		ii) a value greater than zero, if s1>s2 iii) zero, if s1 is equal to s2.
5.	boolean regionMatches (int start, String str, int strstart, int numb	Matches a specific region of the argument string with a specific region of the invoking string. Eg: String s1="Hai! Hello, welcome to Java programming"; String s2="Java lab"; s1.regionMatches(23, s2, 0, 4) returns true
6.	int indexOf(char c)	It finds the index of the given character in the invoking string object. Eg: "java programming".indexOf('r') return 6
7.	int indexOf(String s)	It finds the index of the given string in the invoking string object. Eg: "java programming".indexOf("gram"). return 8
8.	int lastIndexOf(char c)	It finds the last occurance of given character in the invoking string object. Eg: "java programming".lastIndexOf(a) returns 10
9.	int lastIndexOf(String s);	It finds the last occurance of the given string in the invoking string object. Eg: "java programming".lastIndexOf("gram"); returns 8
10.	String SubString(int ind);	Used to extract a substring from the invoking string starting with given index till the end of the string. Eg: "java language".SubString(7) return "nguage"

11.	String SubString(int start, int end)	Used to extract a substring from the invoking string object starting with given start index till given end index. (60) Eg: "java language".SubString(7,11) return "ngua".
12.	Char CharAt(int pos)	It returns the character at a particular position in the invoking string
13.	String toUpperCase( )	Returns a string that has all capital letters of the invoking string.
14.	String toLowerCase( )	Returns a string that has all lower case letters of the invoking string
15.	boolean startswith(String s)	To check whether the invoking string begins with the argument string
16.	boolean endswith(String s)	To check whether the invoking string ends with the argument string

Eg:

The following program illustrates the methods of string class:

```
/* Program to demonstrate methods of string class */
class StringDemo{
    public static void main(String args[ ]){
        String s1="Java programming";
        String s2="Advanced Java";
        String s3="Java Fundamentals";
        String s4="java programming";
        System.out.println("s1.equals(s4): " +s1.equals(s4));
        System.out.println("s1.equalsIgnoreCase(s4): "
                +s1.equalsIgnoreCase(s2));
        System.out.println("Length of s2=" +s2.length());
```

```
System.out.println("s1.CompareTo("Java programming"):"
                                    +s1.CompareTo("Java programming"));
System.out.println("s1.CompareTo(s3): " +s1.CompareTo(s3));
System.out.println("s3.Compare To(s4): " +s3.CompareTo(s1));
System.out.println("s3.SubString(5): " + s3.SubString (5));
System.out.println("s4.startswith("java"):"
                                    +s4.startswith("java"));
System.out.println("s2.endswith("Java"): "
                                    +s2.endswith("Java"));
System.out.println("s3.Substring(10,16):"
                                    +s3.Substring(10,16));
System.out.println("s3.toUpperCase( ): "
                                    +s3.toUpperCase( ));
System.out.println("s2.toLaserCase( ): "+s2.toLaserCase());
System.out.println("s3.indexOf("Fun"): "+s3.indexOf("Fun"));
System.out.println("s1.regionMatches(5,84,5,7): "
                                    +s1.regionMatches(5,84,5,7));
System.out.println("s2.lastIndexOf('a'):"+s2.lastIndexOf('a'));
System.out.println("s3.CharAt(7): " +s3.CharAt(7));
}
}
```

C1

The StringBuffer class: (mutable)

* StringBuffer contains the sequence of characters which can be altered through the methods of this class.

Eg: StringBuffer s=new StringBuffer("Hello");
s = s+"World!";

* The main difference between the String and StringBuffer is, string is immutable where as StringBuffer is mutable.
* A StringBuffer object has a default capacity of 16 bytes and the capacity can be expanded.
* The following table represents some of the methods of StringBuffer class.

S.No	Method Name	Description
1.	int capacity( )	Returns the current capacity of the storage available for characters in the buffer.
2.	StringBuffer append (string str)	Appends string arguments to the buffer.
3.	StringBuffer replace (int short, int end, String str)	The characters from start to end are removed and the string str is inserted at that position.
4.	StringBuffer reverse( )	It reverses the buffer character by character.
5.	char CharAt(int ind)	Return the character at the specified index.
6.	void SetCharAt(int ind, char ch)	Sets a specified characters at the specified index
7.	deleteCharAt(int ind)	Deletes the character at the specified index
8.	delete(int start, int end)	Deletes all the character from the index start to end

```
Class StringBufferDemo{
    public static void main(String args[ ]){
        StringBuffer sb1=new StringBuffer( );
        StringBuffer sb2=new StringBuffer("Hello");
        System.out.println("Capacity of sb1="+sb1.capacity( ));
        System.out.println("Capacity of sb2="+sb2.capacity( ));
        System.out.println("After append sb1="
                +sb1.append("Honesty is the best policy"));
        System.out.println("Capacity of sb1 after append:"
                +sb1.capacity( ));
        System.out.println("sb2 after append:"+sb2.append("Welcom
        sb2.replace(9, 12, "I done");
```

```
        System.out.println("After replacement sb2= "+sb2);
        sb1.setCharAt(6,'y');
        System.out.println("sb1 after setCharAt: "+sb1);
        System.out.println("sb1 reversed: "+sb1.reverse());
        sb2.deleteCharAt(4);
        System.out.println("sb2 deleteCharAt: "+sb2);
    }
}
```

(63)

(30)

The 'StringBuilder' class:

* The StringBuilder class is introduced in java 5.0 as the substitute for StringBuffer class.
* The StringBuilder class is faster than StringBuffer.
* The methods of both the classes are same with the exception that the methods return StringBuilder objects rather than StringBuffer objects.
* The following statement shows how to create a StringBuilder object:

```
StringBuilder sb=new StringBuilder();
```

The 'StringTokenizer' class:

* The StringTokenizer class is available in java.util package.
* This class is used to pass the given text parsing is the division of text into a set of discrete parts is known as tokens.
* To use StringTokenizer, we specify an input string and a string that contains 'delimeters'.
* Delimeters are characters that separate tokens.
* There are 2 ways to create the StringTokenizer object:

1. StringTokenizer st=new StringTokenizer(String str);
2. StringTokenizer st=new StringTokenizer(String str, String delimeter);

* If no delimeter string is given, parsing is done with the default parameter i.e., widespace.
* The following table presents the some of the methods of StringTokenizer class:

S.No	Method Name	Description
1.	int countTokens( )	It returns the no. of tokens after parsing.
2.	boolean hasMoreTokens( )	Returns true if one or more tokens remain in the string and returns false if there are none.
3.	String nextToken( )	Returns the next token as a string
4.	Object nextElement( )	Returns the next token as an object

* The following program demonstrates the StringTokenizer class:

```
import java.util.StringTokenizer;
class StringTokenizerDemo{
    public static void main(String args[ ]){
        String str="595|Komali|34000|Programmer|
                    Komali@gmail.com|Gudimellanka";
        StringTokenizer st=new StringTokenizer(str,"|");
        System.out.println("No. of Tokens: "+st.countTokens( ));
        System.out.print("Tokens are:");
        while(st.hasMoreTokens( )){
            System.out.println(st.nextToken( ));
        }
    }
}
```

Output:

No. of Tokens: 6

Tokens are:

595

Komali

34000

Programmer

komali@gmail.com

Gudimellanka

Exception Handling in Java:

Error:

* An error is a human mistake done while writing the program.
* Errors are of two types:

1. Compilation error
2. Runtime error

1. Compilation Error:

* Compile time errors are due to bad syntax (or) bad typing of your code.
* The Java compiler will not produce the byte code if your program possess any such compile time error.
* Some examples of compile time error's are missing semicolon (;) at the end of the statement and miss typing of keyword using 'else' without defining a 'if' statement etc.,

2. Logical Errors/Runtime Errors:

* Runtime errors are due to bad logic or input/output failures or hardware (or) software related problems.
* This errors cannot be found at the time of compilation of the program.

*Run time errors are more dangerous than compile time errors.

*Some of the examples for runtime errors are:

—> Trying to access the 11th element of an array whose size is 10.

—> When the increment (or) decrement path is omitted in a while loop, it results in an infinite loop or never ended loop.

—> When a program is trying to access a hardware device which is crashed.

Exception:

*An exception is defined as an abnormal situation that araises when a runtime error occurs in a program.

*So exceptions are due to logical errors (or) runtime errors in your program.

*Exceptions can cause any of the following results:

—> Your program may produces wrong result (or) bad output.

—> Program execution may be terminated abruptly (or) suddenly.

—> The system on which we are running the program may get hanged.

—> The system on which the program is running may get crashed.

*To avoid any of the above consequences, we must write some additional code in your programs, so that the exceptions will be caught and handled properly.

Exception Handling in Java:

a) Exception hierarchy

The following diagram shows the exception hierarchy in Java:

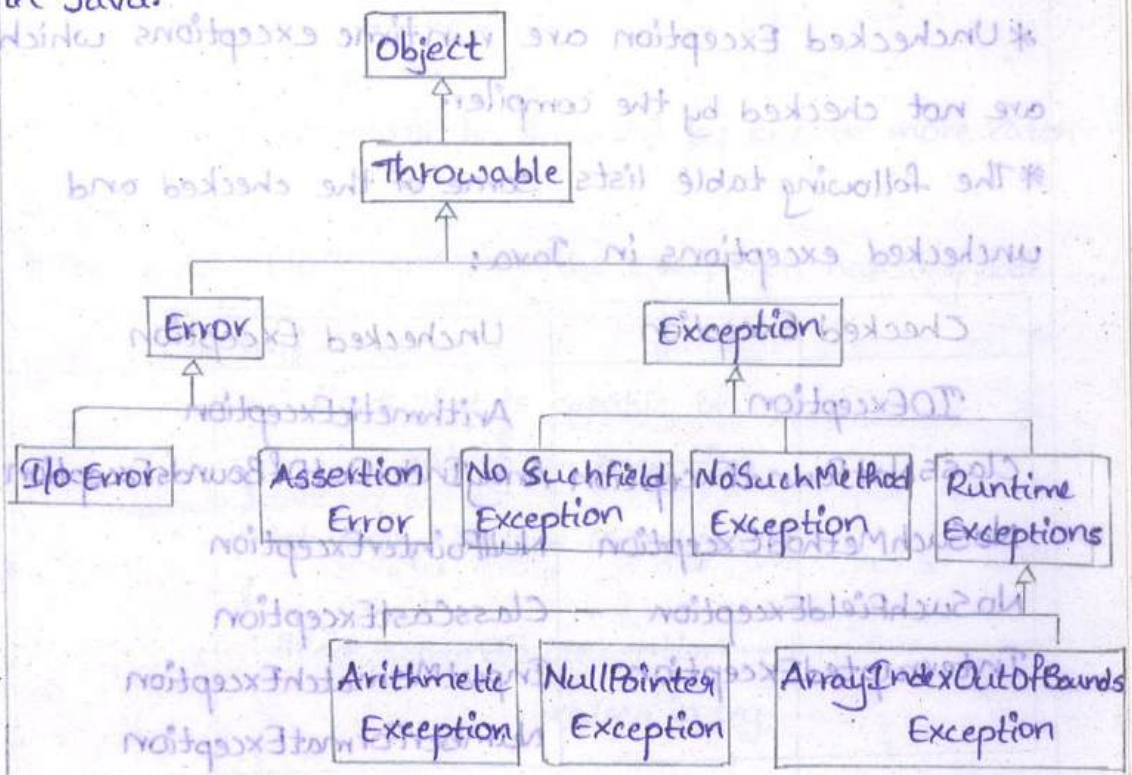


fig: Exception hierarchy

* The topmost class in the exception hierarchy is the 'java.lang.Throwable'
* Throwable class have two subclasses Error and Exception.
* All the exception(built-in & user-defined) are treated as subclass of the Exception class.
* When an exceptional condition is raised by your program Java creates and throws an object that represents the exception and throws out of it.

b) Types of Exceptions

Exceptions in Java are broadly classified into 2 categories:

1. Checked Exception
2. Unchecked Exception

1. Checked Exception :

Checked Exception are those for which the compiler check whether they have been handled or not.

2. Unchecked Exception :

* Unchecked Exception are runtime exceptions which are not checked by the compiler.
* The following table lists some of the checked and unchecked exceptions in Java:

Checked Exception	Unchecked Exception
IOException	ArithmeticException
ClassNotFoundException	ArrayIndexOutOfBoundsException
NoSuchMethodException	NullPointerException
NoSuchFieldException	ClassCastException
InterruptedException	InputMismatchException
	NumberFormatException

C) Exception Handling Techniques in Java

Java provides 5 keywords to handle exceptions.

1. try
2. catch
3. throw
4. throws
5. finally

1. try:

The try block contains the code that is capable of raising an exception. We must place the suspectable code (exception generating code) inside the try block.

```
try{
    //code that is capable of
        raising an exception
}
```


2, catch:

* Every try block must be followed by one or more catch blocks.
* The catch block contains the exception handling code.

```
try{
    //code that is capable of
        raising an exception
}
catch(ExceptionType ObjectName)
{
    //Code to handle exception
        raised in try
}
```

* When we have multiple catch blocks after 'try', it means each catch block is for catching and handling a specific type exception i.e, raised inside the try block.
* The try block may be treated as an exception generator and the catch block as the exception handler.

3, throw:

* When an exceptional situation occurs during program execution Java automatically creates and throws a corresponding exception object out of that block.
* But if you want to throw an exception out of a block explicitly then use 'throw' statement.

```
throw ExceptionObject;
```

Eg: throw new ArithmeticException;

```
try{
    if(b==0)
        throw new ArithmeticException;
}
catch(ArithmeticException e){
    e.printStackTrace();
}
```

4. throws :

* The throws keyword is used to specify that a method is capable of throwing an exception.
* This keyword is placed after the method signature and before the method body begins.

```
returntype methodName(parameters_list)throws ExceptionName
{
    // Method body
}
```

Eg:

```
public static void main(String args[])throws IOException
{
 BufferedReader br=new BufferedReader(new
                        DataInputStreamReader(System.in));
 ...
}
```

* In the above code creation of the BufferedReader object may result in raising an IOException which is a checked exception in Java.

*Since the IOException is checked, the Java compiler checks whether it is caught and handled by inside the main method or not.

*Even if you don't want to handle the exception using try, catch blocks inside the main() method, you must declare that IOException is thrown out of main() method.

5. finally:

*When an exception is raised in the program and it is not handled, the program will get terminated suddenly without executing the remaining code from the point where exception occurs.

*Sometimes we may want to execute certain block of statement at any cost irrespective of whether an exception is raised or not, whether a raised exception is caught and handled properly or not.

*For that purpose, we use the 'finally' block, the finally block generates the execution of statements inside it before the termination of the program. It doesn't matter that whether it is a normal termination (or) abnormal termination due to an exception.

*A try block must follow one (or) more catch blocks and exactly one finally block as shown below:

```
try {
    //Exception generating code
}
catch(ExceptionType1 obj){
    //Exception handling code for type1
}
catch(ExceptionType2 obj){
    //Exception handling code for type2
}
```

```
catch(ExceptionType n obj){
    // Exception handling code for type n
}
finally{
    // This code will be executed at
                                any cost
}
```

User defined Exceptions:

* Java provide an opportunity to create your own exception are known as user-defined exceptions.
* The mandatory requirement for creating user defined exception is that you have to define a class that should be a subclass of 'Exception' class.
* Your own exception can also be thrown using the 'throw' keyword as we have done with built-in exceptions.

Eg:

```
class MyException extends Exception
{
    MyException(String str){
        super(str);
    }
    public String toString(){
        return "Exception Occurred!";
    }
}
```

Assertions:

*Assertions are useful in creating reliable programs that are correct and robust.

*Assertions are boolean expressions that are used to test/validate the code.

*They are basically use during testing and development phases.

*Assertions are used by the programmer when they feel a particular condition to be true.

*Assertions are created in Java using 'assert' keyword.

*The syntax of 'assert' keyword is given below:

```
assert expression1;
        (or)
assert expression1: expression2;
```

where

expression1 is the condition to be evaluated. If the condition is evaluated to false then 'AssertionError' is thrown.

expression2 is the string which is passing to the constructor of the 'AssertionError' object.

Eg: `assert n>=0;`

(or)

`assert n>=0 : "Negative input not allowed!";`

*By default, assertions are disabled so they have to be enabled explicitly.

*Using -ea option we can enable assertions while running the program with Java command.

```
java -ea programName;
```

Eg:

```
/* Program to demonstrate assertions */
import java.util.Scanner;
class AssertionDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the level: ");
        int l=s.nextInt();
        assert l>=0:"Invalid level!";
        System.out.println("Your level is "+l);
    }
}
```

Output:

```
>javac AssertionDemo.java
>java -ea AssertionDemo
Enter the level:
5
Your level is 5
>java -ea AssertionDemo
Exception in thread "main"....AssertionExpression:
                                Invalid level!
```

UNIT-IV

Multithreading And Files

Multithreading:

* When a CPU is performing multiple programs/tasks concurrently at a same time, it is called multitasking.
* For example, a user can type a document while listening to the music and downloading a file from the internet.
* Multitasking is of two types
  1. Program/Process based multitasking
  2. Thread based multitasking
* In program based multitasking, multiple programs run simultaneously on single CPU (or) processor.
* In thread based multitasking, each program has multiple independent execution paths, each path is known as a thread.
* These threads will be executed simultaneously.
* When a program is composed of multiple threads, each thread may be dedicated to achieve an objective.
* A multithreaded program perform multiple tasks at a time.
* The thread based multitasking is also known as multithreading.

Multithreading in Java:

* Multithreading in Java is a process of executing multiple threads simultaneously.
* Creation of threads in Java is quite simple.
* We can create threads via two ways:
  1. Using Thread class
  2. Using Runnable Interface

1, Using Thread class:

* Thread is a built-in class in Java defined in java.lang package
* This class encapsulates any thread of execution.
* The constructor of thread class are given below:

1. Thread( )
2. Thread(String name)
3. Thread(ThreadGroup tg, String name)

* User can create threads as the instances of this class which contain the 'run( )' method in them.
* The run( ) method defines the task to be performed by the threads. It is automatically invoked when the thread object is create and intiated using the 'start( )' method.

Methods of Thread class:

The following table presents some important methods of Thread class.

S.No.	Method Name	Description
1,	static Thread currentThread( )	It returns the reference of the current thread i.e., being executed.
2,	static int activeCount( )	It returns the current no. of active threads.
3,	long getID( )	It returns the identifier of the thread.
4,	final String getName( )	It returns the name of the thread
5,	final void setName(String name)	Set the given string as thread name
6,	final void join( )	Wait for a thread to terminate
7,	final void join(long m)	Waits atmost for m milliseconds for a thread to terminate.

8.	final void join(long m, int n)	Waits atmost for m milliseconds & n nanoseconds for a thread to die.
9.	void run( )	Entry point for the thread
10.	final boolean isDaemon( )	Return true if the invoking thread is Daemon thread.
11.	final void setDaemon(boolean flag)	If the flag is true, then the invoking thread is set to a Daemon
12.	final boolean isAlive( )	Returns true if the thread is still running.
13.	Thread.State getState( )	Returns the current state of the thread
14.	final int getPriority( )	Returns the priority of the thread
15.	final void setPriority(int p)	It sets the priority of the invoking thread to p.
16.	void start( )	It starts a thread execution by call it's run( ) method
17.	static void sleep(long m)	Suspends a thread for a specified period of milliseconds

Eg:

```
//Program to demonstrate methods of thread class
class ThreadDemo{
    public static void main(String args[ ]){
        Thread t=Thread.currentThread( );
        System.out.println("Thread Name= "+t.getName( ));
        t.setName("My main Thread");
        System.out.println("New name of thread=" +t.getName( ));
        System.out.println("Priority of thread = " t.getPriority( ));
        t.setPriority(MAX_PRIORITY);
    } System.out.println("New Priority of thread=" +t.getPriority( ));
```

The above program demonstrates the main thread, which is created by JVM when a normal program starts execution.

Creating threads using Thread class:

We can create our own threads using Thread class by following the steps given below:

1. Define your own Thread class by extending the 'java.lang.Thread'
2. Override run() method inside your class which comprises the task done by your thread.
3. Create an object for your Thread class and invoke start() method.

Eg:

```
class MyThread extends Thread{
  public void run(){
    for(int i=1;i<=5;i++){
      System.out.println("From thread, i= "+i);
      try{
        Thread.sleep(1000);
      }
      catch(InterruptedException e){
        e.printStackTrace();
      }
    }
  }
}
class Demo{
  public static void main(String args[]){
    MyThread t=new MyThread();
    t.start();
    for(int i=1; i<=5; i++){
      System.out.println("From main, i= "+i);
      try{
```

```
            Thread.sleep(2000);
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
    }
  }
}
```

Creating threads using RunnableInterface:

*We can also create threads by implementing RunnableInterface

*RunnableInterface defined in 'java.lang' package.

*It has only one method named run()

*Creating threads using RunnableInterface involved the following steps:

1. Define your own Thread class by implementing Runnable Interface.
2. Override the run() method which comprises the task to be done by Thread instance (object).
3. Create a reference of 'Thread' class and assign your Thread object.
4. Invoke the start() method using the reference of 'Thread' class.

Because there is no start() method available in RunnableInterface.

Eg:

```
class MyThread implements Runnable{
  public void run(){
    for(int i=1;i<=5;i++){
      System.out.println("From thread, i= "+i);
      try{
        Thread.sleep(1000);
      }
```

```
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
class Demo{
    public static void main(String args[]){
        MyThread t=new MyThread();
        Thread t1=new Thread(t);
        t1.start();
        for(int i=1; i<=5; i++){
            System.out.println("From main, i= "+i);
            try{
                Thread.sleep(2000);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

(or)

```
class MyThread implements Runnable{
    MyThread(){
        Thread t=new Thread(this);
        t.start();
    }
    public void run(){
        for(int i=1; i<=5; i++){
            System.out.println("From thread, i= "+i);
            try{
                Thread.sleep(1000);
            }
```

```
                catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        }
    }
class Demo{
    public static void main(String args[]){
        MyThread t=new MyThread();
        for(int i=1; i<=5;i++){
            System.out.println("From main, i="+i);
            try{
                Thread.sleep(2000);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

Thread Lifecycle/States:

* Every Thread that we create has to go through various states during its lifetime.
* Java has Thread states that are similar to the Thread states of an operating system.
* The enumeration type 'Thread.state' is defined as part of the Thread class to represent the Thread state.
* To know the state of the current thread that is being executed we can use the getState() method.
* The following figure shows the lifecycle of a thread:

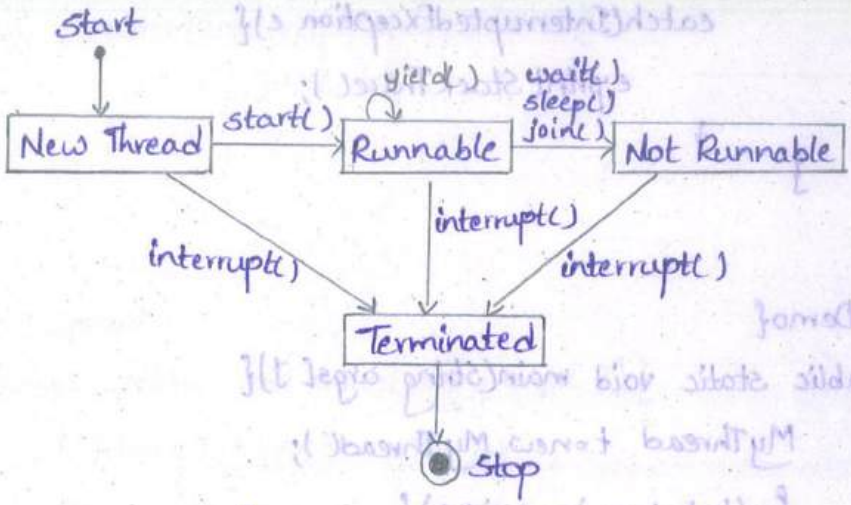


fig: State chart Diagram

*When a new thread is created and its start() method is invoked, the thread will be in Runnable state.

*The thread is said to be in not Runnable state if it is in any one of the following 3 states:

1, waiting(wait(), join())

2, Timed waiting(wait(ms), join(ms), sleep(ms))

3, Blocked(wait())

*When the thread has finished its execution it reaches the terminated state.

wait()	sleep()
1, wait() method is defined in Object class.	1, sleep() method is defined in Thread class.
2, It can be only used within the synchronized() methods or blocks.	2, It can be used outside the synchronized() methods or blocks.
3, wait() state can be terminated by calling Object.notify() method.	3, sleep state can be terminated by calling interrupt() method.
4, wait() method is used in concurrent threads access codes only.	4, sleep() method can be used whenever required.
5, It stops the current thread execution and releases the resources	5, sleep() method causes the current thread to suspend the

held by it. Now other thread can use these resources.	execution for a specified number of milliseconds.

15/5/2020 Assignment-13 18EMIA0525

Thread Priorities:

* Each thread in Java has priority which help the scheduler to decide the order of sequence of thread execution.
* When a high priority thread becomes ready for execution, the currently executing low priority thread will be stopped.
* A low priority thread cannot stop the execution of a currently running high priority thread.
* The thread class has a method setPriority() which helps to set the priority of the thread by programmers.
* The signature of the method is given below:

```
final void setPriority(int p)
```

Here 'p' specifies the value used to indicates threads priority. The value of 'p' can be any one of the constants listed below:

Constant	Value	Meaning
MIN_PRIORITY	1	Minimum Priority
NORM_PRIORITY	5	Normal Priority
MAX_PRIORITY	10	Maximum Priority

Eg:

```
t.setPriority(Thread.MIN_PRIORITY);
```

(or)

```
t.setPriority(1);
```

* If thread priority is not externally assigned by default it is said to NORM_PRIORITY i.e, 5
* A thread's current priority can be obtained by the method getPriority() of the Thread class.

* The signature of the method is given below:

`final int getPriority()`

isAlive() & join() Methods:

* The main() Thread should always be the last thread to terminate. That means all the child thread spawned out of the main() Thread should get terminated before the main().
* The execution of main() Thread can be prolonged using Thread.sleep() method.
* The time for which main() should be made to sleep cannot be estimated exactly.
* To address this problem we have two methods in Thread class i.e., isAlive() and join().
* The isAlive() method can be used to check whether the child thread is running or not.
* The signature of the method is given below:

`final boolean isAlive()`

* This method returns a boolean value. It returns 'true' if the thread is active (it has started and not stopped). It returns 'false' if the thread is either a new thread or dead thread.
* The join() method is used to wait until the thread on which it is called terminate.
* It waits for the child thread to terminate and then joins the main() Thread.

Thread Synchronization:

* There may be a chance where two (or) more threads access a common resource like file.

* Inorder to maintain consistency, the resource should be made available to only one thread at a time.
* For example, there are 2 threads, one responsible for writing a file and other from t for reading from the same file. If both the threads start concurrently both would try to access a file at a time.
* To avoid inconsistency, only one thread should be allowed to access a file at a time.
* Thread Synchronization is a mechanism in Java which allows only one thread use a resource at a time.
* Thread Synchronization is achieved in Java in 2 ways:

1. Using synchronized methods
2. Using synchronized statements

1. Synchronized methods:

* We can make a methods synchronize by using synchronized keyword as shown below:

```
class className{
    synchronized methodName()
    {
      //Method body
    }
}
```

* If more than one thread want to use the synchronized method, the system will not allow them to do so.
* The highest priority thread will be allowed to access the method first and making it inaccessable to other threads.
* Once the thread finishes the executing me the method. The system will allow other waiting threads that has highest priority to use the method.

2. Synchronized statements:

* We can synchronize a block of code by using the keyword synchronize.
* This synchronized statement must specify the object that provides the monitor lock.

```
synchronized(this)
{
    //statement
}
```

Eg: The producer consumer problem is defined as follows:

* There is a finite buffer which has some capacity.
* There are 2 threads:

1. Producer 2. consumer

* The producer has to produce items and place them into the buffer as long as the no. of items in the buffer does not exceed the capacity.
* The consumer thread removes items from the buffer as long as the buffer is not empty.

Write a Java program to synchronized producer and consumer threads.

```
import java.util.LinkedList;
import java.util.Scanner;
class PC{
    LinkedList<Integer> list=new LinkedList<>();
    int capacity;
    PC(int n){
        capacity=n;
    }
    public void produce() throws InterruptedException{
        int value=1;
```

```
            while(true){
                synchronized(this){
                    while(list.size()==capacity)
                        wait();
                    System.out.println("Producer produced item-"
                                                    +value);
                    list.add(value);
                    value++;
                    notify();
                    Thread.sleep(2000);
                }
            }
        }
        public void consume() throws InterruptedException{
            while(true){
                synchronized(this){
                    while(list.size()==0)
                        wait();
                    int val=list.removeFirst();
                    System.out.println("Consumer consumed item-"
                                                    +val);
                    notify();
                    Thread.sleep(2000);
                }
            }
        }
    }
    public class ProducerConsumer{
        public static void main(String[] args)throws InterruptedException
        {
            Scanner s=new Scanner(System.in);
            System.out.println("Enter the size of the buffer:");
            int c=s.nextInt();
            PC pc=new PC(c);
            Thread t1=new Thread(new Runnable() {
                public void run() {
                    try{
                        pc.produce();
                    }
                    catch(InterruptedException e){
```

```
                e.printStackTrace();
            }
        }
    });
    Thread t2=new Thread(new Runnable(){
        public void run(){
            try{
                pc.consume();
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    });
    t1.start();
    t2.start();
    t1.join();
    t2.join();
  }
}
```

Output:

```
Enter the size of the buffer:
3
Producer produced item-1
Producer produced item-2
Producer produced item-3
Consumer consumed item-1
Consumer consumed item-2
Producer produced item-4
Consumer consumed item-3
Producer produced item-5
Consumer consumed item-4
Producer produced item-6
Consumer consumed item-5
Producer produced item-7
```

Consumer consumed item-6
Producer produced item -8
Consumer consumed item-7
Producer produced item -9

Suspending and Resuming Threads:

* The suspend( ) method is used for suspending an executing thread temporarily.
* The resume( ) method is used for resuming the suspension of a thread.

Communication between threads:

* Sometimes the threads need to coordinate among themselves. This communication between threads while they are concurrently executing is known as inter thread communication.
* There are 3 methods which help the threads in communicating with each other:

1. wait( )
2. notify( )
3. notifyAll( )

* One important thing is that these methods are to be called from synchronized methods and synchronized statements.

1. wait( ):

It is defined in java.lang.Object class. It stops the current thread execution and releases the lock of the object.

2. notify( ):

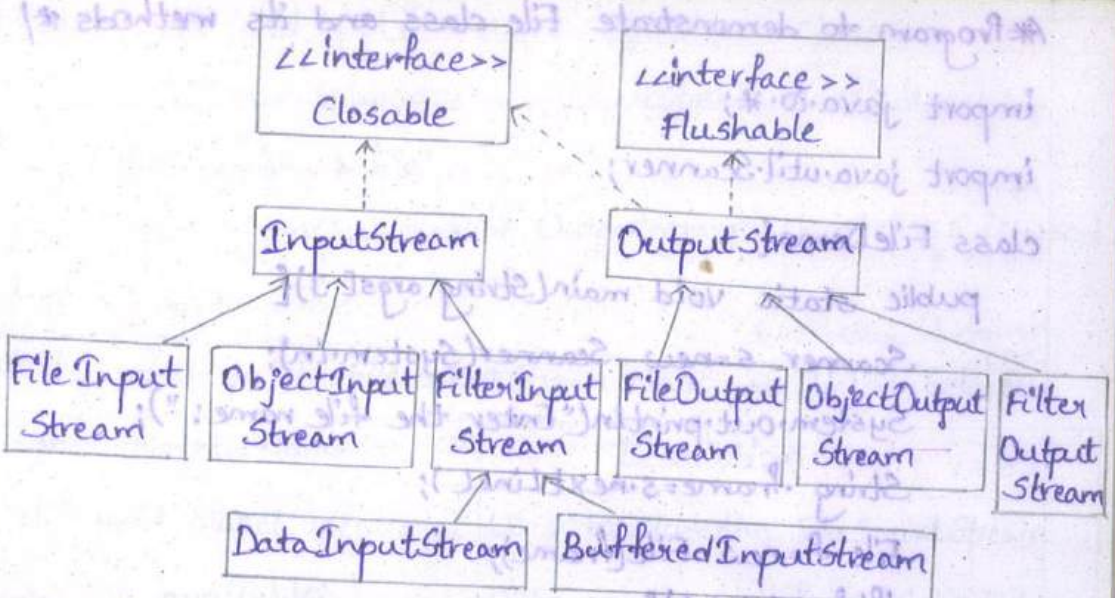
It is defined in java.lang.Object class. It is used to wake up a thread that is in waiting state.

3. notifyAll( ):

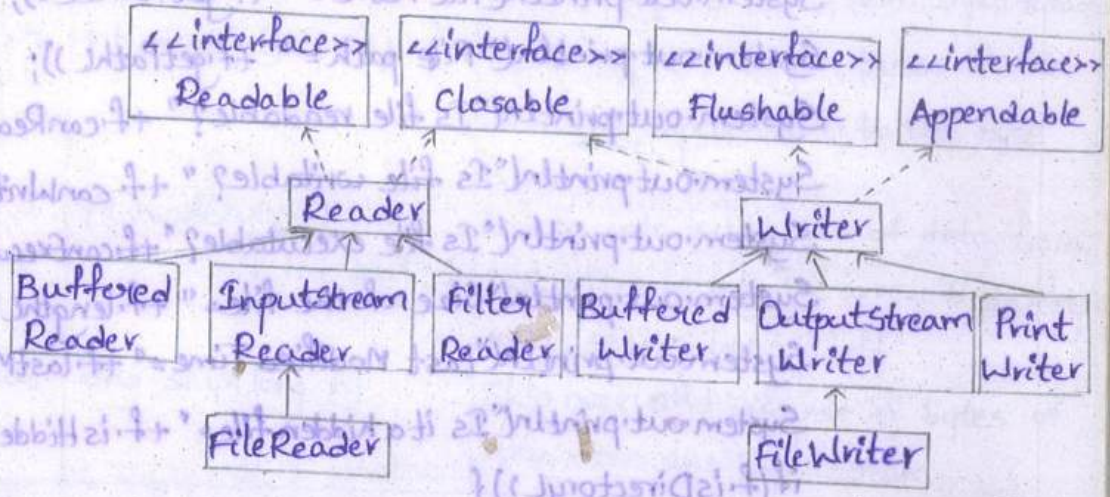
It is defined in java.lang.Object class. It is used to wake up a thread that is in waiting state.

File I/O:

* Java provides 2 pre-defined packages 'io' and 'nio' which contain classes to perform I/O operations.
* The java.io package contains classes that deal with reading and writing to console or files.
* The java.nio package contains classes that support the classes in java.io package and perform advanced operations such as buffering, memory mapping, pattern matching, locking files etc.,
* The Java I/O facility is based on streams.
* The stream is a continuous flow of data.
* The java.io package provides 2 categories of stream classes
  1. ByteStream classes
  2. CharacterStream classes
* Byte Stream classes deals with reading and writing bytes to files.
* CharacterStream classes deals with reading and writing characters to files.
* The java.io package has 2 toplevel Bytestream classes:
  1. java.io.InputStream (For reading)
  2. java.io.OutputStream (For writing)
* It also provides 2 top level CharacterStream classes:
  1. java.io.Reader (For reading)
  2. java.io.Writer (For writing)
* These 4 classes are abstract classes. So the subclasses of these classes are actually used for reading and writing data.
* The following diagram shows few classes under InputStream and OutputStream classes:



* The following figure shows few classes under Reader and Writer classes:



java.io.File class:

* The java.io package provides a class named 'File' that neither belongs to ByteStream nor to the CharacterStream used for reading or writing a file.
* This class is used to know the properties of a file like whether the file exists or not, name of the file, path of the file, whether it is hidden file etc.,
* The following program demonstrates the File class and its methods:

```
/*Program to demonstrate File class and its methods */
import java.io.*;
import java.util.Scanner;
class FileDemo{
    public static void main(String args[ ]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the file name: ");
        String fname=s.nextLine( );
        File f=new File(fname);
        if(f.exists( )){
            System.out.println("Given file exists!");
            System.out.println("File name = " +f.getName( ));
            System.out.println("File path = " +f.getPath( ));
            System.out.println("Is file readable? " +f.canRead(
            System.out.println("Is file writable? " +f.canWrite(
            System.out.println("Is file executable? " +f.canExecute(
            System.out.println("Size of the file = " +f.length( ));
            System.out.println("Last Modified Time = " +f.lastModif
            System.out.println("Is it a hidden file = " +f.isHidden(
            if(f.isDirectory( )){
                System.out.println("It is a directory");
                String l[ ]=f.list( );
                System.out.println("Contents of the directory:
                for(String n: l){
                    System.out.println(n);
                }
            }
            else{
                System.out.println("Sorry! File doesn't exists!");
            }
            f.close( );
        }
}
```

Reading and Writing Files using ByteStream classes:

* FileInputStream (Subclass of Inputstream class) class is used for reading a file.
* FileOutputStream (Subclass of OutputStream class) class is used for writing to a file.
* The methods of FileInputStream class are listed below:

S.No.	Method Name	Description
1.	void close( )	Used to close the FileInputStream
2.	int available( )	Returns the no. of remaining bytes that can be read from a file
3.	int read( )	Read a byte of data from InputStream
4.	int read(byte[ ] b)	Reads upto b.length bytes of data from the Inputstream to the byte array b.
5.	int read(byte[ ] b, int off, int len)	Reads upto 'len' bytes of data from InputStream into the array 'b' starting at offset (address) 'off'
6.	long skip(long n)	Skip over and discards 'n' bytes of data from the InputStream

* The following table shows few important methods of FileOutputStream class:

S.No	Method Name	Description
1.	void close( )	Used to close the FileOutputStream
2.	void write(int b)	Writes the specified byte 'b' to the FileOutputStream
3.	void write(byte[ ] b)	Writes bytes from the specified byte array 'b' to the FileOutputStream
4.	void write(byte[ ] b, int off, int len)	Writes 'len' bytes from the specified byte array 'b' starting at offset 'off' to the FileOutputStream

*The following program illustrates reading data from a file and prints it on the screen using FileInputStream class:

```
/*Program to read the contents from a file using ByteStream
class */

import java.io.*;
import java.util.Scanner;
Class FileReadDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the file name:");
        String fname=s.nextLine();
        File f=new File(fname);
        if(f.exists()){
            FileInputStream fis=new FileInputStream(f);
            int n=fis.available();
            byte b[]=new byte[n];
            fis.read(b);
            System.out.println("Contents of the given file are...");
            for(byte c:b){
                System.out.print((char)c);
            }
            fis.close();
        }
        else{
            System.out.println("File doesn't exists!");
        }
        f.close();
    }
}
```

* The following program illustrates writing data into a file using ByteStream class:

```
/*Program to write data into a file using ByteStream class*/
import java.io.*;
import java.util.Scanner;
class FileWriteDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the file name:");
        String fname=s.nextLine();
        File f=new File(fname);
        FileOutputStream fos=new FileOutputStream(f);
        System.out.println("Enter the content to write :");
        String str=s.nextLine();
        byte b[]=str.getBytes();
        fos.write(b);
        System.out.println("Data written Successfully!");
        fos.close();
        f.close();
    }
}
```

* The following program reads the contents of a source file and writes the same to a destination file:

```
/*Program to copy the contents of one file to another using
ByteStream classes */
import java.io.*;
import java.util.Scanner;
class FileCopyDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
```

```
System.out.println("Enter the source file name:");
String sfname=s.nextLine();
System.out.println("Enter the destination file name:");
String dfname=s.nextLine();
File f1=new File(sfname);
File f2=new File(dfname);
if(!f1.exists()){
    System.out.println("Source file doesn't exists");
}
else{
    FileInputStream fis=new FileInputStream(f1);
    FileOutputStream fos=new FileOutputStream(f2);
    int num=fis.available();
    byte b[]=new byte[num];
    fis.read(b);
    fos.write(b);
    System.out.println("Copying done!");
    fis.close();
    fos.close();
    FileInputStream fis=new FileInputStream(f2);
    System.out.println("After copying contents of
                        destination file are...");
    int c;
    while((c=fis.read())!=-1){
        System.out.print((char)c);
    }
    fis.close();
}
f1.close();
f2.close();
}
}
```

Reading and Writing files using CharacterStream classes:

* Files can be read and written using CharacterStream classes also.
* FileReader class is used to read the contents of a file.
* FileWriter class is used to write the contents from a file.
* But here reading and writing can be done in the form of characters.
* Reading can be done character by character or line by line.
* The following table presents some of the methods of FileReader class:

S.No	Method Name	Description
1.	void close( )	Used to close FileReaderStream
2.	int read( )	Read one character of data from FileReaderStream
3.	int read(char[ ] c)	Reads upto c.length characters of data from the FileReaderStream
4.	int read(char[ ] c, int off, int len)	Reads upto 'len' characters of data from FileReaderStream into the array 'c' starting at offset 'off'.
5.	long skip(long n)	Skip over and discards 'n' characters of data from the FileReader

* The following table presents some of the methods of FileWriter class:

S.No	Method Name	Description
1.	void close( )	Used to close FileWriterStream
2.	void write(int c)	Used to write one character 'c' into the FileWriterStream
3.	void write(String str)	Used to write a given string into the FileWriterStream
4.	void write(String str, int off, int len)	Write 'len' no. of characters from the string str starting at offset 'off'.
5.	void write(char[ ] c)	Write an array of characters into the FileWriterStream

*The following program illustrates reading the contents of a file using CharacterStream classes:

```
import java.io.*;
import java.util.Scanner;
class FileReadDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the file name:");
        String fname=s.nextLine();
        File f=new File(fname);
        if(!f.exists()){
            System.out.println("File doesn't exists!");
        }else{
            FileReader fr=new FileReader(f);
            int c;
            System.out.println("Contents in the given file are...");
            while((c=fr.read())!=-1){
                System.out.print((char)c);
            }
            fr.close();
        }
        f.close();
    }
}
```

*The following program illustrates writing data to a file using CharacterStream classes:

```
import java.io.*;
import java.util.Scanner;
class FileWriteDemo{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
```

```
        System.out.println("Enter the file name: ");
        String fname=s.nextLine();
        File f=new File(fname);
        FileWriter fw=new FileWriter(f);
        System.out.println("Enter the contents to write: ");
        String str=s.nextLine();

        fw.write(str);
        System.out.println("Data Written Successfully!");
        fw.close();
        f.close();
    }
}
```

* The following program illustrates reads the contents of a source file and writes the same to a destination file:

```
/*Program to copy the contents of one file to another using
CharacterStream classes */
import java.io.*;
import java.util.Scanner;
class FileCopyDemo1{
    public static void main(String args[]){
        Scanner s=new Scanner(System.in);
        System.out.println("Enter source file name: ");
        String sfname=s.nextLine();
        System.out.println("Enter destination file name: ");
        String dfname=s.nextLine();
        File f1=new File(sfname);
        File f2=new File(dfname);
        if(!f1.exists()){
            System.out.println("Source file doesn't exists!");
        }
```

```
        else{
            FileReader fr=new FileReader(f1);
            FileWriter fw=new FileWriter(f2);
            int n;
            while((n=fr.read())!=-1)
                fw.write((char)n);
            System.out.println("Copying done!");
            fw.close();
            fr.close();
        }   FileReader fr=new FileReader(f2);
            System.out.println("After copying contents in
                                destination file are...");
            int c;
            while((c=fr.read())!=-1){
                System.out.print((char)c);
            }
            fr.close();
        }
        f1.close();
        f2.close();
    }
}
```

* The following program illustrates the count the no. of lines, words and characters in a given file:

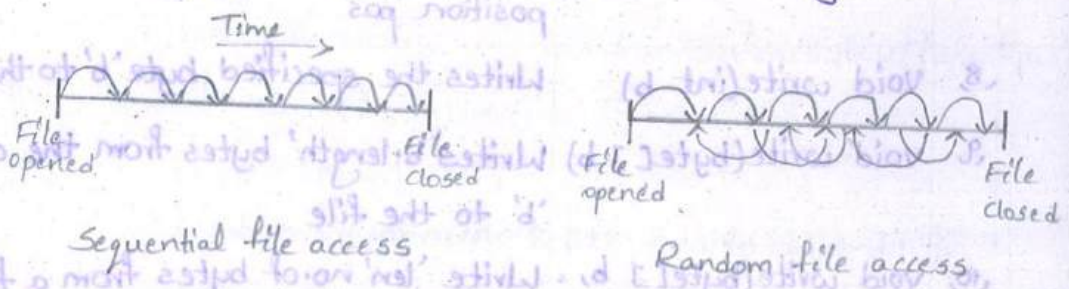
```
/* Program to count the no. of lines, words and characters
in a given file */
import java.io.*;
class FileCountDemo{
    public static void main(String args[])throws IOException{
        BufferedReader br=new BufferedReader(new
                                InputStreamReader(System.in));
```

```
            System.out.println("Enter the file name: ");
            String fname=br.readLine();
            File f=new File(fname);
            if(!f.exists()){
                System.out.println("File doesn't exists! ");
            }
            else{
                FileReader fr=new FileReader(f);
                int n, lcount=0, wcount=0, chcount=0;
                while((n=fr.read())!=-1){
                    chcount++;
                    if((char)n=='\n'){
                        wcount++;
                        lcount++;
                    }
                    if((char)n==' '){
                        wcount++;
                    }
                }
                System.out.println("No.of lines= "+lcount);
                System.out.println("No.of words= "+wcount);
                System.out.println("No.of characters= "+chcount);
                fr.close();
            }
            br.close();
        }
    }
/*Program to count the no.of lines, words and characters in a
given file*/
import java.io.*;
import java.util.Scanner;
import java.util.StringTokenizer;
```

```
class FileCountDemo{
  public static void main(String args[ ]throws IOExceptio
      Scanner s=new Scanner(System.in);
      System.out.println("Enter the file name: ");
      String fname=s.nextLine( );
      File f=new File(fname);
      if(!f.exists( )){
        System.out.println("File doesn't exists!");
      }
      else{
          BufferedReader br=new BufferedReader(new
                                  FileReader(f));
          int lcount=0, wcount=0, chcount=0;
          StringTokenizer st;
          String str;
          while((str=br.readLine( ))!=null){
              lcount++;
              st=new StringTokenizer(str);
              while(st.hasMoreTokens( )){
                  wcount++;
                  str=st.nextToken( );
                  chcount+=str.length( );
              }
          }
          System.out.println("No.of lines=" +lcount);
          System.out.println("No.of words=" +wcount);
          System.out.println("No.of characters=" +chcount
          br.close( );
      }
  }
}
```

Random access of files:

* Files can be accessed either sequentially or randomly. The following figure shows both ways of accessing a file:



Sequential file access

Random file access

* In sequential access, files can be read or written from the first byte/character till the end sequentially.
* In random access, we can read or write files from anywhere in the file.
* The 'RandomAccessfile' class in Java provides an opportunity to read or write files in a random manner.
* This class has a method named 'seek(long pos)'. That sets the file pointer at the specified position 'pos'.
* Now any read or write operation on the file will start from this marked position.
* The following table presents some of the methods of RandomAccessfile class:

S.No.	Method Name	Description
1.	void close( )	Used to close the RandomAccessfile Stream
2.	long length( )	Returns the length of the file
3.	int read( )	Read a byte of data from the file
4.	int read(byte b[ ])	Read 'b.length' of data into the array 'b'
5.	int read(byte b[ ], int off, int len)	Reads 'len' no. of bytes from a file into array 'b' starting from offset 'off'

6.	final String readLine()	Reads the next line of the text from the file.
7.	void seek(long pos)	Sets the file pointer to a specified position 'pos'
8.	void write(int b)	Writes the specified byte 'b' to the file
9.	void write(byte[] b)	Writes 'b.length' bytes from the array 'b' to the file
10.	void write(byte[] b, int off, int len)	Write 'len' no. of bytes from a file into array 'b' starting at offset 'off'

* The constructors of RandomAccessFile class are given below:

1. RandomAccessFile(String fname, String mode)
2. RandomAccessFile(File fileobj, String mode)

* Here 'mode' specifies the file opening mode.
* The 'mode' can be any one of the following:

'r' → File will be opened in read-only mode

'rw' → Read-Write mode

'rws' → File will be opened in read-write mode every updata in file contents and in metadata will be written to the storage device

'rwd' → File will be opened in read-write mode but updata in file contents in metadata will not be written to the storage device

* The following program illustrates using RandomAccessFile class for appending data to a file:

```
/* Program to illustrate random access of a file */
import java.io.*;
class RandomAccessDemo{
    public static void main(String args[]) throws IOException{
```

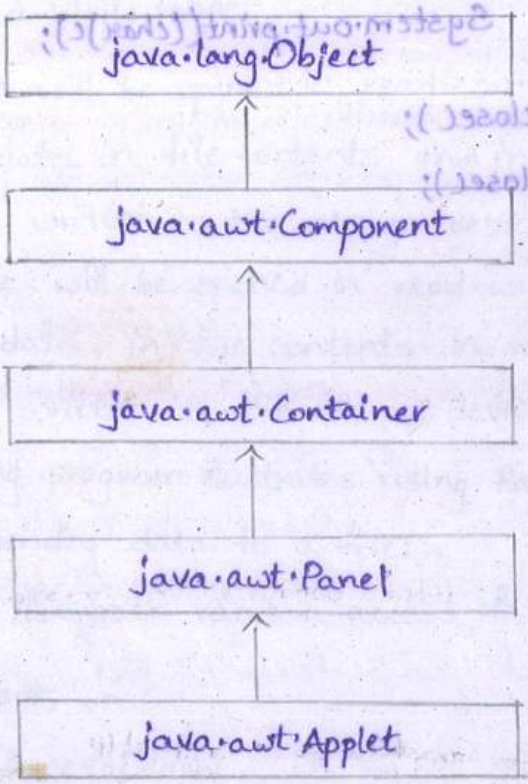
```
BufferedReader br=new BufferedReader(new
                    InputStreamReader(System.in));
System.out.println("Enter the file name:");
String fname=br.readLine();
RandomAccessFile rf=new RandomAccessFile(fname,"rw");
int num=rf.length();
rf.seek(num);
System.out.println("Enter data to append:");
String str=br.readLine();
byte b[]=str.getBytes();
rf.write(b);
System.out.println("Data Appended Successfully!");
System.out.println("After appending, contents of
                    the file are:");
rf.seek(0); //Move fp to beginning of file
int c;
while((c=rf.read())!=-1){
    System.out.print((char)c);
}
rf.close();
br.close();
}
}
```

UNIT-V

Applets and Eventhandling

Applets:

* Applets are small Java programs stored in web server and can easily be transported over the network from the server to client machine.
* The applets have the capability of displaying graphics, creating animation, playing sounds and performing other jobs.
* To execute applets on client machine, the client must use either a Java compatable web browser or a utility called "appletviewer".
* Applets can be created by using 'Applet' class of 'awt' package (or) by using 'JApplet' class of 'swing' package.
* The following figure shows the class hierarchy of Applet class:



*Using Java, we can develop 2 types of applications:

1. Command driven applications
2. Exert driven applications

*The following table shows the differences between a normal Java application and an Applet.

Java application	Applet
1. It posses the main() method and the execution starts from the main().	1. The execution of applet doesnot start at main() method as it doesnot posses main()
2. These can run on their own.	2. Applets cannot run on the own they have to be embedded inside a web page to get executed.
3. They run on command line.	3. Applets run on a web browser or an appletviewer only.
4. They have no inherit security restrictions.	4. Applets have certain security limitations.

Limitations of Applets:

1. Read/Write limitation
2. Connectivity limitation
3. Native library access limitation
4. Process limitation

Structure of an Applet:

The following is the structure of an applet:

```
/*<applet code="AppletName.class" width="300" height="250">
</applet>*/
// This is a mandatory comment line, used for running an applet
import java.awt.*;
//so as to make Graphics class available
import java.applet.Applet; //so as to make Applet class available
```

```
public class AppletName extends Applet {
    //A new applet with name 'AppletName' declared
    ≡
    public void paint(Graphics g){
        //Override paint method to display output
        ≡
    }
≡
}
```

The following is the simple applet program which displays string:

```
/* <applet code="MyApplet.class" width="300" height="250
</applet> */
import java.awt.*;
import java.applet.Applet;
public class MyApplet extends Applet{
    public void paint(Graphics g){
        g.drawString("This is my first applet!", 5,10);
    }
}
```

Save the above program as MyApplet.java

Running Applets:

There are 2 ways to run an applet.

i) Executing applet program using appletviewer tool:

a) Compile the above program using javac

```
$javac MyApplet.java
```

b) Use appletviewer tools to run the applet program.

```
$appletviewer MyApplet.java
```

ii) Executing applet program using webbrowser:

a) Compile the applet program using javac.

```
$javac MyApplet.java
```

b) Open text editor and type the following code.

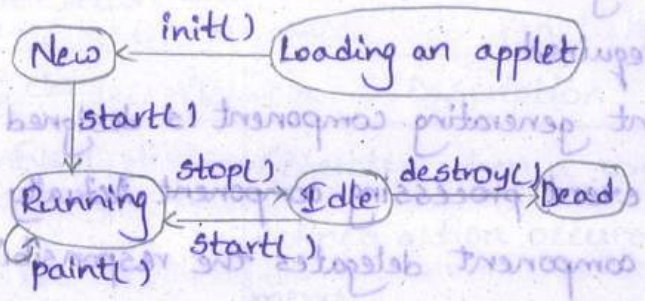
```
<html>
<head><title> My first applet </title></head>
<body>
<applet code="MyApplet.class" width="300" height="150">
</applet>
</body>
</html>
```

c) Save the file as FirstApplet.html

d) Load FirstApplet.html in a Java compatible browser (or) use appletviewer to run the html file.

```
$appletviewer FirstApplet.html
```

Applet Lifecycle:



Applet Lifecycle methods:

1. init()
2. start()
3. paint()
4. stop()
5. destroy()

Event handling in Java:

* An object resides in a particular state until it is made to transists to the other state. This transition occurs due to an event.

Eg: Pressing a key on the keyboard, moving the mouse, clicking on a button etc,

* The object which generates the event is called event generator (or) event source. If a button is pressed for a particular operation. The button is the event generator.
* The object that is responsible for performing the designated task, when the event occurs is called event handler.
* There may be more than one event handler for a single event generator.

Event delegation model:

* Event delegation model is an approach for event handling in Java. In this model a source generate events which are said to one or more listener. The listener are responsible for recovering the event, which one's receive are handled in the way required.
* The event generating component is designed differently than the event processing component. Actually, the event generating component delegates the responsibility of performing on event based action to a separate event programming component.
* this model has 3 dimensions i.e., event, event sources and event listeners.
* An event is an object that describes the state changing the sources. It may be generated directly because of interaction with the components of GUI environment.
* An event sources is an object which generates the event.
* A source can generate more than one event.
* Event listeners are the objects that get notified when

an event occurs on an event source.

java.awt.event package:

* The java.awt.event package provides the classes and interfaces for dealing with different types of events generated by AWT components.
* The java.awt.AWTEvent class is the base class of all AWT events.
* This package includes event classes, listener interfaces and adapter class which forms the basis of event handling in Java.

a) Event class:

Some of main important event classes in java.awt.event package are shown below:

S.No.	Event class	Description
1.	ActionEvent	Generated when a component-defined action occured (button, list, menu)
2.	AdjustmentEvent	When a scroll bar is adjusted
3.	ContainerEvent	When an element is add to (or) removed from the container
4.	FocusEvent	When a component gain (or) lose focus
5.	KeyEvent	When a input is received from a keyboard
6.	MouseEvent	Indicates that a mouse action occurred in a component.
7.	ComponentEvent	When a component is moved (or) changed size (or) changed visibility
8.	WindowEvent	When a window has changed its status
9.	MouseWheelEvent	When a wheel was rotated in a component

Event Sources:

* Sources of events can be either GUI components (or) any class derived from the components (such as applet). Some of the components that generate events are given below: button, checkbox, list, scroll bar, menuitem, window, radio button, text components etc.,

b) Event Listeners:

The following table gives the list of some frequently used listeners and their methods.

S.No.	Listener Interface	Methods
1,	KeyListener	keyPressed(), keyReleased(), keyTyped()
2,	MouseListener	mouseClicked(), mouseEntered(), mousePressed(), mouseReleased(), mouseExited()
3,	MouseMotionListener	mouseMoved(), mouseDragged()
4,	MouseWheelListener	mouseWheelMoved()
5,	ItemListener	itemStateChanged()
6,	ActionListener	actionPerformed()
7,	TextListener	textChanged()
8,	focusListener	focusGained(), focusLosed()
9,	WindowListener	windowActivated(), windowDeactivated(), windowOpened(), windowClosed(), windowClosing(), windowIconified(), windowDeiconified()
10,	AdjustmentListener	adjustmentValueChanged()
11,	ComponentListener	componentResized(), componentMoved(), componentHidden(), componentShown()
12,	ContainerListener	componentAdded(), componentRemoved()

* The following are the steps involved in event delegation model:

1. Implement the appropriate EventListener interface. So as to receive and process the type of event designed.

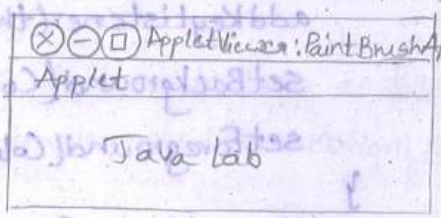
2. Register EventListener with event source using the event registration methods which have the following form:

```
addTypeListener(TypeListener t);
```

Eg: addActionListener(this);
addMouseMotionListener(this);

Example Programs:

o/p: $java PaintBrushApplet.java
$appletviewer PaintBrushApplet.java




```
import javaawt.*;
import java.applet.*;
import java.awt.event.*;
public class
/* <applet code = "PaintBrushApplet.class" width="250" height="150">
</applet> */
public class PaintBrushApplet extends Applet implements
                                    MouseMotionListener{
    public void init(){
        addMouseMotionListener(this);
        setBackground(Color.red);
    }
    public void mouseDragged(MouseEvent me){
        Graphics g=getGraphics();
        g.setColor(Color.green);
        g.fillOval(me.getX(), me.getY(), 5,5);
    }
    public void mouseMoved(MouseEvent me){
        showStatus("X=" +me.getX()+"Y=" +me.getY());
    }
}
```

Program to demonstrate handling keyevents:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
/* <applet code="KeyEventDemo.class" width="300" height="200
</applet> */
public class KeyEventDemo extends Applet implements
                                        KeyListener{
    String msg="Typed character is";
    public void init(){
        addKeyListener(this);
        setBackground(Color.green);
        setForeground(Color.blue);
    }
    public void keyPressed(KeyEvent e){
        showStatus("Key Down");
    }
    public void keyReleased(KeyEvent e){
        showStatus("Key Up!");
    }
    public void keyTyped(KeyEvent e){
        char ch=e.getKeyChar();
        msg+=ch;
        repaint();
    }
    public void paint(Graphics g){
        g.drawString(msg, 20, 30);
    }
}
```

c) Adapter classes:

↓ An interface holds only abstract methods and its implementation classes needs to an override all its metho

This is true for listener interface if are implementing particular interface you have to override all the methods defined in that interface. This will be really annoying when we require only the few methods of the listener interface and listener interface of plenty of methods which are not useful for you.

2, Adapter classes provide empty definition for all the methods of their corresponding listener interfaces. Java defines an adapter class for each an interface.

Eg: for MouseMotionListener, there is an adapter class MouseMotionAdapter.

This MouseMotionAdapter class is defined in Java as follows:

```
public class MouseMotionAdapter implements MouseMotionListener
{
    public void mouseDragged(MouseEvent e){ }
    public void mouseMoved(MouseEvent e){ }
}
```

The following table lists some adapter classes for the listener interfaces:

ListenerInterface	Adapter class	Registration Methods
ComponentListener	ComponentAdapter	addComponentListener( )
ContainerListener	ContainerAdapter	addContainerListener( )
FocusListener	focusAdapter	addFocusListener( )
KeyListener	KeyAdapter	addKeyListener( )
MouseListener	MouseAdapter	addMouseListener( )
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener( )
WindowListener	WindowAdapter	addWindowListener( )

Inner classes in Event Handling:

An inner class can be defined and instantiated inside a class. We can use inner classes in event delegation model. There are 3 types of inner classes:

1. Member classes
2. Local inner classes
3. Annoymous inner classes

* Member classes are defined inside a class like field and methods. A member class can be either static (or) instance.
* Local inner classes are defined within the code block inside a class. An instance of a local inner class exists only during the execution of the methods.
* An annoymous inner class is a local inner class which has no name such as classes are created on the fly i.e., there are created, instantiated, used and garbage collected they are done. We cannot have more than one instances. An annoymous inner class is defined as an argument to a method.

Eg:

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
/* <applet code="AnnoymousDemo.class" width="300" height="200">
</applet> */
public class AnnoymousDemo extends Applet{
    public void init() {
        addKeyListener(new KeyAdapter {
            public void keyPressed(KeyEvent e){
                showStatus("Key Pressed!");
```

```
        }
        public void keyReleased(KeyEvent e){
            showStatus("Key Released!");
        }
    });
  }
}
```

The Graphics class :

The Graphics class is an abstract base class for all Graphics contexts that allows an application to draw onto components that are realized onto various devices. This class has a no. of methods. Some of them are listed below:

1. drawArc(int x, int y, int x2, int y2, int startangle, int enlayer)
2. drawLine(int x_1, int y_1, int x_2, int y_2)
3. drawOval(int x, int y, int width, int height)
4. drawRect(int x, int y, int width, int height)
5. getColor
6. getFont
7. setColor(Color c)
8. setFont(Font f)
9. drawRoundRect(int x, int y, int w, int h, int arcw, int arch)
10. drawString(String s, int x, int y)
11. fillRect(int x, int y, int w, int h)
12. fillOval( )
13. fillArc( )
14. fillRoundRect( )

UNIT-VI

AWT and Swings

AWT:

* AWT stands for Abstract Window Toolkit. It provides many classes for building GUI based applications in Java.
* The package java.awt.Container all classes use for creating Graphical GUI, painting, images and font.
* The following is the list of some major classes of java.awt package.

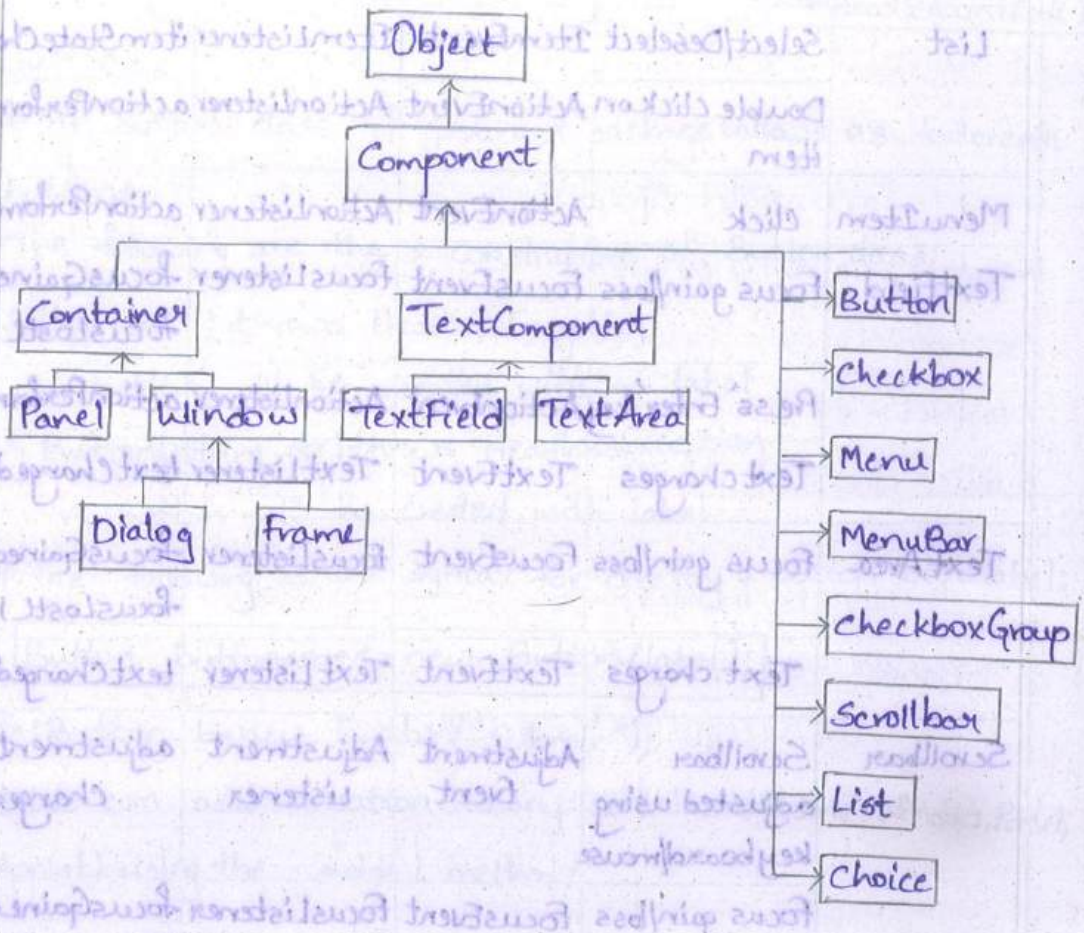
S.No.	Class	Description
1.	AWTEvent	Baseclass for all AWTEvent
2.	Button	Used to create a button
3.	Component	Baseclass for all AWT Component
4.	Container	It is a component that contains other AWT Component
5.	Color	It is used to represent colors.
6.	Checkbox	A component that can be either in 'on' or 'off' state
7.	CheckboxGroup	A group of checkboxes that work as radio buttons
8.	CheckboxMenuItem	For creating checked menu item
9.	Choice	Used to create a popup menu of choices
10.	Dialog	A toplevel window with a title & border
11.	Dimension	This object encapsulates the width and height of a component
12.	FileDialog	A Dialog windaw from which a user can select a file
13.	Font	Represents Fonts
14.	Frame	A toplevel window (container) with a

		title and a border used for containing other components
15,	Graphics	It is the abstract base class that allows an application to draw onto components.
16,	Label	Used for placing text in a container
17,	List	A scrollable list of string items
18,	Menu	A pulldown menu deployed from a menu_bar
19,	MenuBar	To create a MenuBar attached onto a frame
20,	MenuItem	All items in menu are menu items
21,	MenuShortCut	Represents a keyboard shortcut for menu items
22,	Panel	A container class
23,	Scrollbar	A class that encapsulates a scrollbar either vertical (or) horizontal
24,	ScrollPane	A container class which implements automatic horizontal / vertical scrolling for a single child component
25,	TextField	A textcomponent that allows the user to enter or edit a single line of text
26,	TextArea	A text component that allows the user to enter or edit a multiple line of text
27,	TextComponent	A super class of TextField & TextArea classes
28,	FlowLayout	Displays components in a directional flow
29,	GridLayout	Layout components in a rectangular grid
30,	GridBagLayout	It is a flexible layout manager that a line components, vertically or horizontally or along their baseline without requiring the components that be of the same size
31,	GridBagConstrains	It specifies constrains for components

32.	Window	that are layed out using GridBagLayout class Represents a toplevel window with no borders and menu bar it is the super class of Dialog & Frame class.

AWT class hierarchy:

The Component class of java.awt package is the super class of all AWT componerts.



The following table presents all the buttons controls and relevant events, event listeners and eventhandling methods.

Component	Event	Event Type	Event Listener	Method Name
Button	class	ActionEvent	ActionListener	actionPerformed()
	Focus Gain/loss	FocusEvent	focusListener	focusGained() focusLost()
Checkbox	Select/Deselect	ItemEvent	ItemListener	itemStateChanged()
Choice	Select/Deselect	ItemEvent	ItemListener	itemStateChanged()
List	Select/Deselect	ItemEvent	ItemListener	itemStateChanged()
	Double click on item	ActionEvent	ActionListener	actionPerformed()
MenuItem	click	ActionEvent	ActionListener	actionPerformed()
Textfield	Focus gain/loss	FocusEvent	focusListener	focusGained() focusLost()
	Press 'Enter' key	ActionEvent	ActionListener	actionPerformed()
	Text changes	TextEvent	TextListener	textChanged()
TextArea	focus gain/loss	FocusEvent	focusListener	focusGained() focusLost()
	Text changes	TextEvent	TextListener	textChanged()
Scrollbar	Scrollbar adjusted using keyboard/mouse	Adjustment Event	Adjustment Listener	adjustmentValue Changed()
	focus gain/loss	FocusEvent	focusListener	focusGained() focusLost()
	Keyboard event	KeyEvent	KeyListener	keyPressed(), keyReleased(), keyTyped()
	Mouseevent	MouseEvent	MouseListener	mouseClicked(), mouseEntered(), mousePressed(), mouseReleased(), mouseExited()

Frame	MouseMotion Event	MouseEvent	MouseMotion Listener	mouseMoved(), mouseDragged()
	Window events	WindowEvent	WindowListener	windowActivated(), windowDeactivated(), windowOpened(), windowClosed(), windowClosing(), windowIconified(), windowDeiconified()

Button:

*The Button class of java.awt package allows us to create buttons.

The following are the 2 constructors of Button class:

1. Button() throws HeadlessException
   A button will be created with no label
2. Button(String str) throws HeadlessException
   A button will be created with label

*The following is the syntax for creating a button with label:

```
Button buttonname=new Button(label);
```

Eg: Button b=new Button("Submit");

*We can add a button to any container (Frame, Window, Panel, Applet) using the add() method:

Eg: add(buttonname);

*We can remove a button from any container (Frame, Window, Panel, Applet) using the remove() method.

Eg: remove(buttonname)

*The methods of Button class are listed below:

1. String getLabel() - To give name for the button
2. void setLabel(String str) - To change the button name

3, void addActionListener(ActionListener al)

4, void removeActionListener(ActionListener al)

5, String getActionCommand() - the button we have clicked to perform the event

6, void setActionCommand(String str)

* The following program illustrates creation of buttons and handling their events:

```
/* Program to illustrate buttons & their events using AWT */
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
/* <applet code="ButtonDemo.class" width="300" height="250">
</applet> */
public class ButtonDemo extends Applet implements
                                        ActionListener{
    Button b1, b2, b3;
    public void init(){
        setBackground(Color.yellow);
        b1=new Button("Red");
        b2=new Button("Green");
        b3=new Button("Blue");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        add(b1);
        add(b2);
        add(b3);
    }
    public void actionPerformed(ActionEvent e){
        String str=e.getActionCommand();
```

```
        if(str.equals("Red"))
            setBackground(Color.red);
        else if(str.equals("Green"))
            setBackground(Color.green);
        else if(str.equals("Blue"))
            setBackground(Color.blue);
        repaint();
    }
}
```

Labels:

* Labels are the simplest components of awt. Labels are used to display a text string and they never generate any event.
* The constructor of Label class are:

1. Label()
2. Label(String text)
3. Label(String text, int alignment)

Note:

alignment may be Label.LEFT (or) Label.RIGHT (or) Label.CENTER

* The following are the methods of Label class:

1. void setText(String str)
2. String getText()
3. void setAlignment(int alignment)
4. int getAlignment()

Eg:

```
Label l=new Label("This is a label", Label.CENTER);
```

Checkboxes:

* Checkboxes are used as on (or) off switches. If you click on an unchecked checkbox it will get checked and if you click on checked checkbox it will get unchecked.
* The checkbox class is used to create checkboxes.
- This class has the following constructors:

1. Checkbox() – Without label
2. Checkbox(String str) – With label
3. Checkbox(String str, boolean state) – With label & default it is in false, for 'on' state give 'true', for 'off' state give 'false'
4. Checkbox(String str, CheckboxGroup g, boolean state)

Eg: Checkbox c1=new Checkbox("Java", true);

* The following are some of the methods of Checkbox class:

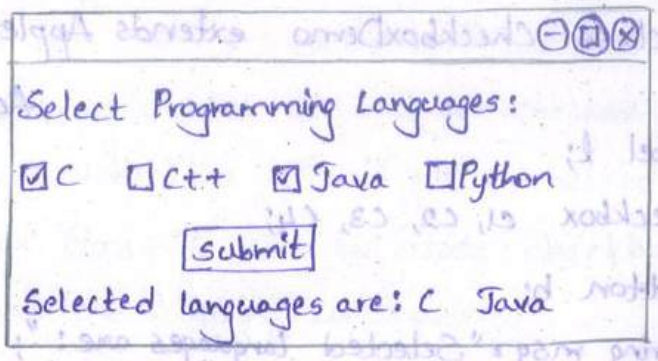
1. void addItemListener(ItemListener it)
2. void removeItemListener(ItemListener it)
3. String getLabel()
4. void setLabel(String str)
5. CheckboxGroup getCheckboxGroup()
6. void setCheckboxGroup(CheckboxGroup g)
7. boolean getState()
8. void setState(boolean b)

The following program illustrate checkboxes:

```
/* Program to illustrate checkbox of awt */
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
/* <applet code="CheckboxDemo.class" width="200" height="300"
```

```
</applet> */
public class CheckboxDemo extends Applet implements
                                        ActionListener{
    Label l;
    Checkbox c1, c2, c3, c4;
    Button b;
    String msg = "Selected languages are: ";
    public void init(){
        l=new Label("Select Programming languages: ");
        c1=new Checkbox("C");
        c2=new Checkbox("C++");
        c3=new Checkbox("Java");
        c4=new Checkbox("Python");
        b=new Button("Submit");
        b.addActionListener(this);
        add(l); add(c1); add(c2); add(c3); add(c4); add(b);
    }
    public void actionPerformed(ActionEvent e){
        String str=e.getActionCommand();
        if(str.equals("Submit"))
            repaint();
    }
    public void paint(Graphics g){
        g.setColor(Color.blue);
        if(c1.getState())
            msg=msg+c1.getLabel()+" ";
        if(c2.getState())
            msg=msg+c2.getLabel()+" ";
        if(c3.getState())
            msg=msg+c3.getLabel()+" ";
        if(c4.getState())
            msg=msg+c4.getLabel()+" ";
    }  g.drawString(msg, 80, 120);
```

Output:



RadioButtons:

*RadioButtons are also called CheckboxGroups. They are a special kind of checkboxes where with a particular group only one box can be selected at a time.

*To create RadioButtons, we use CheckboxGroup class.

*It has only one constructor:

CheckboxGroup()

Eg: CheckboxGroup g=new CheckboxGroup();

*The following are the methods:

1, getSelectedCheckbox()

2, setSelectedCheckbox()

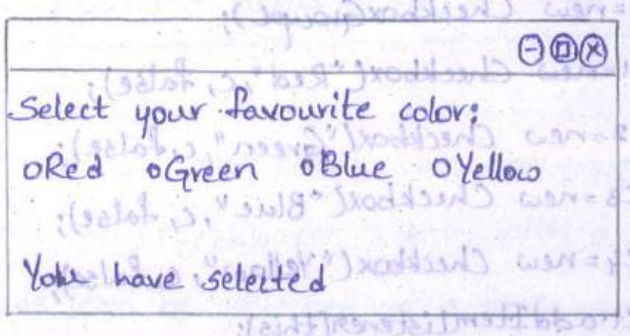
Program:

```
//Program to demonstrate RadioButtons
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*applet code="RadioButtonDemo.class" width="300" height="200">
</applet> */
public class RadioButtonDemo extends Applet implements
                                    ItemListener{
  String msg="You have selected";
  CheckboxGroup c;
```

```
Checkbox c1, c2, c3, c4;
Label l;
public void init(){
    setBackground(Color.green);
    l=new Label("Select your favourite color:");
    c=new CheckboxGroup();
    c1=new Checkbox("Red", c, false);
    c2=new Checkbox("Green", c, false);
    c3=new Checkbox("Blue", c, false);
    c4=new Checkbox("Yellow", c, false);
    c1.addItemListener(this);
    c2.addItemListener(this);
    c3.addItemListener(this);
    c4.addItemListener(this);
    add(l);
    add(c1);
    add(c2);
    add(c3);
    add(c4);
}
public void itemStateChanged(ItemEvent e){
    Checkbox x=c.getSelectedCheckbox();
    String str=x.getLabel();
    if(str.equals("Red"))
        msg+="Red color!";
    else if(str.equals("Green"))
        msg+="Green color!";
    else if(str.equals("Blue"))
        msg+="Blue color!";
    else if(str.equals("Yellow"))
        msg+="Yellow color!";
    repaint();
}
```

```
    public void paint(Graphics g){
        g.setColor(Color.magenta);
        g.drawString(msg, 120, 100);
    }
}
```

Output:



ListBoxes:

*A ListBox is multiple choice, scrolling list of value th may be selected alone or together.

*ListBoxes are created using the List class.

*List class have the following constructors:

1. List()
2. List(int rows)
3. List(int rows, boolean multiple)

Note:

If multiple is said to be 'true' then multiple items can selected in the ListBox.

Eg: List fruits = new List(10, true);

*Once we have created the list, we can add entries the list using add() method as follows:

Eg:
```
fruits.add("Apple");
fruits.add("Grapes");
```

*The add() method can accept the 2nd argument, to s the index, where the item to be appeared in the list

Eg: fruits.add("Apple", 1); //Second item in the list

fruits.add("Grapes", 0); //First item in the list

* The following are some of the methods of List class:

1. void addActionListener(ActionListener al);
2. void addItemListener(ItemListener il);
3. void removeActionListener(ActionListener al);
4. void removeItemListener(ItemListener il);
5. void add(String item);
6. void add(String item, int pos);
7. String[].getItems()
8. int getItemCount()
9. void remove(String item)
10. void remove(int pos)
11. void removeAll()
12. void select(int index)
13. void deselect(int index)
14. String getSelectedItem()
15. String getSelectedIndex()

ChoiceBoxes:

* A ChoiceBox is just like a ListBox but it allows you to conserve space. Since it provides a popup menu of choices. Only single selection can be made in a ChoiceBox.
* To create ChoiceBox, we use the class Choice.

Eg: Choice c=new Choice();

* Once we have created a ChoiceBox, the add() method enables you to add entries to the choice as follows:

Eg: c.add("Red");
c.add("Green");

* The methods of ChoiceBox are listed below:

1. void add(String item)
2. void add(String item, int ind pos)
3. void select(String item)
4. void select(int index)
5. String getSelectedItem()
6. int getSelectedIndex()

Textfield & TextArea:

* The Textfield and TextArea are 2 classes used for creating text components.
* Textfield is used for handling single line of text whereas TextArea is used for handling multiple lines of text.
* The following are the constructors of Textfield class:

1. TextField()
2. TextField(int cols)
3. TextField(String text)
4. TextField(String text, int cols)

Eg: TextField t=new TextField(20);

* The following are the methods of Textfield class:

1. void addActionListener(ActionListener al)
2. int getColumns()
3. void setText(String text)
4. String getText()
5. void setColumns(int cols)
6. void removeActionListener(ActionListener al)
7. void setEchoChar(Char ch)
8. Char getEchoChar()

*The following are the methods of constructors of TextArea class:

1, TextArea( )

2, TextArea(int rows, int cols)

3 TextArea(String text)

4, TextArea(String text, int rows, int cols)

5, TextArea(String text, int rows, int cols, int scrollbar)

* The following are the methods of TextArea class:

1, void append(String str)

2, void setColumns(int cols)

3, int getColumns( )

4 void setRows(int rows)

5, int getRows( )

Container class:

*A Container is a component which holds other components.

* AWT provides 4 Container classes i.e, Panel, Window, Dialog, Frame.

1, Window:

*It is a top level display surface

* It cannot be attached within another container

*An instance of Window doesn't have Border, TitleBar or MenuBar

2, Frame:

*It is a top level window with a border & titlebar.

*An instance of Frame class may have MenuBar or TitleBar & Border.

3, Dialog:

*It is a top level display surface (or) Window with a Border & Title.

*An instance of Dialog cannot exist without an associa instance of Frame.

4. Panel:

*It is a generic container for holding components.

Creating Panel object,

```
Panel p=new Panel();
```

The Frame class:

*The Frame class has the following constructors:

1. Frame() — Just create a Frame
2. Frame(String str) — Frame contains title

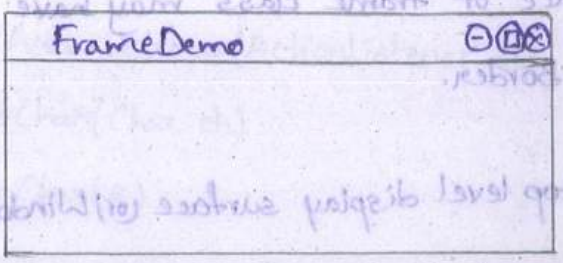
*The following are the methods of Frame class:

1. void setTitle(String title)
2. void setVisible(boolean val)
3. void setSize(int width, int height)

The following program is used to create a frame:

```
class MyFrame extends Frame {
    MyFrame(String title){
        super(title);
    }
    public static void main(String args[]){
        MyFrame f=new MyFrame("FrameDemo");
        f.setSize(300, 250);
        f.setVisible(true);
    }
}
```

Output:



The Frame displayed above will not be closed. When we click on the close button [x] of titlebar. To make it close when we click on the close button, we have to override windowClosing() method of WindowListener interface as follows:

```
class MyFrame extends Frame{
    MyFrame(String title){
        super(title);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
    public static void main(String args[]){
        MyFrame f=new MyFrame("Frame Demo");
        f.setSize(300, 250);
        f.setVisible(true);
    }
}
```

Layouts / LayoutManagers:

* Layout scheme's are used to arrange the components that are add to a frame instance.
* Layout Manager is used to specify how the components can be arranged inside a Container (Frame, Panel, Applet etc.,)
* Various Layout managers are available in 'java.awt' package
* These LayoutManagers are built in class defined in 'java.awt' package and all these classes implement 'LayoutManager' interface (or) 'LayoutManager2'.
* The LayoutManagers of AWT are:

1. FlowLayout

2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout

1. FlowLayout:

* FlowLayout arranges the component from left to right (items) and top to bottom (lines), centering components horizontally.
* There is 5 pixel gap between the components arranged in this Layout.
* FlowLayout recomputes new positions for the components whenever the container size is changed.
* This is the default Layout for the Applet.
* This Layout is one of the simplest Layout available in Java.
* The constructors of FlowLayout class are given below:

1. FlowLayout()
2. FlowLayout(int align)
3. FlowLayout(int align, int hgap, int vgap)

Note:
align can be FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER(day), FlowLayout.LEADING, FlowLayout.TRAILING

Eg: setLayout(new FlowLayout(FlowLayout.LEFT));

2. BorderLayout:

* It is the default Layout of the Frame. The BorderLayout is a Layout where the components can be arranged and resize to fit in 5 different regions i.e., NORTH, SOUTH, EAST, WEST and CENTER.
* There can be only one component in each region and the

regions are identified as constants NORTH, SOUTH, EAST, WEST, CENTER. Any of these 5 components can be used, will adding the components to the container.

Eg:

```
Panel p=new Panel();
p.setLayout(new BorderLayout());
p.add(new Button("Submit"), BorderLayout.CENTER);
```

* The following are the 2 constructors of BorderLayout class:

1. BorderLayout()
2. BorderLayout(int hgap, int vgap)

3. CardLayout:

* Object of CardLayout acts as LayoutManager for container.
* In a container each component is treated as a card by CardLayout object. Each card is kept on another like a stack and only one card can be visible at a time.
* When the container is displayed after add first component then the first component is visible.
* CardLayout defines a set of methods that allow an application to flip through these card sequentially (or) to show a specific card.
* These methods are first(), last(), next(), previous(), show()
* Constructors of CardLayout is are

1. CardLayout()
2. CardLayout(int hgap, int vgap)

4. GridLayout:

* It is a LayoutManager that arranges the components of a container in a rectangular Grid. GridLayout has a specified no. of rows & columns where the container is divided into equal sized rectangles & one component is placed in each rectangle.

* The GridLayout arranges the component in raws & columns order. Each component fills up its respective Gridcells.
* The following are the constructors of GridLayout:

1, GridLayout()
2, GridLayout(int raws, int cols)
3, GridLayout(int raws, int cols, int hgap, int vgap).

5, GridBagLayout:

* Using GridBagLayout, we can arrange the components in more control way in horizontal as well as vertical direction.
* In GridBagLayout the component need not be of same side in a row.
* We can arrange different sizes of component by specifying their position within the cell of a Grid in the same row.
* One more advantage is that each raw can contain dissimilar no. of components.
* In GridBagLayout, the size and position of components are dependent on set of constraints linked to it.
* These constraints can be set using GridBagConstraints object.

Eg: GridBagConstraints.gridx=20;

Menu:

* A MenuBar displays a list of menu's and each menu is a dropdown list of menu choice.
* MenuBar appears in a frame just below the titlebar.
* To implement this concept, we use 3 classes:

1, MenuBar
2, Menu
3, MenuItem

* The following are the steps to add menu's in a Frame:

1, Create a MenuBar instance & set the MenuBar.

```
MenuBar mb=new MenuBar();
setMenuBar(mb);
```

2, Create a Menu

```
Menu m1=new Menu("File");
Menu m2=new Menu("Edit");
```

3, Create MenuItems

```
MenuItem i1=new MenuItem("New");
MenuItem i2=new MenuItem("Open");
MenuItem i3=new MenuItem("Save as");
MenuItem i4=new MenuItem("Save");
MenuItem i5=new MenuItem("Close");
```

* For handling event, addActionListener to the MenuItems.

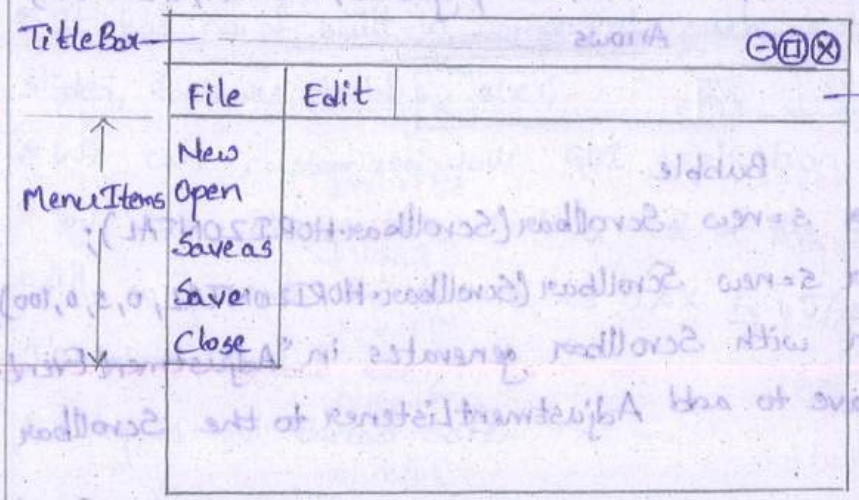
```
i1.addActionListener(this);
i2.addActionListener(this);
i3.addActionListener(this);
i4.addActionListener(this);
i5.addActionListener(this);
```

4 Add MenuItems to the Menu.

```
m1.add(i1); m1.add(i2); m1.add(i3); m1.add(i4); m1.add(i5);
```

5, Add Menu's to the MenuBar.

```
mb.add(m1); mb.add(m2);
```



*If you want to add a KeyBoard ShortCut for selecting MenuItem, we can use MenuShortCut class. First we have to create MenuShortCut instance.

MenuShortcut n=new MenuShortcut(KeyEvent.VK_N);

* To create the MenuItem as follows:

MenuItem i1=new MenuItem("New",n);

* If you want to add a seperator(just line will come) between MenuItems. We can used to addSeperator() method as follows:

Eg: m1.addSeperator();

Scrollbars:

*Scrollbars are used to select continue values through a range of integer values.

* These scrollbars can be set either horizontally or vertically.

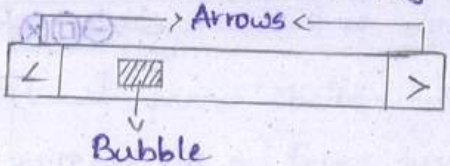
* The Scrollbars are minimum & maximum values can be set along with line increments & page increments.

*for creating a Scrollbar component, we have to create an instance of 'Scrollbar' class. It defines the following constructors:

1, Scrollbar() - By default it creates a vertical Scrollbar

2, Scrollbar(int direction) ↳ Scrollbar.VERTICAL(or) Scrollbar.HORIZONTAL

3, Scrollbar(int dir, int value, int pagesize, int min, int max)



Eg: Scrollbar s=new Scrollbar(Scrollbar.HORIZONTAL);

Scrollbar s=new Scrollbar(Scrollbar.HORIZONTAL, 0, 5, 0, 100);

*Interaction with Scrollbar generates in "AdjustmentEvent."

* So we have to add AdjustmentListener to the Scrollbar component.

*Inorder to handle AdjustmentEvent generated by Scrollbar,

your program must defines adjustmentValueChanged().

Limitations of AWT:

* AWT has heavy memory component.
* Platform dependent.

Swing:

* To overcome the limitation of AWT components a set of GUI related classes called Swing is introduced with JDK 1.2 & later.
* These Swing components are available in `javax.swing` package
* All swing components use Model-View-Controller (MVC) architecture for providing greater flexibility.
* All string controls (components) has 3 components. i.e., a model, a view, a controller.
* Model manages the state & behaviour of the component.
* View manages the display of the component depending upon the state.
* Controller handles the interactions of the user with model. Controller basically determines when & how the state of the model will change.

Features of Swing:

* Swing components are lightweight.
* It has no. of built-in components such as trees, tabbedpanes, slider, toolbars, tables etc.,
* We can customized your GUI application.
* You can change the look & feel of the Swing components.
* All components are named as JXX Eg: JApplet, JFrame, JButton, JLabel etc.,
* Swing is not thread safe.

Differences between AWT and Swings:

AWT	Swings
*1, AWT components are heavy weight.	1, Swing components are light weight.
2, Look & Feel of AWT components is OS based	2, Look & Feel of Swing components is OS independent.
3, AWT components are not pure Java based.	3 Swing components are pure Java based.
4, AWT components are work faster	4, Swing components are work slower
5, Does not support features like icons & tooltip	5, Support features like icons & tooltip.
6, AWT has platform specific limitation for some components.	6, Fewer platform limitation for components.
7, AWT components are defined in java.awt package.	7, Swing components are defined in javax.swing package
8, Events related to AWT components are defined in 'java.awt.event' package.	8, Classes & Interfaces related to eventhandling in Swing are defined in 'javax.swing.event' package

* The following table lists the classes of AWT & their counter parts in Swings.

Class in AWT	Corresponding class in Swing
Frame	JFrame
Applet	JApplet
Panel	JPanel
Label	JLabel
Button	JButton
TextField	JTextField

TextArea	JTextArea
Checkbox	JCheckbox
List	JList
Menu	JMenu
MenuBar	JMenuBar
MenuItem	JMenuItem
Choice	JCombobox

JFrame:

JFrame is a subclass of 'java.awt.frame' class. So it inherits all the features of an awtframe.

*The following program creates a Frame in Swings with label:

```
import javax.swing.*;
class MyFrame extends JFrame{
    MyFrame(){
        JLabel l=new JLabel("This is my first Swing program");
        add(l);
        setTitle("Demo JFrame");
        setSize(300, 200);
        setVisible(true);
        setDefaultCloseOperation(EXIT-ON-CLOSE);
    }
```