UNIT V

MULTITHREADING

MULTITHREADING

- Java provides builtin support for multithreaded programming.
- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a thread, and each thread defines a separate path of execution.
- Thus, multithreading is a specialized form of multitasking.

TYPES OF MULTITASKING

Processbased Multitasking

- A process is, in essence, a program that is executing.
- Thus, processbased multitasking is the feature that allows your computer to run two or more programs concurrently.
- Ex: *Processbased multitasking enables you to run the Java compiler at the same time that you are using a text editor.*

Threadbased Multitasking

- In a threadbased multitasking environment, the thread is the smallest unit of dispatchable code.
- This means that a single program can perform two or more tasks simultaneously.
- Ex: *A text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads.*

Multitasking Threads	Multitasking Processes
Light weight tasks	Heavyweight tasks
Context switching Not Costly	Context switching – Costly
Share the same address space	Share the different address space
Less Idle time in CPU	More Idle time in CPU

The Java Thread Model

- The Java runtime system depends on threads for many things, and all the class libraries are designed with multithreading in mind.
- Java uses threads to enable the entire environment to be asynchronous.
- This helps reduce inefficiency by preventing the waste of CPU cycles.

SingleThreaded system

- Singlethreaded systems use an approach called an event loop with polling.

- In this model, a single thread of control runs in an infinite loop, polling a single event queue to decide what to do next.
- Once this polling mechanism returns with, say, a signal that a network file is ready to be read, then the event loop dispatches control to the appropriate event handler.
- Until this event handler returns, nothing else can happen in the system.
- This wastes CPU time. It can also result in one part of a program dominating the system and preventing any other events from being processed.
- In general, in a singledthreaded environment, when a thread blocks (that is, suspends execution) because it is waiting for some resource, the entire program stops running.

Java's Multithreading System

- The benefit of Java's multithreading is that the main loop/polling mechanism is eliminated.
- One thread can pause without stopping other parts of your program.
- For example, the idle time created when a thread reads data from a network or waits for user input can be utilized elsewhere.
- Multithreading allows animation loops to sleep for a second between each frame without causing the whole system to pause.
- When a thread blocks in a Java program, only the single thread that is blocked pauses.
- All other threads continue to run.

THREADS

Threads exist in several states.

Running

Ready to

Run

Suspended

Resumed

Blocked

Terminated

- A thread can be running.
- It can be ready to run as soon as it gets CPU time.
- A running thread can be suspended, which temporarily suspends its activity.
- A suspended thread can then be resumed, allowing it to pick up where it left off.
- A thread can be blocked when waiting for a resource.
- At any time, a thread can be terminated, which halts its execution immediately.
- Once terminated, a thread cannot be resumed.

The Thread Class and the Runnable Interface

- Java's multithreading system is built upon the Thread class, its methods, and its companion interface, Runnable.

- To create a new thread, your program will either extend Thread or implement the Runnable interface.

The Thread class defines several methods that help manage threads. They are…

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
Join	Wait for a thread to terminate.
Run	Entry point for the thread.
Sleep	Suspend a thread for a period of time.
Start	Start a thread by calling its run method.

The Main Thread

- When a Java program starts up, one thread begins running immediately.
- This is usually called the main thread of your program, because it is the one that is executed when your program begins.

The main thread is important for two reasons:

■ It is the thread from which other "child" threads will be spawned.

■ Often it must be the last thread to finish execution because it performs various shutdown actions.

- Although the main thread is created automatically when your program is started, it can be controlled through a Thread object.
- To do so, you must obtain a reference to it by calling the method currentThread( ), which is a public static member of Thread.
- Its general form is shown here:

static Thread currentThread( )

Program 1

```
class CurrentThreadDemo
{
public static void main(String args[])
{
Thread t = Thread.currentThread();
System.out.println("Current thread: " + t);
t.setName("My Thread");
System.out.println("After name change: "
+ t); try
{
for(int n = 5; n > 0; n)
{
System.out.println(n);
Thread.sleep(1000);
}
}
 catch (InterruptedException e)
{
System.out.println("Main thread interrupted");
}
}
}
```

Sleep()

- The sleep( ) method causes the thread from which it is called to suspend execution for the specified period of milliseconds.
  Its general form is shown here:

static void sleep(long *milliseconds) throws InterruptedException*

Creating a Thread

- You create a thread by instantiating an object of type Thread.
- Java defines two ways in which this can be accomplished:

■ You can implement the Runnable interface.

■ You can extend the Thread class, itself.

Implementing Runnable

- The easiest way to create a thread is to create a class that implements the Runnable interface.
- You can construct a thread on any object that implements Runnable.
- To implement Runnable, a class need only implement a single method called run( ), which is

declared like this:
public void run( )

- run( ) establishes the entry point for another, concurrent thread of execution within your program.
- This thread will end when run( ) returns.

Thread defines several constructors. The one that we will use is:

Thread(Runnable threadOb, String threadName)

- In this constructor, threadOb is an instance of a class that implements the Runnable interface.
- This defines where execution of the thread will begin. The name of the new thread is specified by threadName.
- After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread.
- In essence, start( ) executes a call to run( ).
- The start( ) method is shown here:

void start( )

Extending Thread

- The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
- The extending class must override the run( ) method, which is the entry point for the new thread.
- It must also call start( ) to begin execution of the new thread.
- Here is the preceding program rewritten to extend Thread

Program 3

```
class NewThread extends Thread
{ NewThread() {
// Create a new, second thread
super("Demo Thread");
System.out.println("Child thread: " +
this); start(); // Start the thread
}
// This is the entry point for the second
thread. public void run() {
try {
for(int i = 5; i > 0; i) {
System.out.println("Child Thread: " +
i); Thread.sleep(500);
}
} catch (InterruptedException e)
 {
System.out.println("Child interrupted.");
}
System.out.println("Exiting child thread.");
}
}
class ExtendThread {
public static void main(String args[]) { new NewThread(); // create a new
thread try {
for(int i = 5; i > 0; i) { System.out.println("Main Thread: " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) { System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

Creating Multiple Threads

- Till now we have seen a Main Thread & one Child Thread.
- It is also possible to create multiple Threads.
- The following program demonstrates creation of multiThreads.

Program 4

```
class NewThread implements Runnable
 {
String name; // name of thread Thread t;
NewThread(String threadname)
{
name = threadname;
t = new Thread(this, name); System.out.println("New thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for thread. public void run() {
try {
for(int i = 5; i > 0; i) { System.out.println(name )
}
System.out.println(name + " exiting.");
}
}
```

```
class MultiThreadDemo
 {
public static void main(String args[]) {
new NewThread("One"); // start
threads new NewThread("Two");
new
NewThread("Three"); try
{
// wait for other threads to
end Thread.sleep(10000);
} catch (InterruptedException e)
 {
 System.out.println("Main thread
Interrupted");
}
System.out.println("Main thread exiting.");
}
}
```

Using isAlive( ) and join( )

- How can one thread know when another thread has ended?
- Fortunately, Thread provides a means by which you can answer this question.
- Two ways exist to determine whether a thread has finished.

Alive()

- First, you can call isAlive( ) on the thread.
- This method is defined by Thread, and its general form is shown here:

  final boolean isAlive( )

- The isAlive( ) method returns true if the thread upon which it is called is still running. It returns false otherwise.
- Alive is occasionally used.

Join()

- The method that you will more commonly use to wait for a thread to finish is called join( ), shown here:

  final void join( ) throws InterruptedException

- This method waits until the thread on which it is called terminates.
- Its name comes from the concept of the calling thread waiting until the specified thread *joins it.*

- Program 5

```
class NewThread implements Runnable
{
 String name; // name of thread
Thread t;
NewThread(String threadname)
{ name = threadname;
t = new Thread(this, name);
System.out.println("New thread: " +
t); t.start(); // Start the thread
}
// This is the entry point for
thread. public void run() {
try {
for(int i = 5; i > 0; i) {
System.out.println(name + ": " +
i);


Thread.sleep(1000);
}
} catch (InterruptedException e)
 { System.out.println(name + "
interrupted.");
}
System.out.println(name + " exiting.");
}
}
class DemoJoin {
public static void main(String args[]) {
NewThread ob1 = new NewThread("One");
NewThread ob2 = new NewThread("Two");
NewThread ob3 = new
NewThread("Three");
System.out.println("Thread One is alive: "
+ ob1.t.isAlive());
System.out.println("Thread Two is alive:
" + ob2.t.isAlive());
System.out.println("Thread Three is alive:
" + ob3.t.isAlive());
// wait for threads to
finish try {
System.out.println("Waiting for threads to
finish."); ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch (InterruptedException e) {
System.out.println("Main thread
Interrupted");
}
System.out.println("Thread One is alive: "
+ ob1.t.isAlive());
System.out.println("Thread Two is alive:
" + ob2.t.isAlive());
System.out.println("Thread Three is alive:
" + ob3.t.isAlive());
System.out.println("Main thread exiting.");
}
}
```

Thread Priorities

- Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run.
- A higherpriority thread can also preempt a lowerpriority one.
- For Example, when a lowerpriority thread is running and a higherpriority thread resumes (from sleeping or waiting on I/O, for example), it will preempt the lowerpriority thread.
- To set a thread's priority, use the setPriority( ) method, which is a member of Thread. This is its general form:

  final void setPriority(int level)
- The value of level must be within the range MIN_PRIORITY and MAX_PRIORITY.
- Currently, these values are 1 and 10, respectively.
- To return a thread to default priority, specify NORM_PRIORITY, which is currently 5.
- These priorities are defined as final variables within Thread.

Program 6

```
class clicker implements Runnable
{ int click = 0;
Thread t;
```

```
private volatile boolean running =
true; public clicker(int p) {
t = new
Thread(this);
t.setPriority(p);
}
public void run()
{ while (running)
{ click++;
}
}
public void stop()
{ running = false;
}
public void start()
{ t.start();
}
}
class HiLoPri {
public static void main(String args[]) {
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
clicker hi = new clicker(Thread.NORM_PRIORITY + 2);
clicker lo = new clicker(Thread.NORM_PRIORITY  2);
lo.start();

hi.start()
; try {
Thread.sleep(10000);
} catch (InterruptedException e) {
System.out.println("Main thread
interrupted.");
}
lo.stop();
hi.stop();
//    Wait    for    child    threads    to
terminate. try {
hi.t.join();
lo.t.join();
} catch (InterruptedException e) {
System.out.println("InterruptedException
caught");

}
System.out.println("Lowpriority thread: " +
lo.click); System.out.println("Highpriority thread: "
+ hi.click);
}
}
```

Synchronization

- When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called synchronization.
- Key to synchronization is the concept of the monitor (also called a *semaphore).*

Monitor

- A monitor is an object that is used as a mutually exclusive lock, or mutex.
- Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- These other threads are said to be waiting for the monitor.
- A thread that owns a monitor can reenter the same monitor if it so desires.

Java implements synchronization through language elements in either one of the two ways.

1. Using Synchronized Methods
2. The synchronized Statement

Using Synchronized Methods

```
Program 7(unsynchronized
Thread) class Callme {
void call(String msg) {
System.out.print("[" +
msg); try {
Thread.sleep(1000);
} catch(InterruptedException e)
{
System.out.println("Interrupted"
);
}
System.out.println("]");
}
}
class Caller implements Runnable
{ String msg;
Callme
target;
Thread t;
public Caller(Callme targ, String
s) { target = targ;
msg = s;
t = new
Thread(this);
t.start();
}
public void run() {

target.call(msg);
}
}
class Synch {
public static void main(String
args[]) { Callme target = new
Callme();
Caller ob1 = new Caller(target, "Hello");
Caller ob2 = new Caller(target,
"Synchronized"); Caller ob3 = new
Caller(target, "World");
// wait for threads to
end try {
ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch(InterruptedException e)
{
System.out.println("Interrupted"
);
}
}
}
```

The synchronized Statement

- Second solution is to call the methods defined by the class inside a synchronized block.
- This is the general form of the synchronized statement:

```
synchronized(object)
{
// statements to be synchronized
}
```

Program 8

```
class Callme
{
void call(String msg) {
System.out.print("[" +
msg); try {
Thread.sleep(1000);
} catch (InterruptedException e)
{
System.out.println("Interrupted"
);
}
System.out.println("]");
}
}
class Caller implements Runnable
{ String msg;
Callme
target;
Thread t;
public Caller(Callme targ, String
s) { target = targ;
msg = s;
t = new
Thread(this);
t.start();
}



// synchronize     calls     to
call() public void run() {
synchronized(target)   {   //   synchronized
block target.call(msg);
}
}
}
class Synch1 {
public static void main(String
args[]) { Callme target = new
Callme();
Caller ob1 = new Caller(target, "Hello");
Caller ob2 = new Caller(target,
"Synchronized"); Caller ob3 = new
Caller(target, "World");
// wait for threads to end
try {
ob1.t.join()
;
ob2.t.join()
;
ob3.t.join()
;
} catch(InterruptedException e)
{
System.out.println("Interrupted"
);
}
}
}
```

Interthread Communication

- To avoid polling, Java includes an elegant interprocess communication mechanism via the wait( ), notify( ), and notifyAll( ) methods.

  wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

notify( ) wakes up the first thread that called wait( ) on the same object.

notifyAll( ) wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

These methods are declared within Object, as shown

```
here: final void wait( ) throws
InterruptedException final void notify( )
final void notifyAll( )
```

Program 9 (Correct Implementation of Producer & Consumer Problem)

```
class Q {
```

```
int n;
boolean valueSet =
false; synchronized int
get() { if(!valueSet)
try {
wait(
);
} catch(InterruptedException e) {
System.out.println("InterruptedException
caught");
}
System.out.println("Got: " + n);

valueSet =
false; notify();
return n;
}
synchronized void put(int n)
{ if(valueSet)
try {
wait(
);
} catch(InterruptedException e) {
System.out.println("InterruptedException
caught");
}
this.n = n;
valueSet =
true;
System.out.println("Put: " +
n); notify();
}
}
class Producer implements Runnable {
Q q;
Producer(Q q)
{ this.q = q;
new Thread(this, "Producer").start();
}
public void run()
{ int i = 0;
while(true) {
```

```
q.put(i++);
}
}
}
class Consumer implements Runnable {
Q  q;
Consumer(Q q)
{ this.q = q;
new Thread(this, "Consumer").start();
}
public void run()
{ while(true) {
q.get();
}
}
}
class PCFixed {
public static void main(String
args[]) { Q q = new Q();
new Producer(q);
new
Consumer(q);
System.out.println("Press ControlC to stop.");
}
}
```

Deadlock

- A special type of error that you need to avoid that relates specifically to multitasking is "deadlock".
- It occurs when two threads have a circular dependency on a pair of synchronized objects.

Deadlock is a difficult error to debug for two reasons:

■ In general, it occurs only rarely, when the two threads timeslice in just the right way.

■ It may involve more than two threads and two synchronized objects.

Suspending, Resuming, and Stopping Threads

- suspend( ) and resume( ), which are methods defined by Thread, to pause and restart the execution of a thread. They have the form shown below:

  final void
  suspend( )
  final void
  resume( )

- The Thread class also defines a method called stop( ) that stops a thread. Its signature is shown here:
  final void stop( )

String Handling in Java :

The String class is defined in the java.lang package and hence is implicitly available to all the programs in Java. The String class is declared as final, which means that it cannot be subclassed. It extends the Object class and implements the Serializable, Comparable, and CharSequence interfaces.

Java implements strings as objects of type String. A string is a sequence of characters. Unlike most of the other languages, Java treats a string as a single value rather than as an array of characters.

The String objects are immutable, i.e., once an object of the String class is created, the string it contains cannot be changed.

In other words, once a String object is created, the characters that comprise the string cannot be changed. Whenever any operation is performed on a String object, a new String object will be created while the original contents of the object will remain unchanged.

However, at any time, a variable declared as a String reference can be changed to point to some other String object.

Why String is immutable in Java

Though there could be many possible answer for this question and only designer of String class can answer this, I think below three does make sense

1) Imagine StringPool facility without making string immutable, its not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many reference variables , so if any one of them change the value others will be automatically gets affected i.e. lets say

```
String A = "Test"
String B = "Test"
```

Now String B called "Test".toUpperCase() which change the same object into "TEST" , so A will also be "TEST" which is not desirable.

2) String has been widely used as parameter for many java classes e.g. for opening network connection you can pass hostname and port number as string , you can pass database URL as string for opening database connection, you can open any file by passing name of file as argument to File I/O classes.

In case if String is not immutable, this would lead serious security threat , I mean some one can access to any file for which he has authorization and then can change the file name either deliberately or accidentally and gain access of those file.

3) Since String is immutable it can safely shared between many threads, which is very important for multithreaded programming.

String Vs StringBuffer and StringBuilder

String

Strings: A String represents group of characters. Strings are represented as String objects in java.

Creating Strings:

- We can declare a String variable and directly store a String literal using assignment operator.

String str = "Hello";

- We can create String object using new operator with some data.

String s1 = new String ("Java");

- We can create a String by using character array also.

char arr[] = { 'p','r','o',’g’,’r’,’a’,’m’};

- We can create a String by passing array name to it, as:

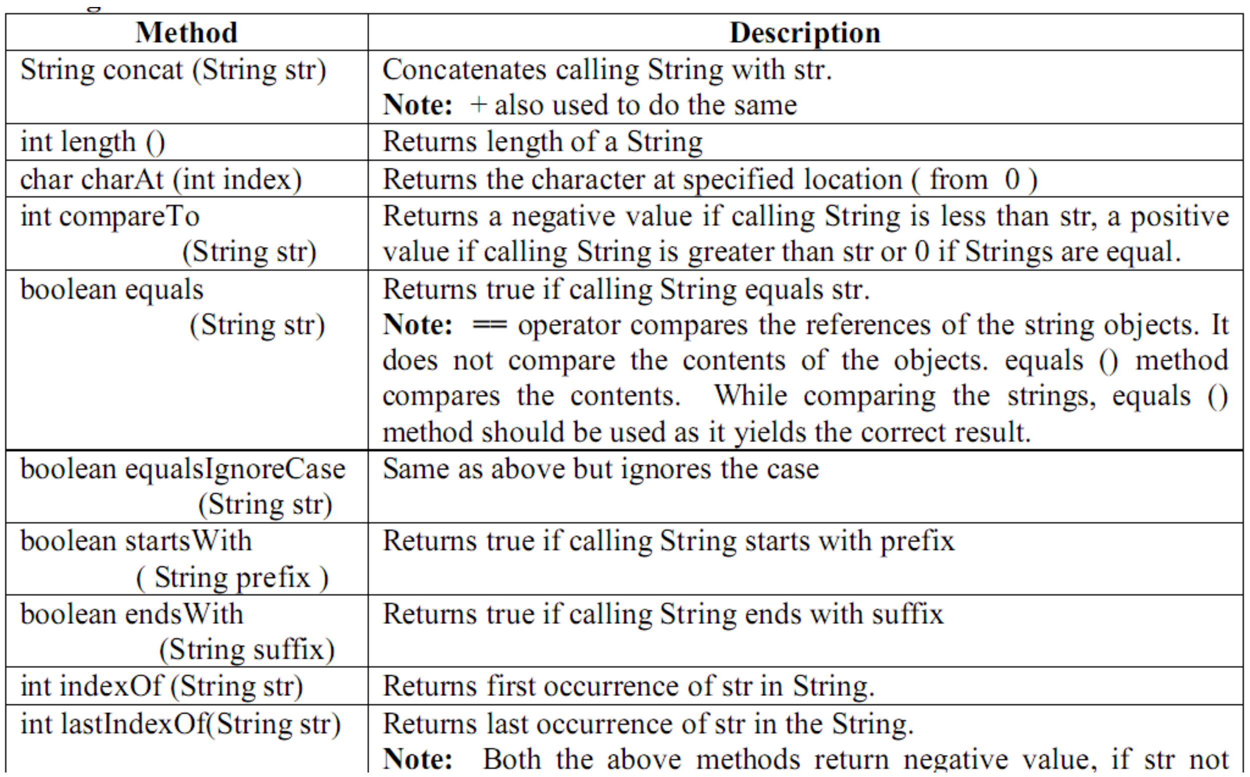
String s2 = new String (arr);

- We can create a String by passing array name and specifying which characters we need:

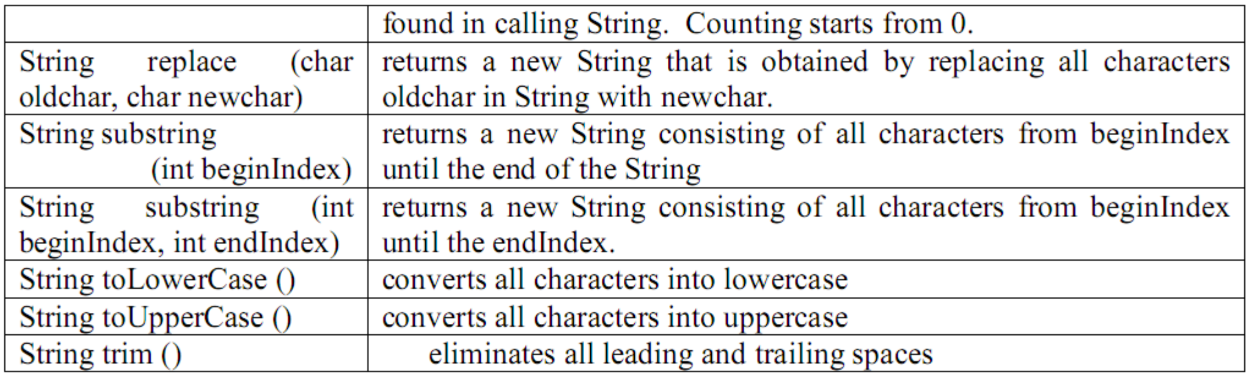
String s3 = new String (str, 2, 3);

Here starting from 2nd character a total of 3 characters are copied into String s3.

String Class Methods:



Method	Description
String concat (String str)	Concatenates calling String with str. **Note:** + also used to do the same
int length ()	Returns length of a String
char charAt (int index)	Returns the character at specified location (from 0)
int compareTo (String str)	Returns a negative value if calling String is less than str, a positive value if calling String is greater than str or 0 if Strings are equal.
boolean equals (String str)	Returns true if calling String equals str. **Note:** == operator compares the references of the string objects. It does not compare the contents of the objects. equals () method compares the contents. While comparing the strings, equals () method should be used as it yields the correct result.
boolean equalsIgnoreCase (String str)	Same as above but ignores the case
boolean startsWith (String prefix)	Returns true if calling String starts with prefix
boolean endsWith (String suffix)	Returns true if calling String ends with suffix
int indexOf (String str)	Returns first occurrence of str in String.
int lastIndexOf(String str)	Returns last occurrence of str in the String. **Note:** Both the above methods return negative value, if str not



	found in calling String. Counting starts from 0.
String replace (char oldchar, char newchar)	returns a new String that is obtained by replacing all characters oldchar in String with newchar.
String substring (int beginIndex)	returns a new String consisting of all characters from beginIndex until the end of the String
String substring (int beginIndex, int endIndex)	returns a new String consisting of all characters from beginIndex until the endIndex.
String toLowerCase ()	converts all characters into lowercase
String toUpperCase ()	converts all characters into uppercase
String trim ()	eliminates all leading and trailing spaces

String represents a sequence of characters. It has fixed length of character sequence. Once a string object has been created than we can't change the character that comprise that string. It is immutable. This allows String to be shared. String object can be instantiated

String str=new String(“Stanford“);
Str +=”Lost”;

Accessor methods:
length(),
charAt(i),
getBytes(),
getChars(istart,iend,gtarget[],itargstart),
split(string,delim),
toCharArray(),

Modifier methods:
concat(g),
replace(cWhich, cReplacement),
toLowerCase(),
toUpperCase(),
trim().

Boolean test methods:
contentEquals(g),
endsWith(g),
equals(g),
equalsIgnoreCase(g), matches(g),

Modifying a String

The String objects are immutable. Therefore, it is not possible to change the original contents of a string. However, the following String methods can be used to create a new copy of the string with the required modification:

substring()

The substring() method creates a new string that is the substring of the string that invokes the method. The method has two forms:

public String substring(int startindex)
public String substring(int startindex, int endindex)

where, startindex specifies the index at which the substring will begin and endindex specifies the index at which the substring will end. In the first form where the endindex is not present, the substring begins at startindex and runs till the end of the invoking string.

Concat()

The concat() method creates a new string after concatenating the argument string to the end of the invoking string. The signature of the method is given below:

public String concat(String str)

replace()

The replace() method creates a new string after replacing all the occurrences of a particular character in the string with another character. The string that invokes this method remains unchanged. The general form of the method is given below:

public String replace(char old_char, char new_char)

trim()

The trim() method creates a new copy of the string after removing any leading and trailing whitespace. The signature of the method is given below:

public String trim(String str)

toUpperCase()

The toUpperCase() method creates a new copy of a string after converting all the lowercase letters in the invoking string to uppercase. The signature of the method is given below:

public String toUpperCase()

toLowerCase()

The toLowerCase() method creates a new copy of a string after converting all the uppercase letters in the invoking string to lowercase. The signature of the method is given below:

public String toLowerCase()

Searching Strings

The String class defines two methods that facilitate in searching a particular character or sequence of characters in a string. They are as follows:

IndexOf()

The indexOf() method searches for the first occurrence of a character or a substring in the invoking string. If a match is found, then the method returns the index at which the character or the substring first appears. Otherwise, it returns -1.

The indexOf() method has the following signatures:

public int indexOf(int ch)
public int indexOf(int ch, int startindex)
public int indexOf(String str)
public int indexOf(String str, int startindex)

lastIndexOf()

The lastIndexOf() method searches for the last occurrence of a character or a substring in the invoking string. If a match is found, then the method returns the index at which the character or the substring last appears. Otherwise, it returns –1.

The lastIndexOf() method has the following signatures:

public int lastIndexOf(int ch)
public int lastIndexOf (int ch, int startindex)
public int lastIndexOf (String str)
public int lastIndexOf (String str, int startindex)

Program : Write a program using some important methods of String class.

```
// program using String class methods
class StrOps
{   public static void main(String args [])
{  String str1 = "When it comes to Web programming, Java is #1.";
String str2 = new String (str1);
String str3 = "Java strings are powerful.";
int result, idx;    char ch;
System.out.println ("Length of str1: " + str1.length ());
 // display str1, one char at a time.
for(int i=0; i < str1.length(); i++)
System.out.print (str1.charAt (i));
System.out.println ();
if (str1.equals (str2) )
System.out.println ("str1 equals str2");
else
System.out.println ("str1 does not equal str2");
if (str1.equals (str3) )
 System.out.println ("str1 equals str3");
else
System.out.println ("str1 does not equal str3");
result = str1.compareTo (str3);
if(result == 0)

System.out.println ("str1 and str3 are equal");

else if(result < 0)

System.out.println ("str1 is less than str3");

else

System.out.println ("str1 is greater than str3");

str2 = "One Two Three One";     // assign a new string to str2

idx = str2.indexOf ("One");

System.out.println ("Index of first occurrence of One: " + idx);

idx = str2.lastIndexOf("One");

System.out.println ("Index of last occurrence of One: " + idx);

}

}
```

StringBuffer

StringBuffer: StringBuffer objects are mutable, so they can be modified. The methods that directly manipulate data of the object are available in StringBuffer class.

Creating StringBuffer:

- We can create a StringBuffer object by using new operator and pass the string to the object, as: StringBuffer sb = new StringBuffer ("Kiran");
- We can create a StringBuffer object by first allotting memory to the StringBuffer object using new operator and later storing the String into it as:

StringBuffer sb = new StringBuffer (30);

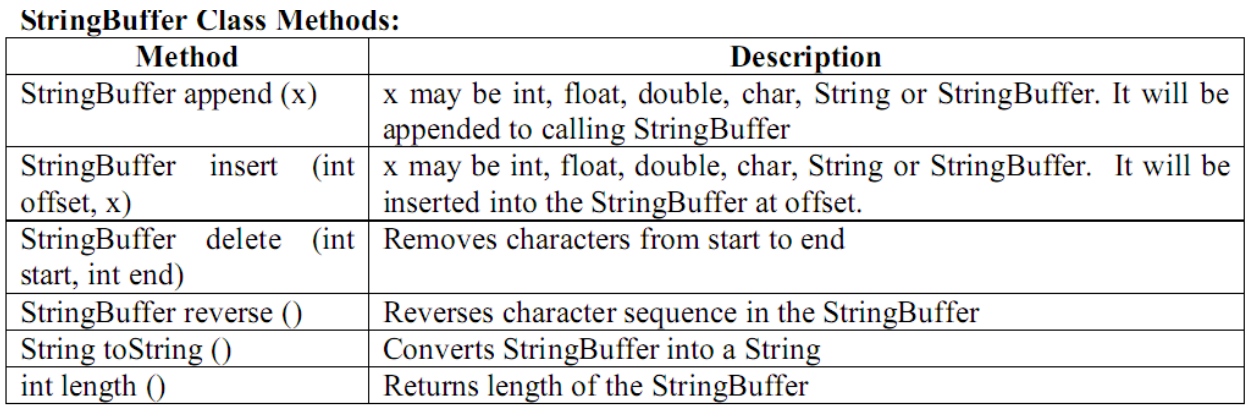
In general a StringBuffer object will be created with a default capacity of 16 characters. Here, StringBuffer object is created as an empty object with a capacity for storing 30 characters. Even if we declare the capacity as 30, it is possible to store more than 30 characters into StringBuffer.

To store characters, we can use append () method as:

```
Sb.append (“Kiran”);
```

This represents growable and writeable character sequence. It is mutable in nature. StringBuffer are safe to be used by multiple thread as they are synchronized but this brings performance penalty.
It defines 3-constructor:
• StringBuffer(); //initial capacity of 16 characters
• StringBuffer(int size); //The initial size
• StringBuffer(String str);

```
StringBuffer str = new StringBuffer ("Stanford ");
str.append("Lost!!");
```



StringBuffer Class Methods:

Method	Description
StringBuffer append (x)	x may be int, float, double, char, String or StringBuffer. It will be appended to calling StringBuffer
StringBuffer insert (int offset, x)	x may be int, float, double, char, String or StringBuffer. It will be inserted into the StringBuffer at offset.
StringBuffer delete (int start, int end)	Removes characters from start to end
StringBuffer reverse ()	Reverses character sequence in the StringBuffer
String toString ()	Converts StringBuffer into a String
int length ()	Returns length of the StringBuffer

Program : Write a program using some important methods of StringBuffer class.

```
// program using StringBuffer class methods

import java.io.*;

class Mutable

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

```
 {  // to accept data from keyboard
 BufferedReader br=new BufferedReader (new InputStreamReader (System.in));
   System.out.print ("Enter sur name : ");
   String sur=br.readLine ( );
   System.out.print ("Enter mid name : ");
   String mid=br.readLine ( );
   System.out.print ("Enter last name : ");
   String last=br.readLine ( );
   // create String Buffer object
   StringBuffer sb=new StringBuffer ( );
   // append sur, last to sb
   sb.append (sur);
   sb.append (last);
   // insert mid after sur
   int n=sur.length ( );
   sb.insert (n, mid);
   // display full name
   System.out.println ("Full name = "+sb);
   System.out.println ("In reverse ="+sb.reverse ( ));
 }
}
```

JavaFX is a Java library and a GUI toolkit designed to develop and facilitate Internet applications, web applications, and desktop applications.

A Visual Layout Tool for JavaFX Applications

JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. Users can drag and drop UI components to a work area, modify their properties, apply style sheets, and the FXML code for the layout that they are creating is automatically generated in the background.

The result is an FXML file that can then be combined with a Java project by binding the UI to the application's logic.

JavaFX Application Structure
JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes. We need to import **javafx.application.Application** class in every JavaFX application. This provides the following life cycle methods for JavaFX application.

-
- public void init()
- public abstract void start(Stage primaryStage)
- public void stop()

in order to create a basic JavaFX application, we need to:

1. Import **javafx.application.Application** into our code.
2. Inherit **Application** into our class.
3. Override **start()** method of Application class.

How to display an image in JavaFX?

The **javafx.scene.image.Image** class is used to load an image into a JavaFX application. This supports BMP, GIF, JPEG, and, PNG formats.

JavaFX provides a class named **javafx.scene.image.ImageView** is a node that is used to display, the loaded image.

To display an image in JavaFX –

- Create a **FileInputStream** representing the image you want to load.

- Instantiate the Image class bypassing the input stream object created above, as a parameter to its constructor.
- Instantiate the ImageView class.
- Set the image to it by passing above the image object as a parameter to the **setImage()** method.
- Set the required properties of the image view using the respective setter methods.
- Add the image view mode to the group object.

Example

```
import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.stage.Stage;

public class ImageViewExample extends Application {

  public void start(Stage stage) throws IOException {

    //creating the image object

    InputStream stream = new FileInputStream("D:\images\elephant.jpg");

    Image image = new Image(stream);
```

```
      //Creating the image view
      ImageView imageView = new ImageView();
      //Setting image to the image view
      imageView.setImage(image);
      //Setting the image view parameters
      imageView.setX(10);
      imageView.setY(10);
      imageView.setFitWidth(575);
      imageView.setPreserveRatio(true);
      //Setting the Scene object
      Group root = new Group(imageView);
      Scene scene = new Scene(root, 595, 370);
      stage.setTitle("Displaying Image");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String args[]) {
      launch(args);
   }
}
```

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event.

For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

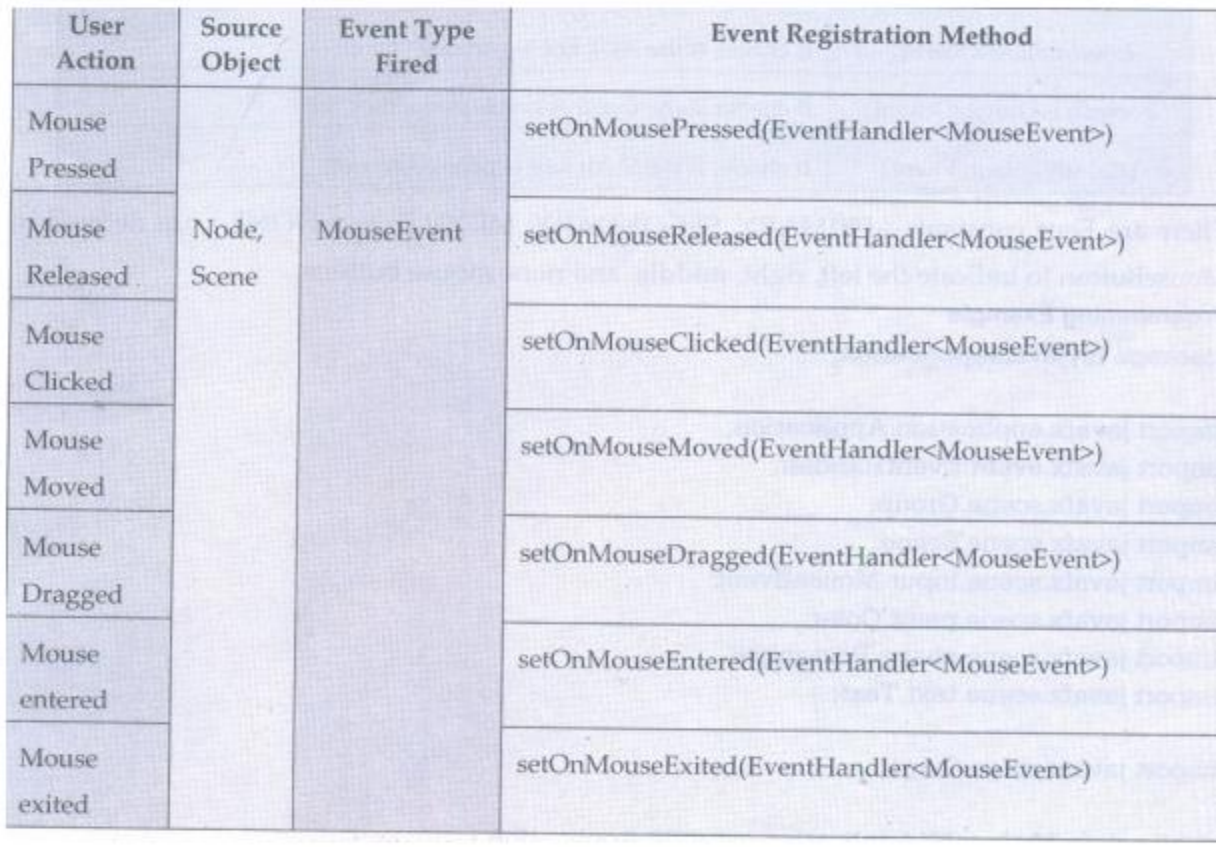
Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener

Handling Mouse Event
Mouse event is fired when the mouse button is pressed, released, clicked, moved or dragged on the node.
Following table shows the user actions and associated event handling activities -

Various mehods for MouseEvent class are enlisted in the following table

Following table shows the user actions and associated event handling activities -



User Action	Source Object	Event Type Fired	Event Registration Method
Mouse Pressed			setOnMousePressed(EventHandler<MouseEvent>)
Mouse Released	Node, Scene	MouseEvent	setOnMouseReleased(EventHandler<MouseEvent>)
Mouse Clicked			setOnMouseClicked(EventHandler<MouseEvent>)
Mouse Moved			setOnMouseMoved(EventHandler<MouseEvent>)
Mouse Dragged			setOnMouseDragged(EventHandler<MouseEvent>)
Mouse entered			setOnMouseEntered(EventHandler<MouseEvent>)
Mouse exited			setOnMouseExited(EventHandler<MouseEvent>)

Connecting to DB

What is JDBC
Driver?
JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.
For example, using JDBC drivers enable you to open database connections and to interact with it

Example to connect to the mysql database in java
For connecting java application with the mysql database, you need to follow 5 steps to perform
database connectivity.
In this example we are using MySql as the database. So we need to know following informations for 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/sonoo** 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 sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.

3.Username: The default username for the mysql database is **root**.

4. **Password:** Password is given by the user at the time of installing the mysql database.

5. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
1. create database sonoo;
2. use sonoo;
3. create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
```

JDBC-Result Sets

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result
set. The *java.sql.ResultSet* interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories −

- **Navigational methods:** Used to move the cursor around.
- **Get methods:** Used to view the data in the columns of the current row being pointed by
the cursor.
- **Update methods:** Used to update the data in the columns of the current row. The updates
can then be updated in the underlying database as well.