OOP (Java)

Kunal

Detail Notes text file is available in Github repo.

Class:- A class is a name group of properties and functions.

Syntax:- class Car { → Name of class.
→ members. (same and diff. data type)
}

→ Name of the class is itself a data type.

Object:- The instance of the class is called object.
→ object occupy space in memory.

→ Using the dot operator (.) to access the member of class.

→ dot operator links reference variable with the instance variable.

Instance Variable:- All the variable that are inside the object is called instance variable.
e.g:- (14, "A1", 37.0)

Reference Variable:- The variables that defined our object is called reference variable.
e.g:- (Student Std1 = new Student();)
↓
Reference Variable

Syntax:-
```
Std1.rollno;
```

new keyword:- It orders to create objects/instance of the class we must use new keyword.

Syntax:-
```
Student std1 = new student();
```

→ This new keyword allocate memory dynamically.

Dynamically (means give memory at run time and return reference).

→ All the class objects in java must be allocated at run time.
→ Default values can assign in the class.

```
Class Student {
    int rollno=30;
}
```

Default values:-
int = 0
string = null
float = 0.0

Constructors:- Student();

→ Same name as class name.
→ A constructor basically defined what happened when the object is created.
→ set default values for all the instance/object of the class.
→ NO return type, Argument or NO Argument constructor.
→ Constructor is the special type of function in the class that have default values. (already built).
→ NO Argument constructor is by default constructor.

```
syntax:- Student() {
            Argument or NO.
         }
```
→ Assign by default value.

this keyword:- this keyword is replace the name of the object. Inside the constructor use this keyword like Ali.rollno=13; → so we use this instead of Ali.

```
Student() {
    this.rollno=13;
}
```

This keyword is mostly use in the Argument constructor to assign the value of argument to member of class and use when we have same variable name.

```
Student(int rollno){
    this.rollno= rollno;
}
```
↓ Defined in class

Constructor overloading:- It means when you create two constructor of a class one is argument and other with no argument it means when you the constructor is arguments it overload the non-argument constructor.

Copy Constructor:- It is the method to call the another constructor or copy the value of one object to the other object.

Syntax:-
```
Student (Student Std1) {
    name = St1.name;
    age = Std1.age;
}
```

Note:- when you create object of class and assign to another object it have the same output.

```
student one = new student();
student two = one;
```

Wrapper Class:- It is the class that converts variable into the objects.

e.g:- `int a=10;` → It is simple

`Integer num=45;` → num is the object.

by using num. → we use many functions.

final keyword:- It is the keyword using for content modified.

e.g:-
```
final int a=2;
a=3;
```
→ It cannot modify.

→ always initialize when declaring because it cannot change.

→ It cannot change value in primitive data type like Int, etc.

-hierarchy. When you

Package :- A package is simple a folder.
for example package com.Ali → It means that the Ali folder is inside the com folder. It has two folders.

→ The use of package is that you cannot create two classes or function of same name in one package. For this create another package.

→ The package show the hirerarchy manner which means one folder inside this another folder and inside another folder.

Import statement:- It is the method to import the statement of one package to another.

→ By using ~~Ctrl + click~~ Alt + Enter after write the function.

This show that:-

```
import static com.kunal.packages.b.Message.message;
```

com.kunal.packages.b → Package Name
Message → class name.
message → function name.

→ private access specificer cannot import.

→ I think this is only happen in ascending package decelaration.

Static : The properties that are not change for any object is the static one. The best example is population. We declare these as static. Static is not related to object.
We class class name instead of this keyword in the case of static.

EMP:.

Note:- when you access, modify the static variable plz don't use the object name use the class name (bcz is better).

The static variable cannot depend on the object. When you don't create the object you can still use the static variable.

P.S.V.main :

Why the main is declared Static?

because you can use the main function without creating any object of that class.

The main method is also in the class so you create object but the use of static access the functions.

The static belongs to class not to object.

Note:- Inside the static functions you cannot use non-static methods bcz static only access static data.

→ Something is not static belongs to object.

→ Inside the static function you can create object of the non static function and then access.

Example:-

```
Static void fun(){
    Main obj = new main();
     obj.greeting();
}
void greeting() {
    System.out.println("Hello");
}
```

Q:- Why don't create the object for non-static methods?

A:- bcz in the end you can call in the main function and the main function is static so create object first and then access the function or variable.

We also don't use this keyword inside the static bcz this is represent the object.

→ Static statement only runs one time for the first object only.

Inner Classes:

The class inside the class is called inner class.

→ only inner classes can be static, no outside class is static.

→ outside classes cannot be static bcz it can depend on another class.

→ The inside class you cannot make of the class bcz it is depend on outer class. so, for this we can create inner class as static or inner class class separated.

Syntax:-

```
Public class A {
static class B {
        int age;
      B (int age) {
       this.age = age;
        }
    }
P.S.V.main (String [] args) {
   B b1 = new B (12);
 }
}
```

```
class B {
    int age;
      B (int age) {
      this.age = age;
      }
Public class A {
    P.S.V.main (String [] args) {
    B b1 = new B(12);
     }
     }
```

Singleton class:

The class can only make one object.

→ It allow to make only one object.

Features:

There are four main fundamental properties and features of Object Oriented Programe.

- Inheritence
- Polymorphism
- Abstraction
- Encapsulation.

Inheritence: In real life the inheritence means the properties of parents pass to children and children use the properties.

→ The class is defined is the Base class and parent class. and the class is inherited with the base class is called child class or derived class.

→ Inheritence means the transfer the property of parent class to the child class.

Syntax: By using extends keyword we inherit the class.

```
Class Student extends faculty {
    int age;
}
```

// It means that all the property of faculty class is also a part of student class and one addition datamember is age in child class.

Super Keyword:

In order to call the parent class constructor we use the super keyword. Super keyword initialize the value that is declared in parent class.

→ Private datamember can't access in superkeyword bcz the private cannot access in another class.

Syntax:

```
Box(int w, int h, int l) {
    super(w, h);
    this.l = l;
}
```

Private: The private keyword is used to access the function and datamember only inside the class. It can't access in outside the class or inherited class.

Note:- The parent class can't access the property of child class.

Q:- what happen when you create object of parent class and assign memory of child class?

A:- It can only initialize the value not access the value bcz for access it can go to the type of object.

e.g:- Box b1 = new Boxweight(10,20,30,40){

S.O.P(b1.w-40); → It show error and can't access.

}

→ when you do the reverse process it shaw the error bcz the parent class or upper class doesn't the behaviour of lower or child class.

e.g:- Boxweight b1 = new Box(5,6,7){

// Error

}

→ Every class has object as super class. Every single class inherit the object (Main) class.

→ you can also access the data member of parent constructor in child constructor using

super(l,b);

super.l;

→ Super class doesn't know what the sub or child class contains but child class care what the parent or super class have.

→ super can also take the copy constructor argument.

Single Inheritence: One class extends with the another class.

- It is a type of inheritence.
- In this the one class is parent and other one is child.

e.g:- zain extends Ali { Parent
// Ali is ~~child~~ class and zain is child class.
}

Multi level inheritence: It is also a one type of inheritence. In this one class inherit with the another class and the derived class from this is the parent of another class.

e.g: Class zain extend Ali {

}

Ali2 extends ~~Ali~~ zain {

}

Multiple Inheritence: It is also a type of inheritence. In this type of inheritence one class extends with more than one classes.

e.g:
Class A and class B both are the parent of class C.

Q:- The Question is if class A has n=5 and class B has n=10. So, when you create object in class C and called the n like (C1.n). What is the ouput?

→ The compiler of java can confuse, that's why the multiple inheritence is NOT possible in java.

→ But there are another ways to support multiple inheritence in java and that is ~~info~~ (interfaces).

Hierarchial Inheritence:

It is the another level of inheritence. In this type the function of multi level inheritence means one class inherited by many classes.

e.g:- Class B, Class C are inherited with class A.

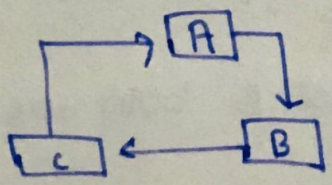
```
e.g:- class B extends A {
      }
      class C extends A{
      }
```

Hybrid Inheritence:

The hybrid inheritence is the combination of single or multiple inheritence. But we know the java doesn't support multiple inheritence that's why hybrid inheritence is also not possible in java.

→ But another method is same as multiple is (Interfaces).

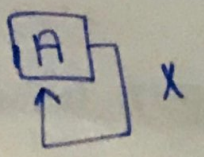
e.g:- class A inherit with class B and class B inherit with class C and class C with A.



Note:-

→ But the class cannot inherit with itself (It is not possible). Simple means the class cannot super class of itself.

```
e.g:- class A extends A{
                // Error.
      }
```



X

triangle();

Polymorphism: The word polymorphism split in two words.

poly~~ce~~ means many morphism means ways to represent so, polymorphism means many ways to represent single entity.

→ Java support polymorphism.

→ Polymorphism occurs during inheritence (mostly).

- One function represent with same name in different classes and for each class object it has different output.
- You can also create object using parent and child class.

e.g:- Ali ali = new Student();

→ In this the object is of student class.

Example:

```
class shape {
    Public void area(){
    S.O.P("This is shape class");
    }
class square extends shape {
    Public void area(){
    S.O.P("The area of square is 2x2");
    }
class triangle extends shape {
    public void display(){
    S.O.P("Triangle l.b.");
    }
Public class Main{
    P.S.V. main (String [] args){
    shapes shapes = new shape();
    shapes.area();
    triangle tri = new triangle();
    tri.area();
}}
```

→ The object select the constructor using the arguments no. is also called type of polymorphism.

Types: Types of polymorphisms are:

① Compile time | static polymorphism.
→ Achieved via method overloading. (Java doesn't have opr. is over loading).

Over loading: when the functions of the class have same name but the arguments and return type are different.

Example:- multiple constructors.

```
A a = new A();
A a1 = new A(3,4);
```

- This is compile time polymorphism. bcz it can check the constructor or which function is call during compile time but memory assign during run time (dynamically).
- when you pass int instead of double java automatically change your int to double. (4 to 4.0);

② Run time | dynamic Polymorphism
→ Achieved via method over riding.

Over riding:

@Override → this is called notation. It is used to check our function is over riding or not. Just write @override.

- Overriding means when the same function is created in both parent and child class with same name and everything will be same so in this case the child function is override the parent function.
- when you created the object of child class it can call the child function and when create the object of parent class it can call the parent class function.

Note:-

Parent obj = new Child();

Here, which method will be called depends on the constructor (Child()).

This is known as UPCASTING. and the entire process is called overriding and how the overriding works.

→ By default in java every class extends with obj class.

Final Keyword:

- Final keyword is used to create constant.
- We cann't override the method which is final.
- Overriding can occur during ~~compile~~ run. time and final is constant that's why it can't override and generate error.
- We prevent the data using final keyword and prevent inheritence using final keyword before class.

e.g:-

```
final class Ali {

}
class Zain extends Ali {

    // Error.
}
```

- When we create class final and all the functions inside the class automatically declared final.

Late Binding

Late Binding also known as dynamic binding or runtime binding occurs when the binding between a method or function call and its implementation is determined at runtime.

Early Binding

Early Binding also known as static binding or compile time binding occurs when the binding between a method or function call and its implementation is determined at compile time.

Static Method:

Static methods doesn't depend on the object of the class. To access static method and change we use class name we already discuss.

e.g: static void greeting(){
    s.o.p("Hi");
}

main → use Box.greeting(); instead of b1.greeting();

main method → class name; object name

The question is can we override the static method?

→ It can't overriding.

Note:-

When you inherit the class and declare the static function with class name it always show parent class function bcz it can't override and don't depend of class object.

Encapsulation:

Wrapping up the implementation of the data members and methods in class.

In simple means it can hide the data from outside.

Abstraction:

Abstraction means hiding the unnecessary details and showing valuable information.

→ All the extra information is hiding from us and show the only relevant data.

e.g: when you have a car and need a key to start the car but don't need what is the mechanism behind the car (start).

e.g:- Array list = new Array();

we use List. method to access all the functions of the array but we don't need to know what are the reasons behind this. (Abstraction).

→ Encapsulation are the hiding the data using access specifiers (Public, private, protected) in the class.

→ we use private keyword to hide the data in the class.

Access specifier:

There are four types of access specifiers.

Access specifiers are the restriction to access the data. It is used before the data member and member functions.

- Public: use public keyword it can access anywhere.

  e.g:- public int x = 6;

- Private: The private can only access inside the class. It cannot access outside the class so we use set the value and get the value function inside the class for private access specifier to access in other and main class.

- Protected: The protected can only access inside the class and the inherited class of the class.

- Default: When you don't use public, private and protected keyword before any data member it means it is default and access in overall the package.

  → so, it is package level access specifier.

Access levels:

	Class	Package	subclass (same Pkg)	subclass (Diff Pkg)	world (diff pkg & not sub)
Public	✓	✓	✓	✓	✓
Protected	✓	✓	✓	✓	
Private	✓				
Default	✓	✓	✓		

→ Private show the high level of restriction or data hiding. we use set and get methods in class to access private in main.

→ Protected is only access outside the package when it is call in sub class.

Packages:

Packages are folder used to divide or catagorized the class.

→ Types of Packages — In-built, User defined

Inbuilt (Java.)

Lang Package
↓
It contain the java language specific stuff.
Example (+,-) etc.
Q:- You think I didn't import this package.
A:- It is already default imported.

io package
↓
Input and output package.
- For read and write the file.
- We use java.io
- .io is the sub-folder.

util
↓
It contains utility stuff. means data structure stuff.
It is a collection of data structure stuff.

awt, applet
↓
For Development.
e.g.. create button, GUI application.
Note:-
No need to learn at this stage.

net package:- related to internet.

Example:-

lang package is already built.

→ you know object class is the parent class of all the classes.

→ We can create the object of object class and call the value. in main.

Hash code:

A unique representation of number. when you print the same value using hashcode it will give you the different output.

→ You can modify the hashcode value to original value.

→ It is used for uniqueness of object or number.

Equal Method:

Equal method is used to check two variable and object to check they are pointing the same object. or not.

Instance of operator:-

It is used to check the object is from the class or not. The answer is true or not.

e.g:-

obj → is the instance of A class.

System.out.println(obj instanceof A); → True

System.out.println(obj instance of B); → False.

system.out.println(obj instance of object); → True.

Bcz every class is subclass of object class and return true in case of inheritence.

getclass method:

It is the method used to know the class of the object. (getclass()).

e.g:- S.O.P(obj1.getclass()); → It give class and package name.

→ We use getclass(). operator to get the all the methods of the class.

e.g: S.O.P(obj1.getclass().getName());

Abstract Classes:

- It give you the only necessarily information.
- In this parent class give the choice to child class use the function according to our choice.
- The super class means parent class only give the method.
- The parent class only give the function not its body, the child define the body using overriding.

Syntax:-

```
abstract void career (String name);
```

(keyword ← abstract, void → return type, career (String name); → name)

- We define the method in the child class according to our choice.

Note:- If a class contain one/two or more abstract method the class must define abstract also.

Example:-

```
abstract class parent {

    abstract void name(string x);

}
```

// we use abstract before class name bcz we make one abstract method in class.

```
class son extends parent {
        void name {          → (string x)
         System.out.println("My mame is" + x);
        }
```

→ just make object and call in main class.
like x1.name("Khaqan");

Note:- we can't create object of abstract class. but make constructor of object class.

→ We can't creat abstract class objects bcz it has no body.

→ We can't create abstract static method, abstract static constructor but we create simple static method and normal functions.

Final abstract class:

We can't create final abstract class bcz you can use this class in inheritence. The final keyword can't change.

so,

```
final public abstract class {
                }              // ERROR
```

→ Multiple inheritence is also not possible in abstract classes.

Interfaces:

It is not the class.

→ By default variables are static and methods are final in interfaces.
→ It is the contract in simple language and all classes flow the contract.
→ multiple inheritence support in interfaces.
→ We use keyword implements instead of extends in interfaces.
→ The main shinning point is we use multiple ~~extends~~ implements keyword.
→ One interface implements with another interface but only abstract classes it is not possible.
→ Class support multiple interfaces using Implements keyword but only single inheritence using extends.
→ Full abstraction.

Syntax:

```
interface Brake{
        void brake();
    }
interface Engine{
    void start();
    void stop();
}
class car implements Brake, Engine {
    // It show error if you don't call the interface method in the class.
    void brake(){
        S.O.P("This is brake");
    }
    void start(){
        S.out("This is start");
    }
```

```
void stop(){
  s.o.p("This is stop");
} }
```

→ These are all override methods.
→ Interfaces break the hirearchy of inheritence. When you use multiple interfaces they don't care and point the same inferface.
→ If I access the variable using this.

```
Engine car = new Car();
```

interface — obj name — constructor.

```
car.a; // show error bcz it is not in Engine.
```

→ It create the object of constructor type but show the data of interface type. and a is not present in engine.
→ In performing critical code not prefer the interfaces.

This concept is variable type.

Extending Interfaces:-

```
interface A {
    }
interface B extends A {
    }
class C implements B {

    }
```

→ Static interface method always have a body.

```
static void greet() {
    S.o.p("Greeting Function");
}
```

→ class with class name in the main.

Nested Interface:

The interface that are inside the class is called nested interface.

→ The nested interface is declared public, private, protected but the top interface we declared public and default.

e.g:

```
class A {
    interface nested {
        boolean isOdd(int num);
    }
}
class B implements A.nested {
    boolean is odd (int num){
        return (num % 1) == 1;
    }
}
```

Enum:

- It is also called enumerations.
- It is a group of variable that we cannot change means it is constant.
- Enum used when we know the number of objects of class.

Syntax:

```
enum week {
    Monday, Tuesday, Wednesday, ....., Sunday
}
```

(enum is the keyword)

- These are enum constants.
- Everyone is public, static and final.
- Final means can't create child Enums.
- static means directly access using **week.**
- Type of all these are week.

```
P.S.V.m (String[] args){
    Week week;
    week = Week.Monday;        ——→ output = Monday.
    for (Week day : Week.values()){
        S.O.P(day);
    }   // Print all days.
```

start from 0.

→ week.ordinal(); → means print the position. (0 →)

→ we don't use constructor for Enum because it create problems.

→ Enums cannot show inheritence.

→ It can implements interfaces.

→ we use and its directly print the value.

JFrame :

to add components.

GUI:

- Graphical user interface
- windows base application. (with data base).

Java swings API → Aleardy built in code.

- Java swing is an enhanced version of AWT.
  AWT (Abstract Windowing Toolkit) → previous version

Package:- javax.swing package → import before GUI.

Diff B/w Java AWT and Java Swing

Java AWT	Java Swing
package → java.awt.*;	package → javax.swing.*;
• AWT components are platform depenent. → run on one platform.	• Java swing components are platform independent. → run on diff. platform.
• AWT components are heavy weight.	• Swing components are lightweight.
• AWT doesn't support pluggable look and feel. → means not change colour etc.	• swing supports pluggable look and feel.
• AWT provides less components than swings.	• swing provides more powerful components such as table, list, scroll panes, colourchoser, etc.
• AWT doesn't support MVC (Model view controller).	• swing support MVC.

conclusion:- Java remove the concept of AWT and follow the new version of AWT is Java swing. Java swing fulfil all the drawbacks of AWT. Java swing is the child of AWT class more prefable is Java swing.

JFrame :

A GUI window to add components.

- First create instance of JFrame in the main.

```
JFrame frame = new JFrame();
```

- Before creating object first import package → import javax.swing.JFrame;
- To make our Frame visible set visibility to true.

```
frame.setVisible(true);
```

- To set the dimension of frame when you run the programe and to set width and height of frame.

```
frame.setsize(420,420);
```

- To set title of frame and its appear on top of the frame use

```
frame.setTitle("KHAQAN FIRST Application");
```

- To exit of application when you click of [x] set.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- To prevent the window Frame resizable sets to False.

```
frame.setResizable(false);
```

- To change Frame logo which is appears on the top left first import and then create an image Icon and then set.

```
ImageIcon image = new ImageIcon("file Name");
frame.setIconImage(image.getimage());
```

- To change frame background colour ~~of the~~ use

```
frame.getContentPane().setBackground(Color.green);
```

also use RGB colours
new color(0,0,0)
Range of colour

also use Hexadecimal color values
new colour(0xFFFFFF);

→Two methods of creating Frame one method is already done and the second method is create constructor of class and use `this.` instead of object name and just create instance in the main like this `new classname();`

Java Labels:-

JLabels is a GUI display area for a String of text, an image, or both.

- First create instance and import the package.

```
import javax.swing.JLabel;
JLabel label = new JLabel();→ object / Instance.
```

- To set text use two methods one is pass inside the constructor like JLabel("——"); and second is use setText.

```
label.setText("Khaqan Notes");
```

- To add label on the frame use

```
frame.add(label);
```

- We also set ImageIcon with the label for this first create image Icon object and then set.

```
ImageIcon image = new ImageIcon("file name");
label.setIcon(image);
```

Horizontal Position:

To set the JLabel Text position horizontally use `label.setHorizontalTextPosition(JLabel.` center / RIGHT / LEFT `);`

This can set the text LEFT, RIGHT or CENTER to Image Icon.

Vertical Position:

To set vertically use

`label.setVerticalTextPosition(JLabel.` TOP / center / Bottom `);`

This set to TOP, BOTTOM or CENTER to Image Icon.

- To set label colour Text use
  `Label.setForeground(color.green);` → same use RGB and Hexadecimal. → `new color( );`

- To set the label font, style and size use
  `Label.setFont(new Font("MV BOLI", Font.PLAIN, 20));`
  Name, type/style, size

- To set GAP between Image Icon and JLabel use
  `Label.setIconTextGAP(20);` → Also Accept -ve int.

- To set label Background color use
  `label.setBackground(color.black);`

- To show the label background color use set opaque
  `Label.setOpaque(true);` → Display Background color.

- To create Border outside the JLabel Text and Image Icon.
  First create object like
  `Border border = BorderFactory.createLineBorder(Color.green,3);`
  Now set to label
  `label.setBorder(border);`

- To set vertical position of Icon + Text within the label use
  `label.setVerticalAlignment(JLabel.TOP);`

- To set Horizontal position of Icon + Text within the Label use
  `label.setHorizontalAlignment(JLabel.CENTER);`

- To set layout of frame use
  `frame.setLayout(null);` → By default null.

- To set a particular position to layout with the frame use
  `Label.setBounds(x, y, width, height);` *: Move in pixels.
  x, y → distance from border (x: Left, y: Right)
  width → of label
  height → of label

- To set automatically size of frame and all the components use `frame.Pack();` method.

→ No need to set frame size and Label size.
→ This method is use after adding all the components in the frame.

JPanels: A GUI component that functions as a container to hold other components.

- First create the object/instance of JPanel

```
JPanel redPanel = new JPanel();
```
according to colour.

- the add in the JFrame.

```
frame.add(redPanel);
```

- To set the background colour of the Panel use

```
redPanel.setBackground(Color.red);
```

- To set the size of the panel in the JFrame use

```
redPanel.setBounds(0,0,250,250);
```

- Create more panels same as red panel and set the size of panel according to choice/demand.

→ Create some labels and import Icon to set the labels and then set the label to the Panel using `panel.`

```
redPanel.add(label);
```

→ we can set the label and border layout of panel and label using

```
redpanel.setLayout(new BorderLayout());
```

→ many more to set horizontal and vertical.

JButtons:

A button that performs an action when clicked on.

- We create an action Listener of each button. Action Listener means perform action when you click on the button.
- It is better to create separate class for action Listener.

→ First create object in the main.

```
JButton button = new JButton();
```

→ To set the size of the button use

```
button.setBounds(200, 100, 100, 50);
```

→ To add button in the frame use

```
frame/this.add(button);
```

→ We need to perform an action when we click on the button for this purpose we need to create an action Listener → for this we implements action listener and them Import action listener of the button.

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == button){
        S.out(——);
    }
}
```

→ After this add action listener in the button.

```
button.addActionListener(this);
```

→ We also use lamda expressions for Action Listener instead of creating Action Listener and implements the action Listener.

```
button.addActionListener(e -> S.O.P(——));
```

↓

lamda

- When we write the Enabled (False) inside the action performed it can disable.
- To add text in the button use
  button.setText("Ok");
- To remove the focus around the text inside the button use
  button.setFocusable(false); → set False.
- To add image in the button first create object of image and then add.
  ImageIcon Icon = new ImageIcon("File Name");
  button.setIcon(Icon);
  ↓ import.
- To set the text position in the button horizontally and vertically use
  button.setHorizontalTextPosition(JButton.CENTER);
  button.setVerticalTextPosition(JButton.BOTTOM);
- To set the Font and size of text of the button use
  button.setFont(new Font("Comic Sans", Font.Bold, 25));
  Font Name ↙ type ↙ size ↓
- To set the GAP between Icon and the text use
  button.setIconTextGap(-15);
- To set the button Text colour use
  button.setForeground(color.cyan);
- To set background color of the button use
  button.setBackground(color.Black);
- To set Border of Button use border Factory
  button.setBorder(BorderFactory.createEtchedBorder());
- To disable the button ~~use~~ set Enable to False
  button.setEnabled(false);

→ When we write the Enabled (False) inside the action performed it can only performed Action onced and then disable.

→ If we need to display a label when we click a button so simply create one lable and set the visiblity False and inside the Action Listner of the button set True.

Java Border layout:

A border layout places components in five areas : EAST, WEST, NORTH, SOUTH and center.

- All extra space is placed in the center area.

Let suppose create five panels:-

like JPanel panel1 = new JPanel();

Assign different background to each panel like.

```
panel1.setbackground(Color.red);
```

Set the size like

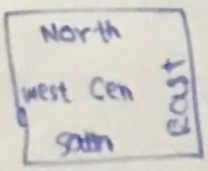
```
panel1.setPreferredSize(new Dimension(100,100));
```

Add in the frame like

```
frame.add(Panel1, BorderLayout.NORTH / EAST / WEST / SOUTH / CENTER);
```

- For add border layout we set the frame layout to new Border Layout.

```
frame.setLayout(new BorderLayout());
```



Frame

- We can also set border margin inside the Border layout (10, 10);
- We can also create subpanels of one panel. like

```
panel5.setLayout(new BorderLayout());
panel5.add(Panel6, BorderLayout.NORTH);
```

→ For add subpanels in the main/center panel so create first layout for this panel and then add like.

```
panel5.setLayout(new BorderLayout());
panel5.add(Panel to, BorderLayout.center);
```

Java Flow Layout:

Places components in a row, sized at their preferred size. If the horizontal space in the container is too small, the FlowLayout class uses the next available row.

- By default Frame use Border Layout. so we use Flow- layout: `frame.setLayout(new FlowLayout());`
- We can create button and add on frame, on same step set layout using FlowLayout.CENTER

```
frame.add(new JButton("1"));
```

- We can also set spacing between the buttons.

```
frame.setLayout(new FlowLayout(FlowLayout.CENTER, 0, 0));
```

(0 → horizontal, 0 → vertical)

- We can create panel and add all the buttons to the panel. By default panel use FlowLayout.
- After this add ~~button to panel~~. panel to frame.

J Java Grid Layout:

Place components in a grid of cells.

- Each component takes all the available space within the cell, and each cell is the same size.
- Create buttons and andadd to frame, it can cover all the space in the button.
- If we set the layout of frame to Grid Layout it can distribute all the button equally.

```
frame.setlayout(new GridLayout(1);
```

- If we set the buttons in the form of rows and columns so pass argument inside the Grid Layout (rows, colomns).

```
frame.setlayout(new GridLayout(3,3));
```

- We can also add horizontal and vertical line spaces between the components, and ~~add~~ buttons pass after rows and columns.

```
GridLayout(rows, columns, horizontal space, vertical);
```

Java LayeredPane:

Swing container that provides a third dimension for positioning components.

e.g:- depth, z-index

- Create Panel and set this Layout to null.
- Create instance/object of Layered Pane.

```
JLayeredPane J1 = new JLayeredPane();
```

- Set the size of the Layered Pane to same as frame.

```
J1.setBounds(0,0,500,500);
```

- Add Layered Pane to the frame.

```
frame.add(J1);
```

- Create Labels and set Bounds the Labels according to our need and then add to the ~~frame~~ LayeredPane.
- By default first component appears on the top in the LayeredPane.
- Default → Palatte → Modal → PopUp → Drag
- we can set the Layers using this name. Default is the lower one and Drag is the top one.

  like → ~~layer~~ J1.add(label1, JLayeredPane.DRAG_Layer);
- We can also do the same things using the Integer numbers. Like.

  J1.add(label1, Integer.valueof(0));

Java New GUI window:

- Create two classes other than main method class. one for one window and other for other window.
- Create Instance of first window class in the main.
- First window class create constructor and before the constructor create Instance of JFrame and JButton.
- Inside the constructor set the frame size and button.
- Create the Action Listener of the button.
  Inside the Action Listener create Instance of another window class.

```
if(e.getSource()==myButton){
    NewClass new1 = new NewClass();
}
```

→ Before creating Instance just dispose the previous window.

```
frame.dispose();
NewClass new1 =new Newclass();
```

JOptionPane:

Pop up a standard dialog box that prompts users for a value or informs them of something.

- To create simple dialog box use

  JoptionPane.showMessageDialog(null, "message", "Title", messag));
  ↙ type"

  JoptionPane.——

- According to the type of message they can show the Icon to you.
- To show the yes/No/cancel option to the dialog box use

  JOptionPane.showConfirmDialog(same , OptionType.)
  JoptionPane.——

  → if we perform action then write inside the System.out.println( );

yes	return 0
NO	" 1
cancel	" 2
X	return -1

- Also store in the some Integer value like int answer = ——— ;
- some input Dialog box are also have some uses like

  JOptionPane.showInputDialog(Arguments);
- sum up all the optionPane in one and write like this

  JOptionPane.showOptionDialog ( );
- Can Also create your own option and Icon in the dialog box so just pass the name.

  → For Icon create Instance and pass the name of Instance.

  → For options create array and pass the name of Array.

  like String[] response = {"Hi", "Hello", "No"};

Java Text Field:

A GUI textbox component that can be used to add, set or get text.

e.g:- Enter UserName and password.

- Before adding text field create new class and frame, action Listener inside the class.
- Create the instance of this new class in the main class.
- Create instance of textfield and add to the frame.

```
JTextfield field = new JTextfield();
field.setPreferredSize(new Dimension(250,70));
```

- field.gettext(); → method to store the text of textfield.
- We can also change the text color and text.
- We can also change the blink line color (I) like

```
field.setCaretColor(Color.white);
```

- We can also show text inside the text-field like.

```
field.setText("_____");
```

Java Check Box:

A GUI component that can be selected or deselected.

- Create object and add to the frame.

```
JCheckBox checkbox = new JCheckBox();
```

- To check the checkbox is selected or not used

```
checkbox.isSelected(); → True / False
```

- We can also add icons to the place of checkbox, just add icon to the check box.

Java Radio Button :

One or more buttons in a grouping in which only 1 may be selected per group

- Set Layout of frame to flowLayout.
- Create object of the Jradio button like.

```
JRadioButton pizza = new JRadioButton();
```

- Create many buttons and set the text of the button

```
pizza.setText("_____");
```

- Add these button to frame.

```
frame.add(pizza);
```

- Create Buttongroup object and add to the object for limit of select only one button.

```
ButtonGroup group = new ButtonGroup();
group.add(pizza);
```

- set the all buttons to the actionlistener and perform operation accordingly.
- Create ActionLister of all the button like.

```
pizza.addActionListener(this);
```

- We can also add ImageIcon of the radio buttons, create the object of the ImageIcon of each button and then add icon to the button like.

```
pizza.setIcon(obj name);
```

Java ComboBox:

A component that combines a button or editable field and a drop-down list.

- setLayout to Flow Layout.
- Create object of the Jcombox Box.

```
JComboBox Box = new JComboBox();
```

- We can pass some array or may be wrapper class inside the constructor of combo Box, to select options.

  like: string[] animals = {_____}; // use wrapper classes for primitive data type.
  JComboBox(animals);

- Add the comboBox to the frame like.

  frame.add(comboBox);

- we can know how ~~the~~ the use select from the combobox using the operation inside the ActionListener.

  combobox.getselected Item();
  comboBox.getselected Index();

- ~~We ca~~ We can perform many operations on combo Box some are:

  combox.setEditable(true); → to edit
  combox.getItemcount(); → to count no. of items
  combox.addItem(); → Add items in combo Box.
  combox.insertItemAt("pig", Index); Add on specific index.
  combox.setselected Index(0); Default selected.
  combox.removeItem("cat"); remove from combo Box.
  combox.removeItemAt(0); remove from index.
  combox.removeAllItem(); → remove all items from combo Box.

Java Slider:

GUI component that lets user enter a value by using an adjustable sliding knob on a track.

- Implement changeListener to the class and implement the method.
- Create instance of frame, Panel, Label and JSlider like.

  JSlider slider = new JSlider();

  pass the value of min, max, starting

- add slider to panel and panel to frame.

J • Set the size of the slider like
slider.setPreferredSize(new Dimension(400,200));

• To visible range more on slider set PaintTricks True.
slider.setPaintTicks(true);

• C
C. To set the range on the slide use
slider.setMinorTickSpacing(10);
slider.setMajorTickSpacing(20);

•
• We can add values to the majorticks on the slider.
slider.setPaintLabels(true);

•
• To move the slide vertical because by default set horizontal
slider.setOrientation(SwingConstants.VERTICAL);

• To show the selected value of slider use
label.setText("0 C" + slider.getValue());
add changeListener and write this code inside the method.
slider.addChangeListener(this);

Java Progress Bar:

Visual aid to let the user know that an operation is processing.

• Create instance of the progress Bar.
JProgressBar bar = new JProgressBar();

• To set the size of the ~~frame~~ progress Bar
bar.setBounds(0,0,500,20);

• To set value and the percentage of progress bar use
bar.setValue(50);
bar.setStringPainted(true); → %

• To run the progress bar many methods but create function, call the function and use while loop inside the function.
like pull();

```
Public void pull() {
    int count = 0;
    While (count ≤ 100) {
        bar.setvalue(count);
    try { Thread.sleep (1000); // pause after second.
    } catch (Interrupted Exception e) {
        e.printstackTrace();
    }
    count += 10;    // increase by 10
    }
    bar.setString("---");
}
```

- To show the message after progress will be done use

Java Menu Bar:

- Create Action Listener of the frame.
- Create object of the menu Bare and then add to frame.

```
JMenuBar menu = new JMenuBar();
frame.setMenuBar (menu);
```

- Create the object to show on the menuBar and then add.

```
JMenu filemenu = new JMenu("_Name_");
menu.add (filemenu);
```

- We can create the submenu of File menu we create instance and add to file menu.

```
JMenuItem exit = new JmenuItem ("  .");
filemenu.add(exit);
```

- We can add action Listener to each item to perform Action.
- We can highlight / Underline the first word of the MenuItem using Mnemonic method.

```
exit.setMnemonic (KeyEvent.VK_E);
```

- Add icons.
- For keyboard shortcut keys set Mnemonic of menu.

Alt + F → like.

Java Select a File:

A GUI mechanism that let's a user choose a file (helpful for opening or saving files).

- Create button and inside the action Listener of button create instance of File Chooser.

```
JFileChooser file = new JFileChooser();
```

- To open the file from the desktop create open dialog.

```
file.showOpenDialog(null); → parent component.
```

- → it can give back the integer response 0 for yes or 1 for cancel so we store in integer to perform action.

```
int x = same code;
if(x==0){   → we also write JFileChooser.APPROVE_OPTION instead of 0.
```

- Create New File and store the selected file.

```
File file = new File(file.getSelectedFile().getAbsolutePath());
S.O.P(file); → print the location.
```

- We can also set the default directory.

```
file.setCurrentDirectory(new File("."));
```

Java Color Chooser:

A GUI mechanism that let's a user to choose a color.

- Create button and Label and create Action listener of button.
- Create and modify label and add to the frame.
- inside the action Listener create instance of color chooser.

```
JColorChooser color1 = new JColorChooser();
Color color = JColorChooser.showDialog(null, "Title", initial);
                                  ↑ open a dialog box
```

- Now in the last change the selected color. label.setForeground(color);

Java KeyListener:

- Set frameLayout to null because we manually moving compon.
- implements the keyListener with the frame; and add.

```
frame.addKeyListener(this);
```

- KeyTyped:- Invoked when a key is typed. Uses KeyChar,
- KeyPressed:- Invoked when a physical key is pressed down. use keycode
- KeyReleased:- called whenever a button is released.

```
e.getKeyChar();    e.getKeyCode();
```

- Create one label and move label in four sides.

→inKeyTyped:-

```
switch(e.getKeyChar()){
case 'a': label.setLocation(label.getX()-1, label.getY());
    break;
```

→set same as it is for remaining keys.

→Pass keycode for arrows moving. and case their code.

Java Mouse Listener:

- Set layout of frame to null and implements MouseList.
- Mouse Pressed:- Invoked when a mouse button has been pressed on a component.
- Mouse Released:- Invoked when a mouse button has been released on a component.
- Mouse Clicked:- Invoked when the mouse button has been clicked(pressed and released) on a component.
- Mouse Entered:- Invoked when the mouse enter the component.

- Mouse Excited:-
  Invoked when the mouse exits a component.
- Create label and add to the frame after modification.
- Add mouse listener to the created label.
  label.addMouseListener(this);
- Add conditions on the method of mouse listener.
- setlocation relative to null and flowlayout for icons.

Java 2D graphics:

- set locationrelative of the frame to null.
  frame.setlocationRelativeTo(null);
- Create one paint function outside the constructor and then pass the ~~thro~~ graphics Argument.

```
Public void paint(Graphics g){
// Convert graphic to 2D for more options.
Graphics2D g2D = (Graphics2D) g;
```

- Draw a simple line

```
g2D.drawLine(0,0,500,500);
g2D.setStroke(new BasicStroke(5)); // thickness
g2D.setPaint(Color.Blue); // Color.
```

- g2D.draw → to draw different components.

```
g2D.drawRect(); → Create rectangle
g2D.FillRect(); → Fill the color.
```

- For polygon set the points.

```
int[] xPoints = {150,250,350};
int[] yPoints = {300,150,300};
g2D.drawPolygon(xPoints, yPoints, 3);
                                  sides.
```

- To set image as a graphics:

```
g2D.drawImage(image, 0.0, null);
```

// Create Instance:

```
Image image = new ImageIcon("Path").getImage();
```

Java 2D Animation: