Day-11

Encapsulation: Wrapping up the implementation of the data members & methods in a class.

Abstraction: Hiding the unnecessary details & only showing valuable information.

Abstraction is solving design level issue.

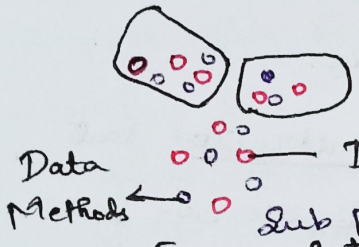
External stuff

Process Gaining information

Encapsulation is solving implementation level issue.

Internal Working

Process of containing information



Encapsulation GETTER & SETTER → Use programming pace.

Syn:

```
<Access Modifier> class <classname>{
    private <Data_Member>;
    private <Data_Methods>;
}
```

Eg:

```
public class Student {
    private String Name;
    private int Rollno;
    private int Age;
    public String getName(){        Access private variable
        return Name;
    }
    public void setName(String Name){
        this.Name = Name;           Set value to private variable
    }
    public int getRollno(){
        return Rollno;
    }
    public void setRollno(int Rollno){
        this.Rollno = Rollno;
    }
    public int getAge(){
        return Age;
    }
    public void setAge(int Age){
        this.Age = Age;
    }
}
public class StudentDetails{
    public static void main(String[] args){
        Student obj = new Student();
        obj.setName("Priya");
        obj.setRollno(29);
        obj.setAge(20);
        S.o.p("Student Name:"+obj.getName());
        S.o.p("Student Rollno:"+obj.getRollno());
        S.o.p("Student Age:"+obj.getAge());
    }
}
```

o/p: Student Name: Priya
Student Rollno: 29
Student Age: 20

Abstract Class in Java

In Abstract class is a class that cannot be initiated by itself.

Syn:
```
abstract class Shape {
    int colour;
    abstract void draw();
}
```

We can have an abstract class without abstract method

Parent Class create or declare Child class as abstract.

Eg:
```
abstract class Sun {
    abstract void point();
}
class Employee extends Sun {
    void point() {
        String name = "Priya";
        int age = 20;
        float salary = 252.5F;
        S.o.p("Name:" + name);
        S.o.p("Age:" + age);
        S.o.p("Salary:" + salary);
    }
}
class Base {
    public static void main(String [] args) {
        Sun s = new Employee();
        s.point();
    }
}
```

Op:
```
Name : Priya
Age : 20
Salary : 252.5
```

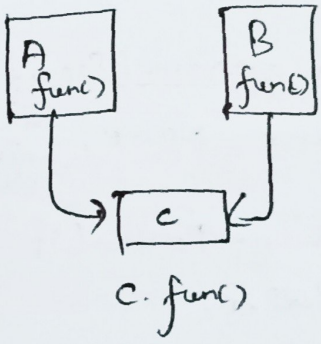
Abstract class where methods can be overridden.

Final behind the class it prevent inheritance.

We create constructor but not the object, we can use super keyword to ref variable

We can call void fun
```
static void Hello() {
    S.o.p("Hey");
}
```

Interface contains abstract class

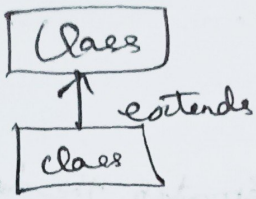


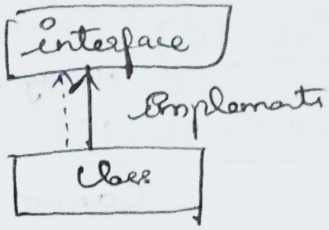
Interfaces in Java:

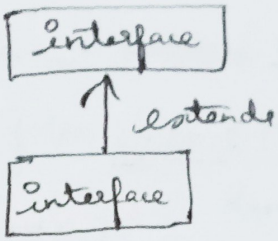
Used to achieve abstraction & multiple inheritance in java.

Syn:
```
Interface {
    // declare methods that abstract
}
```

Eg:
```
interface Player {
    final int id = 10;
    int move();
}
```







Eg:
```
import java.io.*;
interface In1 {                          → Interface
    final int a=10;                      public static final
    void display();                      Public abstract
}                                        Implement interface
class TestClass implements In1{
    public void display(){
        S.o.p("Hi");
    }                                    Driver Code
    public static void main(String[] args){   ← Similar to this we can extends the (interface) class.
        TestClass t = new TestClass();
        t.display();
        S.o.p(t.a);
    }
}
```

Op:
```
Hi
10
```

Interface: Service details
Abs class: Partial implementation
Imp class: Contain all "
Final / Main: Access all interface data

Multiple inheritance