***PROGRAMMING IN***

***JAVA***

***LECTURE NOTES(Semester –IV)*** *for*

***B****achelor of* ***Co****mputer* ***S****cience*



***Department of Computer Science and Applications***

**School of Arts &Science VinayakaMission’sResearch Foundation**

**AV Campus,Chennai–603104.**

**Lecture Note Prepared**

**by**

**G.Ramasubramanian, Associate Professor**

1. **Define token. What are the tokens used in java**

Smallest individual element in a program is called token.

The various tokens are

1. Keywords
2. Identifiers
3. Constants
4. Operators
5. Strings
6. Variables
7. **Define TCP.**

TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

1. **Define UDP**

 UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

1. **Define Proxy server**

**Proxy server** is an intermediary server between client and the interner. Proxy servers offers the following basic functionalities:

* Firewall and network data filtering.
* Network connection sharing
* Data caching

1. **Define Java InetAddress**

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name

**for example**  www.javatpoint.com, www.google.com, www.facebook.com etc.

Define URL

  URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example: http:\\www.yahoo.com\

1. **Define Datagrams**

A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

1. **Define file.**

File is collection of records. Record is a collection of different data.

1. **Define typecasting.**

Type casting is a way to convert a variable from one data type to another data type.

**Example:**

 int a = 350;

 byte b;

 b = (byte) a

1. **Classes and objects in java**

**Classes:** The class is used to create a user-defined data type from the data and code of an object.

**Syntax:**

Class classname

{

Variable declaration;

Method declaration;

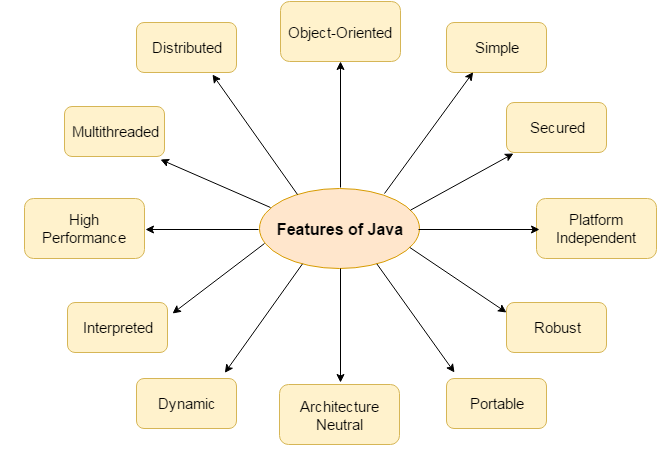
}

**Objects:** Objects are the basic **run-time entities** in object oriented systems. The object contain data, and code to manipulate the data.**Ex:**A Person, A Place, A Bank Account etc.

**Syntax:**

**myobject ob=new myobject();**

**1.Features of java**

****

1. Compiled and interpreted
2. Platform independent and portable
3. Object oriented
4. Robust and secure
5. Distributed
6. Familiar, simple and small
7. Multithreaded and interactive
8. High performance
9. Dynamic and extensible

**Compiled and interpreted**

Java compiler translates source code into bytecode instructions. Bytecodes are not machine instructions therefore java interpreter generates machine code.

**Platform independent and portable**

Java program can easily moved from one computer system to another, anywhere and any time.

Changes and upgrade in operating systems,processors and system resources will not force any changes in java programs.

**Object oriented**

Object oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

**Robust and secure**

Java is a robust language its provides many safeguards to ensure reliable code. Java systems not only verify all memory access but also ensure that no viruses or communicated with an applet.

**Distributed**

Java has ability to share both data and programs .

This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

**Familiar, simple and small**

Java is small and simple language it doesn’t use pointers, pre-processor header files, goto statements, and many others. It also eliminates operator overloading and multiple inheritance.

**Multithreaded and interactive**

Multithreaded means handling multiple tasks simultaneously. This means that we need not wait for the applications to finish one task before beginning another.

Interactive feature greatly improves the interactive performance of graphical applications.

**High performance**

Java architecture design to reduce overheads during runtime.

The multithreading enhances the overall execution speed of java programs.

**Dynamic and extensible**

Java is capable of dynamically linking in new class libraries, methods and objects.

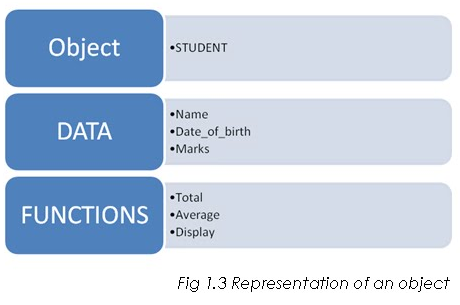
Java program support functions return in another language such as c and c++. These functions are known as native methods.

**2.Basic Concepts of Object Oriented Programming**

**Definition for OOP**

Object oriented programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

**Objects:** Objects are the basic **run-time entities** in object oriented systems. The object contain data, and code to manipulate the data.**Ex:**A Person, A Place, A Bank Account etc.

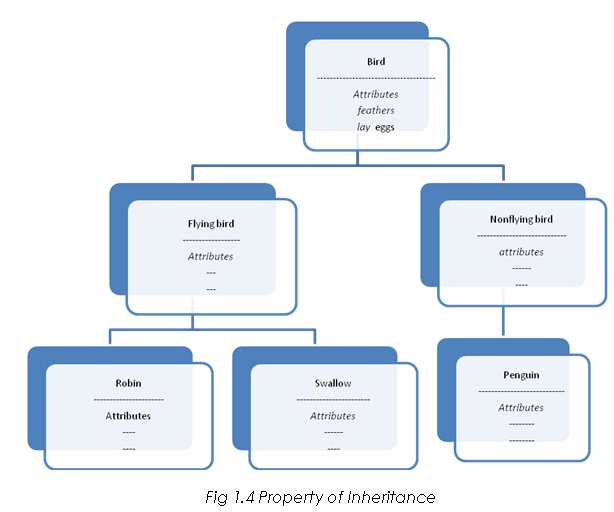


**Classes:** The class is used to create a user-defined data type from the data and code of an object.

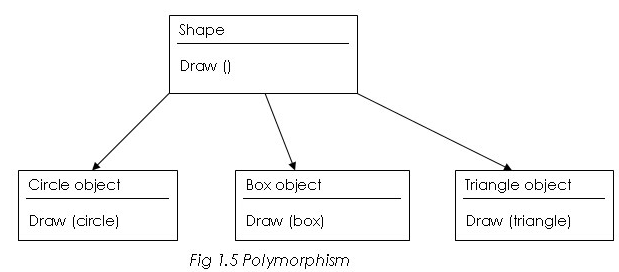
**Data** **Abstraction:** Representing only the essential information to the outside world and hiding their background details,

**Data Encapsulation:** Wrapping up of data and functions into single unit.

**Inheritance:** Process by which objects of one class acquires the properties of objects of another class.

****

**Polymorphism:** Ability to take more than one form. An operation may exhibit different behaviour in different instances. The behaviour depends upon the types of data used in the operation.



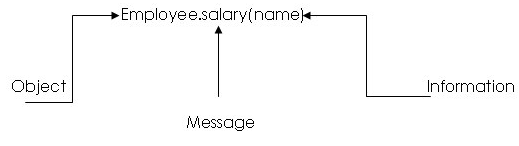
**Dynamic binding:** Binding refers to the linking of a procedure call to the code to be executed in response to the call.

**Message passing:** Set of objects that communicate with each other. Message passing involves the following basic steps:

1.Creating classes that define objects and their behaviour.

2.Creating objects from class definitions.

3.Establishing communication among objects.

****

**===============================================================================**

**3.Control Structures:**

Control structures are specially meant for transfer of control from one statement to another based on condition check or by unconditional branching. The control structures in java is listed below.

1. Sequence
2. Selection
3. Looping

**IF CONDITIONAL STATEMENT**

The different form of if statement is shown below,

1. Simple if
2. If . . . else
3. Nested If . . . else
4. Else if Ladder

**Simple If Statement**

The simple if is the powerful decision making statement and is used to control the program flow. It is used for two way decision making. The simple if meant for single condition check.

The syntax for simple if is shown below,

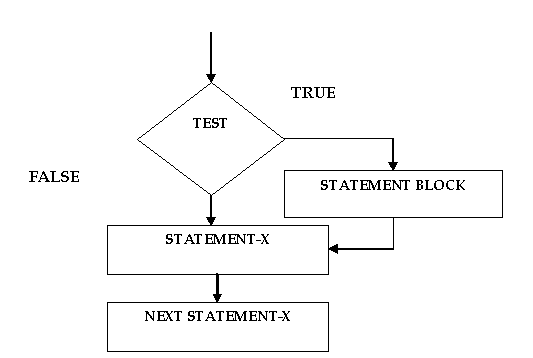
if (Test Condition){

Statement Block-1

}

Statement Block-N;

**Flow control of Simple if**



**If . . . Else Statement**

It is the extension of simple if. The test condition is evaluated first, and when the test condition is true, the true block will be executed else the false block will be executed.

Syntax:

if (Test Condition)

{

True Statement Block;

}

else

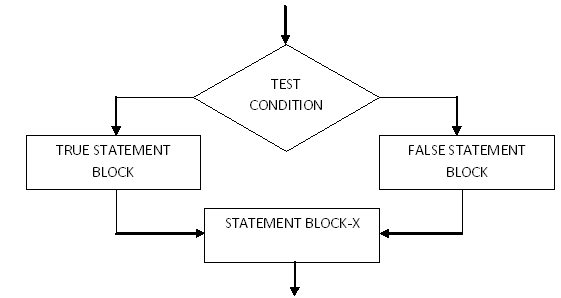
{

False Statement Block;

}

Statement-X;

**Flow control of Simple if else**

****

**Nested if else**

If statement within another if statement is called nested if

**Syntax**

if(condition)

{

if(condition)

{

statements;

}

else

{

statements;

}

}

else

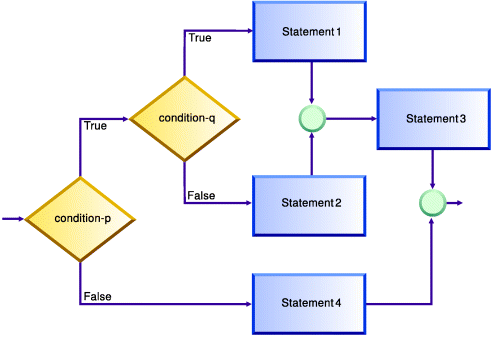
{

statements;

}

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and again checks the next condition. If it is true then it executes the block of statements associated with it else executes else part.

**Flow Control**

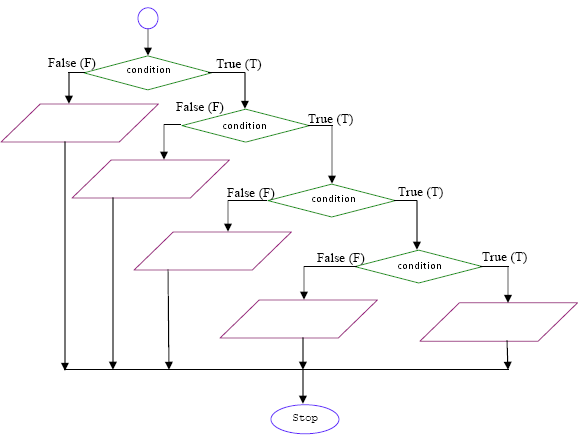
****

**Else if Ladder**

Too many if, else block may be very difficult. This situation can be avoided using else if ladder. The **syntax of else if ladder** is:  
**Syntax**

if ( condition 1)  
true block of the above decision  
else if ( condition 2)  
true block of the above decision  
else if (condition 3)  
true block of the above decision  
............................................  
else  
default block.

**FlowChart**



**SWITCH CASE**

Switch is a **multiple-branch selection statement**, which successively tests the value of an expression against a list of integer or character constants.

**The general form of the switch statement is,**

switch (expression) {

case constant1:

statement sequence

break;

case constant2:

statement sequence

break;

case constant3:

statement sequence

break;

.

..

default

statement sequence

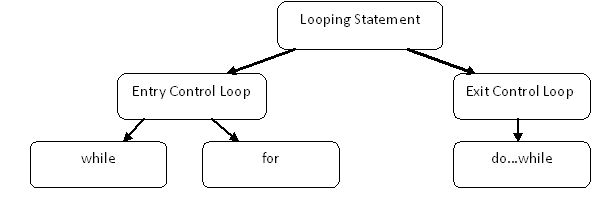
}

**LOOPING STATEMENT**

**Definition:**

The Looping is the programming construct, which is meant for executing set of statements repeatedly for countable number of times. The process of repeating the steps is called as iteration.

**Types of looping**



**WHILE LOOP:**

The while loop is the entry control loop. The test condition is evaluated first and the body of the loop will be executed. When the test condition is true the body of the loop will be executed, after executed the loop variable will be incremented or decremented. Again the condition is checked when the condition is true the body of the loop will be executed. The body of the loop will be executed until the condition is true.

**Syntax: while (condition)**

**{**

**…… Body of the loop**

**……**

**}**

**FOR LOOP:** For loop is an entry control loop, but for loop is very concise than the while loop. The initialization of counter variable, condition check and the increment and decrement of variable are set in one statement.

**Syntax**

for (Initialization of counter variable; condition check; Inc/Dec)

{

Body of loop

}

**Example**

**for (count=1;count<=10;count++)**

Count++ which is the increment of Counter variable

**Steps involved in executing the for loop**

1. Initializing the loop control variable.
2. Condition check
3. If condition is true execute the body of the loop, then increment or decrement the loop control variable for iteration.

**DO-WHILE LOOP:** In do.. While loop the body of the loop will be executed at least once. The loop condition will be checked only at the end of the loop. This type of loop will be suitable for execution of loop unconditionally once. The syntax for the do… while loop is shown below.

**Syntax**

**Do**

**{**

**Body of loop**

**} While (condition);**

**4. Constructors**

* A constructor is a ‘special’ member function whose task is to initialize the object of its class.
* It is special because its name is the same as the class name.
* The constructor is invoked whenever an object of its associated class is created.
* It is called constructor because it construct the value of data members of the class.

**Example:**

class integer

{

int m,n;

integer()

{

m=0;

n=0;

.......

};

The declaration, integer I1; creates the object I1 of type integer and initializes its data members m and n to zero. A constructor that accepts no parameters is called the default constructor.

**5.Inheritance**

Process by which objects of one class acquires the properties of objects of another class.

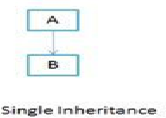
The main advantages of the inheritance are:

* Reusability of the code
* To increase the reliability of the code.
* To add some enhancements to the base class

**Types of Inheritance:**

1. **Single Inheritance**
2. **MultipleInheritance**
3. **Hierarchical Inheritance**
4. **Multilevel Inheritance**
5. **Hybrid Inheritance**

**Single Inheritance**

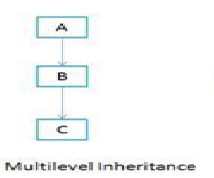
Inheritance hierarchy wherein one derived class inherits from one base class.  
****

|  |
| --- |
| **Syntax:**  class derived class name :: visibility name base class name  {  ----------  ----------  }  **Example:**  class B  {  ----------  };  class D: private B  {  ------------  };  void main()  {  clrscr();  D d;  ---------------  } |

|  |
| --- |
|  |

**Multilevel Inheritance:**

Inheritance hierarchy wherein one derived class inherits from multiple base classes.class B is derived from class A and class c is derived from class B . Class B is intermediate class it act as a base class and derived class.

****

**Example:**

class A

{

------

}

class B :: public A

{

------

}

class C :: public B

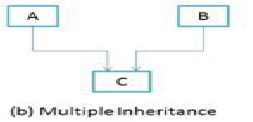
{  
 --------

}

**Multiple Inheritance:**

A class can inherit the attributes of two or more classes. This type of inheritance is called multiple inheritance.

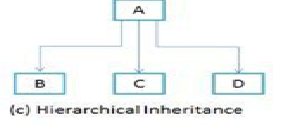
Java doesn’t support multiple inheritances. This is accomplished by the concept of interface.



**Hierarchical Inheritance:**

Inheritance hierarchy wherein multiple subclasses inherit from one base class.

Another interesting application of inheritance is to use it as a support to the hierarchical design of a program. Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below that level



class A

{

-------

}

class B :: public A

{

}  
class C :: public A

{

}  
class D :: public A

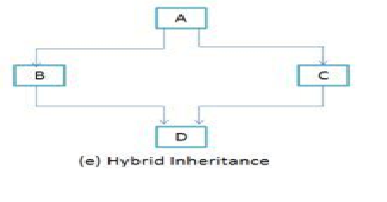
{

**}**

**Hybrid Inheritance:**

Inheritance hierarchy that reflects any legal combination of other four types of inheritance.

Some situations do arise where two or more type of inheritance are used to design a program. The following example gives an example of two types of inheritance being used. i.e. multilevel and multiple inheritance.

****

class A

{

}

class B :: public A

{

}

class C

{

}

class D :: pubic B public C

{

}

1. **Method Overloading**

A class can have same function name but different parameter with different task is called method overloading.

The compiler match the function call with the exact function code by checking the number and type of the arguments.

1. Function with no argument
2. Function with argument

**Example:**

class over

{

void test()

{

System.out.println(“ no operation”);

}

void test(int a)

{

System.out.println(a);

}

void test(int a,int b)

{

int c;

c=a+b;

System.out.println(c);

}

}

class methover

{

public static void main(String args[])

{

over f=new over();

f.test();

f.test(10);

f.test(10,20);

}

}

1. **Method overriding**

By defining the method in the subclass that has the same name same arguments and same return type as a method in a superclass. Then when that method is called the method defined in the subclass is invoked and executed instead of the one in the super class. This is known as method overriding.

Class super

{

int x;

super(int x)

{

this.x=x;

}

Void display()

{

System.out.println(“super x:”+x);

}

}

Class sub extends super

{

int y;

super(int x,int y)

{

Super(x);

this.y=y;

}

Void display()

{

System.out.println(“super x:”+x);

System.out.println(“sub y:”+y);

}

}

Class override

{

Public static void main(String args[])

{

Sub s1=new sub(100,200);

S1.display();

}

}

1. **Interface**

An interface is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

**General form**

interface interfacename

{

Variable declaration;

Method declaration;

}

**Example**

interface area

{

float pi=3.14;

void display();

}

**Extending interface**

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Interface interfacename1 extends interfacename2

{

Variable declaration;

Method declaration;

}

**Example**

interface area

{

float pi=3.14;

-------

-------

}

interface peri extends area

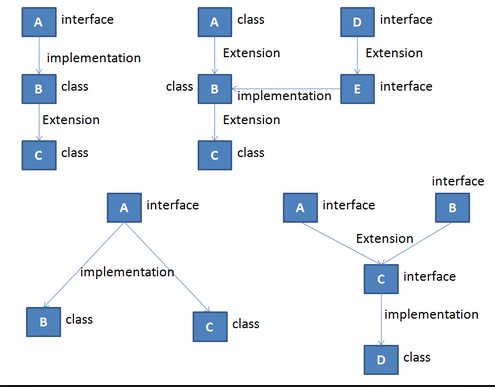
{

void display();

-------

-------

}



**Implementing interface**

A class uses the **implements** keyword to implement an interface.

Class classname implements interfacename

{

Body of class

}

Class classname extends superclass implements interface1,interface2….

{

Body of class

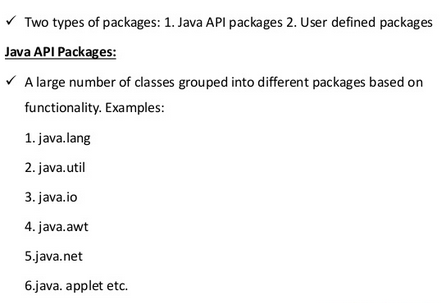
}

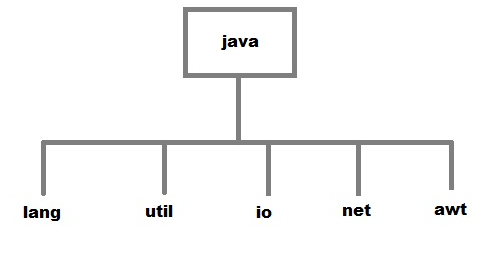
1. **Package**

A package is a collection of classes and interfaces which provides a high level of access protection and names space management.

syntax: package packagename;

Eg: package mypack;





**Defining & Creating package:**

1. Declare the package at the beginning of a file using the form

**Package packagename;**

1. Define the class that is to be put in the package and declare it **public**.
2. Create a subdirectory under the directory where the main source files are stored.
3. Store the listing as the **classname.java** file in the subdirectory created.
4. Compile the file. This creates .classfile in the subdirectory.

package mypack;

public class first

{

--------

---------

Body of class

}

**Accessing a package**

A java package can be accessed either using a fully qualified classname or using import statement.

We generally use import statement.

syntax: import pack1.pack2.pack3.classname;

or

import pack1.pack2.pack3.\*;

here pack1 is the top level package, pack2 is the package which is inside in pack1 and so on. In this way we can have several packages in a package  hierarchy. \* indicates  that the compiler should search this entire package hierarchy when it encounters a class name.

**Adding a class to a package**

package pack1;

public class A

{

Public void displayA()

{

System.out.println(“Class A”);

}

}

import pack1.classA

class packtest

{

public static void main(String args[])

{

A a=new A();

a.displayA();

}

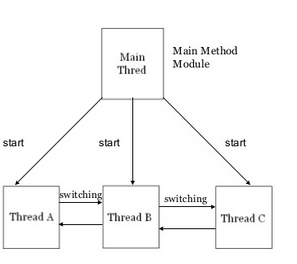
}

**10.Thread**

A thread is similar to a program that has a single flow of control. It has a beginning, a body and an end, and executes command sequentially.

**Multithreading**

Multithreading means handling multiple tasks simultaneously. This means that we need not wait for the applications to finish one task before beginning another.



**Creating Threads:**

A new thread can be created in two ways.

1. By extending Thread class: Define a class that extends **Thread** class and override its run() method with the code required by the thread.
2. By implementing Runnable interface: Define a class that implements Runnable interface. The Runnable interface has only one method, run(), that is to be defined in the method with the code to be executed by the thread.

The **run()** method is the heart and soul of any thread.

public void **run()**

{

…………….

…………….(statements for implementing thread)

…………….

}

**Extending Thread Class**

We can make our class runnable as thread by extending the class java.lang.Thread. This gives us access to all the thread methods directly. It includes the following steps

1. Declare the class as extending the Thread class
2. Implement the run() method that is responsible for executing the sequence of code that the thread will execute.
3. Create a thread object and call the start() method to initiate the thread execution.

Example:

class A extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println(“From Thread A : i ”+i);

}

}

}

class ThreadTest

{

 public static void main(String ar[])

 {

A a=new A();

a.start();

  }

}

**Stopping and blocking a thread**

**Stopping a thread**

Stop() method is used to stop a thread from running. It causes the thread to move to the dead state.

athread.stop()

**Blocking a thread**

A thread can also be temporary suspended or blocked from entering into the runnable and running state. The following methods are used to blocking a thread

* 1. sleep() - blocked for a specified time
  2. suspend() - blocked until further orders
  3. wait() - blocked until certain condition occurs

**Thread priority**

Thread Scheduler handles the context switching of threads with same priority. The priority of thread lies between 1 to 10. Default priority of each thread is NORM\_PRIORITY, which is 5. Thread priority can be defined as:

* MAX\_PRIORITY, which is 10
* NORM\_PRIORITY, which is 5
* MIN\_PRIORITY, which is 1

We can get and set the priority of thread using Thread.getPriority() method and Thread.setPriority() method.

**Sytax**

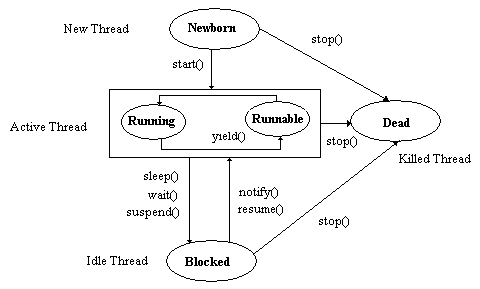
Thread.setPriority(Priority value);

**11.Life cycle of a Thread**

During the lifetime of a thread, there are many states it can enter. They include:

1. NewBorn State
2. Runnable State
3. Running State
4. Blocked State
5. Dead State

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



**NewBorn State:**

When we create a thread object, the thread is  born and is said to be in newborn state. The thread is not yet scheduled for running. At t his state, we can do only one of the following things with it:

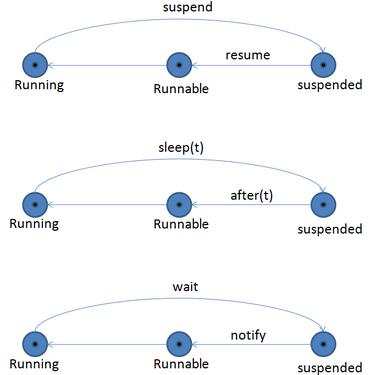
* Schedule it for running using start() method
* Kill it using stop() method

**Runnable State:**

The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. If we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the **yield()**

**Running State:**

Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.



**Blocked State:**

A thread can also be temporary suspended or blocked from entering into the runnable and running state. The following methods are used to blocking a thread

* 1. sleep() - blocked for a specified time
  2. suspend() - blocked until further orders

wait() - blocked until certain condition occurs

**Dead State:**

Stop() method is used to stop a thread from running. It causes the thread to move to the dead state.

athread.stop()

**12.Exception handling**

An exception is a situation, which occurred by the run-time error. In other words, an exception is a run-time error. An exception may result in loss of data or an abnormal execution of program.

Types of error

* + 1. compile time error- all syntax errors will be detected and displayed by the compiler such as
       1. Missing semicolons
       2. Missing double quotes
       3. Misspelling of identifiers and keywords
    2. run time error- error that occurs while the program is running
       1. dividing an integer by zero
       2. converting invalid string to a number
       3. Accessing an element that is out of the bounds of an array

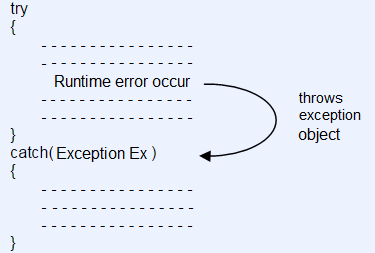
Exception handling is a mechanism that allows you to take appropriate action to avoid run-time errors.

Java provides five keywords to support exception handling.

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

* **Try :** The try block contain statements which may generate exceptions.
* **Catch :**The catch block defines the action to be taken, when an exception occur.

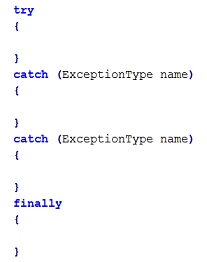
The general form of try-catch block in Java.



* **Throw :** When an exception occur in try block, it is thrown to the catch block using throw keyword.

throw new Throwable-subclass;

* **Throws :** Throws keyword is used in situation, when we need a method to throw an exception.
* **Finally :** If exception occur or not, finally block will always execute.



Example:

**class** ExceptionDemo

{

**public** **static** **void** main(**String**[] args)

{

**int** a=10,b=0,c;

**try**

{

c = a/b;

**System**.out.println(c);

}

**catch**(**Exception** e)

{

**System**.out.println("Divide by zero”);

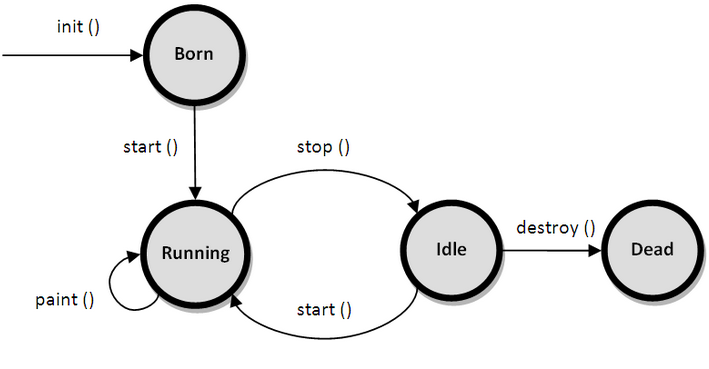
}

}

}

**13.Applet life cycle**

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.



* **init**()− This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

**public void init()**

**{**

**---**

----

}

**start()** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

**public void start()**

**{**

**---**

----

}

* **stop()** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

**public void stop()**

**{**

**---**

----

}

* **destroy()** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

**public void destroy()**

**{**

**---**

----

}

* **paint()** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

**public void paint()**

**{**

**---**

----

}

**14.Visibility control(Access protection)**

The visibility modifiers are also known as *access modifiers.* Access modifiers determine the accessibility of the members of a class.

Java provides three types of visibility modifiers:

1. **Public,**
2. **Private**
3. **Protected.**

They provide different levels of protection as described below.

**Public Access:** Any variable or method is visible to the entire class in which it is defined. But, to make a member accessible outside with objects, we simply declare the variable or method as public. A variable or method declared as **public** has the widest possible visibility and accessible everywhere.

**Friendly Access (Default):** When no access modifier is specified, the member defaults to a limited version of public accessibility known as "friendly" level of access. The difference between the "public" access and the "friendly" access is that the **public** modifier makes fields visible in all classes, regardless of their packages while the friendly access makes fields visible only in the same package, but not in other packages.

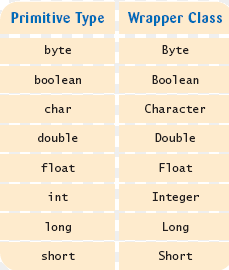
**Protected Access:** The visibility level of a "protected" field lies in between the public access and friendly access. That is, the **protected** modifier makes the fields visible not only to all classes and subclasses in the same package but also to subclasses in other packages

**Private Access:** private fields have the highest degree of protection. They are accessible only with their own class. They cannot be inherited by subclasses and therefore not accessible in subclasses. In the case of overriding public methods cannot be redefined as private type.

**Private protected Access:** A field can be declared with two keywords **private** and **protected** together. This gives a visibility level in between the "protected" access and "private" access. This modifier makes the fields visible in all subclasses regardless of what package they are in. Remember, these fields are not accessible by other classes in the same package.

**15.Wrapper classes**

**Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive.

****

**1.valueOf() method**

Returns an Integer object holding the value of the specified primitive.

## Syntax

static Integer valueOf(int i)

static Integer valueOf(String s)

static Integer valueOf(String s, int radix)

**2.toString() method**

Returns a String object representing the value of a specified int or Integer.

## Syntax

String toString()

static String toString(int i)

**3.parseInt()**

This method is used to get the primitive data type of a certain String.

## Syntax

static int parseInt(String s)

static int parseInt(String s, int radix)

EXAMPLE

import java.io.\*

class fahren

{

public static void main(String args[])

{

int f;

float c;

DataInputStream ds=new DataInputStream(System.in);

System.out.println(“Enter the temperature:”);

f=Integer.parseInt(ds.readLine());

c=(f-32)\*1.8

System.out.prinln(“celcius is:”+c);

}

}

**16.The AWT classess**

The AWT classes are contained in the **java.awt** package. It is one of Java's largest packages. some of the AWT classes.

**AWT Classes**

* 1. AWTEvent:Encapsulates AWT events.
  2. AWTEventMulticaster: Dispatches events to multiple listeners.
  3. BorderLayout: The border layout manager. Border layouts use five components: North, South, East, West, and Center.
  4. Button: Creates a push button control.
  5. Canvas: A blank, semantics-free window.
  6. CardLayout: The card layout manager. Card layouts emulate index cards. Only the one on top is showing.

1. Checkbox: Creates a check box control.
2. CheckboxGroup: Creates a group of check box controls.
3. CheckboxMenuItem: Creates an on/off menu item.
4. Choice: Creates a pop-up list.
5. Color: Manages colors in a portable, platform-independent fashion.
6. Component: An abstract super class for various AWT components.
7. Container: A subclass of Component that can hold other components.
8. Cursor: Encapsulates a bitmapped cursor.
9. Dialog: Creates a dialog window.
10. Dimension: Specifies the dimensions of an object. The width is stored in width, and the height is stored in height.
11. Event: Encapsulates events.
12. EventQueue: Queues events.
13. FileDialog: Creates a window from which a file can be selected.
14. FlowLayout: The flow layout manager. Flow layout positions components left to right, top to bottom.
15. Font: Encapsulates a type font.
16. FontMetrics: Encapsulates various information related to a font. This information helps you display text in a window.
17. Frame: Creates a standard window that has a title bar, resize corners, and a menu bar.
18. Graphics: Encapsulates the graphics context. This context is used by various output methods to display output in a window.
19. GraphicsDevice: Describes a graphics device such as a screen or printer.
20. GraphicsEnvironment: Describes the collection of available Font and GraphicsDevice objects.
21. GridBagConstraints: Defines various constraints relating to the GridBagLayout class.
22. GridBagLayout: The grid bag layout manager. Grid bag layout displays components subject to the constraints specified by GridBagConstraints.
23. GridLayout: The grid layout manager. Grid layout displays components i n a two-dimensional grid.Scrollbar: Creates a scroll bar control.
24. ScrollPane: A container that provides horizontal and/or vertical scrollbars for another component.
25. SystemColor: Contains the colors of GUI widgets such as windows, scrollbars, text, and others.
26. TextArea: Creates a multiline edit control.
27. TextComponent: A super class for TextArea and TextField.
28. TextField: Creates a single-line edit control.
29. Toolkit: Abstract class implemented by the AWT.
30. Window: Creates a window with no frame, no menu bar, and no title.

**17.AWT CONTROLS**

**The AWT supports the following types of control**

* 1. Labels
  2. Push button
  3. Choice list
  4. Test field
  5. Check box
  6. Text area
  7. Scroll bar

**1. Labels:**

Creates a label that displays a string.

**Label** defines the following constructors:

* + Label( )
  + Label(String str)
  + Label(String str, int how)

The value of *how* must be one of these three constants: **Label.LEFT**, **Label.RIGHT**, or **Label.CENTER**.

1. **Push button**

A *push button* is a component that contains a label and that generates an event when it is pressed.

Push buttons are objects of type **Button**. **Button** defines these two constructors:

Button( )

Button(String str)

1. **scrollbars**

Scrollbar generates adjustment events when the scroll bar is manipulated. Scrollbar creates a scroll bar control. Scroll bars may be oriented horizontally or vertically.

**Scrollbar** defines the following constructors:

Scrollbar( )

Scrollbar(int style)

Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

If *style* is **Scrollbar.VERTICAL**, a vertical scroll bar is created. If *style* is **Scrollbar.HORIZONTAL**, the scroll bar is horizontal.

1. **TextField**

The **TextField** class implements a single-line text-entry area, usually called an *edit*

*control.*

**TextField** defines the following constructors:

TextField( )

TextField(int numChars)

TextField(String str)

TextField(String str, int numChars)

1. **check box**

A *check box* is a control that is used to turn an option on or off. It consists of a small box that can either contain a check mark or not.

**Checkbox** supports these constructors:

* + 1. Checkbox( )
    2. Checkbox(String str)
    3. Checkbox(String str, boolean on)
    4. Checkbox(String str, boolean on, CheckboxGroup cbGroup)
    5. Checkbox(String str, CheckboxGroup cbGroup, boolean on)

**6.choices**

The **Choice** class is used to create a *pop-up list* of items from which the user may choose. A **Choice** control is a form of menu.

**Choice** only defines the default constructor, which creates an empty list.

**lists**

The **List** class provides a compact, multiple-choice, scrolling selection list.

**List** provides these constructors:

List( )

List(int numRows)

List(int numRows, boolean multipleSelect)

Ex: List os = new List(4, true);

**18.Layout manager**

A layout manager automatically arranges your controls within a window Layout manager class defines the following types of layout managers

* Border Layout
* Grid Layout
* Flow Layout
* Card Layout
* GridBag Layout

**BorderLayout**

**BorderLayout** class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center.

The constructors defined by **BorderLayout**:

BorderLayout( )

BorderLayout(int horz, int vert)

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center.

BorderLayout.CENTER

BorderLayout.SOUTH

BorderLayout.EAST

B orderLayout.WEST

BorderLayout.NORTH

# GridLayout

|  |
| --- |
| The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle. |

GridLayout( )

GridLayout(int numRows, int numColumns )

GridLayout(int numRows, int numColumns, int horz, int vert)

**Flow layout**

**FlowLayout** is the default layout manager.

Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right.

The constructors are

 FlowLayout( )

FlowLayout(int how)

FlowLayout(int how, int horz, int vert)

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

**Card layout**

The **CardLayout** class is unique among the other layout managers in that it stores several different layouts.

Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.

**CardLayout** provides these two constructors:

CardLayout( )

CardLayout(int horz, int vert)

**GridBag Layout**

The Grid bag layout displays components subject to the constraints specified by GridBagConstraints.

**GridLayout** lays out components in a two-dimensional grid.

The constructors are

GridLayout( )

GridLayout(int numRows, int numColumns )

GridLayout(int numRows, int numColumns, int horz, int vert)

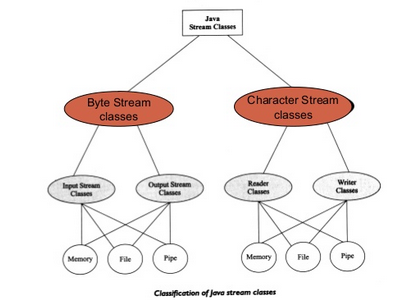
**19. Stream classes**

A stream is a sequence of bytes and serves as a source or destination for an I/O data.

Types of streams are

1. Byte stream

2. Character stream



**Byte stream classes**

Byte stream classes that provide functional features for creating and manipulating streams and files for reading and writing bytes

Types of Byte streams are

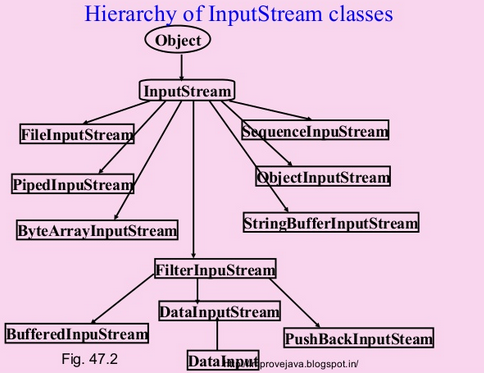
1. Input stream classes
2. Ouput stream classes

**Input stream**

Input stream objects can read and interpret input from sequences of characters

The input stream classes define methods for performing input functions such as

* 1. Reading bytes
  2. Closing streams
  3. Marking position in streams
  4. Skipping ahead in a stream
  5. Finding the number of bytes in a stream

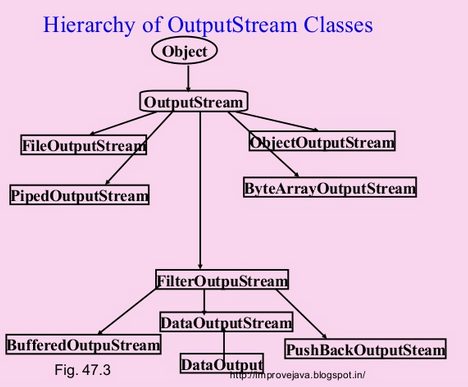


**Output stream**

**Output stream** objects can write sequences of characters and represent other kinds of data.

The output stream classes define methods for performing output functions such as

* + 1. Writing bytes
    2. Closing streams
    3. Flushing streams



**Character stream**

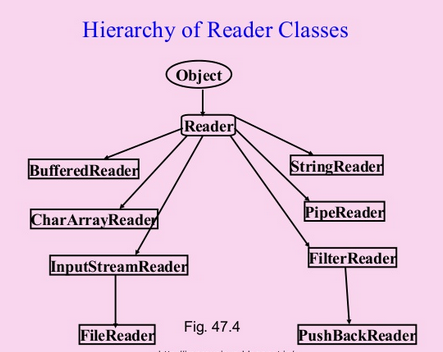
The character streams can be used to read and write 16 bit Unicode characters .

Types of character stream

* + - 1. Reader stream
      2. Writer stream

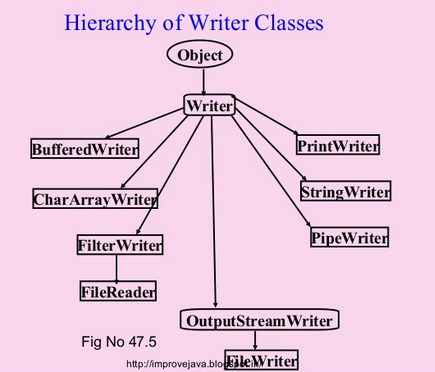
**Reader stream**

Reader stream classes are designed to read character from the files.

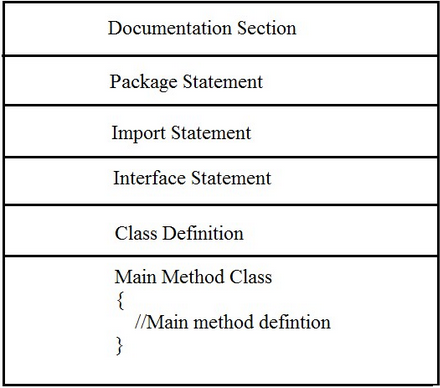


**Writer stream**

Writer stream classes are designed to write character to the files.



**20. Structure of java program**



(i)**Documentation section**: It comprises a set of comment lines giving the name of the program and other details.

Java comment s are

1. single line//

2. multiple line /\*….\*/

3. documentation comment /\*\*…….\*/

(ii**)package statements** The first statement allowed in a Java file is a **package** statement . This statement declares a package name and informs the compiler that the classes defined here belong to this package .

**Exampe :** package student;

(iii)**Import Statements:**  This is similar to #include statement in C .

**Example :**  import student.test;

This statement instructs the interpreter to load he test class contained in the package student .

(iv)**Interface Statement :** An interface is like a class but includes a group of method declarations. It is optional and is used only when we wish to implement the multiple inheritance feature in the program .

(v)**Class Definition :** A java program may contain multiple class definitions . Classes are the primary and essential part of a java program . These classes are used to map the real world problems .

(vi)**Main Method Class :**  Every java program requires a main method as its starting point , this clss is the essential part of a java program . The main method creates objects of various classes and establishes communications between them.

1. **String handling methods in java**

String is nothing but a sequence of characters, for e.g. “Hello” is a string of 5 characters.

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

**String methods**

### toUpperCase()-method converts this string into uppercase letter

### toLowerCase()-method converts this string into lowercase letter

Example:

String s="Sachin";

System.out.println(s.toUpperCase());

System.out.println(s.toLowerCase());

System.out.println(s);

1. trim()- method eliminates white spaces before and after string
2. charAt()- method returns a character at specified index.
3. length()- method returns length of the string.

Example:

String s="Sachin";

System.out.println(s.length());

**StringBuffer**

The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

StringBuffer()- creates an empty string buffer with the initial capacity of 16.

**StringBuffer methods**

The append()- method concatenates the given argument with this string.

The insert()- method inserts the given string with this string at the given position.

The replace()- method replaces the given string from the specified beginIndex and endIndex.

The delete()- method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

The capacity()- method of StringBuffer class returns the current capacity of the buffer.

**Don’t omit (refer notes)**

1. java statements and expressions

2. data types in java

3. constants

4. operators

5. variables

