

cache.accessPage(3);

cache.printCacheState();

cache.accessPage(7);

cache.printCacheState();

cache.accessPage(8);

cache.printCacheState();

cache.accessPage(3);

cache.printCacheState();

}

}

class DoublyLinkedList {

private final int size;

private int currSize;

private Node head;

private Node tail;

public DoublyLinkedList(int size) {

this.size = size;

currSize = 0;

}

public Node getTail() {

return tail;

}

public void printList() {

if (head == null) {

return;

}

Node tmp = head;

while (tmp != null) {

System.out.print(tmp);

tmp = tmp.getNext();

}

}

public Node addPageToList(int pageNumber) {

Node pageNode = new Node(pageNumber);

if (head == null) {

head = pageNode;

tail = pageNode;

currSize = 1;

return pageNode;

} else if(currSize < size) {

currSize++;

} else {

tail = tail.getPrev();

tail.setNext(null);

}

pageNode.setNext(head);

head.setPrev(pageNode);

head = pageNode;

return pageNode;

}

public void movePageToHead(Node pageNode) {

if(pageNode == null || pageNode == head) {

return;

}

if (pageNode == tail) {

tail = tail.getPrev();

tail.setNext(null);

}

Node prev = pageNode.getPrev();

Node next = pageNode.getNext();

prev.setNext(next);

if (next != null) {

next.setPrev(prev);

}

pageNode.setPrev(null);

pageNode.setNext(head);

head.setPrev(pageNode);

head = pageNode;

}

public int getCurrSize() {

return currSize;

}

public void setCurrSize(int currSize) {

this.currSize = currSize;

}

public Node getHead() {

return head;

}

public void setHead(Node head) {

this.head = head;

}

public int getSize() {

return size;

}

}

class Node {

private int pageNumber;

private Node prev;

private Node next;

public Node(int pageNumber) {

this.pageNumber = pageNumber;

}

public int getPageNumber() {

return pageNumber;

}

public void setPageNumber(int data) {

this.pageNumber = data;

}

public Node getPrev() {

return prev;

}

public void setPrev(Node prev) {

this.prev = prev;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

public String toString() {

return pageNumber + " ";

}

}

**4. How do you implement Executor Service?**

[Executor Service](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html) is an interface that extends [Executor](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executor.html) class and represents an asynchronous execution. It provides us mechanisms to manage the end and detect progress of the asynchronous tasks.

In this example we are going to see some basic functionalities of Executor Service, as well as handle the [Future](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html) object, the result of asynchronous computation.

**Example:**

Package com.concurrency.executerservicetest;

Public class MyThread implements Runnable {

Private String myName;

Private int count;

Private final long timesleep;

MyThread(String name, int newcount, long newtimeSleep) {

this.myName = name;

this.count = newcount;

this.timeSleep = newtimesleep;

}

@Override

Public void run () {

int sum = 0;

for (int i = 1; i <= this.count; i++) {

sum = sum + i ;

}

System.out.println(myName + “thread has sum =” + sum + “and is going to sleep for “ + timeSleep);

try {

Thread.sleep(this. timeSleep);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**4.Describe the singleton design pattern how do you implement in java?**

The Singleton design pattern addresses all these properties

* Ensure that only one instance of a class is created
* Provide a global point of access to the object
* Allow multiple instances in the future without affecting a singleton class's in program.

**Example:**

public class ClassicSingleton {

private static ClassicSingleton instance = null;

protected ClassicSingleton() {

// Exists only to defeat instantiation.

}

public static ClassicSingleton getInstance() {

if(instance == null) {

instance = new ClassicSingleton();

}

return instance;

}

}

**6.Describe properties of java string?**

String is a sequence of characters. In java strings are immutable objects which means it is a constant and cannot be changed once it has been created. References to a java string is mutable. Java string is final.

Creating a copy of existing java string is easier as there is no need to create a new instance but can be easily created by pointing to already existing string.JVM maintains a memory pool for strings.

**Example:**

public class Example {

public static void main (String [] args) {

// creating a string by java string literal

String str1 = “Begineersbook”;

Char arrch[]={‘H’, ’e’, ’l’, ‘l’, ‘o’};

String str2 = new String(arrch);

//creating another java string str3 by using new keyword

String str3 = new String (“Java String Example”);

//displaying all three strings

System.out.println(str1);

System.out.println(str2);

System.out.println(str3);

}

}

**Output:**

Beginnersbook

Hello

Java String Example