www.Notesfree.in

1 CS3391 - Object Oriented Programming.

Unit-1 INTRODUCTION TO OOP AND JAVA FUNDAMENTAL

Overview of OOP - object oriented programming paradigms - Features of object oriented programming - Java Buzzwords - Overview of Java - Data types, variables and arrays - operators - control statements - programming structures in java - Defining classes in Java constructors - Methods - Access specifiers - static members - Java Doc comments.

Overview of OOPS:

In object oriented programming approach there is a collection of objects.

Classification:

1. classes and objects
2. Abstraction
3. Encapsulation
4. Inheritance
5. Polymorphism

www.Notesfree.in

www.Notesfree.in

OOP Paradigm :

Classes :

Class can be defined as an entity in which data and function are put together.

Syntax :

```
class name_of_class
{
  private :
        variable declaration;
        function declaration;
  public :
        variable declaration;
        function declaration;
};
```

Objects :

Object is an instance of class.

Syntax :

```
class name object_Name;
```

Abstraction :

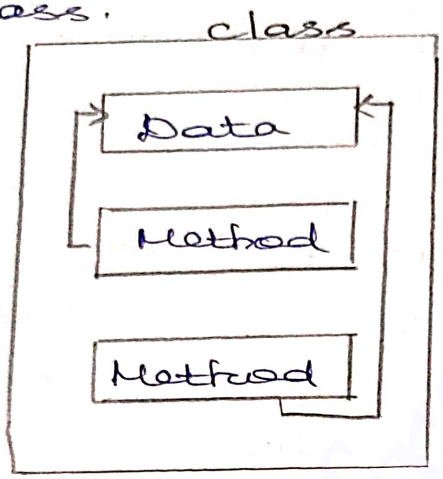
Abstraction means representing only essential features by hiding all the implementation details.

www.Notesfree.in

www.Notesfree.in

Encapsulation:

Encapsulation means binding of data and method together in a single entity called class.



Inheritance:

Inheritance is the property by which new classes are created from old classes

Types:

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Polymorphism:

Polymorphism is the ability to take more than one form and refers to an operation exhibiting different behaviour in different instances.

www.Notesfree.in

www.Notesfree.in

characteristics of Java:

Java is getting popular because of its features. Following is a list of buzzwords that makes Java popular.

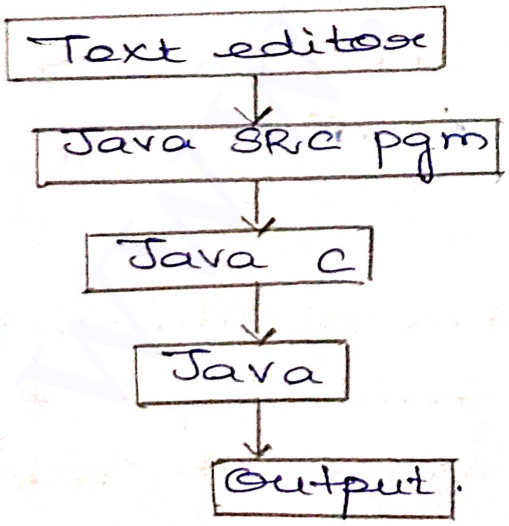
Simple
Secure
Platform independent
Portable
Robust

Overview of Java:

JDK - Java Development kit

Java development kit is a collection of tools used for development and run time programs.

Execution Process:



www.Notesfree.in

www.Notesfree.in

Data types:

1. int (2-4 bytes)
2. long (8 byte)
3. float (4 byte)
4. char (1 byte)
5. double (8 byte)
6. Boolean (True or False)

Variable:

Variable is an identifier that denotes storage location.

Syntax:

datatype name_of_variable;

Array:

Array is a collection of similar type of elements

Syntax:

datatype array_name [];

Types:

1. One-dimensional array
2. Two-dimensional array

One dimensional array:

ex:

```
Class Sample
{
 public static void main (String args[])
{
 int a [] = new int [3];
 system.out.println ("Storing array:");
```

www.Notesfree.in

www.Notesfree.in

```
        a[0] = 1;
        a[1] = 2;
        a[2] = 3;
        System.out.println("Element at a[2]="+a[
    }
}
```

Two dimensional array:

Two dimensional array are the arrays in which elements are stored in rows and columns.

Syntax:

datatype - array-name = new datatype[size]

Operators:

Operators are the symbol used in the expression for evaluating them.

Types:

1. Arithmetic operator
2. Relational operator
3. Logical operator
4. Assignment operator
5. Increment operator
6. Decrement operator

Arithmetic Operator:

Arithmetic operator are used to perform basic arithmetic operations +, -, *, /, % are the operators.

www.Notesfree.in

www.Notesfree.in

ex:

```
class Arithmetic
{
 public static void main (String args[])
{
 int a=10, b=20, c;
 c=a+b;
 System.out.println(" Addition = "+c);
 }
}
```

Relational operator:

Relational operator establish the relation among the operators. The result is in Boolean value.

<, >, <=, >= are relational operators.

ex:

```
class Relational
{
 public static void main(String args[])
 {
 int a=10, b=20, c;
 if (a>b)
 {
  System.out.println("a is big");
 }
 else
 {
 System.out.println(" b is big");
 }
```

www.Notesfree.in

www.Notesfree.in

Logical operator:

Logical operators are used to co, ce two operands these two are of boolean operators.

ex:

```
class Logical
{
  public static void main (String args[])
  {
    boolean op1 = true, op2 = false;
    boolean ans1, ans2;
    ans1 = op1 & op2;
    ans2 = op1 | op2;
    System.out.println (+ans1);
    System.out.println (+ans2);
  }
}
```

Assignment Operator:

Assignment operator are used to assign a value to a variable. '=' equal to is an assignment operator.

Conditional operator:

The conditional operator is "?".

Syntax:

Condition ? statement 1; statement 2

www.Notesfree.in

www.Notesfree.in

Control statements:

Programmers can take decision in their program with the help of control statement.

i) if statement
ii) if - else
iii) while
iv) Do - while
v) switch
vi) for loop

If statement:

Types:

1. Simple
2. Compound

Syntax for simple:

```
if (condition)
statement;
```

Syntax for compound:

```
if (condition)
{
  statement 1;
  statement 2;
}
```

www.Notesfree.in

www.Notesfree.in

ex:

```
class demo
{
  public static void main (String args)
  {
   int a=10, b=5;
   if (a>b)
  System.out.println ("a is big");
  }
}
```

If-else statement:

syntax:

```
if (condition)
  statement;
else
  statement;
```

ex:

```
class demo
{
 public static void main (String args[])
 {
 int a=2, b=5;
 if(a>b)
  System.out.println ("a is big");
  else
  System.out.println (" b is big");
 }
}
```

www.Notesfree.in

www.Notesfree.in

if-else-if :

Syntax :

```
if (condition)
  statement;
elseif (condition II)
statement;
else
  statement;
```

ex :

```
class demo
{
public static void main (String args[])
{
int a=5, b=5;
if (a>b)
System.out.println("a is big");
else if (b>a)
System.out.println("b is big");
else
  System.out.println("Both are equal")
}
}
```

www.Notesfree.in

www.Notesfree.in

while statement :

Syntax:

```
while (condition)
{
  statement 1;
  :
  statement n;
}
```

ex:

```
class demo
{
 public static void main (String args[])
 {
  int count=1;
  int i=0;
  while (count <=2)
  {
   i=i+1;
   System.out.println (+i);
   count ++;
  }
 }
}
```

do-while statement :

Syntax:

```
do
{
  statement 1;
  :
  statement n;
}while (condition);
```

www.Notesfree.in

ex:

www.Notesfree.in

```
class Sample
{
 public static void main (String [] args)
 {
 int count =1, i =0;
 do
  {
   i = i+1;
   system.out.println ("+i);
   count++;
  }while (count <=3)
 }
}
```

For loop:

Syntax:

```
for (initialization ; condition ; inc/dec)
 {
   statement 1;
      :
   statement n;
 }
```

ex:

```
class Sample
{
 public static void main (String args[])
{ for (int i=0; i<=3; i++)
```

www.Notesfree.in

www.Notesfree.in

Switch case :

```
class Sample
{
 public static void main(String args[])
 throws java.io.IO.Exception
 {
  int choice;
  System.out.println("1. A");
  System.out.println("2. B");
  System.out.println("Enter choice");
  choice=(int) system.in.read();
  switch(choice)
  {
  case1:
   System.out.println("You selected A");
   break;
  case2:
   System.out.println("You selected B");
   break;
  default:
    System.out.println("None");
  }
 }
}
```

www.Notesfree.in

www.Notesfree.in

Programming Structures in Java.

Java tokens:

The smallest and logical, individual unit of the java statements are called tokens.

Types:

1. Reserved words
2. Identifiers
3. Literals
4. Operators
5. Separators.

Reserved words:

Keywords are reserved words.

int, float, char are keywords.

Identifiers:

Identifiers are defined by users. They are used for naming class, object, variables etc.

Literals:

Literals are used to store the sequence of characters for representing the constant values.

Types:

1. Integer
2. Floating Point
3. Boolean

www.Notesfree.in

www.Notesfree.in

4. Character

5. String

Constructors :

Syntax :

```
class Test
{
   Test ()
   {
   }
}
```

ex:

```
class Customer
{
  int km;
  int m;
  Distance (int k, int m)
  {
    km=k;
    meter=m;
  }
}

Public static void main (String args[])
{
 Distance shop = new Distance (1, 200);
 Distance hotel = new Distance (2, 300);
}
```

www.Notesfree.in

www.Notesfree.in

Access Specifiers:

Access modifiers control access to data fields, methods and classes.

Types:

1. Public
2. Private
3. Default modifier

Public:

Allows classes, methods and data fields accessible from any class.

Private:

Allows classes, methods and data field accessible from only within the own class

ex:

```
Package Test
Public class class1
{
  public int a;
  int b;
  private int c;
  public void fun1()
  {
  }
  void fun2()
  {
  }
Private void fun3()
```

www.Notesfree.in

www.Notesfree.in

```
    }
    }
}
Public class class 2
{
  void my-method ()
  {
  class 1 obj = new class 1();
  obj. a;
  obj. b;
  obj. c;
  obj. fun 1();
  obj. fun 2();
  obj. fun 3();
  }
}
Package another Test
public class class 2
{
  void my meth()
  {
  class 1 obj = new class 1();
  obj. a;
  obj. b;
  obj. c;
  obj. fun 1();
  obj. fun 2();
  }
}
```

www.Notesfree.in

www.Notesfree.in

Doc Comments :

Javadoc is a convenient, standard way to document your java code. Javadoc is actually a special format of comments

Types:

1. class level comments
2. Member level comments

Methods:

ex:

```
void get_data (int a, int b)
{
int c = a+b;
}
public class Method
{
public static void main (String args[])
{
Method m= new Method;
m sum(10,20);
}
void sum(int a, int b)
{
int c;
c = a+b;
System.out.println(c);
}
}
```

www.Notesfree.in

www.Notesfree.in

Static members:-

Static members are those members which can be accessed without using object

example:

```
Class ex
{
  Static int a=10;
  {
   System.out.println("a="+a);
  }
}

class anotherclass
{
  public static void main (String [] args)
  {
   System.out.println("a="+ ex.a);
  }
}
```

o/p

a =10

www.Notesfree.in

www.Notesfree.in

Unit-II

Inheritance, Packages and interfaces.

Overloading methods - objects as parameters - returning object - static, Nested and Inner classes. Inheritance Basics - Types of inheritance - super keyword - method Overriding - Dynamic method dispatch - Abstract classes - final with inheritance. Packages and interfaces: Packages - packages and member access - Importing packages - Interfaces.

Overloading method:

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

ex:

```
int sum( int a, int b);
double sum (double a, double b);
int sum (int a, int b, int c);
```

www.Notesfree.in

www.Notesfree.in

object as parameters :-

Call by value

Call by reference.

Call by value:

The value of the actual argument is assigned to the formal parameter.

```
public class ObjDemo
{
int height;
    int width;
    ObjDemo (int h, int w)
    {
    height = h; width = w;
    }

 Void area(ObjDemo o)
 {

 int result = (height + o.height)*(width +
                                  o.width);
 System.out.println("The area is" + result)
 }
}
class Demo
{
 public static void main (String args[])
```

www.Notesfree.in

www.Notesfree.in

```
}
objDemo obj1 = new objDemo (2,3);
objDemo obj2 = new objDemo (10,20);
obj1.area (obj2);
}
}
```

O/P

The area is 276

class

call by reference:

For passing the parameter by reference we pass the object of the class as a parameter.

ex:

```
public class objRetDemo{
int a;
objRetDemo (int val)
{
 a = val;
}
objRetDemo fn()
{
 objRetDemo temp = new objRetDemo (a+5);
 return temp;
}
}
```

www.Notesfree.in

www.Notesfree.in

```
class ObjRet {
  public static void main (String args[]
  {
   ObjRetDemo obj2 = new ObjRetDemo(20);
   ObjRetDemo obj1;
   obj1 = obj2.fn();
   System.out.println("The returned value is
                        = "+obj1.a);
  }
}
```

output:

The returned value is = 25.

Static, Nested and Inner classes.

We can declare a class static by using the static keyword. A class can be declared static only if it is a nested class.

Inner class:

The classes that are non-static and nested are called inner classes.

Outer class:

The class in which nested class is defined is called outer class.

www.Notesfree.in

www.Notesfree.in

Nested class:

A class within a class is known as nested class.

```
public class Class1
{
 static class Class 2
 {
  public void fn()
  {
   System.out.println("This a static nested
                                  class");
  }
 }
public static void main (String args [])
{
class1.class2 obj = new Class1.Class2();
obj.fn();
}
}
```

o/p

This is a static nested class.

www.Notesfree.in

www.Notesfree.in

Inner classes:

Syntax:

```
{
 . . . .
 Access - modifier class InnerClass
 {
  . . . .
  }
}
```

Types:

→ static member classes
→ Member classes
→ Local classes
→ Anonymous classes.

ex:

```
class MyInnerClass implements Runnable
 {
 public void run()
 {
 System.out.println("Hello");
 }
class DemoClass
{
 public static void main (String [] args)
 {
 MyInnerClass my= new MyInnerClass();
Thread th = new Thread (my);
 my.start();
 }
}
```

www.Notesfree.in

www.Notesfree.in

Inheritance :

Inheritance is a mechanism in Java by which derived class can borrow the properties of base class.

Advantages :

* Reusability
* Extensibility
* Data hiding
* Overriding.

Types:

Single inheritance:

There is one parent per derived class.

Subclass can be defined as:

```
class name of subclass extends superclass
{
  variable declaration
  method declaration
}
```

Multiple inheritance :

The derived class is derived from more than one base class.

www.Notesfree.in

www.Notesfree.in

Multilevel inheritance:

When a derived class is derived from a base class which itself is a derived class then it is multilevel inheritance.

Hybrid inheritance:

When two or more types of inheritance is combined together then it forms the hybrid inheritance.

Super Keyword:

It is used to access the immediate parent class from subclass.

Three ways to use super keyword

* Variable invocation
* Method invocation
* Constructor invocation

ex:

```
class A
{
    int x=10;
}
class B extends A
{
    int x=20;
    void display()
```

www.Notesfree.in

www.Notesfree.in

```
    }
    public void main (String args[])
    {
        B obj = new B();
        obj.display();
    }
}
```

Method overriding:

Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

Dynamic method dispatch:

It is run time polymorphism in which a call to overriden function is resolved at runtime instead of at compile time.

Syntax:

Super class refvariable = new subclass_obj();

www.Notesfree.in

www.Notesfree.in

ex: for method Overriding & dynamic method dispatch:

```
class x
{
    void show()
    {
        system.out.println("x class");
    }
}
class Y extends x
    void show()
    {
        System.out.println("Y class")
    }
}
class override
{
    Public static void main (String[] args)
    Y obj = new Y();
    obj.show();
}
```

O/P Y class

Now,

```
x ref = new y()
ref. show();
```

→ dynamic method dispatch

www.Notesfree.in

www.Notesfree.in

ract classes:

* Abstract class specifies only the member function.

Properties:

* Abstract method must be present in abstract class only.
* Abstract class cannot be instantiated using new operator.
* Abstract class must be parent class.
* Abstract method has no definition part.

Example:

```
abstract class A
{
    abstract void fun1();  <- This is an abstract method
    void fun2()
    {
        System.out.println("A: fn fun2");
    }
}
class B extends A
{
    void fun1()
    {
        System.out.println("B: fn fun1");
    }
}
class C extends A
{
    void fun1()
    {
        System.out.println("C: fn fun1");
    }
}
```

www.Notesfree.in

www.Notesfree.in

```
public class Demo
{
    public static void main(String[]
    {
        B b = new B();
        C c = new C();
        b.fun1(); // calls method of class B
        b.fun2();
        c.fun1(); //calls method of class C
        c.fun2();
    }
}
```

Output:

B: In fun 1
A : In fun 2
C : In fun 1
A : In fun 2

Final with Inheritance

Final keyword can be applied at 3 places.

* For declaring variables.
* For declaring the methods
* For declaring the classes.

Final variables and methods:

The final keyword can also be applied to method. On applying, method overriding is avoided. The method declared with final keyword cannot be

www.Notesfree.in

www.Notesfree.in

```
        final void fun()
        {
            System.out.println("Hello!");
        }
    }
```

Output :

1 error

cannot inherit from final test.

Packages :

Package is a mechanism in which variety of classes and interface can be group together.

Syntax :

Package name_of_package.

Eg :

```
Package My_package
Public class A
{
    int a;
    public void set_val(int n)
    {
        a=n;
    }
    Public void display()
    {
        System.out.println(+a);
    }
}
```

www.Notesfree.in

www.Notesfree.in

overriden.

```
class Test
{
    final void fun()
    {
        System.out.println("Hello");
    }
}
class Test1 extends Test
{
    final void fun()
    {
        System.out.println("Hello1");
    }
}
```

Output:

1 error

fun() in Test1 cannot override fun() in Test.

Final classes to stop Inheritance:

If we declare class as final, no class can be derived from it.

```
final class Test
{
    void fun()
    {
        System.out.println("Hello");
    }
}
class Test1 extends Test
{
```

www.Notesfree.in

www.Notesfree.in

```
import my package A
class package demo
{
    Public static void main(String[] args)
    {
        A obj = new A();
        obj.set_val(10);
        obj.display();
    }
}
```

INTERFACES :

Java does not support more than one superclass. Java does not implement multiple inheritance diredly, but make use of this concept using interfaces.

Defining an Interface :

Interface is a kind of class. Like classes, Interface contains methods and variables, but the method are not defined.

Interface define only abstract method and final fields.

Syntax :

```
interface interface_name
{
    variable decleration;
    Methods decleration;
}
```

www.Notesfree.in

www.Notesfree.in

Variables are declared as
static final type variable name

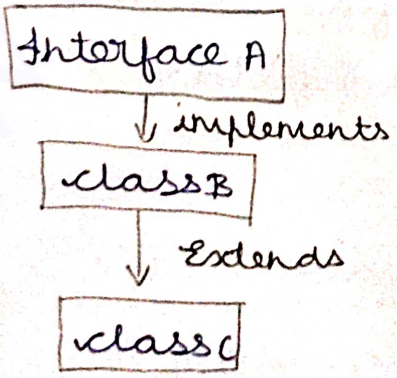
Eg: Static final int X = 10;

Difference between class and Interface:

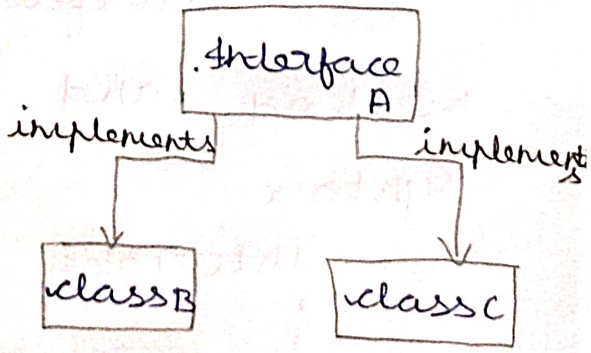
Class	Interface
* Class is denoted by keyword class.	* Interface is denoted by keyword interface.
* Instance of class can be created.	* Cannot create instance of class.
* Public, Private & Protected access specifiers can be used.	* Only public access specifiers is used.
* Class contains data Member & Methods. But methods are defined.	* Interface contain data members and methods. But methods are not defined.
* Data members can be constant or final	* Data members are always declared as final.

Implementing Interface:

Various way of interface implementation:



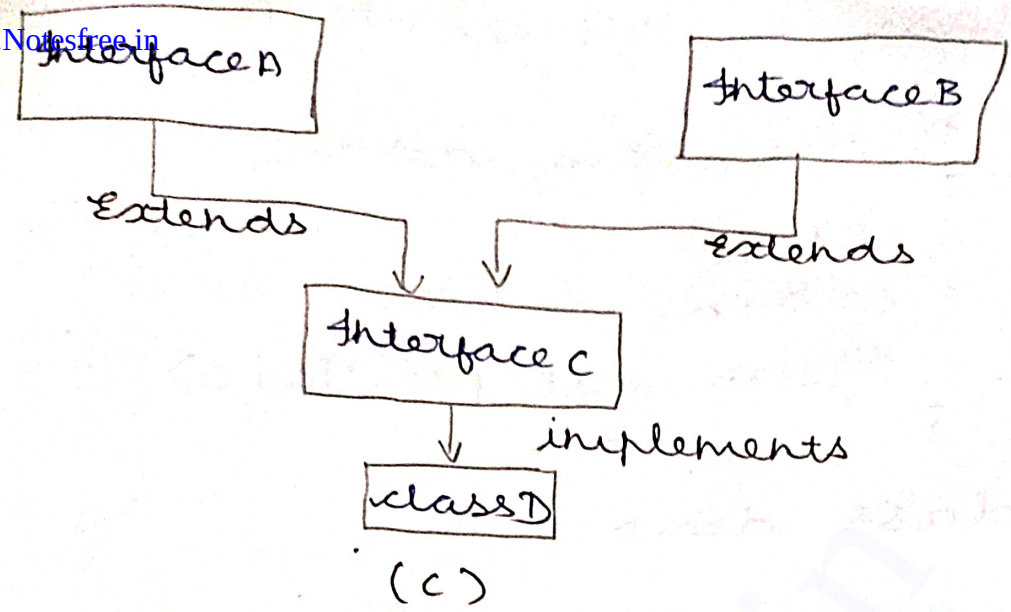
(a)



(b)

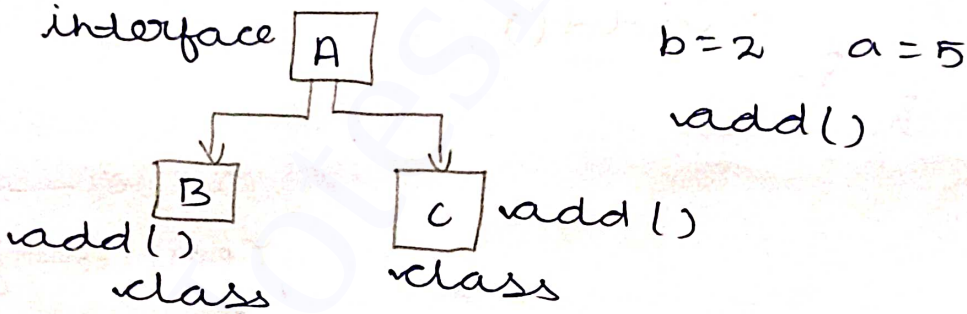
www.Notesfree.in

www.Notesfree.in



(c)

class - class → extends
interface - interface → extends
class - interface → implements
interface - class → implements.



Eg:

```
interface A
{
    void add();
    int a=5 , b=2 ;
}
class B implements A
{
   Public void add()
   {
      int c;
      c=a+b ;
      System.out.println(c);
   }
}
```

www.Notesfree.in

www.Notesfree.in

```
class C implements A
{
    Public void add()
    {
        int c;
        c=a+b;
        System.out.println(c);
    }
}
class demo
{
    Public static void main(String args[])
    {
        B b=new B();
        C c =new C();
        b.add(); ->7
        c.add(); ->7
    }
}
```

Save → demo.java
Compile → javac demo.java
Run → java demo

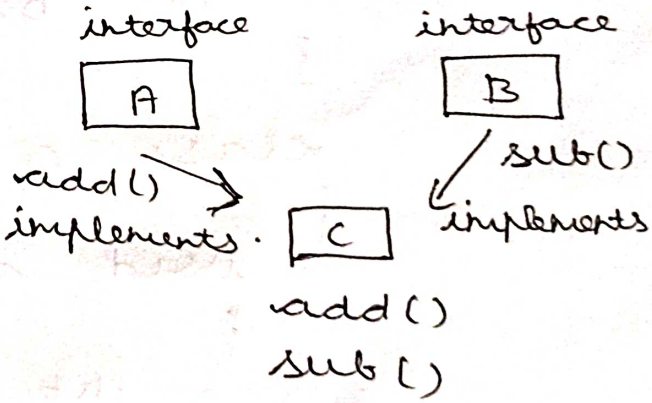
Multiple inheritance (using interface)

```
interface A
{
    void add();
    int a=5 , b=2;
}
interface B
{
    void sub();
    int a=5,b=2;
}
class C implements A,B
{
    Public void add()
    {
```

www.Notesfree.in

www.Notesfree.in

```
        int c ;
        c = a + b ;
        System . out . println(c);
    }
    Public void sub()
    {
        int c ;
        c = a - b ;
        System . out . println(c);
    }
}
class demo
{
    Public static void main(String args[])
    {
        C c = new c();
        c.add();
        c.sub();
    }
}
```



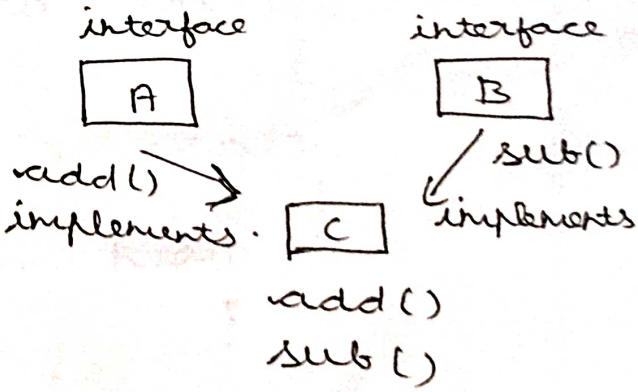
Extending Interface :

Interface are extended similar to classes. We can derive sub interfaces from main interfaces by using keyword extends.

www.Notesfree.in

www.Notesfree.in

```
        int c ;
        c = a + b ;
        System . out . println(c);
    }
    Public void sub()
    {
        int c ;
        c = a - b ;
        System . out . println(c);
    }
}
class demo
{
    Public static void main(String args[])
    {
        C c = new c();
        c.add();
        c.sub();
    }
}
```

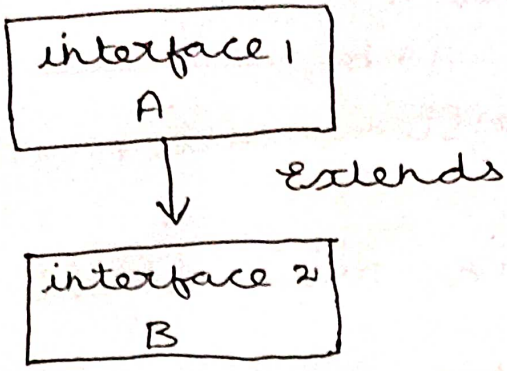


Extending Interface :

Interface are extended similar to classes. We can derive sub interfaces from main interfaces by using keyword extends.

www.Notesfree.in

www.Notesfree.in



Syntax :

```
interface  Interface_name2 extends interface_name1
{
    // body of interface
}
```

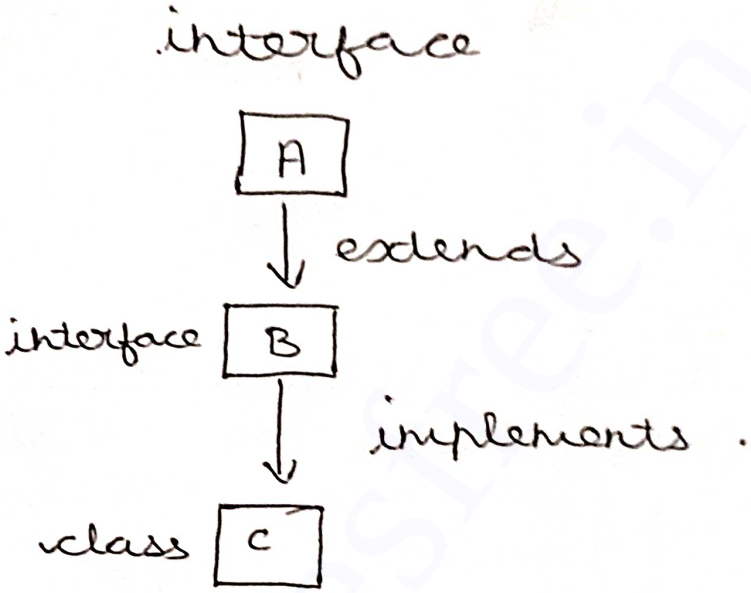
Example :

```
interface A
{
    int a=5;
}
interface B extends A
{
    int b=2;
}
class C implements B
{
    void add()
    {
        int c;
        c=a+b;
        System.out.println(c);
    }
}
class demo
{
    Public static void main(String
                                 args[])
```

www.Notesfree.in

www.Notesfree.in

```
{
    C c = new c();
    c.add();
}
}
```



www.Notesfree.in

www.Notesfree.in

UNIT III - EXCEPTION HANDLING AND MULTITHREADING

Exception Handling basics - Multiple catch clauses - Nested try Statements - Java's Built-in Exceptions - User defined Exception. Multithreaded Programming : Java Thread Model - Creating a Thread and Multiple Threads - Priorities - Synchronization - Inter Thread Communication - Suspending - Resuming and Stopping Threads - Multithreading Wrappers - Auto boxing.

1. EXCEPTION HANDLING BASICS :

Exception Handling is a mechanism in which the statements that are likely to cause an exception are enclosed within 'try' block. As soon as exception occurs it is handled using 'catch' block.

Thus exception handling is a mechanism ~~which~~ that prevents the program from crashing when some unusual situation occurs.

Types of Exception :

1. Checked Exception - Need to be handled explicitly by the code itself either by using 'try - catch' block or by using 'throws'.

2. Unchecked Exception - Need not be handled explicitly. The JVM handles it.

www.Notesfree.in

www.Notesfree.in

Keywords used in Exception Handling:

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

try - catch block:

→ The statements that are likely to cause an exception are enclosed within a <u>try</u> block.

→ The <u>catch</u> block is responsible for handling the exception throwm by the try block.

Syntax:

```
try
{
        //code
}
catch (exception)
{
        //code
}
```

finally block:

It specifies the code that must be executed even though exception may or may not occur.

Syntax:

```
finally
{
        // clean up code
}
```

www.Notesfree.in

www.Notesfree.in

Example:

```
Public class Example
{
    Public static void main (String[] args)
    {
        try
        {
            int a = 5/0;
        }
        catch (Exception e)
        {
            System.out.println("Can't divide by zero");
        }
        finally
        {
            System.out.println("Final block executed");
        }
    }
}
```

Output:

Can't divide by zero

Final block executed.

Program Explanation:

On occurance of 'divide by zero' exception in the try block, the control goes to the catch block and the exception gets caught and the statement will be printed as output. The finally block always execute. to free the resources.

www.Notesfree.in

www.Notesfree.in

throw:

→ 'throw' keyword is used for throwing the exception.

→ It is normally used within a method.

→ we cannot throw multiple exceptions using throw.

Syntax:

throw new Throwable's subclass

Example:

```
Public class Example
{
    Static void checkAge (int age)
    {
        if (age < 18)
        {
            throw new ArithmeticException ("can't vote");
        }
        else
        {
            System.out.println ("can vote");
        }
    }
    Public static void main (String[] args)
    {
        checkAge (15);
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException:
                                        can't vote
        at Example.checkAge (Example.java:4)
        at Example.main (Example.java:12)
```

www.Notesfree.in

www.Notesfree.in

throws:

→ When a method wants to throw an exception then keyword 'throws' is used.

→ 'Throws' is used with method signature.

→ We can declare multiple exceptions using 'throws'.

Syntax:

```
methodname(Parameters) throws exceptionlist
{
    // method code
}
```

Example:

```
Public class Example
{
  Static void checkAge (int age) throws ArithmeticException
  {
      if (age < 18)
      {
          throw new ArithmeticException ("can't vote");
      }
      else
      {
          System.out.println(" can vote");
      }
  }
  public Static void main (string[] args)
  {
      checkAge (15);
  }
}
```

www.Notesfree.in

www.Notesfree.in

Program Explanation:

Throw an exception if age is below 18 (Print "can't vote"). If age is 18 or older, Print "can vote".

The 'throws' keyword indicates what exception type may be throwm by a method.

2. MULTIPLE CATCH CLAUSES:

→ It is not possible for the try block to throw a single exception always.

→ Different exceptions may be raised by a single try block statements.

→ Depending upon the type of exception thrown, it must be caught.

→ To handle such situation, multiple catch blocks may exist for the single try block statements.

Syntax:

```
try
{
    // Exception occurs
}
catch (Exception_type e)
{
    // Exception is handled here
}
...
catch (Exception_type e)
{
    //Exception is handled here
}
```

www.Notesfree.in

www.Notesfree.in

Example:

```
~~Public~~ class Example
{
    Public static void main (string args[ ])
    {
        try
        {
            int a[] = new int [5];
            a[5] = 30/0;
            System.out.println (a[10]);
        }
        catch (ArithmeticException e)
        {
            system.out.println ("Arithmetic Exception occurs");
        }
        catch (ArrayIndexOutofBoundException e)
        {
            system.out.println ("AIoutofBound Exception occurs");
        }
        catch (Exception e)
        {
            System.out.Println ("Parent Exception occurs");
        }
        System.out.Println ("rest of the code");
    }
}
```

Output:

Arithmetic Exception occurs

rest of the code.

Explanation:

Here, 30/0 throws arithmetic exception, so it is caught by the first catch block. After handling exception by the catch block, statements

www.Notesfree.in

www.Notesfree.in

after try-catch block has been executed.

3. NESTED TRY STATEMENTS:

→ when a try block is defined within another try. it is called nested try block.

→ when there are chances of occuring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.

Syntax:

```
try
{
    Statements;
    try
    {
        statements;
    }
    catch (Exception1 e1)
    {
        Statements;
    }
}
catch (Exception2 e2)
{
    statements;
}
```

Inner try-catch

Outer try-catch

www.Notesfree.in

www.Notesfree.in

Example:

```
Class Example
{
    Public static void main(String args[])
    {
        try
        {
            int a[] = new int[10];
            System.out.println(a[12]);
            try
            {
                int res = 5/0;
            }
            catch(ArithmeticException e1)
            {
                System.out.println("Divide by 0 is ∞");
            }
        }
        catch(ArrayIndexOutofBoundException e2)
        {
            System.out.println("ArrayIndex Out of Bound");
        }
    }
}
```

Output:

Array Index Out of Bound

Explanation:

The outer try code tries to display the element at index 12 which gives rise to ArrayIndexOutofBoundException. Since the outer catch block handles this exception, we get the above output.

www.Notesfree.in

www.Notesfree.in

4. JAVA'S BUILT-IN EXCEPTIONS:

→ Exceptions that are already available in Java Libraries are referred to as built-in exception.

→ Java defines several exception classes inside the standard package java.lang.

→ It can be categorized into two broad categories

1. Checked exceptions
2. Unchecked exceptions

→ Some of the common checked exceptions are classNotFoundException, IOException, FileNotFoundException etc,

→ Some of the common Unchecked exceptions are ArithmeticException, ArrayIndexOutofBoundEXception, NullPointerException etc.

Examples:

1. Arithmetic Exception:

It handles arithmetic exceptions such as 'divide by zero'.

```
Class Example
{
    Public Static void main (string args[])
    {
        try
        {
            int a = 30, b = 0;
            int c = a/b;
            System.out.Println (" Result = " + c);
        }
```

www.Notesfree.in

www.Notesfree.in

```
        catch (ArithmeticException e)
        {
            System.out.println(" can't divide by zero");
        }
    }
}
```

Output:

Can't divide by zero.

2. ArrayIndexOutofBoundException:

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

```
Class Example
{
    Public static void main(String args[])
    {
        try
        {
            int a[] = new int[5];
            a[6] = 9;   //accessing 7th element in an
                        //array of size 5.
        }
        catch (ArrayIndexOutofBoundException e)
        {
            System.out.println("Array Index is out of Bound");
        }
    }
}
```

Output:

Array Index is Out of bound.

www.Notesfree.in

www.Notesfree.in

3. FileNotFound Exception:

This exception is raised when a file is not accessible or does not open.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
class Example
{
    Public static void main (String args[])
    {
        try
        {
            File file = new File ("E://file.txt");
                                    //This file does not exist
            FileReader fr = new FileReader (file);
        }
        Catch (FileNotFoundException e)
        {
            System.out.println ("File does not exist");
        }
    }
}
```

Output:

File does not exist.

4. IOException:

It is throwm when an input-output operation failed or interrupted.

www.Notesfree.in

www.Notesfree.in

```
import java.io.*;

class Example
{
    public static void main(String args[])
    {
        FileInputStream f = null;
        f = new FileInputStream("abc.txt");
        int i;
        while ((i = f.read()) != -1)
        {
            System.out.Print((char)i);
        }
        f.close();
    }
}
```

Output:

error: unreported exception IOException; must be caught or declared to be throwm.

5. class NotFound Exception:

This exception is raised when we try to access a class whose definition is not found.

6. Null Pointer Exception:

This exception is raised when referring to the members of a null object. Null represents nothing.

www.Notesfree.in

www.Notesfree.in

```
class Example
{
    Public static void main (String args[])
    {
        String a = null;
        System.out.Println(a.charAt(0));
    }
}
```

Output:

Exception in thread "main" java.lang.NullPointerException

5. **USER DEFINED EXCEPTION:**

[Creating own Exception or custom Exception]

→ In Java, we can create our own exception class and throw that exception using 'throw' keyword. These exceptions are known as user-defined or custom exceptions.

→ The syntax for throwing out own exception is

throw new Throwable's subclass

→ Here the Throwable's subclass is actually a subclass derived from the Exception class.

Example:

```
throw new ArithmeticException();
```

↑ Throwable's subclass.

www.Notesfree.in

www.Notesfree.in

User-defined exception must extend Exception class.

Example Program:

```
import java.lang.Exception;
class MyException extends Exception
{
    MyException(String msg)
    {
        Super(msg);
    }
}
Class Example
{
    Public static void main(String args[])
    {
        int age = 15;
        try
        {
            if (age < 21)
            throw new MyException ("Age is less than condition");
        }
        catch (MyException e)
        {
            System.out.println("This is my Exception block");
            System.out.println (e.getMessage());
        }
    }
}
```

Output:

This is my Exception block.

Age is less than condition.

www.Notesfree.in

www.Notesfree.in

Explanation:

In the above code, the age value is 15 and in the try block - the 'if' condition throws the exception if the value is less than 21. As soon as the exception is thrown, the catch block gets executed. Hence as an output we get the message "This is my Exception block".

Then the control is transferred to the class MyException. The message is set and then printed by the catch block using the system.out.print statement by means of e.message.

www.Notesfree.in

www.Notesfree.in

6. MULTITHREADED PROGRAMMING:

→ In Java, we can write the programs that perform multitasking using the multithreading concept.

→ Thus Java allows to have multiple control flows in a single program by means of multithreading.

7. JAVA THREAD MODEL:

→ Thread is a tiny program running continuously. It is sometimes called as 'Light-weight process'.

Thread	Process
* Thread is a light-weight process * Do not require separate address space for execution.	* Process is a heavy weight process. * Requires separate address space to execute

Thread Life cycle:

→ Thread life cycle specifies how a thread gets processed in the Java program, by executing various methods.

→ During the life time of a thread, it can enter many states like new, ready (or) runnable, running, blocked (or) wait and dead (or) terminated state.

www.Notesfree.in

www.Notesfree.in

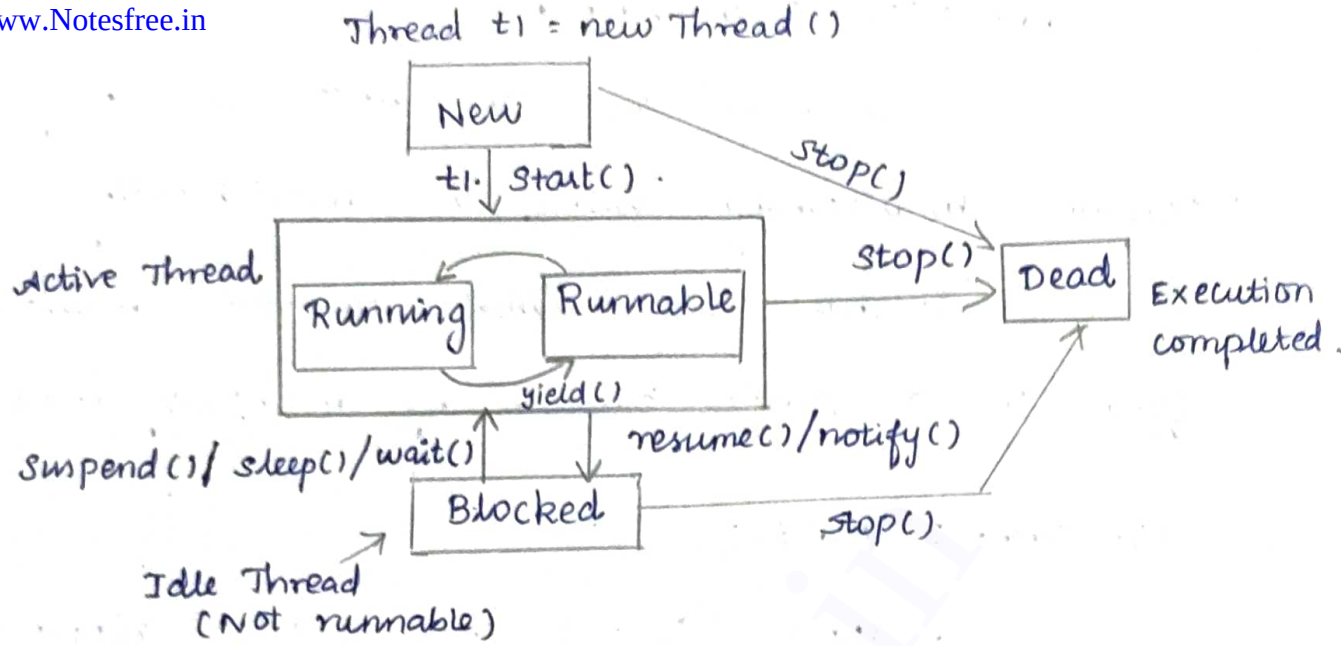


Fig: Thread Life cycle.

(i) New State:

→ When a thread object is created, then the thread is said to be in New state.

→ This state is also known as Born state.

```
Thread t1 = new Thread();
```

(ii) Runnable state:

→ This is a state in which a thread is ready for execution and is waiting for the availability of processor.

```
t1.start();
```

→ If all the threads that are waiting have equal priority, CPU allocates time slots for thread execution on the basis of first-come, first-serve manner.

www.Notesfree.in

www.Notesfree.in

→ The process of allocating time to threads is known as time slicing.

→ If a thread wants to give the control to another thread, it is done by using yeild() method.

(iii) Running state:

→ when the thread gets the CPU, it moves from the runnable to the running state.

→ The run() method of a thread called automatically by the start() method.

→ A thread can change its state to Runnable, Dead or Blocked from running state depending upon time slicing, thread completion of run() method or waiting for some resources.

(iv) Blocked State:

→ A thread in running state may move into the blocked state due to various reasons like sleep() method called, wait() method called, suspend method called etc.

```
Thread.sleep(1000);
wait(1000);
```

www.Notesfree.in

www.Notesfree.in

→ From blocked state, a thread can move back to runnable state when sleep time completed, waiting time completed, resume() method called etc.,

t1.resume();

(v) Dead State:

→ The dead state is also known as terminated state.

→ After successful completion of the execution, the thread in runnable state enters the terminated state.

→ A thread can also be killed by using stop() method.

During thread life cycle, a thread moves from one state to another state in a variety of ways. This is because in multithreading environment, when multiple threads are executing, only one thread can use CPU at a time. All other threads live in some other states.

www.Notesfree.in

www.Notesfree.in

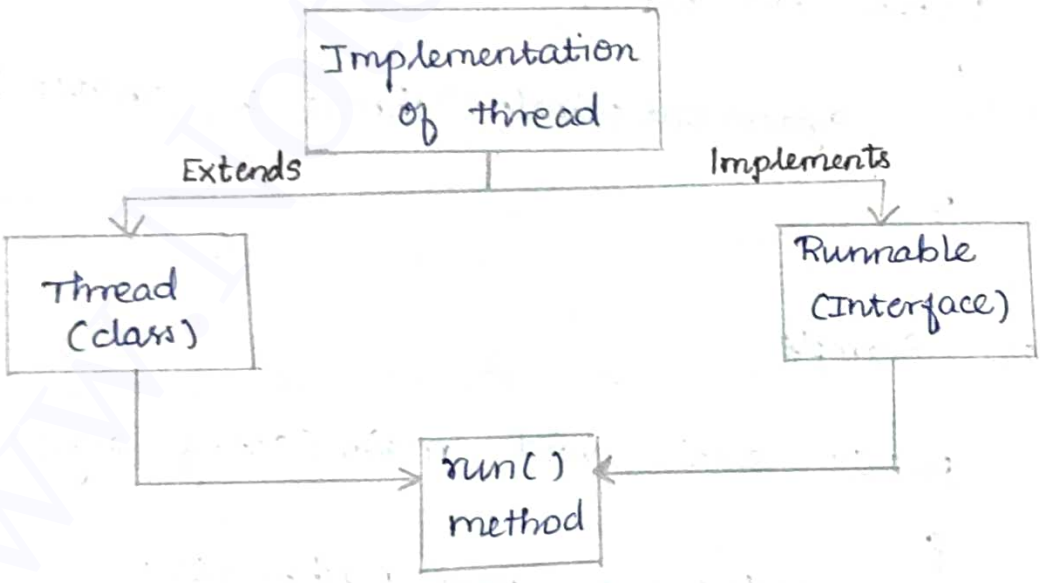
8. CREATING A THREAD AND MULTIPLE THREADS:

Creating a Thread:

→ When we run a java Program, the Program begins to execute its code starting from the main method. Therefore the JVM creates a thread to start executing the code present in main method. This thread is called as main thread.

→ In Java we can implement the thread programs using two ways

1. Using Thread class
2. Using Runnable interface.



→ run() method is the most important in threading Program. By using this, the thread's behaviour can be implemented.

```
Public void run()
{
    // Stmts for implementing thread
}
```

www.Notesfree.in

www.Notesfree.in

For invoking run(), thread object is needed. It can be obtained by creating & initiating a thread using the start() method.

(i) Extending Thread class.

Threads can be created using the Thread class. Using 'extend' keyword, your class ~~can~~ extends the Thread class.

Ex: class A extends Thread

Program:

```
class MyThread extends Thread
{
    Public void run()
    {
        System.out.println("Thread is created");
    }
}
class Example
{
    Public static void main(String args[])
    {
        MyThread t = new MyThread();
        t.start();
    }
}
```

Output:

Thread is created

www.Notesfree.in

www.Notesfree.in

Explanation:

In the above program, we have created two class named MyThread and Example.

MyThread extends the Thread class. It has run() method which is called by t.start() in main method of class Example.

The thread gets created and executes by displaying the given message.

(ii) Implementing Runnable Interface:

Implementing thread Program using Runnable Interface is Preferable because

1. If a class extends a thread class, then it cannot extends any other classes which are needed

2. If Thread class is extended, then all its functionilities get inherited. This is expensive operation.

Program:

```
class MyThread implements Runnable
{
    Public void run()
    {
        System.out.Println("Thread is created");
    }
}
class Example
{
    Public static void main(String args[])
    {
```

www.Notesfree.in

www.Notesfree.in

```
        MyThread Obj = new MyThread();
        Thread t = new Thread(obj);
        t.Start();
    }
}
```

Output:

Thread is Created.

Explanation:

→ While using the interface, it is necessary to use 'implements' keyword.

→ Inside main method

* create instance of class MyThread & pass this as a parameter to Thread class.

* Using the instance of class Thread, invoke the start() method. The Start() method in-turn calls the run() in class MyThread. Then it executes and displays the given message.

CREATING MULTIPLE THREADS:

Multiple threads can be created both by extending Thread class and by implementing Runnable interface.

www.Notesfree.in

www.Notesfree.in

Program

Extending Thread class

```
class A extends Thread
{
    Public void run()
    {
        for(int i=0; i<=5; i++)
        {
            System.out.println(i);
        }
    }
}
class B extends Thread
{
    Public void run()
    {
        for (int i=10; i>=5; i--)
        {
            System.out.println(i);
        }
    }
}
Class Example
{
    Public static void main
                (String args[])
    {
        A t1 = new A();
        B t2 = new B();
        t1.start();
        t2.start();
    }
}
```

Output:
0
1
2
3
4
5
10
9
8
7
6
5

Implementing Runnable Interface

```
class A implements Runnable
{
    Public void run()
    {
        for (int i=0; i<=5; i++)
        {
            system.out.println(i);
        }
    }
}
class B implements Runnable
{
    Public void run()
    {
        for (i=10; i>=5; i--)
        {
            system.out.println(i);
        }
    }
}
class Example
{
    Public static void main (String
                    args[])
    {
        A obj1 = new A();
        B obj2 = new B();
        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);
        t1.start();
        t2.start();
    }
}
```

Output:
0
1
2
3
4
5
10
9
8
7
6

www.Notesfree.in

www.Notesfree.in

9. THREAD PRIORITIES:

→ Thread Priority is a number assigned to a thread that is used by Thread scheduler to decide which thread should be allowed to execute

→ Each thread have a priority. Priorities are represented by number between 1 and 10.

→ In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling).

→ But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

→ Three constants defined in Thread class are

- Public static int MIN_PRIORITY
- Public static int NORM_PRIORITY
- Public static int MAX_PRIORITY

→ Default priority of a thread is 5 (NORM_PRIORITY)

→ The value of MIN_PRIORITY is 1

→ The value of MAX_PRIORITY is 10.

Methods used:

→ getPriority() method returns priority of given thread.

→ setPriority(int newPriority) method changes the priority of thread to the value newPriority.

→ Void start() - to start the specified thread

→ Boolean isAlive() - Returns true if thread is started

www.Notesfree.in

www.Notesfree.in

and not died.

→ void run() - This method encapsulates the functionality of a thread.

Example: (using Thread class)

```
class A extends Thread
{
    Public void run()
    {
        for(int i=1; i<=3; i++)
        {
            System.out.println("From thread A: i =" + i);
        }
        System.out.println("exit from A");
    }
}
class B extends Thread
{
    Public void run()
    {
        for(int j=1; j<=3; j++)
        {
            System.out.println("From thread B: j =" + j);
        }
        System.out.println("exit from B");
    }
}
class Main
{
    Public static void main(String args[])
    {
        A ta = new A();
        ta.setPriority(Thread.MIN_PRIORITY+1);
```

www.Notesfree.in

www.Notesfree.in

```
        B tb = new B();
        tb.setPriority(6);
        ta.start();
        tb.start();
    }
}
```

Output:

From thread B: j = 1

From thread B: j = 2

From thread B: j = 3

exit from B

From thread A : i = 1

From thread A : i = 2

From thread A : i = 3

exit from A

Explanation:

→ Suppose there are two threads ta and tb with priorities 2 (MIN-PRIORITY+1) and 6, the thread tb will execute first based on maximum priority 6. After that thread ta will execute.

→ The above Program can also be implemented using interface.

~~Disadvantage~~:

→ Thread Priority does not guarantee execution order. They are highly OS specific.

www.Notesfree.in

www.Notesfree.in

10. SYNCHRONIZATION:

→ The process of ensuring one access at a time by one thread is called synchronization.

→ There are two ways to achieve the synchronization.

1. Using synchronized Methods
2. Using Synchronized Blocks (statements)

1. synchronized method:

→ If we declare any method as synchronized, it is known as synchronized method.

→ Synchronized method is used to lock an object for any shared resource.

→ When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Example:

```
class Table
{
    Synchronized void PrintTable(int n)
    {
        System.out.println("TABLE" +n);
        for(int i=1; i<=5; i++)
        {
            System.out.println(n*i);
        }
    }
}
```

www.Notesfree.in

www.Notesfree.in

```
Public class Main
{
    Public static void main (String[] args)
    {
        Table t = new Table ();
        Thread t1 = new Thread ()
        {
            Public void run()
            {
                t.printTable (5);
            }
        };
        Thread t2 = new Thread ()
        {
            Public void run()
            {
                t.printTable (10);
            }
        };
    t1.start();
    t2.start();
    }
}
```

Output

```
TABLE 5
5
10
15
20
25
TABLE 10
10
20
30
40
50
```

Explanation :

In the above program, we have created two threads t1 and t2. PrintTable() method is invoked with parameters 5 and 10 respectively. This method executes thread t1 at first and then t2 in synchronized manner.

www.Notesfree.in

www.Notesfree.in

2. Synchronized Block:

→ Synchronized block can be used to perform synchronization on any specific resource of the method.

→ Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

Example:

```
Class Table
{
    Void PrintTable (int n)
    {
        Synchronized (this)
        {
            System.out.println ("TABLE" + n);
            for (int i = 1; i <= 5; i++)
            {
                system.out.println (n * i);
            }
        }
    }
}
Public Class Main
{
    Public Static void main (String[] args)
    {
        Table t = new Table ();
        Thread t1 = new Thread ()
        {
            Public void run()
            {
                t.PrintTable (5);
            }
        };
```

← This is a synchronized block

www.Notesfree.in

www.Notesfree.in

```
        Thread t2 = new Thread()
        {
            public void run()
            {
                t.printTable(10);
            }
        };
        t1.start();
        t2.start();
    }
}
```

Output:

```
TABLE 5
5
10
15
20
25
TABLE 10
10
20
30
40
50
```

→ If we put all the codes of the method in the synchronized block, it will work same as synchronized method.

Advantages & Limitations:

* Synchronization is a good way to achieve mutual exclusion.
* All methods of a class need not be synchronized.
* we cannot synchronize the constructors.
* object has one lock. Thread can acquire more than one lock.
* Java synchronization does not allow concurrent reads.
* Synchronized methods run very slowly and can degrade the performance. So synchronize the method only when it is absolutely necessary.

www.Notesfree.in

www.Notesfree.in

11. INTER - THREAD COMMUNICATION:

→ Two or more threads communicate with each other by exchanging the messages. This mechanism is called inter-thread communication.

→ There are three in-built methods that take part in inter-thread communication.

1. notify() - If a particular thread is in the sleep mode, then that thread can be resumed using the notify call.

2. notifyall() - This method resumes all the thread that are in suspended state.

3. wait() - The calling thread can be send into a sleep mode.

Example:

```
class myclass
{
    int val;
    boolean flag = false;
    synchronized int get()
    {
        if(!flag)
        try
        {
            wait();
        }
        catch (InterruptedException e)
        {
            System.out.println("Interrupted Exception!");
        }
```

www.Notesfree.in

www.Notesfree.in

```
        System.out.println("consumer consuming:" + val);
        flag = false;
        notify();
        return val;
    }
    synchronized void Put(int val)
    {
        if(flag)
        try
        {
            wait();
        }
        catch(InterruptedException e)
        { }
        this.val = val;
        flag = true;
        System.out.println("Producer Producing:" + val);
        notify();
    }
}
class Producer extends Thread
{
    myclass t1;
    Producer(myclass t)
    {
        t1 = t;
    }
    Public void run()
    {
        for(int i = 0; i < 5; i++)
        {
            t1.put(i);
        }
    }
}
```

Producer thread is writing the numbers from 0 to 4

www.Notesfree.in

www.Notesfree.in

```
class consumer extends Thread
{
    myclass t2;
    consumer(myclass t)
    {
        t2 = t;
    }
    Public void run()
    {
        for(int i=0; i<3; i++)
        {
            t2.get();
        }
    }
}
class Interthread
{
    Public Static void main(String[] args)
    {
        myclass obj = new myclass();
        Producer P = new Producer(obj);
        consumer C = new consumer(obj);
        P.start();
        C.start();
    }
}
```

consumer thread is reading the numbers from 0 to 3

Output:

Producer producing: 0
Consumer consuming: 0
Producer Producing: 1
Consumer consuming: 1
Producer Producing: 2
Consumer Consuming: 2

www.Notesfree.in

www.Notesfree.in

Explanation:

The above program has two threads producer and consumer.

• Inside get() → The wait() suspends the execution, by that time producer writes the value. when the data is ready, it notifies other thread.

→ When the consumer reads the data, execution inside get() is suspended. After the data is obtained, get() calls notify().

→ This tells the Producer can write new data in the queue.

• Inside put() → The wait() suspends the execution, by that time, the consumer removes the item from queue.

→ when execution resumes, the next item is put in the queue and notify() is called.

→ when notify() is issued, the consumer can remove the corresponding item for reading.

www.Notesfree.in

www.Notesfree.in

UNIT IV I/O, GENERICS, STRING HANDLING

I/O Basics - Reading and writing console I/O - Reading and writing Files. Generics: Generic Programming - Generic classes - Generic Methods - Bounded Types - Restrictions & Limitations. Strings: Basic string class, methods & Buffer class

1. IO BASICS:

Java I/O is used to process the i/p and Produce the o/p based on i/p.

Streams:

Based on type of operation

1. System.in
2. system.out
3. system.err

Depending on type of file

1. Byte Stream
2. character Stream.

2. READING AND WRITING CONSOLE I/O :

Reading console input:

Method 1: Reading input using Buffered Reader, Inputstream Reader, system.in.

Syntax. Buffered_Reader obj_name = new BufferedReader(new InputStreamReader (system.in));

www.Notesfree.in

www.Notesfree.in

Program:

```
Class Readchar
{   Public static void main(String args[]) throws
                                        IOException
    {
        int count = 0;
        char ch;
        System.out.println("\n Enter 5 characters");
        while (count<5)
        {
            ch = (char) obj.read();
            System.out.println(ch);
            count++;
        }
    }
}
```

O/P:

```
Enter 5 characters
hello
h
e
l
l
o
```

Method 2: Reading console input using scanner and system.in

Pgm:

```
Public class Example
{
    PSVM( )
```

www.Notesfree.in

www.Notesfree.in

```
{
    Scanner sc = new scanner (string [] args)
    System.Out.print ("Name?");
    String name = sc.next();
    S.O.pln(+name);
  }
}
```

O/P

Name?
Sduge

Writing console output:

The simple method used for writing the O/P on the console is write().

Syntax: System.out.write (int b)

---

3. READING AND WRITING FILES:

For Reading files in Java, we have

1. Buffered Reader:

BufferedReader in = new BufferedReader (Reader in, int size);

2. FileReader class:

```
Public class ReadingFile
{
    PSVM
    {
        FileReader fileread = new FileReader ("c:\\users\\test.txt");
        int i;
        while ((i = fileread.read()) != -1)
        S.O.pln((char) i);
    }
}
```

www.Notesfree.in

www.Notesfree.in

3. scanner class:

A scanner can parse the primitive types.

Ex:
```
import java.io.File;
import java.util.scanner;
Public class Main
{
    psvm
    {
     try
     {
        File file = new File (pathname = "c:\\test.txt");
        Scanner sc = newscanner(file);
        while (sc.hasNextline())
        {
            S.O.pln (sc.hasNextLine());
        }
     }
     catch (Exception e)
     { S.O.pln ("Error") }
    }
}
```

Writing text File in Java:

1. Java BufferedWriter class

Public class BufferedWriter extends Writer.

2. Java FileWriter class

```
import java.io.File;
import java.io.FileWriter;
Public class Main
{
    psvm()
    {
```

www.Notesfree.in

www.Notesfree.in

```
try
{
    FileWriter fw = new FileWriter(Pathname:"c:\\test.txt");
    fw.write(str:"Hello");
    fw.close();
    S.o.pln("successfully written");
}
catch(Exception e)
{ S.o.pln("error"); }
}
}
```

O/P
Hello

4. GENERIC PROGRAMMING:

Generic is a mechanism for creating a general model in which generic methods and generic classes enable prgmrs to specify a single method & single class for performing desired task.

Generic class:

To create object of generic class, we use the following syntax:

UDCN<type> obj = new UDCN<type>()

UDCN - User defined class name.

type - any one of valid data types.

www.Notesfree.in

www.Notesfree.in

Example:

```
Class Test<T>
{
    T obj;
    Test (T obj)
    {
        this.obj = obj;
    }
    Public T getobject()
    {
        return this.obj;
} }
    class gsample
    {
        Psvm ()
        {
            Test<Integer> iobj = new Test<Integer>(15);
            S.o'pln (iobj. getobject());
            Test<String> sobj = new Test<String>("Hello");
            S.o.pln (sobj. getobject());
        }
    }
```

O/P

15
Hello

Generic Method:

It allows programmer to write a generalised method for the methods of different data types.

www.Notesfree.in

www.Notesfree.in

Example:

```
class Test
{
    static <T> void genericDisplay (T element)
    {
        S.O.pln (element.getclass().getName() + "=" + element);
    }
    PSVM()
    {
        genericDisplay(11);
        genericDisplay("hello");
        genericDisplay(1.0);
    }
}
```

O/P

```
java.lang.Integer = 11
java.lang.String = hello
java.lang.Double = 1.0
```

Bounded Types:

Using bounded types, we can make the object of generic class to have data of specific derived types. To restrict the type that can be used as argument

Syntax: <T extends superclass>

Example:

```
class GenericClass <T extends Number>
{
    Public void display()
    {
        S.O.pln("Bounded");
    }
}
```

www.Notesfree.in

www.Notesfree.in

```
class Main
{
    PSVM()
    {
        Genericclass<String> obj = new Genericclass<>();
    }
}
```

RESTRICTIONS AND LIMITATIONS:

→ Runtime execution is possible only if it is used along with raw type.

→ Primitive type parameter is not allowed.

→ Generic class throw & catch not allowed.

→ Instantiation of generic parameter T is not allowed. new T() // Error new T[10].

→ Parameterized types are not allowed.

Eg: new Stack<String>[10];

---

5. STRINGS

→ String is a collection of characters.

→ Two ways to create strings in Java

1. String s = "Hello"
2. String s = new String ("Hello");

Methods:

s.length() - Gives length of string s1.

s1.concat(s2) - concatenation of string s1 and s2.

s1.equals(s2) - If s1 and s2 equal, it returns True.

www.Notesfree.in

www.Notesfree.in

s1.charAt(position) - Returns character present at index position.

String toLowerCase - returns string in lowercase

Example:

```
import java.util.*;
class Main
{
    psvm()
    {
        String s = "welcome";
        S.O.pln (" String length = " + s.length());
        S.O.pln (" character at 3rd pos = " + s.charAt(3));
        S.O.pln ("Substring" + s.substring(3));
        String s1 = " Home";
        S.O.pln ("concatenated String = " + s.concat(s1));
        Boolean out = "hello".equals("hello");
        S.O.pln ("checking Equality" + out);
        Out = "hello".equalsIgnoreCase("HELLO");
        S.O.pln ("checking Equality " + out);
        S.O.pln ("changing to uppercase" + s.toUpperCase());
    }
}
```

O/P

String length = 7

character at 3rd pos = c

substring come

concatenated string = welcome Home

checking Equality true

checking Equality true

changing to UPPER case WELCOME

www.Notesfree.in

www.Notesfree.in

STRING BUFFER CLASS:

StringBuffer class is used to create mutable string. ie) it can be changed.

Syntax:

StringBuffer <obj-Name> = new StringBuffer();

Example:

```
class Main
{
  psvm()
  {
      StringBuffer sb = new StringBuffer("Hello");
      sb.append("Java");
      S.opln(sb);
      sb.insert(6, "welcome");
      S.o.pln(sb);
      sb.replace(9, 13, "done");
      S.o.pln(sb);
      sb.delete(0,13);
      S.o.pln(sb);
      sb.reverse();
      S.o.pln(sb);
  }
}
```

O/P:

Hello Java

Hello welcomeJava

Hello weldone Java

Java

avaJ

www.Notesfree.in

www.Notesfree.in

UNIT V

JAVAFX EVENT HANDLING, CONTROLS & COMPONENTS

JavaFx Events and controls: Event Basics - Handling Key and Mouse Events. controls: checkBox, ToggleButton - RadioButtons - List View - comboBox - choiceBox - Text controls - ScrollPane. layouts - FlowPane - HBOX and VBOX - Border Pane - StackPane - GridPane. Menus - Basics - Menu - Menubars - Menu Item.

---

1. JavaFx Events and controls:

Basics:

JavaFx is a set of packages that allow Java Programmers to create rich graphics and media applications such as 2D and 3D games, animations etc,

Advantages:

1. Hardware - accelerated graphics
2. High performance media engine.

Types of event.

1. Foreground Events
2. Background Events

→ JavaFx, event is an instance of the javaFx.event.Event class.

www.Notesfree.in

www.Notesfree.in

JavaFX provides several events.

* Drag Event
* Mouse Event
* Key Event.

Events are handled by implementing the EventHandler interface. which is in javafx.event.

interface EventHandler <T extends Event>

HANDLING KEY AND MOUSE EVENTS:

Key Event:

This is an input event that indicates the key stroke occured on a node. It is represented by the class named KeyEvent. This includes actions like key pressed, key released and key typed.

Mouse Event:

This is an input event that occurs when a mouse is clicked. It is represented by the class named MouseEvent. It includes actions like mouse clicked, mouse pressed, mouse released, mouse moved, etc.

www.Notesfree.in

www.Notesfree.in

xample:

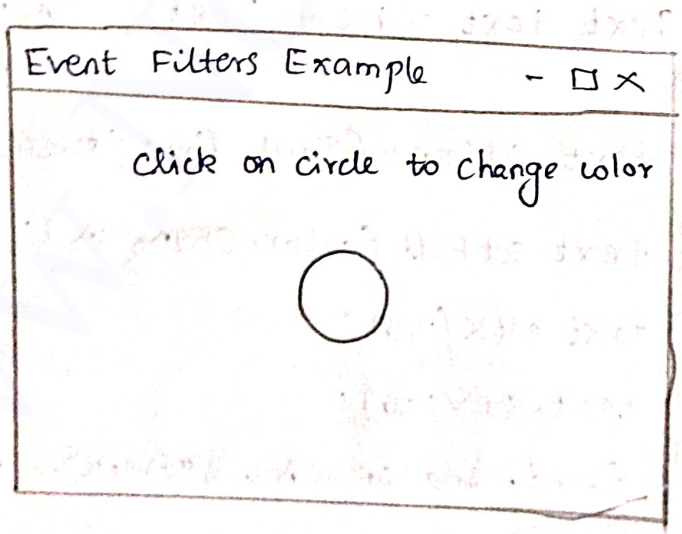
```
import javaFx.application.Application,
import static javafx.application.Application.launch;
import javafx.event.EventHandler;
import javafx.Scene.*;
Public class Example extends Application
{
    public void start(Stage stage)
    {
        Circle circle = new Circle();
        Circle.setCenterX(300.0f);
        circle.setcenterY(135.0f);
        circle.setRadius(25.0f);
        circle.setFill(color.BROWN);
        circle.setStrokeWidth(20);
        Text text = new Text("click on circle to
                                change colour");
        text.setFont(Font font(null, FontWidth BOLD, 15));
        text.setFill(color.CRIMSON);
        text.setX(150);
        text.setY(150);
        EventHandler<MouseEvent> eventHandler = new
                        EventHandler<MouseEvent>()
        {
            public void handle(MouseEvent e)
            {
                System.o.pln("Hello World");
                Circle.setFill(color.DARKSLATEBLUE);
            }
        };
```

www.Notesfree.in

www.Notesfree.in

```
circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventHan
Group root = new Group(circle, text);
Scene scene = new Scene(root, 600, 300);
Scene.setFill(color.LAVENDER);
Stage.setTitle("Event Filters Example");
Stage.setScene(scene);
Stage.show();
}
Public static void main(String args[])
{
    launch(args);
}
}
```

Output:



2. **CONTROLS**: checkBox:

The checkBox Group component allows the user to make one and only one selection at a time.

www.Notesfree.in

www.Notesfree.in

Syntax:

checkBox (String str, checkBoxGroup cbg, Boolean val);

check Box can also called as Radio Buttons.

Example:

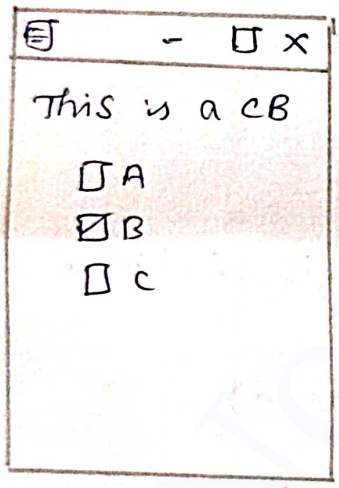
```
~~import java.awt.*;~~
import javafx.application.Application;
import javafx.scene.*;
import javafx.event.*;
import javafx.collections.*;
import javafx.stage.Stage;
public class checkbox_1 extends Application
{
    public void start(Stage s)
    s.setTitle("Creating CB");
    TitlePane r = new TitlePane();
    Label L = new Label("This is CB");
    String st[] = {"A", "B", "C"};
    r.getchildren().add(l);
    for (int i = 0; i < st.length; i++)
    {
        checkBox c = new checkBox(st[i]);
        r.getchildren().add(c);
        c.setIndeterminate(true);
    }
```

www.Notesfree.in

www.Notesfree.in

```
        Scene sc = new Scene (r, 150, 200);
        s.setScene(sc);
        s.show();
    }
public static void main(String args[])
{
        launch(args);
    }
}
```

Output:



JavaFx Toggle Button:

It is a button that can be selected or not selected. It is represented in the class javafx.scene.control.ToggleButton.

Example:

```
ToggleButton toggleButton1 = new ToggleButton ("Left");
```

Radio Button:

It is similar to Toggle Button but the difference is RadioButton cannot be unselected.

www.Notesfree.in

www.Notesfree.in

Example:

```
RadioButton rb = new RadioButton("Left");
```

LISTVIEW:

Listview control enables user to choose one or more options from a predefined list of choices.

Example:

```
Listview listview = new Listview();
```

Adding Items:

```
listview.getItems().add("Item 1");
```

COMBO BOX:

ComboBox control enables users to choose an option from a predefined list of choices. or type in another value if none of the predefined choices matches what the user want to select.

Example:

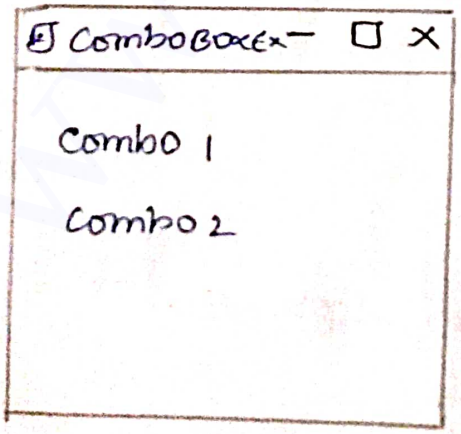
```
import javafx.application.Application;
import javafx.scene.*;
import javafx.stage.Stage;
```

www.Notesfree.in

www.Notesfree.in

```
Public class comboBox1 extends Application
{
    Public void start (stage primarystage) throw
                                        Exception
    {
        Primarystage.setTitle ("comboBox Ex1");
        comboBox comboBox = new comboBox();
        comboBox.getItems().add ("ch 1");
        comboBox.getItems().add ("ch 2");
        HBox hbox = new HBOX (comboBox);
        Scene scene = new Scene (hBox, 200, 120);
        primary stage.setscene (scene);
        primaryStage.show();
    }
    psvm()
    {
        Application.launch(arg);
    }
}
```

Output:



www.Notesfree.in

www.Notesfree.in

CHOICE BOX:

→ It enables user to choose an option from a predefined list of choices.

→ It is represented by the class

javafx.scene.control.ChoiceBox.

Creating ChoiceBox:

```
ChoiceBox choiceBox = new ChoiceBox();
```

Adding choices:

```
choiceBox.getItems().add("choice 1");
ChoiceBox.getItems().add("choice 2");
ChoiceBox.getItems().add("choice 3");
```

TEXT CONTROLS:

→ TextField controls enable users of a JavaFx application to enter text which can then be read by the application.

→ It is represented in the class

javafx.Scene.control.TextField

Creating a TextField:

```
TextField textfield = new Textfield();
```

Adding:

The text field object must be added to the scene graph.

www.Notesfree.in

www.Notesfree.in

SCROLL PANE:

JavaFX Scroll pane control is a con that has two scrollbars. The scroll bar ena the user to scroll around the components shown inside the scrollPane, so that different parts of the components can be seen.

It is represented by the JavaFx class.

javafx.scene.control.scrollpane.

Create a Scroll Pane:

ScrollPane scrollpane = new ScrollPane();

3. LAYOUTS:

JavaFx provides several predefined layouts such as HBOX, VBOX, BorderPane, Flow Pane, Stack Pane and GridPane.

Flow Pane:

→ JavaFx FlowPane is a layout component which lays out the child components either vertically or horizontally.

→ It is represented by the class

javafx.scene.layout.FlowPane

www.Notesfree.in

www.Notesfree.in

Creating a FlowPane:

```
FlowPane flowpane = new FlowPane();
```

Adding children:

```
Button button1=new Button ("Button NO 1");
Button button2 = new Button ("Button NO 2");
FlowPane flowpane = new FlowPane();
flowpane.getchildren().add(button1);
```

HBOX and VBOX:

HBOX:

→ HBOX component is a layout component which positions all its child nodes in a horizontal row.

→ It is represented by the class javafx.scene.layout.HBOX.

Creating HBOX:

```
HBOX hbox = New HBOX();
Button b1 = new Button ("Button NO 1");
```

V BOX:

It is a layout component which positions all its child nodes in a vertical column on top of each other.

www.Notesfree.in

www.Notesfree.in

→ It is represented by the class

javafx.scene.layout.VBOX.

creating a VBox:

```
VBox vbox = new VBox();
```

Border Pane:

→ BorderPane class lays its children in top, bottom, center, left and right positions.

→ It also inherits Pane class.

Constructors:

* Border pane ()
* Border pane (Node c)
* BorderPane (Node center, Node top, Node right, Node bottom, Node left)

StackPane:

→ StackPane places all the nodes into a single stack where every new node gets placed on the top of the previous node.

→ It is represented in

javafx.scene.layout.StackPane.

Constructors:

1. Stackpane ()
2. Stackpane (Node ? children)

www.Notesfree.in

www.Notesfree.in

Grid pane:

Grid Pane is a layout component in a grid. The size of the cells in the grid depends on the components displayed in the GridPane.

Rules:

All cells in the same row will have same height and the cells in the same column will have the same width.

Creating a Grid Pane:

```
GridPane gridPane = new GridPane();
```

Adding children:

```
gridPane.add(button1, 0, 0, 1, 1);
```

---

4. MENUS: BASICS:

Menu:

Menu Bar is the main component of any application. In JavaFx, javafx.scene.control, menu class provides all the method to deal with menus.

Menu Bar:

MenuBar Provides Javafx application with a visual dropdown menu similar to that most desktop application.

www.Notesfree.in

www.Notesfree.in

Creating MenuBar Instance:

```
MenuBar menuBar = new MenuBar();
```

Menu Items:

once you have created a Menu instance you must add one or more Menu Item instances to it.

Example:

```
Menu menu = new Menu("Menu 1");
MenuItem menuItem1 = new MenuItem("Item 1");
MenuItem menuItem2 = new MenuItem("Item 2")
menu.getItems().add(menuItems1);
menu.getItems().add(menuItems2);
MenuBar menuBar = new MenuBar();
menuBar.getMenus().add(menu);
```

www.Notesfree.in