◦ Java Programming -

- History of Java :-

The history of java starts from Green Team

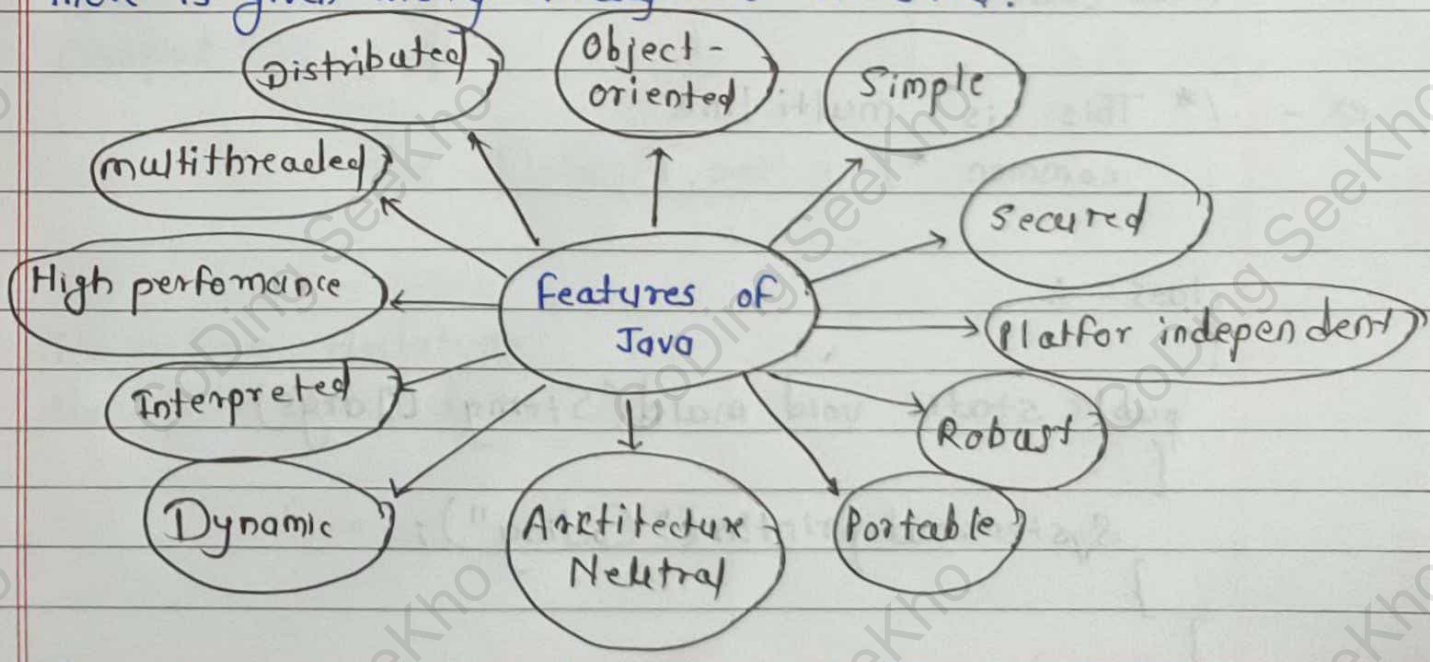
1) James Gosling, Mike Sheridan, and Patrik Naughton initiated the java language project in June 1991. The small team of sun engineers called Green Team.
2) Originally designed for small, embedded systems in electronic applicans like set-topboxes.
3) Firstly, it was called "Greentalk" by James Gosling and file extension was .gt.
4) After that, it was called Oak and was developed as a part of the Green project.

◦ Java Version History :-

Java SE 8. (18th March, 2014)

◦ features of Java -

There is given many freagtures of Java.



- Java Comments - it can also be used to hide program code for specific time

- Types of Java Comments :-

1) Single Line Comment - The single line comment is used to comment only one line.

ex -
```
class A {
  public static void main(String []args)
  {
    int i = 10; // Here, i is a variable
    System.out.println(i);
  }
}
```
Output : 10

2) Java MultiLine Comment -
the multi-line comment is used to comment multiple lines code.

ex -
```
/* This is multi line
   commen */

class A
{
  public static void main(String []args)
  {
    System.out.println("Coding");
  }
}
```

// Coding : Output.

③ Documentation comment :-

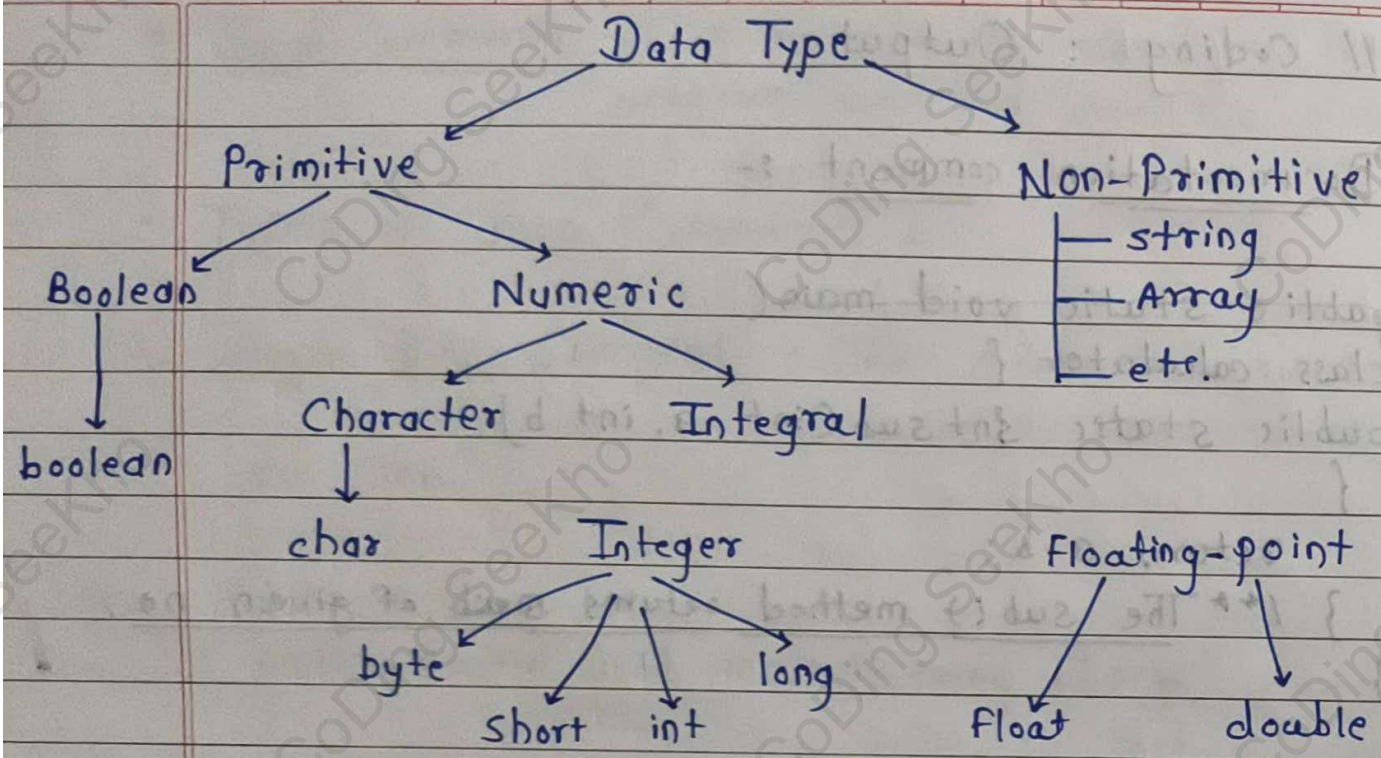
ex -
```
public static void main(
class calculator {
public static int sub(int a, int b)
  {
    return a-b;
  } /** The sub() method returns sub of given no.;
}
```

◦ Java Variable ◦

ex -
```
class Simple {
public static void main(String []args) {
  int a = 10;
  int b = 10;
  int c = a + b;
  System.out.println(c);
}}
```
Output : 20.

◦ Data Types ◦

- Primitive datatype
- Non-primitive datatype.



Data Type	Default Value	Default size
boolean	False	1 bit
char	'\u000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0F	4 byte
double	0.0d	8 byte

- Variables and Data Types in Java -

Variable is name of memory location. There are three types of variables in java, local, instance and static.

- Types of Variable -

① Local Variable - A variable which is declared inside the method is called local variable.

② Instance Variable - A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static.

③ static variable - A variable that is declared as static is called static variable.

- Constants in Java :-
A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword.

- Scope & Life time of variables -
The scope of a variable defines the section of the code in which the variable is visible.

Instance variable - Instance variables are those that are defined within a class itself & not in any method or constructor of the class.

Argument variables - these are the variables that are defined in the header oaf constructor or a method.

Local Variables - A local variable is the one that is declared within a method or a constructor. The scope and lifetime are limited to the method itself.

- Operators in java - Operator in java is a symbol
  - Unary Operator.
  - Arithmetic Operator
  - shift Operator
  - Relational Operator
  - Bitwise Operator
  - Logical Operator
  - Ternary Operator and
  - Assignment Operator.

- Operators Hierachy

Operator Arcedence

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~
multiplicate	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&

logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

- Expressions - expressions builiding blocks of any java program, usually created to produce a new value, although sometimes an expression simply assign a value to a variable.

Types of Expression -
While an expression produces a result, it doesn't always.
- those that produce a value, i.e. result of (1+1)
- those that assign a variable, ex - (v = 10)

- Java type casting & Type conversion -

Widening or Automatic Typic Conversion -
Widening conversion takes place when two data types are automatically converted.
- the two data types arecompatible.
- When we assign value of a smaller data type to a bigger datatype.

Byte → Short → Int → Long → Float → Double.

Widening or Automatic conversion

○ Narrowing or Explicit Conversion ○

Date
Page

- If we want to assign a value of larger data type to a smaller data type we plat perform explicit type casting or narrowing.

Double → Float → Long → Int → Short → Byte.
Narrowing or Explicit Conversion.

○ Java Enum - Enum in java is data type that contains fixed set of constants.

- Simple example of java enum -

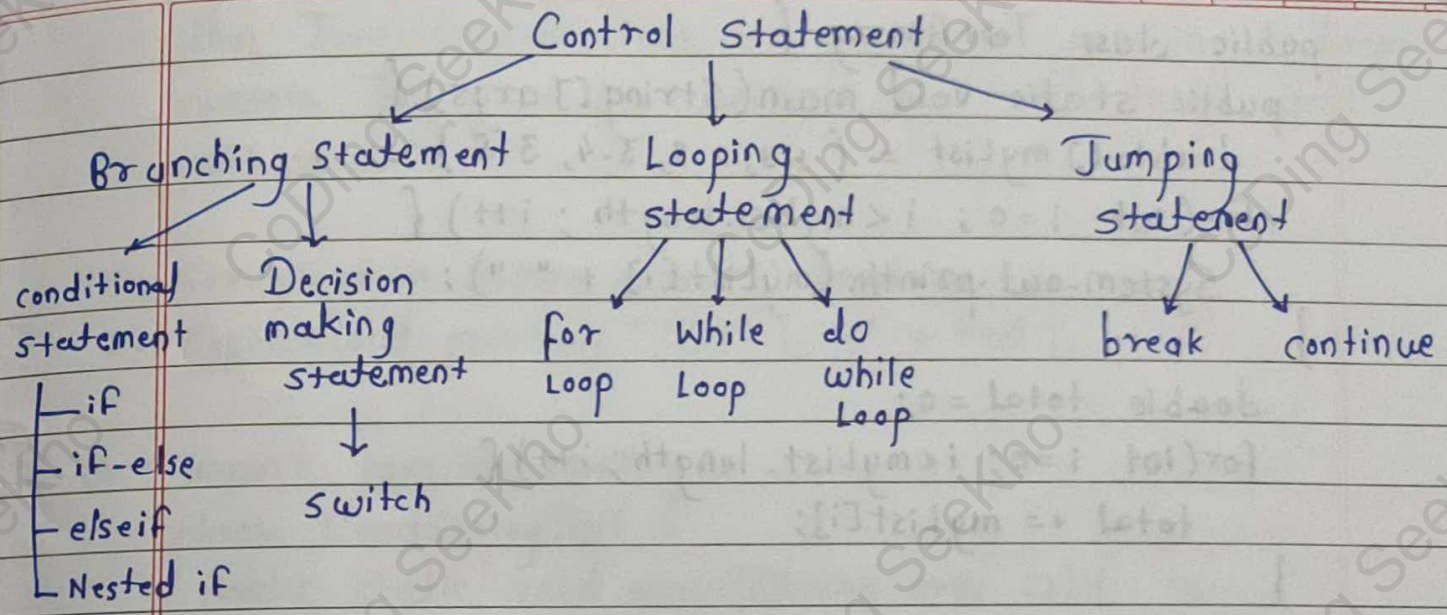
```
class EnumExample1 {
 public enum Season {
    WINTER, SPRING, SUMMER, FALL }
 public static void main (String [] args){
   for (season s: Season.values ())
   System.out.println(s);
 }}
```

Output : WINTER
SPRING
SUMMER
FALL

○ Control Flow Statements - the control flow statements in java allow you to run to skip blocks of code when special cond^n are met.

- 'if' statement - the "if" statement in Java works exactly like in programming most languages.

```
if (condition){
  —
}
```



- Creating a stand-Alone Java Application - Write a main method that runs your program. You can write this method anywhere.

- Declaring Array Variable - you must declare a variable to reference the array, and you must specify the type of array the variable can reference.

```
double[] myList;  // prefered way.
Or.
double myList[];  // works but not preferred way.
```

- Processing Arrays - When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type & the size of the array is known.

```
public class TestArray {
  public static void main(String[] args) {
    double[] myList = { 1.9, 2.9, 3.4, 3.5 }
    for( int i=0 ; i<myList.length ; i++) {
      System.out.println (myList[i] + " ");
}
    double total = 0;
    for(int i=0; i<myList. length; i++){
       total += myList[i];
  }
 System.out.println ("Total is " + total);

  double max = myList[0];
  for(int i = 1; i<myList.length; i++){ if
     (myList[i]>max) max = myList[i];
  }
  System.out.println ("Max is " + max);
  }
}
// 1.9          Total is 11.7
   2.9          Max is 3.5
   3.4
   3.5


public class TestArray {
  public static void main(String []args) {
    double [] myList = { 1.9, 2.9, 3.4, 3.5};
   for (double element: myList) {
     System.out.println(element);
  }
}
}
```

∘ Java Console class ∘

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

1. `Stringtext = System.console().readLine();`
2. `System.out.println("Text is :" + text);`

ex-
```
import java.io.Console;
class ReadStringTest {
 public static void main(String args []) {
   Console c= System.console();
   System.out.println("Enter your name:");
   String n= c.readLine();
   System.out.println("Welcome" +n); }
}
```

Output:
Enter your name : Nakul Jain
Welcome Nakul Jain

∘ **Constructors** - Constructor in java is a special type of method that used to initialize the object.

1. Constructor name must be same as its classname
2. Constructor must have no explicit return type.

∘ **Types of Java Constructor**-

① Default Constructor
② Parameterized Constructor.

Date
Page

① Java Default Constructor - A constructor have no parameter is known as default constructor.

ex -
```
class Bike1 {
 Bike1() {
  System.out.println("Bike is created");
 }
 public static void main(String args[]){
  Bike1 b = new Bike1();
}}
```
Output : Bike is created.

② Parameterized Constructor -
```
class Student4 {
 int id;
 String name;
 Student4(int i, String n) {
  id = i;
  name = n;
 }
 void display() { System.out.println(id + " " + name); }
 public static void main(String args[]) {
  Student4 s1 = new Student4(111, "Karan");
  Student4 s2 = new student4(222, "Aryan");
  s1.display();
  s2.display();
 }
}
```
Output :

111 Karan
222 Aryan

∘ Constructor Overloading ∘

Constructor overridding is technique in Java in which can have any number of constructors that differ in parameter lists.

ex-
```
class Demo
{
   private int x,y;
    Demo()
    {
      x=5;  y=6;
      System.out.println(x);
      System.out.println(y);
    }
  Demo(int p, int q) {
    x=p ;   y=q;
    System.out.println(x);
    System.out.println(y); }}
class Construct {
    public static void main(String []args){
    Demo. d1 = new Demo(7,9);
    Demo d1 = new Demo();
  }}
```

Output = 5, 6
7, 9

∘ Copy Constructor ∘

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

ex-
```
class Student6{
 int id;
 String name;
 Student6(int i,String n){
 id = i;
 name = n; }
 Student6(Student6 s){
 id = s.id;
 name = s.name; }
 void display(){
 System.out.println(id+" "+name);}
 public static void main(String args[]){
Student6 s1= new Student6(111,"Karan");
Student6 s2 = new Student6(s1);
 s1.display();
 s2.display(); }}
```
Output : 111 Karan
111 karan

◦ Java Methods ◦

A java method is a collection of statements that are grouped together to perform an operation.

- public static - modifier
- int - returntype
- methodName - name of the method
- a,b - formalparameters
- int a, int b - list of parameters.

- modifier - it defines the access type of the method & it is optional touse.
- returnType - Method may a value
- nameOfMethod - method signature consists of themethod name & the parameter list.
- parameter List - is the type, order & no of parameters of a method.
- method body - the method body defines what the method does with the statements.

* Call by Value & Call by Reference - there is only call by value in java, not call by reference. If we call a method passing a value, it known as by call value.

ex -
```
class Operation {
  int data = 50;
  void change(int data) {
   data = data + 100; }
  public static void main(String args[]) {
  Operation op = new Operation();
  System.out.println("before change " + op.data);
  op.change(500);
   System.out.println("after change " + op.data); }}
```
Output : before change 50
after change 50

o Static Field & methods o

The static keyword in java is used for memory manggement mainly

1. variable
2. method
3. block
4. nested class

Date
Page

- **Static Variable** -
- the static variable can be used to refer the common property of all objects.
- the static variable gets memory only class area at the classloading.

Ex -
```
class Demo {
  static private int x;
  static int y;
  static {
   x = 27;
   System.out.println(x);
    y = 23;
    System.out.println(y);
 }}
class staticInitial {
  public static void main(String []args){
   Demo.y = 29;
   System.out.println(Demo.y);
  }
}
```

- **Static Block** - is used to initialize static data member.
- it is executed before main method at the time of class loading.

ex -
```
class A2 {
static {System.out.println("static block is invoked");}
public static void main(String []args){
 System.out.println("Hello main");
}}
```

∘ Access Control ∘

∘ Access Modifiers - ① private ③ protected
② default ④ public

-∘- private access modifier -
the private class modifier is accessible only within class.

Ex-
```
class A {
 private int data = 40;
 private void msg() { System.out.println("Hello java");}}
 public class Simple {
 public static void main(String []args) {
  A obj = new A();
  System.out.println(obj.data);
  obj.msg(); }
}
```

-∘- default access modifier - it is treated as default bydefault. don't useany modifier.

Ex-
```
package pack;   // save by A.java
class A {
  void msg() {
    System.out.println("Hello"); }
}

// save by B.java
package mypack;
import pack.*;
class B {
  public static void main(String args[]) {
   A obj = new A();
   obj.msg(); }
}
```

③ protected access modifier - the protected access modifier is accessible within package and outside the package but through inheritance only

Ex-
```
//save by A.java
package pack;
public class A{
 protected void msg(){
  System.out.println("Hello");}}

//save by B.java
package mypack;
class B extends A{
 public static void main(String []args){
  B obj = new B();
 }}
```

④ public access modifier - the public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Ex.
```
//save by A.java
package pack;
public class A{
 public void msg(){
  System.out.println("Hello");}}

//save by B.java.
package mypack;
import pack.*;
class B{
 public static void main(String []args){
  A obj = new A();
  obj.msg();}}
```

Understanding all java access modifiers

Access modifier	within class	within package	outside package by sub class only	outside package
private	Y	N	N	N
Default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

this keyword -

1. this can be used to refer current class instance variable
2. this can be used passed as an argument in method call, constructor call.

Ex -

```
class Student {
int rollno;
String name;
Float fee;
Student(int rollno, String name, Float fee) {
this.rollno = rollno;
this.name = name;
this.fee = fee; }
void display() {
System.out.println(rollno + " " + name + " " + fee); }}
class TestThis2 {
public static void main(String args[]) {
Student s1 = new Student(111, "ankit", 5000f);
Student s2 = new Student(112, "sumit", 6000f);
s1.display();
s2.display();
}}
```

Output : 111 amit 5000
112 sumit 6000.

Difference between constructor and method.

	Constructor	Method
1.	used to initialize the state of an object	1. is used to expose behaviour of an object.
2.	must not have return type	2. must have a return type.
3.	invoked. implicity	3. invoked explicity.
4.	same as class name	4. name may or may not be same as class name.

- **Method Overloading** : changing no. of arguments.
we have created two methods, first add() method performs addn of two no & second method performs addn of 3 no.

Ex-
```
class Adder {
  static int add(int a, int b) {
    return a+b ; }
  static int add(int a; int b, int c) {
    return a+b+c; }
class Test0 {
  publicstatic void main(String [Jargs) {
   System.out.println(Adder.add(11, 11));
   System.out.println(Adder.add(11, 11, 11);
  }}
```

Output : 22
33

- Method Overloading - changing data type of arguments.

- Recursion - Recursion in java is a process in which a method calls itself continously.

Ex -
```
public class Recursion {
 static int factorial(int n) {
  if (n == 1)
   return 1;
  else
   return(n * factorial(n-1);
 }}
public static void main(String [] args){
 System.out.println("Factorial of 5 is: " + factorial(5));
 }}
```
Output :
factorial of 5 is 120.

- Garbage Collection - garbage means unreferenced object.

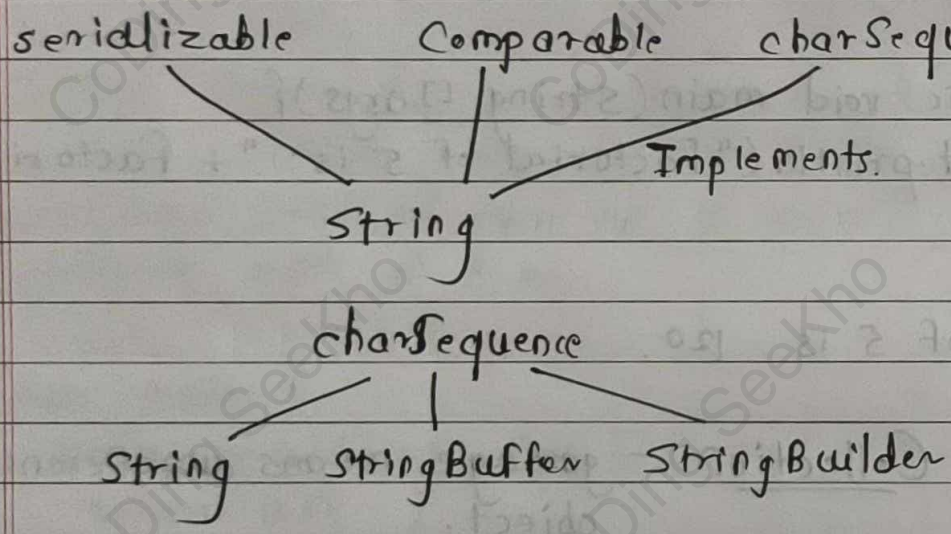
- get() method - is used to invoke the garbage collecto to perform cleanup processing.

Ex -
```
public class TG {
 public void finalize() {
  System.out.println(" garbage collected ");}
  public static void main (String args []){
  TG s1 = new TG();
  TG s2 = new TG();
  s1.null;
  s2.null;
  System.get();
 }}
```

Output :- ~~object~~ garbage collected
garbage collected.

o **String** - String is basically an object that represents sequence of char values. An array of characters works same as java string.

1. char [] ch = {"j", "a", 'v', 'a'}
2. String s = new string (ch);



-: Two ways String Object.

(1) **String Literal** - Java string literal is created by using double quotes.

Ex -
```
public class String {
public static void main(String args[]){
 String s1 = "java"
 char ch[]= {'s', 't', 'r', 'i', 'n', 'g', 's'}
 String s2 = new String (ch);
 String s3 = new String ("example.");
 System.out.println(s1);
 System.out.println(s2);
 System.out.println(s3); }}.
```

Output : java
Strings

∘ Inheritance ∘

Inheritance in java is a mechanism in which one object acquires all the properties & behaviours of parent object.

- ∘ Overriding
- ∘ Code Reusability.

- the extends keyword indicates that you are making a new class that derives from an existing class.

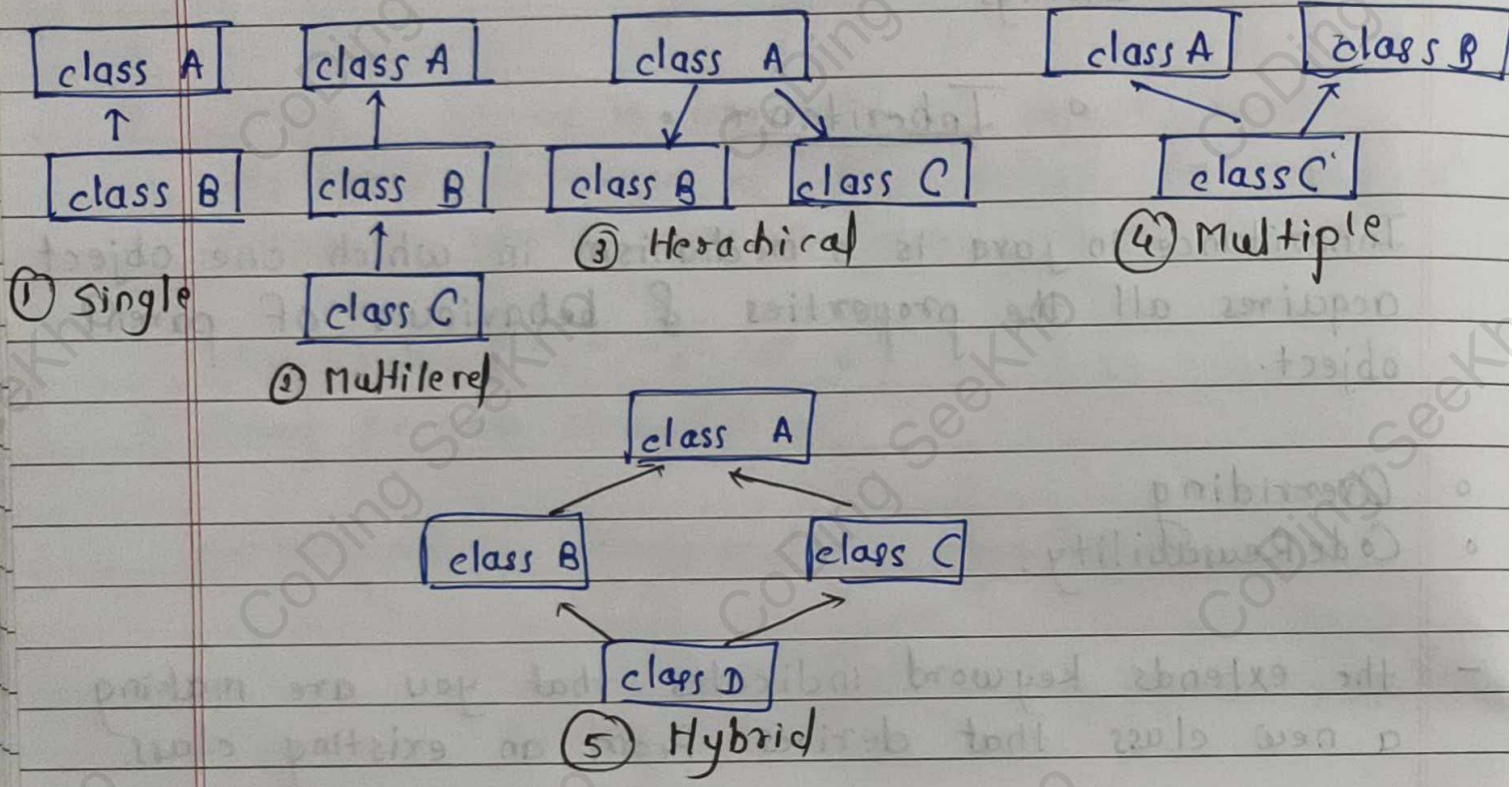
Ex -
```
class Employee {
    float salary = 40000; }
class Programmer extends Employee {
 int bonus = 10000;
 public static void main(String args[]){
  Programmer p = new Programmer();
  System.out.println(" Programmer salary is : "+p.salary);
  System.out.println(" Bonous of Programmer is :"+p.bonus);
 }}
```

Output :

programmer salary is : 40000.0.
Bonous of Programmer : 10000.

Date
Page

Types of Inheritance



① Single Inheritance

```
class A {
  int x, y;
  void f1() {
   System.out.println(" Fun1 executed. ");
  }}
class B extends A {
  int z;
  void f2() {
   System.out.println(" Fun2 executed.");
  }}
class Inherit {
 public static void main (String []args) {
   B b1 = new B();
   b1.x = 5;
   b1.y = 7.
```

```
b1.f1();
b1.z = 67;
b1.f2();
System.out.println(b1.x);
System.out.println(b1.y);
System.out(println(b1.z);
}}
```

Output :- fun1 executed
fun2 executed
5
7
67.

② Multilevel Inheritance -

Ex-

```
class A {
  A() {
  System.out.println("A constructor."); }
class B extends A {
  B(){
  System.out.println("B Constructor.");}
}
class C extends B {
  C() {
  System.out.println("C constructor."); }
}
class Inherit {
 public static void main(String []args){
   C c1 = new C(); }
}
```

∘ Hierachical Inheritance ∘

Ex-
```
class A {
  void f1() {
  System.out.println(" f1 executed. ");
}}
class B extends A {
  void f2() {
  System.out.println(" f2 executed. ");
}}
class C extends A {
  void f3() {
  System.out.println(" f3 executed. ");
}}
class Inherit {
public static void main(String []args)
{
   C c1 = new C();
   c1.f3();
}}
```

∘ Member access and Inheritance ∘

∘ **Super keyword** - The super keyword in java is a reference variable which is used to refer immediate parent class object.

Ex-
```
class Animal {
   String color = "white"; }
class Dog extends Animal {
   String color = "black";
void printcolor() {
   System.out.println(color);
   System.out.println(super.color); }}
```

```
class TestSuper {
  public static void main(String []args){
   Dog d = new Dog();
   d.printColor();
  }}
```

Output : black
white.

- **Final keyword** - The final keyword in java is used to restrict the user. The java final keyword can be used in many context.

1. variable 2. method 3. class

- **Object class** - the object class is the parent class of all the classes in java by default.

- **Method Overriding** - If subclass has the same method as declared in the parent class, is known as method overrieding in java.

Ex -

```
class Vehicle {
  void run() {
   System.out.println("Vehicle is running");}}
class Bike extends Vehicle {
  void run(){
  System.out.println("Bike is running safely");}
  public static void main(String []args){
   Bike obj = new Bike();
  obj.run();
  }
```

Output :- Bike is running safely.

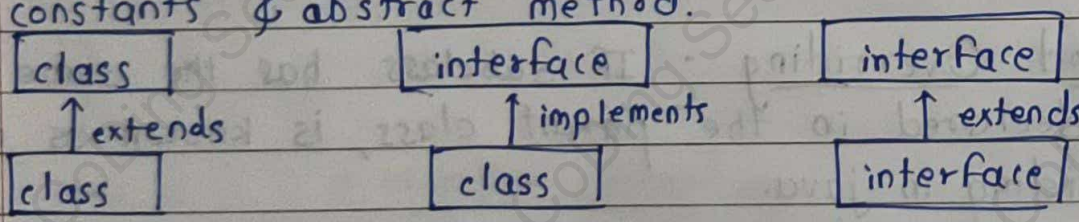
∘ Abstract Class ∘

A class that declared with abstract keyword is known as abstract class in java.

Ex-
```
abstract class Bike {
  abstract void run(); }
class Honda extends Bike {
  void run() {
  System.out.println("running safely..."); }
  public static void main(String []args){
   Bike obj = new Honda(){;
   obj.run();
  } }
```
Output :- running safely...

∘ Interface ∘

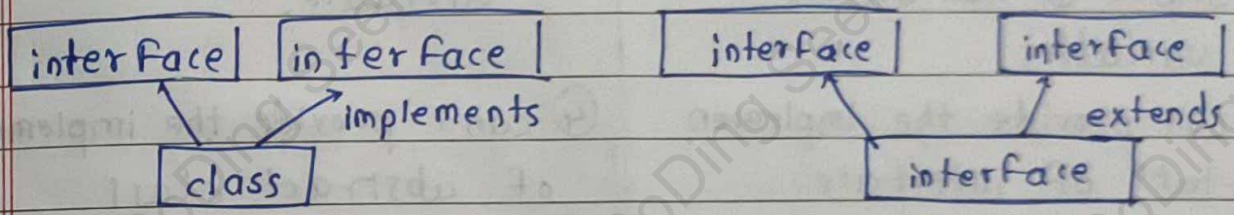
An interface in java is blueprint of class. The static constants & abstract method.



Ex-
```
interface Bank {
   int x = 5;
   void f1();
   static void f2() {
   System.out.println("static f2 of Bank");
   }}
interface RBI {
   int y= 7;
   void f1();
   static void f2(){
   System.out.println("static f2 of RBI"); }}
```

```
class SA implements Bank, RBI {
 public static void F1() {
  SA s1= new SA();
  s1.F2();
 }}
```

° Multiple inheritance °



Ex-

```
interface Printable {
  void print(); }
interface Showable {
   void show(); }
class A implements printable, Showable {
 public void print() {
 System.out.println(" Hello ");}
 public void show() {
  System.out.println( " Welcome ");}
  public static void main(String []args) {
 A obj = new A();
 obj.print();
 obj.show();
 }}
```

Output :- Hello
Welcome.

	Abstract class	Interface
①	Abstract class can have abstract & non-abstract methods.	① only abstract method. default & static methods also
②	dosen't support multiple inheritance	② Supports multiple inheritance.
③	can have final, non-final, static & non-static variables	③ only static and final variable.
④	can provide the implemen-tatn of interface	④ can't provide the implementatn of abstract class
⑤	abstract keyword is used.	⑤ interface keyword is used.

◦ Types of Nested Classes ◦

- Inner class is a part of nested class.
- Non-static class nested class are known as inner classes.

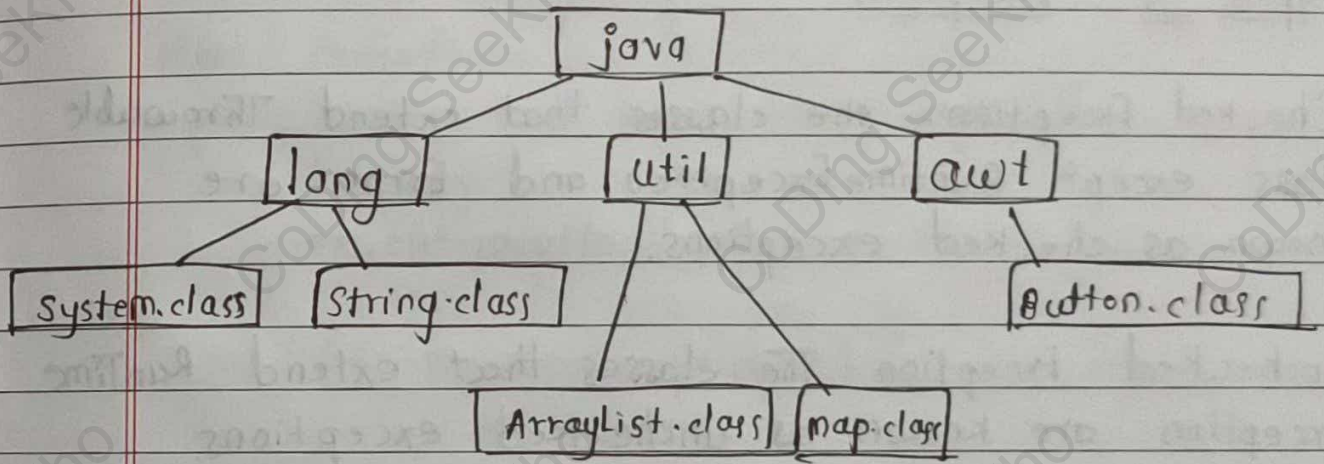
◦ Types of nested classes -

- Non-static nested-class
  1. Member inner class
  2. Anonymous inner class
  3. Local inner class.
- static nested class.

Package

A java package is a group of similar types of classes, interfaces and sub-packages.

Ex-
```
package mypack;
public class Simple {
  public static void main(String []args){
    System.out.println("Welcome to package");
}}
```



To Compile - javac -d . Simple.java
To run - java mypack.Simple.

Ex.
```
// save by A.java
package pack;
public class A {
  public void msg(){
  System.out.println("Hello");}}

// save by B.java  package mypack;
class B{
  public static void main(String args[]){
  pack.A obj = new pack.A();
  obj.msg();
}}
```
Output : Hello.

• Exception Handling •

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that

Defn- exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

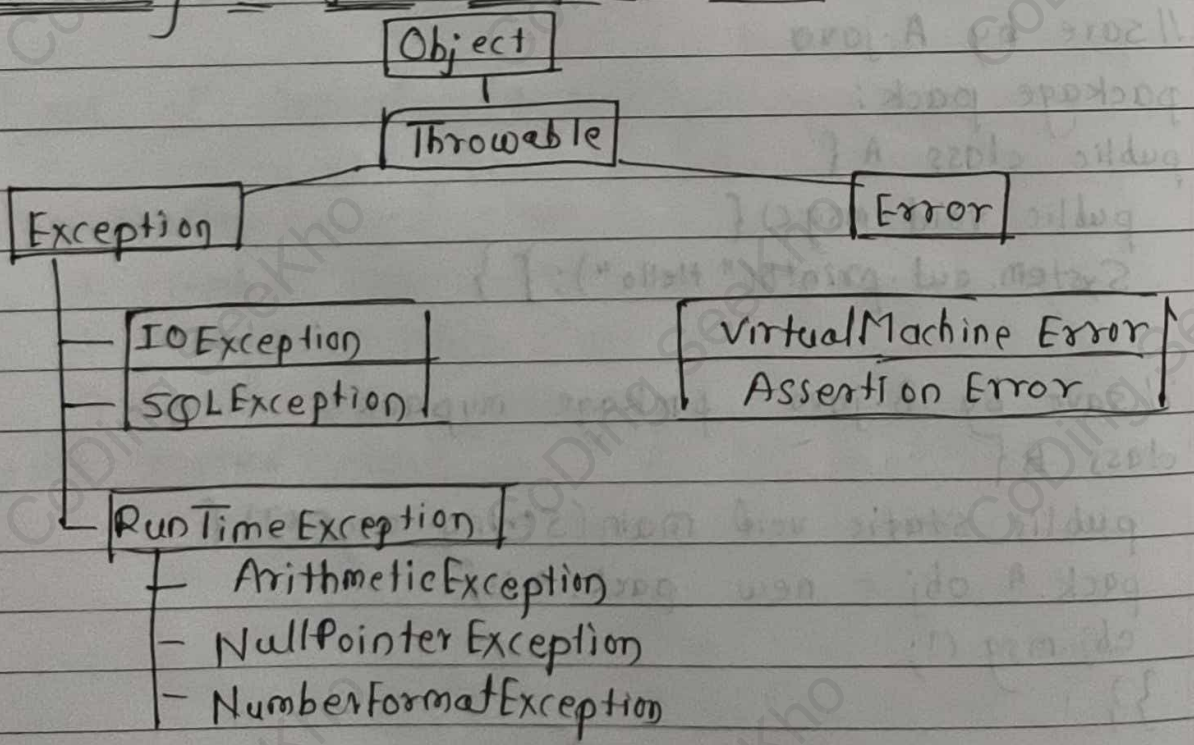
• Types of Exception -

(1) Checked Exception - the classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.

(2) Unchecked Exception - The classes that extend RunTime Exception are known as uncheeked exceptions.

(3) Error - error is irrecoverable.

• Hierarchy of java Exception classes -



- try block - try block is used to enclose the code that might throw an exception. It must be used within the method.

- catch block - catch block is used to handle the Exception. It must be used after the try block only.

- Nested try example -

```
class Excep{
 public static void main(String args[]){
 try { try {
  System.out.println("going to divide");
  int b=39/0; }
  catch(ArithmeticException e) {
  System.out.println(e): }
 try {
  int a[] = new int[5];
  a[5]=4; }
  catch (ArrayIndexOutofBoundsException e){
System.out.println(e); }
  System.out.println("other statement"): }
 catch( Exception e){
  System.out.println("handleled");}
  System.out.println("normal Flow..."); } }
```

- Finally block :- block that is used to execute important code such as closing connection, stream.

Ex-

```
class TestFB {
  public static void main(String []args){
  try{
  int data = 25/5;
  System.out.println(data); }
```

```
catch (NullPointerException e) {
  System.out.println(e); }
  Finally {
    System.out.println("Finally block is always
      executed");}
  System.out.println("rest of the code...");
}}
```

Output - 5

Finally block is always executed
rest of the code...

- throw keyword - the throw keyword is used to explicity throw an exception.

Ex -
```
public class TestThrow {
  static void validate(int age) {
  if(age<18)
  throw new ArithmeticException("not valid");
  else
  System.out.println("Welcome to vote"); }
  public static void main(String []args){
  validate(13);
  System.out.println("rest of the code..."); }}
```

Output =

Exception in thread main java.lang.ArithmeticException:
notvalid.

- throws keyword - i sused to declcare an example exception. It gives an informatn to be programmer that there many occur an exception so it is better for the programmeo to provide the exception handling code so that normal flow can be maintained.

Ex-

```
import java.io.IOException;
class Testthrows {
 void m() throws IOException {
  throw new IOException("device error"); }
void n() throws IOException {
  m(); }
  void p(){
  try{
  n(); } catch(Exception e) {
System.out.println("exception handled"); }}
  public static void main(String []args){
   Testthrows obj = new Testthrows();
   obj.p();
System.out.println("normal flow..."); }}
```

Output - exception handled
normal flow...

- Custom Exception - creating your own Exception that is known as custom exception or user-defined exception.

Ex-

```
class InvalidAgeException extends Exception {
 InvalidAgeException(String s) {
  super(s); }}
class TestCustomException {
 static void validate(int age) throws InvalidAgeException {
 if(age<18)
  throw new InvalidAgeException("not valid");
 else
 System.out.println("Welcome to vote"); }
 public static void main(String args[]) {
  try{
  validate(13); }
```

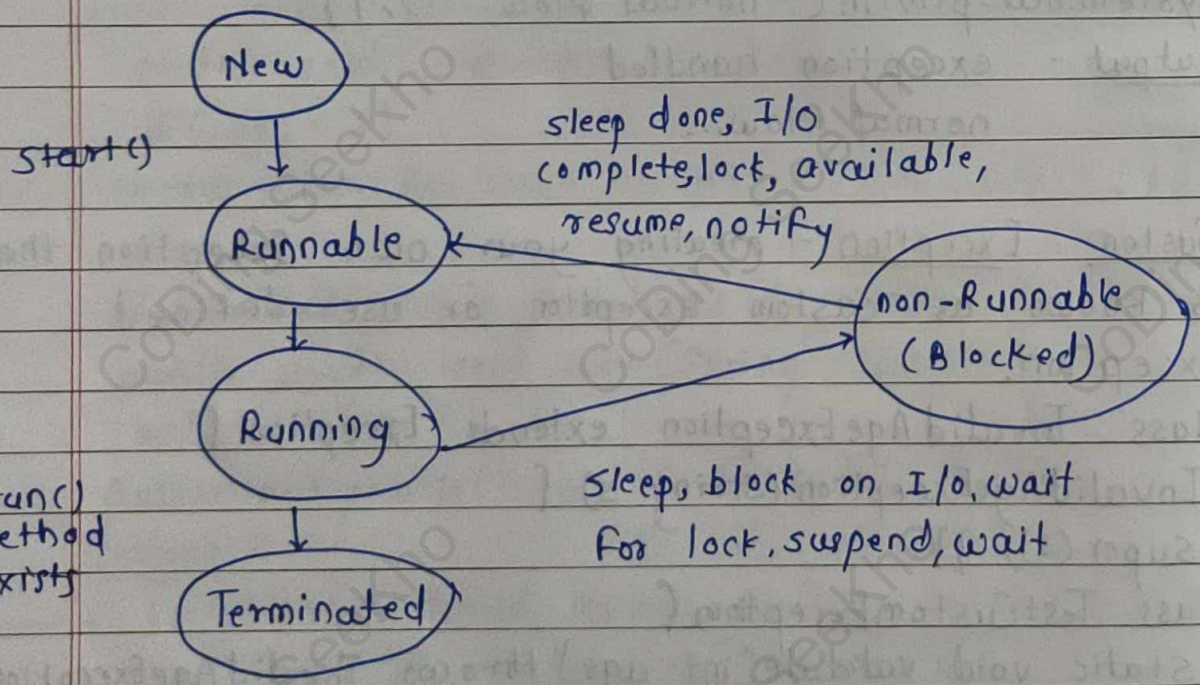
```
catch(Exception m){
System.out.println("Exception occured: "+m);}
 System.out.println("rest of the code...");
}}
```

Output : Exception occured: InvalidAgeException: not valid rest of the code...

• Multithreading •

Multithreading in java is a process of executing multiple threads simultaneously.

• Life cycle of a Thread -



• Commonly used Methods of Thread class -

1) public void run()
2) public void start()
3) public void sleep()
4) public void join()
5) public int getPriority()
6) public String getName()

7) public void setName()
8) public Thread currentThread()
9) public int getId()
10) public Thread.State getState()
11) public boolean isAlive()
12) public void yield()
13) public void suspend()
14) public void resume()
15) public void stop()
16) public boolean isDaemon()
17) public void setDaemon
18) public boolean isInterrupted()
19) public void interrupt()
20) public static boolean interrupted()

- Runnable interface - the Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.

(1) public void run() :- is used to perform action for a thread

- Starting a thread - start() method of Thread class is used to start a newly created thread.

Ex-
```
class Multi extends Thread {
  public void run() {
  System.out.println("thread is running...");}
  public static void main(String []args){
  Multi t1 = new Multi();
  t1.start();
}}
```
Output - thread is running...

° Synchoronized Method °

- If you declare any method as synchronized, it is known as synchronized method.

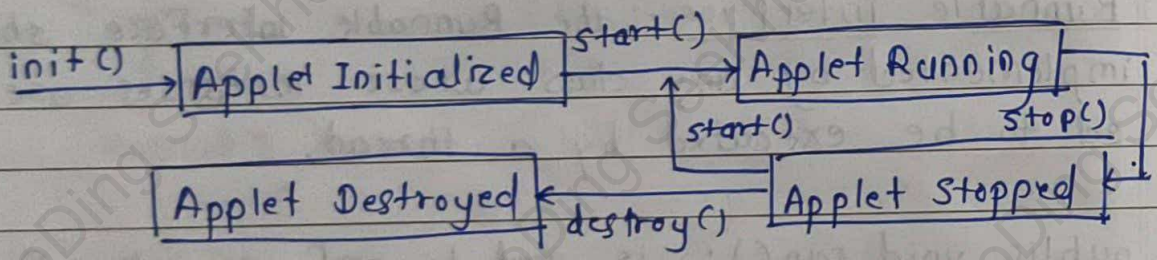
° ThreadGroup °

Java provides a convenient way to group multiple threads in a single object.

° Applets °

An a applet is a program that comes from server into a client and gets executed at client and displays the result.

- Life cycle of an applet.



- public void start () - After init() method is executed, the start method is executed automally.
- public void stop () - The method is executed when the applet focus.
- public void destory() - this method is executed only once when the applet is terminted from the memory.

o Types of Applets - An applet developed locally & stored in a local system is called local applets.

1) Local Applets.
2) Remote Applets.

ex.1. Write a program to pass empoly name & id number to an applet.

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet {
  String n;
  String id;
  public void Init() {
   n = getParameter("t1");
   id = getParameter("t2"); }
public void paint( Graphics g) {
  rawString("Name is :" +n, 100, 100);
  g.drawString("Id is :" + id. 100.150); }}
```

ex- Hai.java

```
import java.applet.*;
import java.aw.*;
class hai extends Applet {
  private String defaultMessage = &quot;Hello!&quot;;
  public void paint( Graphics g) {
String input FromPage = this.getParameter (&quot;Message&quot;) if
  (inputFromPage == null) inputFromPage = defaultMessage;
  g.drawString (inputFromPage, 50, 55); }}
```

- **Event Handling** - event handling is at the core of successful applet programming.

- The Delegation Event Model - The modern approach to handling events is based on the delegation event model, which defines standard & consistent mechanisms to generate and process events.

- ActionEvent Class - An ActionEvent is generated when a button is pressed, a list item is double-clicked, or a menu item is selected.

- **MouseEvent class** -

MOUSE_CLICKED	MOUSE_MOVED
MOUSE_DRAGGED	MOUSE_PRESSED
MOUSE_ENTERED	MOUSE_RELEASED
MOUSE_EXITED	MOUSE_WEEL

- **WindowsEvent class** -

WINDOW_ACTIVATED	WINDOW_GAINED FOCUS
WINDOW_CLOSED	WINDOW_ICONIFIED
WINDOW_CLOSING	WINDOW_LOST FOCUS
WIDOW_DELICONIVATED	WINDOW_OPENED
WINDOW_STATE_CHANGED.	

- **Files & Streams :-**

Text stream classes for reading data.

Reader

- InputStreamReader
  - FileReader
- BufferedReader
  - LineNumberReader
- FilterReader
  - PushbackReader
- CharArrayReader
- PipedReader
- StringReader

Text stream classes for writing data

Writer

- OutputStreamWritier
  - FileWriter
- BufferedWriter
- FilerWriter
- CharArrayWriter
- PipedWriter
- StringWriter

- Handling Keywords Events -

When a key is pressed, a KEY_PRESSED event is pre' generated. This results in a call to the keyPressed() event handler, When the key is released, a KEY_RELEASED event is generated & the keyReaksed() handler is executed.

Ex -

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class SimpleKey extends Applet implements Key
  Listener {
  String msg = " ";
  int X=10, Y=20;
  public void init() {
  addKeyListener(this);
  requestFocus(); }
```

```
public void keyPressed(keyEvent ke) {
    showStatus("Key Down");}
public void keyReleased(keyEvent ke){
    showStatus("Key Up");}
public void keyTyped(KeyEvent ke){
    msg += ke.getKeyChar();
    repaint(); }
public void paint(Graphics g) {
    g.drawString(msg, X, Y); }}
```

○ Stream ○

A stream can be defined as a sequence of data.

- InputStream - read data from a source.
- OutputStream - writing data to a destinatn.

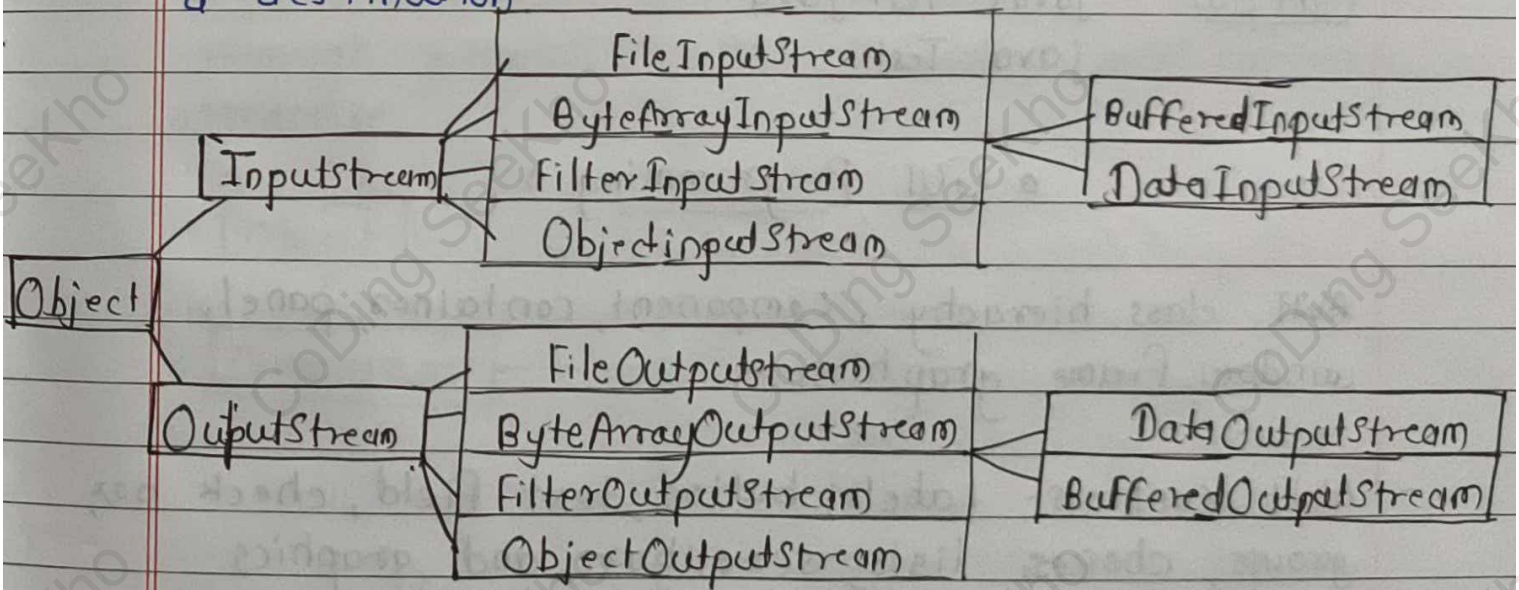
Source ⟶ Program ⟶ Destination

○ **Byte Streams** - to perform input & output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently but the most frequently used classes are, FileInputStream & FileOutputStream.

Ex -
```
import java.io.*;
public class CopyFile {
  public static void main(String []args)throws
        IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
try {
    in = new FileInputStream("input.txt");
    out = new FileOutputStream("Output.txt");
```

```
int c;
while ((c = in.read()) != -1) {
    out.write(c); } }
finally {
if (in != null) {
    in.close(); }
if (out != null) {
    out.close(); } } }
```

- Reading & Writing Files Text input/Output - a stream can be defined as a sequence of data. The InputStream is used to read data from a source & the OutputStream is used for writing data to a destination



Ex- Write a program to read data from the keyword & write myfile.txt. file.

```
import java.io.*;
class Test {
 public static void main(String [Jargs) {
   DataInputStream dis = new DataInputStream(System.in);
   FileOutputStream fout = new
   FileOutputStream("myfile.txt");
   System.out.println("Enter text @ at the end;");
   char ch;
while ((ch=(char)dis.read())!="@")
   fout.write(ch);
   fout.close(); }}
```

Output - javac Test.java
java Test

o GUI Programming o

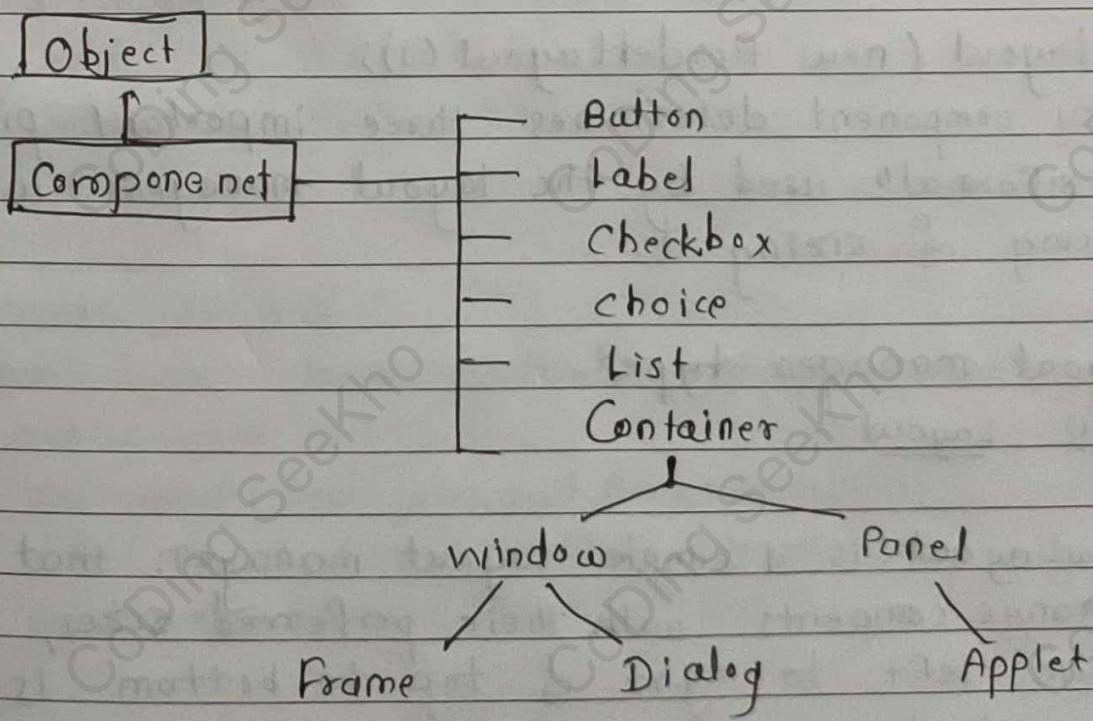
AWT class hierachy, component, container, panel, window, frame, graphics.

- AWT controls- Labels, button, text field, check box, groups, choices, lists, scrollbars, and graphics.
- Layout Manager - Layout manager types: border, grid & flow.

1. AWT Classes -

Class	Description.
AWTEvent	Choice
AWTEventMulticaster	Color
BorderLayout	Component
Checkbox	container
CheckboxGroup	Cursor
CheckboxMenuItem	Dialog
Dimension	Frame
Event	Graphics
FlowLayout	

2. GUI (Graphical User Interface) - the applicatn through graphics. User Friendly. application attractive.



The classes that extend Container class are known as containes Frame, Dialog & Panel.

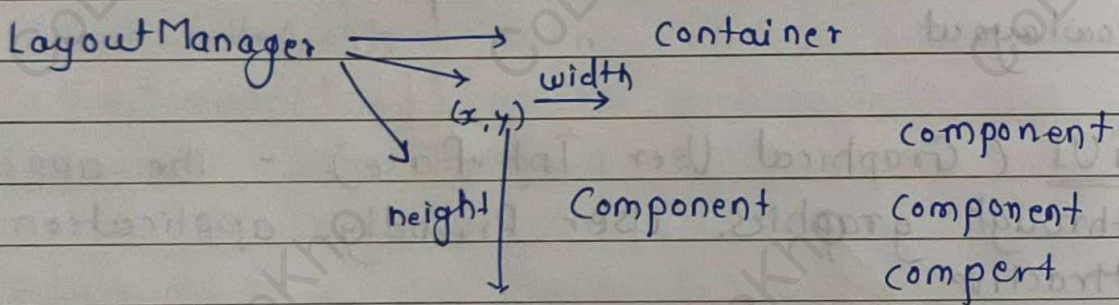
Date
Page

- Window - the window is the container that has no borders & menu bars.

- Panel - the is the container that doesn't contain title bar and menu bars.

- Frame - the Frame is the container that contain title bar and can have menu bars.

- LayoutManager at work. =



- setLayout (new BorderLayout());
Every component determines three important pieces of inFormatn used by the layout manager in placing & sizing it.

- Layout manager types-
Flow Layout -

FlowLayout is a simple layout manager that tries arrange compents with their perferred sizes, from left to right & top to bottom is the display.

```
import java.aw.*;
public class Flow extends java.applet.Applet {
 public void init() {
    add(new Button("Two"));
    add(new Button("Three"));
 }}
```

- **Grid Layout** -

arranges components into regularly spaced rows & columns. The components arbitarily resized to fit in the resulting areas; their minimum & preferred sizes are consequently ignored.

Ex-
```
import java.awt.*;
public class Grid extends java.applet.Applet {
 public void init() {
  setLayout(new GridLayout(3,2));
  add(new Button("One"));
  add(new Button("Two"));
 }}
```

Ex -
```
import java.awt.*;
public class Border extends java.applet.Applet {
 public void init() {
  setLayout(new java.awt.BorderLayout());
  add(new Button("North"),"North");
  add(new Button("East"),"East");
 }}
```

Output :- Compile : javac. Border.java
Run : appletviewer Border.java.

- Simple example of AWT by inheritance -
- Simple example of AWT by association.

★ AWT Controls -

- Labels - the easist control to use is a label.

Ex.
```
public class LabelDemo extends Applet {
 public void init() {
  Label one = new Label("One");
  Label two = new Label("Two");
  add(one);
  add(two);
 }}
```

- Buttons - the most widely used control is the push button.

```
import java.awt.*;
import java.awt.event.*;
import. java.applet.*;
public class ButtonDemo extends Applet implements
     ActionListner {
  String msg = " ";
  Button yes, no, maybe;
 }}
```

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

- TextField - the textfield class implements a single-line textentry area, usually called on edit control.

- TextArea - Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multipine editor called TextArea.

- CheckboxGroup - it is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time.

- choice Controls - the choice is used create a pop-up list of items from which the user may choose.

- Graphics - AWT supports a rich assortment of graphics methods

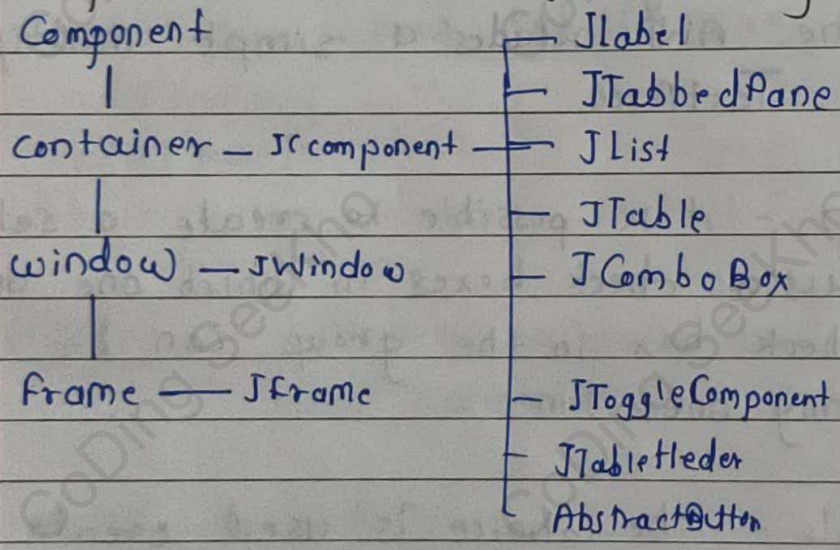
```
import java.awt.*;
java.applet.*;
public class Lines extends Applet {

    g.drawLine(0, 0, 100, 100);
}
```

- Drawing Rectangles - the drawRect() & fillRect() methods display an outlined & filled rectangle, respectively

- Swings :- Swing is a set of classes that provides more powerful & flexible components than are possible with the AWT.
  - Swing is a GUI widget toolkit for Java.

- Hierarchy for Swing components -
  Important classes of javax.swing.



- JApplet - Fundamental to Swing is the JApplet class, which extends Applet. Applets that use Swing must be subclasses of JApplet.
  - the add() method of Container can be used to add a component to a content pane

- **Window Panes** - swing which contains a free area inside it. this free area is called 'window panel'.

1) Glass Pane
2) Root Pane
3) Layered Pane
4) Content Pane

Q. Write a program to display text in the frame.

```
import javax.swing.*;
import java.awt.*;
class MyPanel extends JPanel {
public void paintComponent (Graphics g) {
super.paintComponent (g);
setBackground (color.red);
g.setColor (Color.white);
  etFont (new Font ("Courier New", Font.BOLD, 30)); }}
class FrameDemo extends JFrame {
  FrameDemo () {
   Container c = getContentPane ();
MyPanel mp = new MyPanel ();
c.add (mp);
setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE); }
 public static void main (String [] args) {
FrameDemo ob = new FrameDemo ();
ob.setSize (600, 200);
ob.setVisible (true); }}
```

- **TextFields** :- the Swing text field is encapsulated by the ~~area~~ JTextComponent class, which extends JComponent. It provides functionality that is common to Swing text components.

```
import java.awt.*;
import javax.swing.*;
public class JTextFieldDemo extends JApplet {
 JTextField jtf;
 public void init() {
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
   pane jtf = new
   JTextField(15);
   contentPane.add(jtf); }}
```

◦ Buttons ◦

Swing buttons provide features that are not found in the Button class defined by the AWT.

Ex

```
import java.awt.*;
import java.awt-event.*;
import javax.swing.*;
public class JButtonDemo extends JApplet implements
  ActionListener {
 JTextfield jtf;
 public void init(){
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
 }}
```

o CHECK BOXES - which provides functionality of a check box, is a concrete implementation of AbstractButton. Its Immediate superclass is JToggleButton.

~~JCheck~~

```
import java.awt.*;
import java.awt.event.*;
import javax.swing;*;
public class JCheckBoxDemo extends JApplet impleme-
    nts ItemListener {
  JTextfield jtF;
public void init() {
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout());
ImageIcon normal = new ImageIcon("nomal.gif");
JcheckBox cb = new JCheetBox("C", normal);
cb.setRolloverIcon(rollover);
cb.setSelectedIcon(selected);
cb.addItemlistener(this);
ContentPane.add(cb);

cb = new JCheckBox("C++"), normal);

public void itemStateChanged(ItemEvent ie) {
 JCheckBox cb = (JCheck Box) ie.getItem();
jtf.setText(cb.getText());
 }
}
```