

26 Polymorphism and abstract keyword in Java

Agenda :-

- ① Polymorphism
- ② Abstraction
 - ↳ Abstract Keyword.
- ③ ~~Encapsulation~~
- ④ Has - A relationship
- ⑤ Interface
- ⑥ functional interface
- ⑦ Lambda Expression.

↳ ~~Hyderabbasfir~~

↳ 15 Nov live class

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..



GADDALA ASHISH



LinkedIn

→ Compile time polymorphism

referred 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 resolves 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 referred w.r.t to Method overriding.

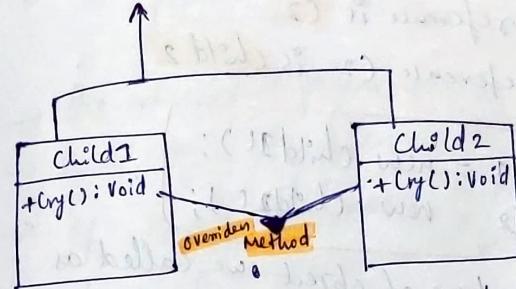
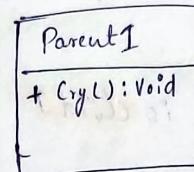
Advantages - ① flexibility

② Code size reduces.

→ One thing is doing many tasks

(IM)

④ Tight Coupling Program



→ cry() Method is child 2 and child 2 is overridden 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();
  
```



GADDALA ASHISH

in LinkedIn

@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.

↳ if 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 incuron = 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 void M (String [] args) {
        Parent C1 = new Child1();
        Parent C2 = new Child2();
    }
}
```



GADDALA ASHISH
in LinkedIn

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() {
        System.out.println("Parent");
    }
}

class child1 extends Parent {
    public void disp() {
        System.out.println("child1");
    }
}

class child2 extends Parent {
    public void disp() {
        System.out.println("child2");
    }
}

public class Poly {
    public static void main (String args) {
        Child c1 = new child1();
        Child c2 = new child2();
        Parent ref;
        ref = c1;
        ref.disp();
        ref = c2;
        ref.disp();
    }
}
```

① Loose Coupling :-

→ first rule (or) principle to achieve polymorphism is whenever 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;

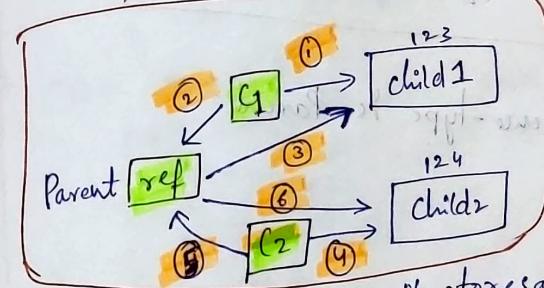
ref = c1;

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 stores the value of c2.
⑥ ref variable now refers to c2.

t (a) 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.

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

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

Eg:

```
Parent d1 = new Child1();
Parent d2 = new Child2();
```

→ here reference type is Parent.

Example ③ Polymorphism

class Plane {

```
public void takeoff() {
    System.out.println("plane is taking off");}
```

```
public void fly() {
    System.out.println("plane is flying");}
```

```
public void landing() {
    System.out.println("plane is landing");}
```

class PassengerPlane Extends Plane {

```
public void fly() {
```

```
System.out.println("Passenger plane flies at medium height");}
```

class CargoPlane Extends Plane {

```
public void fly() {
```

```
System.out.println("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();
```

~~```
OrganicPlane op = new OrganicPlane();
```~~

```
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 poly {

        Public static Void main(String[] args)

        {

            Passengerplane PP = new Passengerplane();

            CargoPlane CP = new CargoPlane();

~~permitted~~

            Airport a = new Airport()

            a. Permit(PP)

            a. Permit(CP)

plane = CP, PP

→ here one line doing multiple tasks.  
→ object creation happens at run time.

→ plane. takeoff();  
    plane. fly();  
    plane. landing();

    involving is happening at  
    run time

    ② ~~poly~~  
    ↳ genuine polymorphism

→ plane is parent of all three classes.

Polymorphism :- Creating parent type reference  
for child object

↳ loose coupling.

↳ we can achieve polymorphism

→ In above code by using ~~permitted~~  
parent type reference you can call  
child class specialized methods.

→ fly() → specialized Method

→ land(), takeoff() → inherited Methods.

→ ① PP is given to plane (plane = PP).  
    ② it involves all methods in classes

③ CP is given to plane (plane = CP)  
    it involves all methods in classes

    ↳ takeoff()  
    ↳ fly()  
    ↳ landing()



GADDALA ASHISH  
LinkedIn

## 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 // overridden Method

// P1.Eat(); → ②

((Child)P1).Eat(); → ③ // Specialized Method

## Code Explanation

But

① At ① :- P1.Cry(); // Overridden 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.

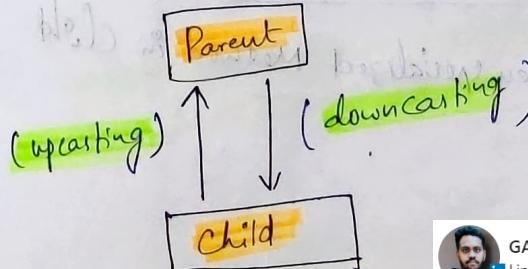
At ③ :- downcasting

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 Child object.



GADDALA ASHISH  
LinkedIn

## ① Upcasting (or) Box 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 downcast to child type

`((Child) ref). Eat();`

→ To access specialized Method in Child class.

Eg:- Class Demo Extends Servlet

{

    }     ≡  
    }     // Specialized Method → To access this  
    }     we have to  
    }     downcast

Servlet ref = new Demo();

~~((Demo) ref)~~. Specialized Method;

~~Written in code.~~

↓ Downcasting

~~Overecast~~  
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 ~~child~~ 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 achieved by abstract keyword.

→ you may achieve 100% or not achieve.

↳ it is achieved by interface.

→ 100% abstraction is achieved by interface.

→ there is possibility of achieving 100% abstraction.

→ ~~abstraction~~ ~~high~~

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

→ In Java we can have Methods without the implementation or definition or body.

Ex:- Public void fly();

→ However those Methods must be declared abstract.

⑧ Abstract keyword used to Method:

⑨ Abstract Methods :-

Abstract methods are such methods which has only method signature but body and implementation is not there.

Ex:- Abstract public void fly();

↳ no body for this method  
↳ no implementation inside the method.

⑩ In one class if one method is also abstract then must and should class also declared as abstract.

Ex:-

~~abstract~~ abstract class plane h

abstract public void takeoff();

abstract public void fly();

abstract public void landing();

→ Abstract Methods.

→ it is 100% of Abstraction

→ there is no implementation.

⑪ ~~Abstract~~ ~~Abstract class~~ ~~can't~~ ~~have~~ ~~Abstract~~ ~~Method~~

⑫ Abstract class ~~can~~ can have Abstract Concrete Method and also normal Method.



GADDALA ASHISH  
in LinkedIn

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

Eg:- Abstract class plane

```
abstract class plane {
 abstract public void takeoff();
 abstract public void fly();
 public void airport()
 {
 System.out.println("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.

→ Variables cannot be abstract

Eg:- ~~abstract~~ abstract class plane {

abstract int a;

}

we get Error.

Variables 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()
 {
 System.out.println("All planes need airport");
 }
}
```

Class PassengerPlane extends Plane {

public void takeoff()  
 {  
 System.out.println("PassengerPlane need medium runway to takeoff");  
 }  
}

public void fly()  
{  
 System.out.println("it flies at Medium level");  
}

Class CargoPlane extends Plane {

public void fly()  
 {  
 System.out.println("it flies at low level");  
 }  
}

public class AbstractL

public static void main(String[] args){

PassengerPlane pp=new PassengerPlane();

CargoPlane cp=new CargoPlane();



GADDALA ASHISH  
in LinkedIn

- ① whenever we extend abstract class it says
- ② either implement the body of abstract method in abstract class.

~~(or)~~

(or)

- ③ declare the class as abstract.

- ④ implementing body of abstract Method.

~~Abstract Class loan1~~

Eg:- Abstract class loan1 h

abstract Public Void dispInt();

Public Void welcomeNotl();

{  
    s.o.p("welcome");

}

Class HomeLoan1 Extends Loan1

{

    Public Void dispInt();

        body of  
        abstract  
        method is  
        implemented.

{  
    s.o.p("RD is 12%");

}

- ⑤ declare the class as abstract.

Abstract class Loan1 l

abstract Public Void dispInt();

}

Abstract class HomeLoan1 Extends

{

}

- ⑥ We can create reference for abstract class

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

- ⑦ We Cannot Create object of abstract class

Loan1 loan = new Loan1();

- ⑧ Abstraction should not affect polymorphism.

→ refer code in next page.

- ⑨ We can have abstract class without abstract Methods.

→ We Cannot Create object st1.



GADDALA ASHISH



LinkedIn

## Code : Abstract 2

abstract class Loan1 {

    abstract public void dispInt();

    public void welcomeNote() {

        System.out.println("Welcome");

}

Class HomeLoan1 Extends Loan1

    public void dispInt() {

        System.out.println("RBI is 10%");

}

Class EducationLoan1 Extends Loan1

    public void dispInt() {

        System.out.println("RBI is 15%");

}

Public class Abstract1 {

    public static void main(String[] args) {

        Loan1 loan = new HomeLoan1();

        loan.dispInt();

        loan.welcomeNote();

        Loan1 loan = new EducationLoan1();

        loan.dispInt();

        loan.welcomeNote();

## Important Keynote on Abstraction :-

① abstract Keyword used for

    ② Variable X ✓

    ③ Method ✓

    ④ Class ✓

⑤ we cannot create abstract class object.

⑥ abstract class can have all abstract Methods

↳ it is 100% abstraction.

⑦ abstract class can have both abstract & concrete methods.

⑧ abstract class can have all methods concrete.

⑨ ~~abstract if one abs~~

⑩ In one class if one method is also abstract then must and should class also declared as abstract.

⑪ Declare the class as abstract then no one create object of that class.

⑫ The sub class / child class if extending abstract class then either have to implement or declare class abstract.

⑬ Constructor cannot be made abstract.

↳ it has body by default.



GADDALA ASHISH  
in LinkedIn

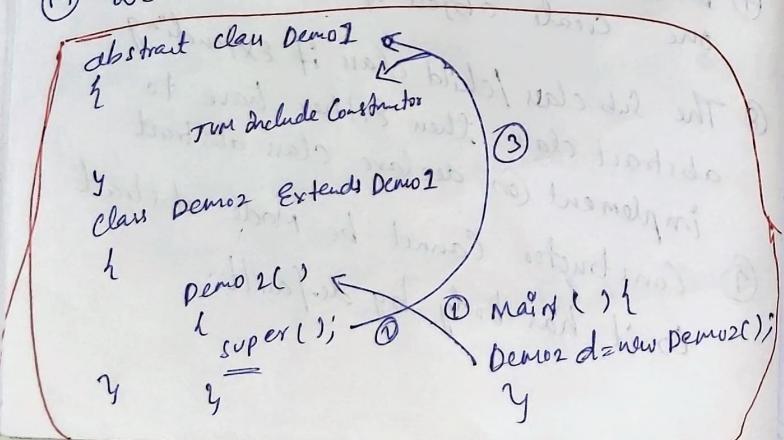
- (10) abstract classes incomplete class.
    - ↳ child class will implement
  - (11) abstract class is incomplete class.
    - ↳ child class will implement (inheritance)
    - ↳ ~~overriding~~ Method in child class.
  - (12) We Cannot make abstract class as final.
    - ↳ final and abstract is illegal.

→ if we use final then inheritance of Abstract class not possible.

→ we cannot override in child class.

~~final~~

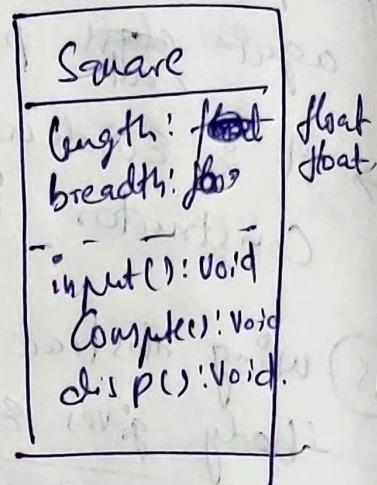
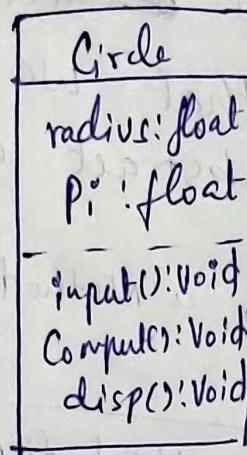
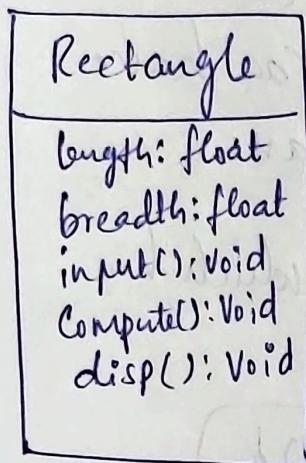
(13) final Method will participate in inheritance but Cannot overridden.
  - (14) Variable, Cannot be abstract.
  - (15) we can have constructor in abstract class.
- when Constructor is called in main by creating object then JVM Create a constructor in class with zero parameters.
- Super() method present in that constructor again ~~calls~~ parent ~~constructor~~ class constructor.
- it is evident that we can have constructor in abstract class.
- using abstract for a method which has body gives error.
- E.g. `abstract public void disp() {  
 System.out.println("Hello");  
}` → we get Error.
- (16) Abstraction and polymorphism depends on inheritance.



GADDALA ASHISH  
Linkedin

Small project :- To calculate area of Rectangle, Circle,

planning - UML



To refer abstract 3 and 4 in github.



GADDALA ASHISH

