(26) Polymorphism and abstract keyword in Java

↳ [redacted]
↳ Hyder abbas sir
↳ 15 Nov live class

Agenda :-

(1) Polymorphism
(2) Abstraction
    ↳ Abstract Keyword.
(3) [redacted]
(3) Has-A relationship
(4) Interface
(5) functional interface
(6) Lamba Expression.

(1) Polymorphism :- is a Concept by which we can Perform a single action in different ways.

→ two types of Polymorphism
  (i) Compile time polymorphism
  (ii) runtime polymorphism.

(i) Compile time polymorphism :-

↳ Compile time polymorphism is also known as static polymorphism (or) Early binding.

↳ Compile time polymorphism is a polymorphism that is resolved during the Compilation Process.

↳ if you overload a static Method. in Java. it is example of Compile time polymorphism..

→ Compile time polymorphism

reffered w.r.t to → Method overloading

↳ Compiler will resolve during Compile time

↳ Compiler Consider few things

no. of Parameters, data type of parameters, order of data types of parameters.

↳ by Considering this it resolve the issues.

## (ii) Run time polymorphism (or true polymorphism)

→ Run time polymorphism (or Dynamic Method Dispatch) is a process in which a call to an overridden Method is resolved at runtime rather than Compile-time.

→ it is reffered w.r.t to Method overriding.

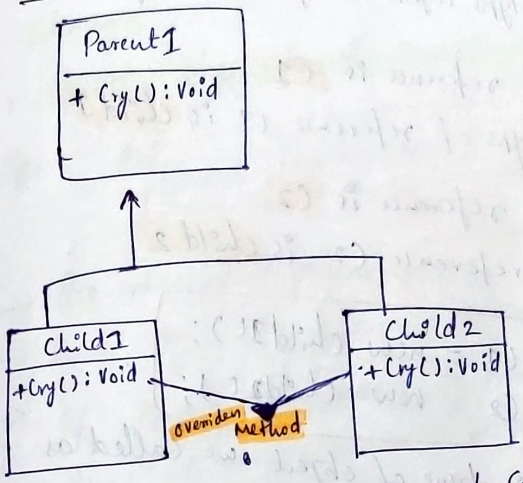
Advantage :- ① flexibility

② Code size reduces.

→ one thing is doing Many tasks

(1M)

## ① Tight Coupling Program




→ Cry() Method is child1 and child2 is overidden Method.

Code :-

```
Class Parent {
    Public void Cry() {
        S.o.p("Parent Crying");
    }
}
class Child1 Extends Parent {
    Public Void Cry() {
        S.o.p("Child1 Cries at low Voice");
    }
}
Class Child2 Extends Parent {
    Public Void Cry() {
        S.o.p("Child2 Cries at high Voice");
    }
}
Public class Poly {
    Public static Void Main(String[] args) {
        Child1 C1 = new Child1();
        Child2 C2 = new Child2();
        C1.Cry();
        C2.Cry();
    }
}
```

① Tight Coupling :- refer above Code

Creating child type reference for child type object.

→ for child1 reference is c1
type of reference c1 is child1

→ for child2 reference is c2
type of reference c2 is child2

```
Child1 c1 = new Child1();
Child2 c2 = new Child2();
```

↳ Creating this type of object we called as tight Coupling.

c1.Cry(); → it calls Cry() Method of child1

c2.Cry(); → it calls Cry() Method of Child2

↳ it is non-polymorphic Version.

→ Whenever we create a object the reference whose going to refer the object. the reference type Must be same as that of the object.

Example

→ for child1 reference is c1

→ type of reference it should be same that of the object.

→ type of reference c1 is child1 in above example.

② Lose Coupling :-

→ In one case the type of reference can be change.

→ if the type is of Parent type.

Eg: P.

```
Parent ineuron = new Child1();
```

↳ Variable name can be anything.

→ By we achieve polymorphism.

→ here type of reference is parent

→ if Parent class name is demo then type of reference will be demo.

→ Code :-

```
class Parent {

}
class child1 extends Parent {

}
Class child2 Extends Parent
{

}
Public class Poly {
    Public s v M (String[] args) {
        Parent c1 = new child1();
        Parent c2 = new child2();
    }
}
```

① Whenever we Create any object reference type Must same as that of the object.

② In one Case the reference can be different. reference type must be parent type.

↳ we achieve polymorphism.

Code :-

```
class Parent {
    Public Void disp() {
        S.o.p("Parent 1");
    }
}
class child1 Extends Parent
{
    Public Void disp() {
        S.o.p("child 1");
    }
}
class child2 Extends Parent
{
    Public Void disp() {
        S.o.p("child2");
    }
}
Public class poly {
    Public static Void Main(String[] args){
        Child c1 = new child c1();
        Child c2 = new child c2();
        Parent ref;
        ref = c1;
        ref.disp();
        ref = c2;
        ref.disp();
    }
}
```

① Loose Coupling :-

→ first rule (or) principle to achieve polymorphism is whatever object is created. type of object Must be of Parent type.

→ this only called as loose Coupling.

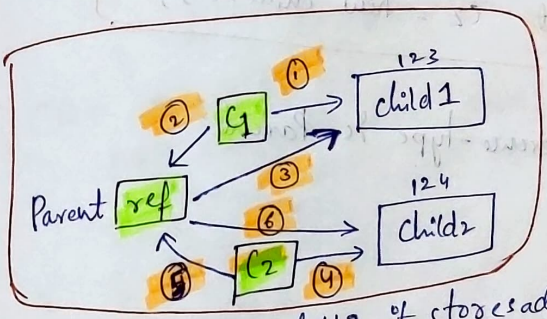
```
child1 c1 = new child1();
child2 c2 = new child2();

Parent ref;

ref = c1;
ref.disp();  → 1:M → one line doing Multiple tasks.

ref = c2;
ref.disp();
```

↳ refer previous Code




→ ① c1 refers to child1. it stores address of child1
② ref variable stores the value of c1
③ ref variable refer to child1
④ c2 refers to child2. it stores address of child2
⑤ ref variable store the value of c2.
⑥ ref variable now refers to c2.

ⓐ Tight Coupling :- Creating child type reference for child type object.

Eg:-
```
Child1 c1 = new child1();
Child2 c2 = new child2();
```

→ In this case we cannot achieve polymorphism.

ⓑ Loose Coupling :- Creating parent type reference for child type object.

→ In this we achieve polymorphism. (1:M)

Eg:-
```
Parent c1 = new child1();
Parent c2 = new child2();
```

→ here reference type is Parent.

Example ③ Polymorphism

```
class plane {
    Public Void takeoff() {
        S.o.p("plane is taking off");
    }
    Public Void fly() {
        S.o.p("plane is flying");
    }
    Public Void landing() {
        S.o.p("plane is landing");
    }
}

Class Passengerplane Extends plane {
    Public Void fly() {
        S.o.p("Passenger plane flies at Medium height");
    }
}

Class Cargoplane Extends plane {
    Public Void fly() {
        S.o.p("Cargo plane flies at high height");
    }
}

Class Airport {
    Public Void Permit(plane plane) {
        plane.takeoff();
        plane.fly();
        plane.landing();
    }
}

Public class poly {
    Public static Void main(String[] args) {
        Passengerplane PP = new Passengerplane();
        Cargoplane CP = new Cargoplane();
        Airport a = new Airport();
        a.Permit(CP);
        a.Permit(PP);
    }
}
```

## Code Explanation

```
Class Airport {
    Public Void Permit(Plane plane)
    {
        plane.takeoff();
        plane.fly();
        plane.landing();
    }
}
Public class poly4 {
    Public static Void main(String[] args)
    {
        Passengerplane PP = new Passengerplane();
        Cargoplane CP = new Cargoplane();
        Airport a = new Airport()
        a.Permit(PP)
        a.Permit(CP)
    }
}
```

plane = CP, PP

1, 2, 3, 4

→ ① PP is given to plane (plane = PP).
② it invokes all Methods in classes
③ CP is given to plane (plane = CP)
④ it invokes all Methods in classes
↳ takeoff()
↳ fly()
↳ landing()

→ here one line doing Multiple tasks.
→ object Creation happens at run time.
→ plane.takeoff();
plane.fly();
plane.landing();
} invoking is happening at run time
↳ runtime polymorphism

→ plane is parent of all three classes.

Polymorphism :- Creating parent type reference for child object
↳ lose coupling.
↳ we can acheive polymorphism

→ In above code by using Parent type reference you can call child class specialized Methods.

→ fly() → Specialized Method
takeoff(), land() → Inherited Methods.

## Poly Morphism Example ④

↳ Code flexibility.

```
Class Parent {
    Public Void Cry() {
        S.o.p("Parent Crying");
    }
}
Class Child1 Extends Parent {
    Public Void Cry(){
        S.o.p("Child1 Crying");
    }
    Public Void Eat(){
        S.o.p("Child1 Eats less");
    }
}
Class Child2 Extends Parent {
    Public Void Cry(){
        S.o.p("Child2 Crying");
    }
    Public Void Eat(){
        S.o.p("Child2 Eats More");
    }
}
Public class Poly {
    Public static Void main(String[] args) {
        P1.Cry(); → ① // overriden Method
        // P1.Eat(); → ②
        ((Child)P1).Eat(); → ③ // Specialized Method
    }
}
```

## Code Explanation

At ① :- `P1.Cry();` // Overriden Method.

→ overridden Method can be called directly by Parent type reference.

At ② :- `P1.Eat();` // Specialized Method

↳ by using Parent type reference we cannot call Specialized Methods.

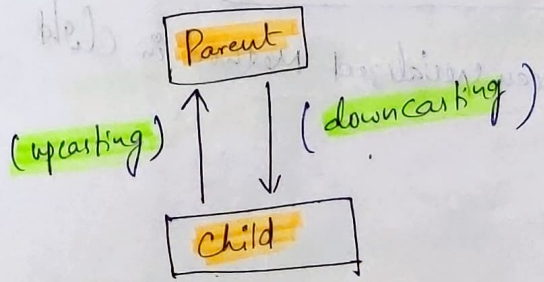
At ③ :- down casting

In order to call specialized Method by using Parent type reference we have to downcast.

```
((Child)P1).Eat();
```

① Upcasting :- typecast a child object to Parent object.

② downcasting :- typecast a Parent object to Parent object.



① upcasting (or) loose coupling

→ type casting a child object to Parent object

→ for child type object creating Parent type reference

⇒ Child promoted to parent (upcasting).

Eg:

```
Parent c1 = new child1();
```

→ to achieve polymorphism we do upcasting

② downcasting :-

to call specialized Method we do downcast.

```
Parent ref = new child();
```

→ ref is parent type

↳ it is down cast to child type

```
((child) ref).eat();
```

→ to accen specialized Method in child class.

Eg: class Demo Extends Servlet

{

≡

// specialized Method → to accen this we have to down cast

}

```
Servelet ref = new Demo();
((Demo) ref).specializedMethod;
```

↳ written in Code.

↳ downcasting

important Points :-

① To achieve polymorphism we do upcasting.

② To call specialized Method in child class we do Downcasting in Java.

③ Creating child type reference for child type object is tight coupling

```
child c = new child()
```

④ Creating Parent type reference for child type object

```
Parent c = new child()
```

→ to achieve polymorphism.

⑦ Abstraction :- it is a process of hiding the implementation details and showing only functionality to the user.

→ it shows only Essential things to the user and hides the internal details.

→ Abstraction :-

↳ it is acheived by abstract keyword.
→ you may achieve 100% (or) not achieve.

↳ it is acheived by interfaces
→ 100% abstraction is acheived by interface.
→ there is possibility of acheiving 100% abstraction.

→ ~~Abstraction - lack high~~

→ Abstraction :- hiding actual implementation only features are shown.

→ In Java we can have Methods without the implementation (or) defination (or) body

Eg: Public Void fly();

→ However those Methods must be declare abstract.

⑦ Abstract keyword used to Method.

⑦.① Abstract Methods :-

① Abstract Methods are such Methods which has only Method Signature but body and implementation is not there.

Eg: Abstract Public Void fly();
↳ no body for this Method
↳ no Implementation inside the Method.

② In one class if one Method is also abstract then must and should class also declared as abstract.

Eg:
```
abstract class plane {
    abstract public Void takeoff();
    abstract public Void fly();
    abstract public Void landing();
}
```
→ Abstract Methods.

→ it is 100% of Abraction
→ there is no implementation.

④ ~~Abstract Abstract class cant abstract~~

⑤ Abstract class can have Abstract Method and also normal Concrete Method.

④ Abstract class can have normal Concrete and Abstract Method in it.

Eg:-

```
Abstract class plane
{
    abstract Public Void takeoff();
    abstract public Void fly();
    Public Void airport()
    {
        S.o.p("All planes need airport");
    }
}
```

abstract Methods

normal Method

⑤ Abstract Keyword can be used for class and Method.

⑥ Abstract keyword cannot be applied to Variable.

→ Variable Cannot be abstract

Eg:-

```
abstract class plane{
    abstract int a;
}
```

↳ we get Error. Variable Cannot be abstract

Code:-

```
abstract class Plane {
    abstract public Void takeoff();
    abstract public Void fly();
    abstract public Void landing();
    abstract public Void landing();
    Public Void airport()
    {
        S.o.p("All planes need airport");
    }
}
Class PassengerPlane extends Plane {
    Public Void take off() {
        S.o.p("Passengerplane need medium runway to takeoff")
    }
    Public Void fly() {
        S.op("it flies at Medium level");
    }
}
Class Cargoplane extends plane {
    Public Void fly() {
        S.o.p("it flies at low level");
    }
}
Public class Abstract {
    Public static V M(String() args){
        Passengerplane PP = new Passengerplane();
        Cargoplane CP = new Cargoplane();
    }
}
```

① Whenever we Extend abstract class it says

① Either implement the body of abstract Method in abstract class.

(or)

② declare the class as abstract.

Eg:- ① implementing body of abstract Method.

```
Abstract class Loan1 {
    abstract Public Void dispInt();
    Public Void welcomenote() {
    {
      s.o.p("welcome");
    }
}

class HomeLoan1 Extends Loan1
{
    Public Void dispInt()
    {
      s.o.p("RI is 12%");
    }
}
```

body of abtract method is implemented.

② declare the class as abstract

```
Abstract class Loan1 {
abstract Public Void dispInt();
}
abstract class HomeLoan1 Extends
{

}
```

③ We can create reference for abstract class

```
Loan1 loan = new HomeLoan1();
loan.dispInt();
```

④ We Cannot Create object of abstract class.

```
Loan1 loan = new Loan1();
```

⑤ Abstraction should not affect polymorphism.

⇒ refer Code in next page.

⑤ We can have abstract class without abstract Methods.

→ We Cannot Create object sbl.

Code : Abstract 2

```
abstract class Loan1 {
    abstract Public Void dispInt();
    Public Void WelcomeNote() {
        s.o.p("Welcome");
    }
}
Class HomeLoan1 Extends Loan1
{
    Public Void dispInt() {
        s.o.p("RoI is 10%");
    }
}
Class EducationLoan1 Extends Loan1
{
    Public Void dispInt() {
        s.o.p("RoI is 15%");
    }
}
Public class Abstract1 {
    Public static Void main(String[] args) {
        Loan1 loan = new HomeLoan1();
        loan.dispInt();
        loan.WelcomeNote();
        Loan2 loan = new EducationLoan1();
        loan.dispInt();
        loan.WelcomeNote();
    }
}
```

Important Keynote on Abstraction :-

① abstract Keyword used for
   - (a) Variable X
   - (b) Method ✓
   - (c) class ✓

② we cannot create abstract class object.

③ abstract class can have all ~~methods~~ abstract Methods
   ↳ it is 100% abstraction.

④ abstract class can have both abstract & Concrete Methods.

⑤ abstract class can have all Methods Concrete.

⑥ ~~abstract if one abs~~

⑥ In one class if one Method is abs abstract then must and should class also declared as abstract.

⑦ Declare the class as abstract then no one create object of that class.

⑧ The Sub class / child class if extending abstract class then Either have to implement (or) declare class abstract.

⑨ Constructor Cannot be Made abstract.
   ↳ it has body by default.

(10) abstract class is incomplete class.
↳ child class will implement (inheritance)
↳ overriding Method in child class.

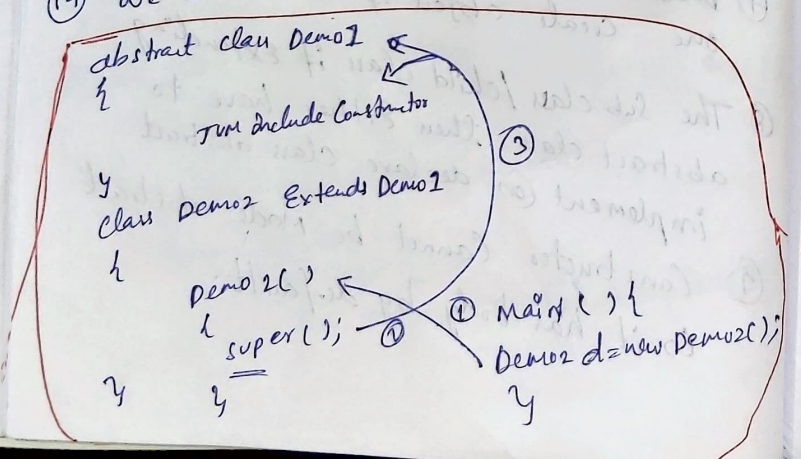
(11) We cannot make abstract class as final.
↳ final and abstract is illegal.

→ if we use final then inheritance of Abstract class not possible.
→ we cannot override in child class.

(12) final Method will participate in inheritance but cannot overridden.

(13) Variable cannot be abstract.

(14) we can have Constructor in abstract class.




→ when Constructor is called in main by creating object then JVM create a Constructor in class with zero parameters.

→ super() Method present in that Constructor again calls parent class Constructor.

→ it is evident that we can have Constructor in abstract class.

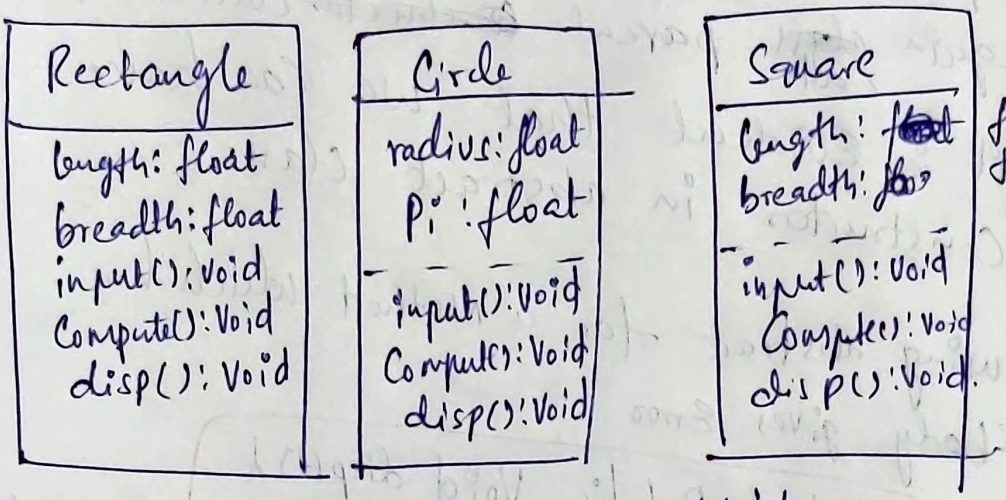
(15) using abstract for a Method which has body gives Error.

Eg:
```
abstract Public Void disp()
{
    S.O.P ("Hello");
}
```
→ we get Error.

(16) Abstraction and polymorphism depends on inheritance

Small Project :- To Calculate area of Rectangle, Circle,

planning - UML




↳ refer abstract 3 and 4 in github.