UNIT V

String Handling in Java: Introduction, Interface Char Sequence, Class String, Methods for Extracting Characters from Strings, Methods for Comparison of Strings, Methods for Modifying Strings, Methods for Searching Strings, Data Conversion and Miscellaneous Methods, Class String Buffer, Class String Builder.

Multithreaded Programming: Introduction, Need for Multiple Threads Multithreaded Programming for Multi-core Processor, Thread Class, Main Thread- Creation of New Threads, Thread States, Thread Priority-Synchronization, Deadlock and Race Situations, Inter-thread Communication - Suspending, Resuming, and Stopping of Threads.

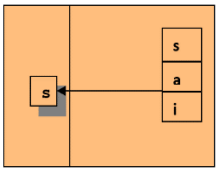
Java Database Connectivity: Introduction, JDBC Architecture, Installing MySQL and MySQL Connector/J, JDBC Environment Setup, Establishing JDBC Database Connections, ResultSet Interface, Creating JDBC Application, JDBC Batch Processing, JDBC Transaction Management.

String Handling in Java:

String: String is a group of characters surrounded by double quotes, in java String class is used to represent these group of characters.

Ex:

String s="sai"



1. Address of the object is assigned to the variable s. This object will be like a character array.
2. Once a string object is created, the data inside the object cannot be modified, that's why we say strings are immutable.

Different ways of creating String objects:

String s1=new String();

String s2=new String("sai");

String s3="sai";

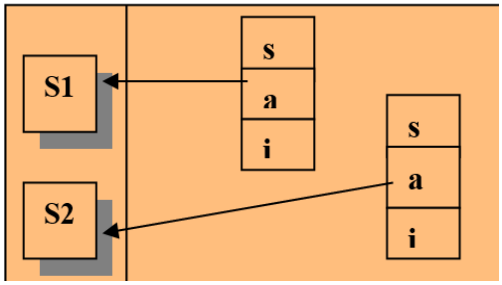
Difference between new and " "

1. Using new keyword any number of objects are created for the same class.

String s1=new String("sai");

String s2=new String("sai");

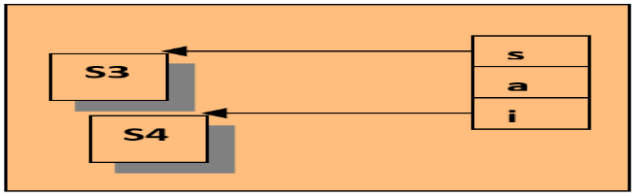
The above statements will create two different objects, with different address(same contents) assigned to different variable s1 and s2 respectively.



2. Assignment operator is used to assign the string to the variable s3. in this case, JVM first of all checks whether the same object is already available in the string constant pool or not. If it is available, then it creates another reference to it. If same object is not available, then it creates another object with the with the content "PREM" and stores it into the string constant pool.

String s3="sai";

String s4="sai";

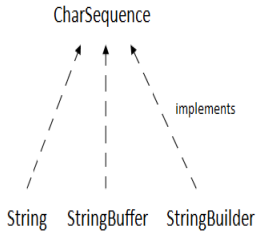



```
class StringDiff
{
 public static void main(String args[])
 {
  String s1=new String("sai");
  String s2=new String("sai");
  String s3="sai";
  String s4="sai";
  if(s1==s2)
            System.out.println("Equal..");
  else
            System.out.println("NOT Equal..");
  if(s3==s4)
            System.out.println("Equal..");
  else
            System.out.println("NOT Equal..");
 }
}
```

Note: The java. lang. String class implements *Serializable*, *Comparable* and *Char Sequence* interfaces.

Interface Char Sequence:

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

Class String:

The java. lang. String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	boolean equals(Object another)	checks the equality of string with the given object.
10	boolean isEmpty()	checks if string is empty.
11	String concat(String str)	concatenates the specified string.

12	String replace(char old, char new)	replaces all occurrences of the specified char value.
13	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
14	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
15	String[] split(String regex)	returns a split string matching regex.
16	int indexOf(int ch)	returns the specified char value index.
17	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
18	int indexOf(String substring)	returns the specified substring index.
19	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
20	String toLowerCase()	returns a string in lowercase.
21	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
22	String toUpperCase()	returns a string in uppercase.
23	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
24	String trim()	removes beginning and ending spaces of this string.
25	static String valueOf(int value)	converts given type into string. It is an overloaded method.

String concatenation:

Appending a string to another String is known as String concatenation.

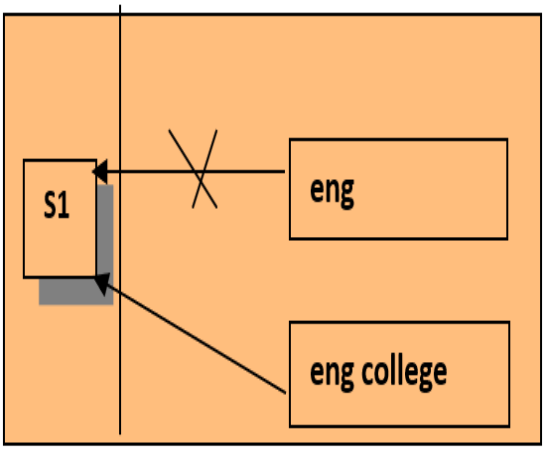
The + symbol acts as concatenation operator.

```
class SCat
{
 public static void main (String args[])
 {
  String s1="eng";
  s1=s1+" college";
  System.out.println(s1);
 }
}
```

We said that String objects immutable(contents of the object are not changed once created).

```
s1=s1+" Computers";
```

when this statement is executed a new string is created with the contents of LHS and RHS of the + operator. And the old object will eligible for garbage collection.



Example to work with String functions:

```
class StringClass
{
        public static void main(String args[])
        {
        String name="ADITYA";
        String name1="college";
        String name3="";
        String name4="Aditya College of Engineering and Technology";
        String name5="      Aditya College of Engineering and Technology       ";
        char ch=name.charAt(4);//returns the char value at the 4th index
        System.out.println(ch);
        System.out.println("string length is: "+name.length());
        System.out.println("sub string example: "+name.substring(2,4));
        System.out.println("contains example: "+name.contains("IT"));
        System.out.println("join example: "+String.join("-","02","07","2021"));
        System.out.println("equals example: "+name.equals(name1));
        System.out.println("concat example: "+name.concat(name1));
        System.out.println("replace example: "+name.replace('a','A'));
```

```
        System.out.println("isEmpty example: "+name3.isEmpty());
        System.out.println("isEmpty example: "+name1.isEmpty());
        System.out.println("name 4 before spliting: "+name4);
        String s[]=name4.split("\\s");
        for(String word:s)
                                System.out.println(word);
        System.out.println("indexof example: "+name.indexOf('t'));
        System.out.println("tolowercase example: "+name.toLowerCase());
        System.out.println("touppercase example: "+name1.toUpperCase());
        System.out.println("name 5 is:-"+name5);
        System.out.println("trim example: "+name5.trim());
        }
}
```

Methods for Extracting Characters from Strings

The following methods are used to extract Characters from Strings

1. charAt()
2. getChars()
3. toCharArray()
4. getBytes()

Example:

```
class Extract
{
        public static void main(String args[])
        {
        String s=new String("Aditya College of Engineering and technology");
        System.out.println("character at 7th index is-"+s.charAt(7));
        System.out.println(s.length());
        char[] des=new char[44];
        s.getChars(1,21,des,0);
        System.out.println("Extracted characters-"+des);
        String s1="ABCDEF";
```

```
        byte b[]=s1.getBytes();
        for(byte a:b)
        System.out.println(a);
        String s2="Aditya";
        char[] ch=s2.toCharArray();
        for(char c:ch)
            System.out.print(c);
        }
}
```

Methods for Comparison of Strings

The following methods are used to compare Strings

1. equals()
2. compareTo()

Example:

```
class Compare
{
        public static void main(String args[])
        {
                                String s1="Aditya";
                                String s2="Sai";
                                String s3="aditya";
                                String s4="Aditya";
                                System.out.println(s1.compareTo(s4));
                                System.out.println(s1.compareTo(s2));
                                System.out.println(s2.compareTo(s1));
                                if(s1.equals(s4))
                                   System.out.println("Equals");
                                else
                                  System.out.println("Not Equals");
                                if(s1==s4)
                                   System.out.println("Equals");
```

```
                    else
                        System.out.println("Not Equals");
        }
}
```

Methods for Modifying Strings

The following methods are used to modify Strings

1. concat()
2. replace()
3. trim()
4. substring()

Example:

```
class StringClass
{
        public static void main(String args[])
        {
        String name="ADITYA";
        String name1="college";
        String name3="";
        String name4="Aditya College of Engineering and Technology";
        String name5="      Aditya College of Engineering and Technology       ";
        System.out.println("sub string example: "+name.substring(2,4));
        System.out.println("concat example: "+name.concat(name1));
        System.out.println("replace example: "+name.replace('a','A'));
        System.out.println("trim example: "+name5.trim());
        }
}
```

Methods for Searching Strings

The following methods are used searching Strings

1. indexOf()
2. lastIndex()
3. charAt()
4. contains()

Example:

```
class StringClass
{
	public static void main(String args[])
	{
	String name="ADITYA";
	String name1="college";
	String name3="";
	String name4="Aditya College of Engineering and Technology";
	String name5="      Aditya College of Engineering and Technology      ";
	char ch=name.charAt(4);//returns the char value at the 4th index
	System.out.println("contains example: "+name.contains("IT"));
	System.out.println("indexof example: "+name.indexOf('t'));

	}
}
```

Data Conversion and Miscellaneous Methods

Example:

```
class StringClass
{
	public static void main(String args[])
	{
	String name="ADITYA";
	String name1="college";
	String name3="";
	String name4="Aditya College of Engineering and Technology";
	String name5="      Aditya College of Engineering and Technology      ";
	char ch=name.charAt(4);//returns the char value at the 4th index
	System.out.println(ch);
	System.out.println("string length is: "+name.length());
```

```
        System.out.println("sub string example: "+name.substring(2,4));
        System.out.println("contains example: "+name.contains("IT"));
        System.out.println("join example: "+String.join("-","02","07","2021"));
        System.out.println("equals example: "+name.equals(name1));
        System.out.println("concat example: "+name.concat(name1));
        System.out.println("replace example: "+name.replace('a','A'));
        System.out.println("isEmpty example: "+name3.isEmpty());
        System.out.println("isEmpty example: "+name1.isEmpty());
        System.out.println("name 4 before spliting: "+name4);
        String s[]=name4.split("\\s");
        for(String word:s)
                                System.out.println(word);
        System.out.println("indexof example: "+name.indexOf('t'));
        System.out.println("tolowercase example: "+name.toLowerCase());
        System.out.println("touppercase example: "+name1.toUpperCase());
        System.out.println("name 5 is:-"+name5);
        System.out.println("trim example: "+name5.trim());
        }
}
```

StringBuffer class:

1. String buffer is a peer class of string. The string buffer class creates mutable strings of flexible length which can be modified in terms of both length and content.
2. StringBuffer reserves space for 16 characters.
3. EnsureCapacity () adds double+2 spaces.
4. The following are methods of StringBuffer Class
    1. length()
    2. capacity()
    3. charAt()
    4. append()
    5. toString()
    6. reverse()
    7. ensureCapacity()
    8. setCharAt()

9. delete();
10. deleteCharAt();
11. replace();
12. substring();
13. insert()

Example 1:

```
class StringBufferDemo
{
 public static void main(String args[])
 {
  StringBuffer sb=new StringBuffer("Engineering Student");
  System.out.println("Buffer:"+sb);
  System.out.println("Length:"+sb.length());
  System.out.println("Capacity:"+sb.capacity());
  // Displaying all the characters of the string
  for(int i=0;i<s.length();i++)
  {
        System.out.println("Character at position:" +i+" is "+s.charAt(i));
  }
   // Appending at the end
   s.append(" CSE");
   System.out.println("New string is:"+s);
 }
}
```

Example 2:

```
class StringBufferDemo1
{
public static void main(String args[])
{
 StringBuffer sb1=new StringBuffer();
 StringBuffer sb2=new StringBuffer(10);
```

```
StringBuffer sb3=new StringBuffer("Prem");

System.out.println("sb3 toString:"+sb3.toString());
System.out.println("sb1 length is:"+sb1.length());
System.out.println("sb2 length is:"+sb2.length());
System.out.println("sb3 length is:"+sb3.length());
System.out.println("sb1 capacity is:"+sb1.capacity());
System.out.println("sb2 capacity is:"+sb2.capacity());
System.out.println("sb3 capacity is:"+sb3.capacity());
sb1.ensureCapacity(50);
System.out.println("sb1 capacity is:"+sb1.capacity());
StringBuffer sb4=new StringBuffer("hi are u");
System.out.println("sb4 character is:"+sb4.charAt(0));
System.out.println("sb4 character is:"+sb4.charAt(4));
sb4.setCharAt(0,'H');
sb4.setCharAt(4,'H');
sb4.reverse();
System.out.println(sb4);

sb4.insert(4," Prem");
String s="Good Bye";
boolean b=true;
int i=99;
double d=99.99;

StringBuffer sb5=new StringBuffer();

sb5.append('r');
sb5.append('a');
sb5.append('j');
```

```
System.out.println(sb5);
StringBuffer sb6=new StringBuffer("Raj Jain");
sb6.insert(4," Kumar ");
System.out.println(sb6);
sb6.delete(1,3);
sb6.deleteCharAt(3);
sb6.replace(0,4,"Hello");
String s1=sb6.substring(4);
String s2=sb6.substring(0,4);
System.out.println(s1);
System.out.println(s2);
}
};
```

equals() and ==:

equals() and == performs two different operations. equals() method compares the characters inside a String object.

The == operator compares two object references to see whether they refer to the same instance.

StringBuilder class:

StringBuilder class has been added in JDK1.5 which has same features like StringBuffer class. StringBuilder class objects are also mutable as the stringBuffer Objects.

Difference:

StringBuffer is class is synchronized and StringBuilder is not.

Multithreaded Programming:

Introduction, Need for Multiple Threads Multithreaded Programming for Multi-core Processor

1. Multithreading in Java is a process of executing multiple threads simultaneously.
2. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Advantages of Java Multithreading

1. It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
2. You **can perform many operations together, so it saves time**.

3. Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

Thread Class

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

- **public void run():** is used to perform action for a thread.
- **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
- **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
- **public void join():** waits for a thread to die.
- **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
- **public int getPriority():** returns the priority of the thread.
- **public int setPriority(int priority):** changes the priority of the thread.
- **public String getName():** returns the name of the thread.
- **public void setName(String name):** changes the name of the thread.
- **public Thread currentThread():** returns the reference of currently executing thread.

- **public int getId():** returns the id of the thread.
- **public Thread.State getState():** returns the state of the thread.
- **public boolean isAlive():** tests if the thread is alive.
- **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
- **public void suspend():** is used to suspend the thread(depricated).
- **public void resume():** is used to resume the suspended thread(depricated).
- **public void stop():** is used to stop the thread(depricated).
- **public boolean isDaemon():** tests if the thread is a daemon thread.
- **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
- **public void interrupt():** interrupts the thread.
- **public boolean isInterrupted():** tests if the thread has been interrupted.
- **public static boolean interrupted():** tests if the current thread has been interrupted.

Main Thread:

- A Thread represents a separate path of execution.
- Group of statements executed by JVM one by one.

Program to find the thread used by JVM to execute the statements(Main Thread):

```
class ThreadName
{
   public static void main(String args[])
   {
      System.out.println("Welcome ");
      Thread t=Thread.currentThread();
      System.out.println("Current Thread is : "+t);
      System.out.println("Current Thread Name is : "+t.getName());
   }
}
```

In the above program currentThread() is a static method of Thread Class, which returns the reference of the current running thread.

Thread[main,5,main]

- Here Thread indicates that t is thread class object.
- The First main indicates the name of the thread that is executing the current code.
- 5 the is priority of thread.

The next main indicates the thread group name to which this thread belongs.

Creation of New Threads

Creating threads in java is simple. Threads are created in the form of objects that contain a method called run(). The run() is heart and soul of any thread. The whole code that constitutes a new thread is written inside the run() method.

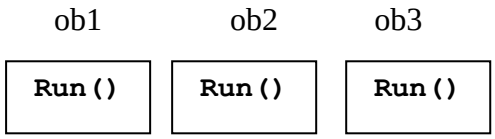
Threads can be created by the user in two ways

1. By extending Thread class
2. By implementing Runnable interface

Creating thread by extending Thread class

Syntax:-

```
class <class name> extends Thread
{
  public void run()
  {
    ----------
    ---------- override run() method
    ----------
  }
}
```



Note:

- Methods of Thread class are inherited in our own class, so every object created for our class is a thread.
- Every object(thread) has its own start(),run() and many more methods.
- Whenever start() method is called run() is excuted.
- The run() method is just like your main() method where you can declare variables, use other classes, and can call other methods.
- The run() is an entry point for another thread. This thread will end when run() returns.

Example:

```
class NewThread1 extends Thread
{
 public void run()
 {
  try
  {
  for(int i=1;i<=5;i++)
  {  System.out.println(Thread.currentThread().getName()+" : "+i);
```

```
   Thread.sleep(1000);
  }
  }
  catch(InterruptedException e){}
  System.out.println("Child thread exiting....");
};
}
class NewThreadDemo1
{
 public static void main(String args[])throws InterruptedException
 {
  NewThread1 t1=new NewThread1();
  t1.setName("Child Thread");
  t1.start();
}
};
```

By implementing Runnable interface:

- We can't extend more than one class to our class because multiple inheritance is not possible, in such situations we can implement runnable interface.
- Runnable implementation is slightly less simple. To run a separate thread, you still need a thread instance.
- To implement Runnable interface, a class need to implement a single method called run().

Syntax: Public void run()

Example:

```
class NewThread3 implements Runnable
{
 public void run()
 {
  try
  {
  for(int i=1;i<=5;i++)
  {
   System.out.println(Thread.currentThread().getName()+" : "+i);
   Thread.sleep(1000);
  }
  }
```

```
  catch(InterruptedException e){}
  System.out.println("Child thread exiting....");
};
}
class NewThreadDemo3
{
 public static void main(String args[])throws InterruptedException
 {
  NewThread3 obj=new NewThread3();
  Thread t1=new Thread(obj);
  t1.setName("Child Thread");
  t1.start();
}
};
```

Thread Priorities:

When the threads are created and started, a 'thread scheduler' program in JVM will load them into memory and execute them. This scheduler will allot more JVM time to those thread which are having priority.

The priority numbers will change from 1 to 10.

Thread.MAX_PRIORITY – 10

Thread.MIN_PRIORITY – 1

Thread.NORM_PRIORITY - 5

Example:

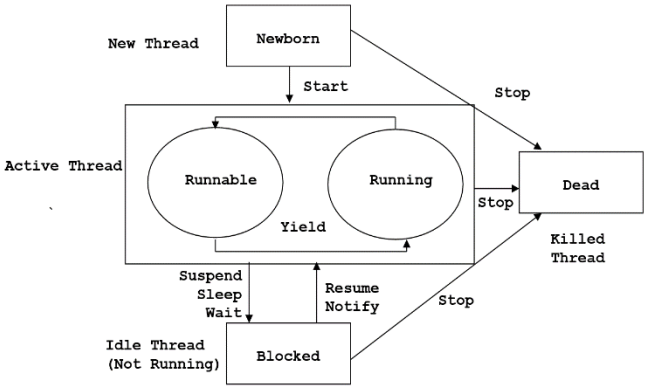
```
class MyClass extends Thread
{
 int count=0;
 public void run()
 {
  for(int i=1;i<=10000;i++)
   count++;

  System.out.println("Completed Thread:"+Thread.currentThread().getName());
  System.out.println("It's Priority:"+Thread.currentThread().getPriority());
 }
}
class Prior
{
 public static void main(String args[])
 {
  MyClass m=new MyClass();
  Thread t1=new Thread(m,"One");
  Thread t2=new Thread(m,"Two");
  t1.setPriority(2);
  t2.setPriority(Thread.NORM_PRIORITY);

  t1.start();
  t2.start();
 }
}
```

Thread States (OR) Life Cycle of the Thread:

During the life time of a thread, there aremany states it can enter.They are

1. Newborn State
2. Runnable State
3. Running State
4. Blocked State
5. Dead Sate

A thread is always in one of these five states. It can move from one state to another state via a variety of ways.

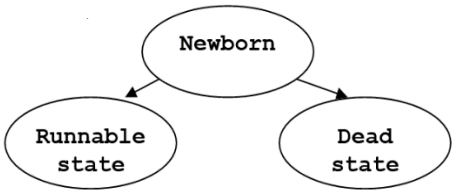


Newborn state:

When a thread object is created it is said to be in newborn state. It is not yet scheduled for running. At this stage we can do only following things.

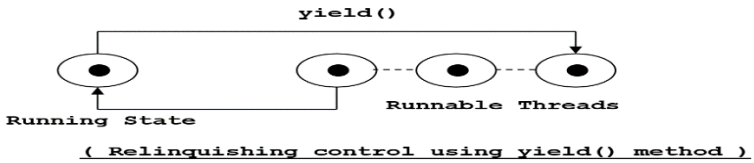
- Schedule it for running using start() method.
- Kill it using Stop() method.

If scheduled, it moves to the runnable state. If any other method is used an exception will be thrown.



Runnable state:

- Runnable state means the thread is ready for execution and is waiting for the availability of the processor. i.e The thread has joined the queue of threads that are waiting for execution.
- If all threads have equal priority, they are given time slots for execution in round robin fashion. i.e first come first serve manner.
- The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is knows as time slicing.



(Relinquishing control using yield() method)

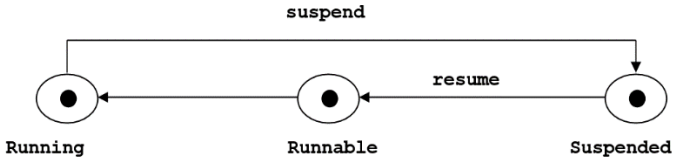
If we want a thread to relinquish control to another thread of equal priority before its turn comes, we can do so by using the yield().

Running State:

- Running means the processor has given its time to the thread for execution.
- A running thread may relinquish its control in one of the following situtations.

Suspend:

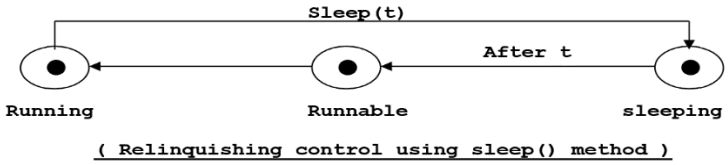
It has been suspended using suspend() method. A suspended thread can be revived using resume() method. This is useful when we want to suspend a thread for some time due to certain reason. But do not want to kill.



(Relinquishing control using suspend() method)

Sleep:

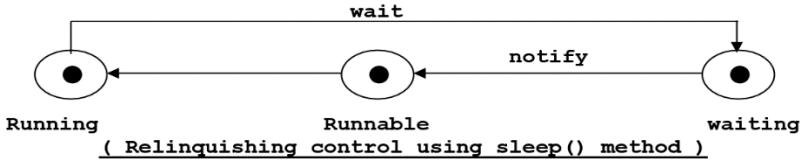
It has been made to sleep. We put a thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds. This means that the thread is out of the queue during this time period. It re-enters the runnable state as soon as time period is elapsed.



(Relinquishing control using sleep() method)

Wait:

It had been told to wait until some event occurs. This is done using wait() method. The thread can be scheduled to run again using notify() method.



(Relinquishing control using sleep() method)

Blocked State:

- A thread is said to be blocked when it is prevented from entering into the Runnable state. Subsequently the running state.

- This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements.
- A blocked thread is considered "NOT RUNNABLE" but not dead and therefore fully qualified to run again.

Dead state:

- Every thread has a life cycle.
- A running thread ends its life when it has completed executing its run() method.
- It is a natural death.
- We can kill a thread by sending stop message to it at any state thus causing a premature death.
- A thread can be killed as soon as it is born, or while it is running or even when it is in "NOT RUNNABLE"(blocked) condition.

Daemon Threads:

- Threads that work continuously without any interruption to provide services to other threads (works in the background to support the runtime environment) are called *daemon threads*.
  Eg: the clock handler thread, the idle thread, the garbage collector thread, the screen updater thread etc.
- By default a thread is not a daemon thread.
  - *setDaemon(true)* turns a thread into deamon thread.
  - *isDaemon()* to know whether a thread is Daemon or not.

Synchronization

- When a thread is already acting on an object, preventing any other thread from acting on the same object is called "Thread synchronization" or "Thread safe".
- Synchronized object is like a locked object, When a thread enters the object, it locks it, so that the next thread cannot enter till it comes out.
- This means the object is locked mutually on threads, so this object is called mutex(Mutually exclusive lock).

How to synchronize the object:

There are two ways

- Synchronized block
- Synchronized keyword

Synchronized block:

Here we can embed a group of statements of the object(inside run() method) within a synchronized block.

```
synchronized(this)
{
 Statements;
}
```

Synchronized keyword:

We can synchronize an entire method my using synchronized keyword.

synchronized void display()

{

Statements;

}

Example for Synchronized block can be executed only by one thread:

```
class Reserve implements Runnable
{
  int avaliable=1;
  int wanted;
  Reserve(int i)
   {
          wanted=i;
    }
  public void run()
   {
           synchronized(this)
            {
             System.out.println("Avaliable Berths= " +avaliable);
             if(avaliable>=wanted)
              {
               String name=Thread.currentThread().getName();
               System.out.println(wanted+"  berths reserved for "+name);
               try
               {
                 Thread.sleep(1000);
                 avaliable=avaliable-wanted;
                }
               catch(InterruptedException ie)
               { ie.printStackTrace(); }
                 }
                 else
                   System.out.println("Sorry no berths....");
          }
   }
}
class Sync
{
          public static void main(String args[])
          {
            Reserve obj=new Reserve(1);
            Thread t1=new Thread(obj);
```

```
        Thread t2=new Thread(obj);

        t1.setName("First person...");
        t2.setName("Second Person....");
        t1.start();
        t2.start();
      }
};
```

Deadlock and Race Situations

- Deadlock in Java is a condition where two or more threads are blocked forever, waiting for each other.
- This usually happens when multiple threads need the same locks but obtain them in different orders.
- Multithreaded Programming in Java suffers from the deadlock situation because of the synchronized keyword.

Example:

```
public class Example
{
    public static void main(String[] args)
    {
       final String r1 = "surya";
       final String r2 = "java";
       Thread t1 = new Thread()
       {
            public void run()
           {
               synchronized(r1)
               {
                    System.out.println("Thread 1: Locked r1");
                    try
                   {
                         Thread.sleep(100);
                   }
                   catch(exception e)
                  {
                   }
             }
           }
};
Thread t2 = new Thread()
  {
     public void run()
      {
     synchronized(r1)
      {
      System.out.println("Thread 2: Locked r1");
      try{ Thread.sleep(100);} catch(exception e) {}
```

```
    }
   }
};
t1.start();
t2.start();
}
}
```

Inter-thread Communication:

In some cases, two or more threads should communicate with each other. For example, a consumer thread is waiting for a producer to produce the data. When the producer completes the production of data, then the consumer thread should take that data and use it.

Methods used for inter -thread communication:

obj.wait() : This method makes a thread wait for the object(obj) till it receives a notification from a notify() or notifyAll() method.

obj.notify() : This method releases an object(obj) and sends a notification to a waiting thread that the object is available.

obj.notifyAll() : This method is useful to send notification to all waiting threads at once that the object is available.

Example:

```
class Communicate1
{
 public static void main(String args[])throws Exception
 {
  Producer obj1=new Producer();
  // pass producer object to Consumer
  Consumer obj2=new Consumer(obj1);
  //Create two threads and attach to producer and consumer
  Thread t1=new Thread(obj1);
  Thread t2=new Thread(obj2);
  t2.start();//consumer waits
  t1.start();//producer starts production
 }
}
```

```
class Producer extends Thread
{
 StringBuffer sb;
 Producer()
 {
  sb=new StringBuffer();
 }
 public void run()
 {
  synchronized(sb)
  {
  for(int i=1;i<=10;i++)
  {
          try{
                  sb.append(i+":");
                  Thread.sleep(100);
                  System.out.println("Appending...");
             }
             catch(Exception e){}
   }
   //Production is over, so it is sending a notification to consumer
   //thread that sb object is avaliable
   sb.notify();
  }
 }//end of run
}//end of producer class
class Consumer extends Thread
{
 Producer prod;
 Consumer(Producer p)
```

```
{
 prod=p;
}
public void run()
{
 synchronized(prod.sb)
  {
   // wait till a notification is received from producer thread. Here
   //there is no wastage of time of even a single millisecond
   try{
            prod.sb.wait();
     }
     catch(Exception e){}
     System.out.println(prod.sb);
  }//end of sync
 }//end of run
}//end of consumer class
```

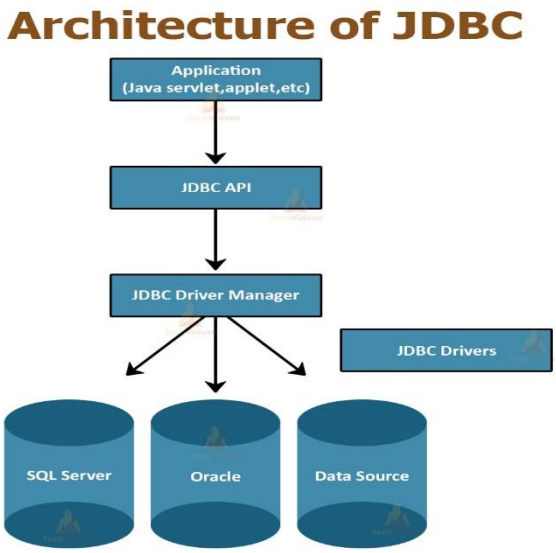
Java Database Connectivity:

Introduction

1. The term JDBC stands for Java Database Connectivity and it is a specification from Sun microsystems.
2. JDBC is an API (Application programming interface) in Java that helps users to interact or communicate with various databases.
3. The classes and interfaces of JDBC API allow the application to send the request to the specified database.
4. Using JDBC, we can write **programs required to access databases**. JDBC and the database driver are capable of accessing databases and spreadsheets.
5. Interacting with the database requires efficient database connectivity, which we can achieve using ODBC (Open database connectivity) driver. We can use this ODBC Driver with the JDBC to interact or communicate with various kinds of databases, like Oracle, MS Access, MySQL, and SQL, etc.

6. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:
    - Driver interface
    - Connection interface
    - Statement interface
    - PreparedStatement interface
    - CallableStatement interface
    - ResultSet interface
    - ResultSetMetaData interface
    - DatabaseMetaData interface
    - RowSet interface

    A list of popular classes of JDBC API are given below:
    - DriverManager class
    - Blob class
    - Clob class
    - Types class
7. We can use JDBC API to handle database using Java program and can perform the following activities:
    1. Connect to the database
    2. Execute queries and update statements to the database
    3. Retrieve the result received from the database.



Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/aditya** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and aditya is the database name. We may use any database, in such case, we need to replace the aditya with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Example:

Id	Name	City
2415	Vahida	KKD
117	Ail	RJY

Example:

```
import java.sql.*;
class MysqlCon
{
	public static void main(String args[])
	{
		try
		{
			Class.forName("com.mysql.jdbc.Driver");
			Connection con=DriverManager.getConnection(  "jdbc:mysql://localho
		st:3306/aditya","root","root");
			//here sai is database name, root is username and password
			Statement stmt=con.createStatement();
			ResultSet rs=stmt.executeQuery("select * from emp");
			while(rs.next())
```

```
                    System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.get
            String(3));
                    con.close();
            }
            catch(Exception e)
            {
                    System.out.println(e);
        }
}
}
```

JDBC Batch Processing:

- Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.
- When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.
- JDBC drivers are not required to support this feature. You should use the DatabaseMetaData.supportsBatchUpdates() method to determine if the target database supports batch update processing. The method returns true if your JDBC driver supports this feature.
- The **addBatch()** method of Statement, PreparedStatement, and CallableStatement is used to add individual statements to the batch. The **executeBatch()** is used to start the execution of all the statements grouped together.
- The **executeBatch()** returns an array of integers, and each element of the array represents the update count for the respective update statement.
- Just as you can add statements to a batch for processing, you can remove them with the **clearBatch()** method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.

Step to Batching with Statement Object

Here is a typical sequence of steps to use Batch Processing with Statement Object −

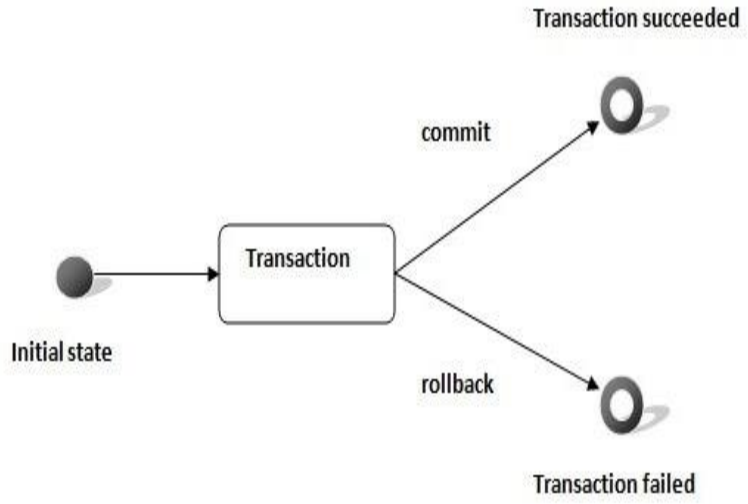
- Create a Statement object using either createStatement() methods.
- Set auto-commit to false using setAutoCommit().
- Add as many as SQL statements you like into batch using addBatch() method on created statement object.
- Execute all the SQL statements using executeBatch() method on created statement object.
- Finally, commit all the changes using commit() method.

Example:

```
// Create statement object
Statement stmt = conn.createStatement();
// Set auto-commit to false
conn.setAutoCommit(false);
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " + "VALUES(200,'ravi', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);
// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +  "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);
// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +"WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);
// Create an int[] to hold returned values
int[] count = stmt.executeBatch();
//Explicitly commit statements to apply changes
conn.commit();
```

Transaction Management in JDBC

- Transaction represents **a single unit of work**.
- The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.



In JDBC, **Connection interface** provides methods to manage transaction.

Method	**Description**
void setAutoCommit(boolean status)	It is true bydefault means each transaction is committed bydefault.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Example:

```
import java.sql.*;
class FetchRecords
{
  public static void main(String args[])throws Exception
  {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:xe","system","oracle");
        con.setAutoCommit(false);
        Statement stmt=con.createStatement();
        stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
        stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
          con.commit();
con.close();
   }}
```