Deliverable

- ***GITHUB URL with the project checked into the GITHUB***
    - Code
    - Documentation
    - Answers to the challenges

Part 1

Programming Challenge 1:

You are to create two small applications for this programming task; one is called Randomizer, the other Prime Randomizer's job is to generate a series of positive random integers and send those to Prime via a distributed queue of integers.

Primes job is to receive the integers and calculate whether the integer is a prime or not and return the answer to Randomizer via a distributed queue (just a java Queue implementation , no need to implement JMS etc) that contains the original number and a Boolean; which Randomizer will print to system out.

Points
1. Use only the standard java library
2. Both Applications will run on the same server
3. The system should be as fast as possible
4. The results do not have to be returned in the same order as received
5. You don't have to go overboard tweaking the prime check

Deliverable

1) A project with code – eclipse or any IDE that you are using. However, eclipse is preferable
2) A Readme.txt should contain the following
    a. What is your design and implementation?
    b. Sample Output
    c. Further work – if you have all the time in the world how would you implement

Design Approach:

I thought of using RMI server client API provided in Java. But RMI allows to call the object remotely from sewer but I wanted something which provides sharing the data structure. Providing methods to access the common queue was the idea.

Instead I used simple concurrent access provided by Java util package. It provides ConcurrentLinkedQueue which can be accessed by various thread by time sharing and updated. Its synchrnized and the java class accessing just need to implement runnable interface.

Files are uploaded on github under folder name: RandomisedPrimeChecker The RandomizedPrime class calls the randomizer thread and prime thread by passing the input and output queue. Inputqueue: integer concurrent linked queue outputQueue: concurrent linked queue holding two objects num as well as its Boolean result.

Randomizer thread generates the number and adds it to the input queue and checkes outputqueue for any result for printing. Prime thread checks input queue for input and calls the isPrime(int num) method and add the output to output queue.

The sample run works for 10 numbers and prime number within range 0 to 100 which can be alter by asking user to give range before starting the threads and option to end the thread.

Programming Challenge 2:

1) Reverse of a string without reverse method.

```
class StringReverse
{
    public static void main(String[] args)
    {
        String input = "This is String reverse";
        char[] temp = input.toCharArray();
        int i, j=0;
        j = temp.length-1;

        for (i=0; i < j ; i++ ,j--)
        {
            // Swap values of left and right
            char tempchar = temp [i];
            temp [i] = temp [j];
            temp [j]=tempchar;
        }

        for (char ch : temp)
            System.out.print(ch);
        System.out.println();
    }
}
```

2) Find a palindrome

```
Class Palindrome{

    public static void main(String[] args)
    {
        String str = "Madam, I'm Adam";
        String reverse = new StringBuffer(s).reverse().toString();
        if (str.equals(reverse))
             System.out.println("Yes, it is palindrome");
        else
             System.out.println("Not a palindrome");
    }
}
```

2) Please create a class for the below and send it,
We have a table which has 4 columns as id, name, phone and address.
You need to have a method which will return me the data.
Create a method where you can hard code the data and print the details.
The main aim of above example is how you use data structure.

```
Class Customer{

private long id;
private String name;
private long phoneNo;
private String address;

<getters> and <setters>

public Customer(long id, String name, long phoneNo,String
address){

this.id = id;
this.name = name;
this.phoneNo= phoneNo;
this.address = address;

}

Public void printDetails(){

return "id : "+this.id+ " name :"+ this.name+" phone:
"+this.phoneNo+" address: "+ this.address;

}
}

Public static void main(String [] args){

 Customer cust = new
Customer(10L,"FirstName",90000000L,"City,State,Country");
String details =   cust.printDetails();
System.out.println(details);
}
```

Part 2

Representative Questions – Please write your answers with an example for each questions.

- How do you design an application with JMS messaging?
- How do you handle exception in JMS consumers and how to you recover?
- How do you implement LRU or MRU cache?
- How would you implement Executor Service?
- Describe singleton design pattern – how would you implement?
- Describe properties of Java String.

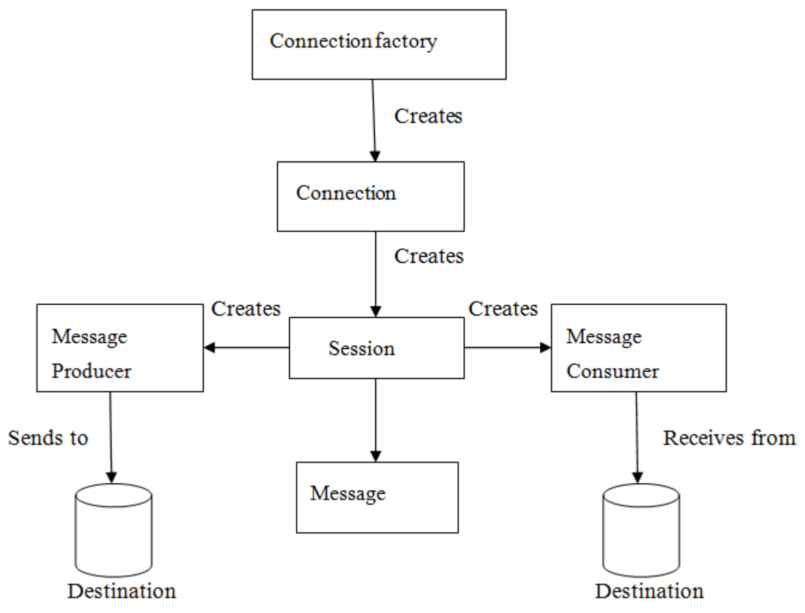
Deliverable

1) Simple document with Question and corresponding answers. If you have used data from anywhere please include reference.

How do you design an application with JMS messaging?

JMS application has some basic building blocks, which are:

1. Administered objects – Connection Factories and Destination
2. Connections
3. Sessions
4. Message Producers
5. Message Consumers
6. Message Listeners



1) 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 so it doesn't need to connect to receive the message as well as reliable and there is no retransmission of messages causing duplication. The Java EE platform, moreover, enhances the JMS API by providing support for distributed transactions and allowing for the concurrent consumption of messages.
2) To design an enterprise level application with asynchronous JMS messaging we can go in following manner: We make 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:
3) Connection: Provides access to the underlying transport, and is used to create Sessions.
4) Session: Provides a context for producing and consuming the messages, using methods MessageProducers (used to send messages) and MessageConsumers (used to receive messages). Destination and ConnectionFactory are for administrative purpose which are pushed on to the cosumers to administers the application.
5) There are more interfaces specific to type of messaging service: point-to-point (Queue, QueueSender, QueueBrowser, QueueReceiver) or publisher /consumer(TopicSubscriber, TopicPublisher, TopicSessions). Application should refer to the predefined administered objects to WebSphere Application Server and are also bound to JNDI namespace. We can directly use them in the application without worrying internal implementation. This allows the required encapsulation with all the benefits of JMS API. JMS resource for Point-to-point: Connection Factory(or QueueConnectionFactory) , Queue and resource for Publish/Subscribe: ConnectionFactory(or TopicConnectionFactory) , Topic. A connection factory is used to create connections from the JMS provider to the messaging system, and encapsulates the configuration parameters needed to create connections.
6) 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. WebSphere Application Server closes these objects when a bean or servlet completes, and so any attempt to use a cached object will fail with a javax.jms.IllegalStateException exception.
7) To further increase performance of the application, we should cache JMS objects that have been looked up from JNDI. For eg: createConnection method on each instantiation is needed, and due to pooling connections and sessions with the JMS provider, there wont be performance impact.
8) A non-durable subscriber can only be used in the same transactional context (for example, a global transaction or an unspecified transaction context) that existed when the subscriber was created.
9) Whereas a durable subscription on a JMS topic enables a subscriber to receive a copy of all messages published to that topic, even after periods of time when the subscriber is not connected to the server. So if an application creates a durable subscription, it is added to the runtime list that administrators can display and act on through the administrative console. Each durable subscription is given a unique identifier, clientID##subName where client identifier is used to associate a connection and its objects with the messages maintained for applications. Application administrator can even use naming which are familiar to help identify the applications. subName is the subscription name to identify the durable subscription uniquely within a given client identifier. For durable subscriptions created by message-driven beans, these values are set on the JMS activation specification. For other durable subscriptions, the

client identifier is set on the JMS connection factory, and the subscription name is set by the application on the createDurableSubscriber operation.

10) To create a durable subscription to a topic, an application uses the createDurableSubscriber operation defined in the JMS API: public TopicSubscriber createDurableSubscriber(Topic topic, java.lang.String subName, java.lang.String messageSelector,Boolean noLocal throws JMSException) where topic is the name of the JMS topic to subscribe to which is an object supporting Topic interfaces, subName is name to identify this subscription, messageSelector defines the expression of messages allowed to consumers(null or no values means all messages to be delivered) and noLocal is true to prevent delivery of messages published on the same connection as the durable subscriber.
11) Alternatively, we can use two argument createDurableSubscriber operation that takes only topic and subName parameters. It directly invokes the four argument version shown previously, but sets messageSelector to null (so all messages are delivered) and sets noLocal to false (so messages published on the connection are delivered). For example, Session createDurableSubscriber(myTopic, "mySubscription");
12) If this operation fails, it throws a JMS exception that provides a message and linked exception to give more detail about the cause of the problem.
13) 8.To delete a durable subscription, an application uses the unsubscribe operation defined in the JMS API. In normal operation there can be at most one active subscriber for a durable subscriber at a time. However, the subscriber application can be running in a cloned application server, for failover and load balancing purposes. In this case the "one active subscriber" restriction is lifted to provide a shared durable subscription that can have multiple simultaneous consumers.
14) Selection of Message Selectors: We can use the JMS message selector mechanism to select a subset of the messages on a queue so that this subset is returned by a receive call. The selector can refer to fields in the JMS message header and fields in the message properties.
15) On receiving the message we can process it using business logic. Also we need to handle all sorts of Exception at this point the message is of the correct type and extract the content of the message to more specific such as TextMessage. JMS applications using the default messaging provider can access, without any restrictions, the content of messages that have been received from WebSphere Application Server embedded messaging or WebSphere MQ (provide by IBM Stack).
16) Using a listener to receive messages asynchronously. In a client, not in a servlet or enterprise bean, an alternative to making calls to QueueReceiver.receive() is to register a method that is called automatically when a suitable message is available. When a message is available, it is retrieved by the onMessage() method on the listener object.

Example:

```
 import javax.jms.*;
public class MyClass implements MessageListerner{
public void onMessage(Message message) {
    System.out.println("message is "+ message);
    //application logic

}}
```

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to receive() methods. To cope with this situation, we can register an ExceptionListener, which is an instance of a class that implements the onException() method. When an error occurs, this method is called with the JMSException passed as its only parameter.
In short: 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.
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.

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

Handling Exceptions in a Non-Transacted Session
If a connection is failed-over for a producer in a non-transacted session, a client application may receive a JMSException. 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 MessageConsumer.receive during a failover will raise a JMSException. The consumer should call Session.recover to recover or re-deliver 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.
- In dups-OK-acknowledge mode, the synchronous consumer should pause a few seconds after getting an exception and then call MessageConsumer.receive to continue receiving messages. In this case, it's possible that messages delivered and acknowledged (before the failover) could be redelivered.

Failover Producer Example
The following code sample illustrates good coding practices for handling exceptions during a failover. It is designed to send non-transacted, persistent messages forever and to handle JMSExceptions when a failover occurs. The program is able to handle either a true or false setting for the imqReconnectEnabled property

Failover Consumer Example
The following code sample, FailoverConsumer, illustrates good coding practices for handling exceptions during a failover. The transacted session is able to receive messages forever. The program sets the auto reconnect property to true, requiring the Message Queue runtime to automatically perform a reconnect when the connected broker fails or is killed.

- 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. I am 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 a 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 due to the transaction is committed successfully and the call to Session.Commit doesn't not return an exception, JMS consumer doesn't have to do anything. Or if the runtime throws a TransactionRolledbackException and does not commit the transaction. The transaction automatically rolled back by Message Queue runtime and the consumer must retry the transaction. And if a JMXException is thrown

means the transaction state is unknown. So the consumer should consider it as failure and try again after a pause for some time and calling Session.rollback() to resume the operations. But in case the transaction as successful to handle such cases the producer should set application specific properties on the messages it resends to signal that these might be duplicate messages. In other words, to endure that messages are not duplicated both producers and consumers need to handle this edge case separately which is rare occurrence.

- 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.
- There are couple of best practices to be followed while developing JMS application:
    1. We should avoid extra features in the JMS feat if not needed in our application like using the correct acknowledgement mode for the message and using non persistence message mode wherever possible greatly increase the throughput of the application.
    2. Message selectors that only consider message header fields will run the fastest.A selector that uses the message properties is slower, and if a selector examines the message body, it will be even slower. So implementations depend on the type of requirement and can be optimized more to increase performance. Message Selectors is well resemblance to JMS topics. When using point-to-point message domain, queue retain message even if the consumer haven't selected for message. So Message Selector are more efficient when used with topics than queues.
    3. Use of Asynchronous consumers than synchronous wherever possible allows use of less resources and no block of resources so exceptions can be prevented.
    4. Many JMS applications require transactional messaging. Instead of using Transacted Sessions, we should use JTA UserTransaction interface enables the non-transacted session to participate in a transaction that can encompass other components, including JDBC or EJB.
    5. If the message is corrupt or invalid, when receiving it on consumer side if the transaction aborts, the message will be returned to the queue and delivered again to the consumer. Unless this error is transient, the second delivery will cause a transaction rollback, and the process continues. This should be handled separately by sending it to separate exception handling block as soon as received then sent to read block. Or the acknowledgement should be separated from EJBor JDBC transaction. The JMS consumer uses the CLIENT_ACKNOWLEDGE or AUTO ACKNOWLEDGE modes for acknowledgement.

- How do you implement LRU or MRU cache?

  LRU stands for Least Recently used and MRU for Most Recently Used meaning the same. 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 function calls in recursive function also uses the same phenomenon, the base case gets called at the end and implemented first. 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. 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 O(1) 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

  Approach: Cache is used to store request to process it faster in future but keeping them in least recently used order and remove least recently item used when a new one comes. So
  cache hits needs to be 0(1). We also need a list of items to keep update of new objects at top and discard old one at bottom. For 0(1) access we use HashMap as insertion and deletion is also 0(1) as compared to Arrays. And we store the items in LinkedList where we can pin point head and tail. Java has LinkedHashMap in util package to use.

  Alternative: 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.

- How would you implement Executor Service?
  An ExecutorService is thus very similar to a **thread pool**. In fact, the implementation of ExecutorService present in the java.util.concurrent package is a thread pool implementation
  - Describe singleton design pattern – how would you implement?
  - Describe properties of Java String.

Since ExecutorService is an interface, you need to its implementations in order to make any use of it. The ExecutorService has the following implementation in the java.util.concurrent package:

- **ThreadPoolExecutor**
- **ScheduledThreadPoolExecutor**

1.) 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.
2.) 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. Java provides different factory methods to use executor:

ExecutorService e Executors.newSingleThreadExecutora It contains only a single thread so all task passed to this executor will be run by this single thread in sequential order and rest of the task waits in queue. WE use this when we want our tasks to be performed sequentially.

ExecutorSerivce e=Executors.newFixedThreadPool(5),

It contains fixed number of threads. All task passed are executed by any of these thread. Whereas ExecutorService e= Executors.newScheduledThreadPool(5); allows to schedule the process of task.

3.) ExecutorService contains methods to pass task for execution: execute 0- which takes runnable method as parameter of task similar to thread. submit 0- takes runnable as well as callable as parameter. Callable method parameter has return property not present with runnable instances. And other methods like invokeAny 0, invokeAll 0 and shutdown() to shutdown the executor when the application doesn't need to parallel task executor.

Implementation Example:

```
TaskThread class

 package Part2;

public class TaskThread implements Runnable {

private String command; //constructor of TaskThread

class public TaskThread(String s){ this.command

} //override run method of thread as it implements runnable

public void run() {

 System.out.println(Thread.currentThreada.getName()+" Starting. enter your
command = "+command); processCommand();
```

```
System.out.println(Thread.currentThreada.getName()+" End of task"); }
//processCommand method ib private void processCommand() {

try {

a Thread.sieep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}



} //to return the command in string format

public String toString(){return this.command; }

}

ExecutorSerivce class:

package Part2;

import java.util.concurrent.Executors;

public class ExecutorService {

public static void main(String[] args) {

 Java.util.concurrent.ExecutorService e Executors.newFixedThreadPool(5);

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

Runnable task = new TaskThread("" + e.execute(task);

}}
```

- Describe singleton design pattern – how would 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.

  Singleton pattern is used for logging, drivers objects, caching and thread pool.

  Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.

Singleton design pattern is used in core java classes also, for example java.lang.Runtime, java.awt.Desktop.

To implement Singleton pattern, we have different approaches but all of them have following common concepts.

- 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.

Singleton pattern is one of the design paradigm in Java. It comes as the creational design patterns as its used mostly for creating an object of the class but only one object of it. This pattern has a single class which is responsible for creation of object and also makes sure at the same time that only single object is created. The single class also provides method to

access its only object which can be directly accessed instead of instantiating the object of the class. The class is called Singleton Class.

Implementation: We will create a class with private constructor so no other class or method can access it outside this singleton class and create more objects. And we will create a static instance of the class within itself which we will return by one get method() to be accesses outside the class. Just declaring the instance variable doesn't stop to create the new instance by calling new Object() even though the variable is static and stored as single copy in the memory so to stop that we can make class singleton. We make the class as singleton class when we need to control the instances of an object. When we share the same object state across the application we should make that class singleton so to avoid creation of more than one object. Sample Code:

```
Singleton class:  package Part2; import java.util.*; import 'avasio.*;

public class Singleton{

private static Singleton s = new Singleton();

private int value;

private String msg;

private Singleton(){ //constructor to assign the other property of
class

value= 10;

 msg "Default message";

}
public static Singleton getInstance(){  return s; }



public void printvalues(){

 System.out.println("Value: "+value+"Msg:"+msg);

 }

}

SingletonDemo class:  package Part2;

import java.io.*; import java.util.*;

public class SingletonDemo {

public static void main (String args[])throws IOException{

Singleton obj= Singleton.get/nstance(); obj.printvalues();

 }

}
```

- Describe properties of Java String.

1.) In Java, String is a class and immutable object. So for an immutable object we cannot modify any of its attributes values. Once declared and assigned it cannot be manipulated.
2.) String is collection of characters representing a word, similar to array which stores values of same type in contiguous fashion. Likewise, String is continuous sequence of character ending with a null to signify the end of the string. There are different ways to declare and initialize string data type. Eg: String str= "Hello World"; Whenever string Literal is encountered, the compiler stores this in memory but before it checks in string pool memory (String intern pool in Java heap) if the same string is there then only the reference is returned to old memory location otherwise it is stored in the pool and reference is returned. That's why String object is immutable so that it doesn't alter the value of location as multiple references are referencing to the same memory location holding the String value.
3.) So whenever a String is needed to be altered or manipulated a lot we should use String Buffer and Builder class. String Builder is unsynchronized and String buffer is used for thread safe code because its synchronized.
4.) Also we can use new Constructor to declare the String. Eg: String s= new String("Hello World"); The String class provides the various constructor to create String from char arrays, etc. Whenever new constructor is used it doesn't store it in String pool and create the new string object which is immutable. So if we compare the object referencing the same location in memory using "==" will gives us false as it creates new object references pointing to same

location when we use new constructor. We can use intern method() to specify for the constructor to check in String pool and store it in pool if not present already. So when we need to create a new String object holding same old value we should use new Constructor to declare it.

5.) Java String class provides various inbuild method for manipulations of string. Like String.length(); gives the length of the string. We can concate two string using strl.concat(str2); We can also format the string using String.format(); Strl.equals(str2); checks if two string references hold same value or not. Whereas Str1== Str2; checks if two references are equal or not. Strl= str2; is valid and it assign the strl reference to point to where str2 is referencing. Various other method charAt(), compareTo(),copy(),index0f() provides other functionalities.