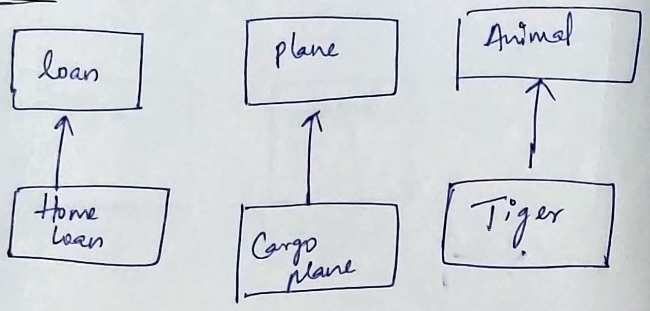
(25) Inheritance, (inherited vs Specilised Methods) Super, final Keyword in Java

↳ Hyder Abbasi
↳ 14th Nov live class

① Inheritance Example



→ loan → Personal loan, Education loan, Car loan, Home loan } Types of loans

→ Common data regarding loans are written in loan class.

→ this Common data inherited in all types of loans.

② Access Specifiers in Java :-

→ the access modifiers in Java Specifies the accessibility (or) Scope of a field, Method, Constructor, (or) class.

→ we can change the access level of fields, Constructors, Methods, and class by applying the access Modifier on it.

→ there are four types of Java access Modifiers

① Private :- the access level of a private Modifier is only within the class. it cannot be accessed from outside the class.

② Default :- the access level of and default Modifier is only within the package.
→ it cannot be accessed from outside the Package.
→ if you do not Specify any access level, it will be the default.

③ Protected :- the access level of a protected Modifier is within the package and outside through child class.

→ if you do not Make child class, it cannot be accessed from outside the package.

④ Public - the access level of a public Modifier is Everywhere. it can be accessed from within the class, outside the class, with in the package and outside the Package.

⇒ Access specifiers :- 4 type of access specifiers

① Public
② Protected
③ Default
④ Private

} Access specifiers applied to class, Method, Constructor, Variable.

Example :- Default Modifier

```
Void disp()
{
}
```

→ if we not specify any access level it will be the default.

② Public Void disp()

{ → Public access specifier.

}

③ Private int age;

↳ Restrict access within class

⇒ Project → Multiple functionality

folder { ↳ addition → Multiple classes
↳ subtraction → Multiple classes and Methods
↳ Multiplication → }

→ these folders we call it as package.

→ Java project

↳ combination of Packages.

↳ Modularity

③ Understanding Java Access Modifiers

	Within a class	Outside class within Package	Outside Package (is-A relationship)	outside Package no is-A relationship
Public	✓	✓	✓	✓
Protected	✓	✓	✓	X
default	✓	✓	X	X
Private	✓	X	X	X

↓ outside Package by Subclass only.

✓ → can be accessed

X → not accessed

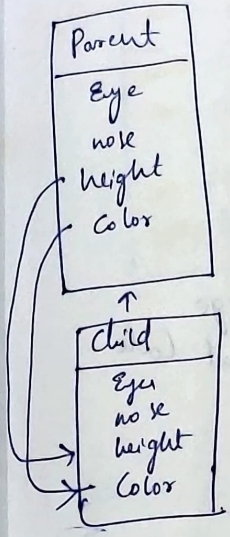
④ Access Modifier Visibility :-

1. Public
2. Protected
3. default
4. Private

↑ Visibility increasing.

→ Public → strongest access specifier

→ Private → weak access specifier.



→ features inherited from Parents

↳ We Cannot Modify some inherited features.

⑤ Inherited Methods :-

Methods which are inherited from parent to child. In child class we are using as it is without any modification.

⑤ Overiding Methods :-

Methods which are inherited from parent to child. In child class as per the requirement it is modified then these Methods are called overiding Methods.

⑥ Specialized Methods :-

A Method which are present only in child not present in parent are called Specialized Methods.

→ Remember these symbols

① '+' → Public
② '#' → Protected
③ '~' → default
④ '-' → Private

→ UML : unified Modeling language
↳ pictorial representation of what code you have to write.

default class

Plane
Cost: int Color: string
fly: void
Land: Void

(1) ⓐ UML (unified Modeling language) :- is a general purpose, graphical Modeling language in the field of software Engineering.

→ UML is used to specify, visualize, Construct and document the artifacts (Major Elements) of the software system.

Eg1 ~~of plane~~.

UML diagram

Eg1 ①

~ Plane
- Cost : int - Color : string
+ fly() : Void # Land() : Void

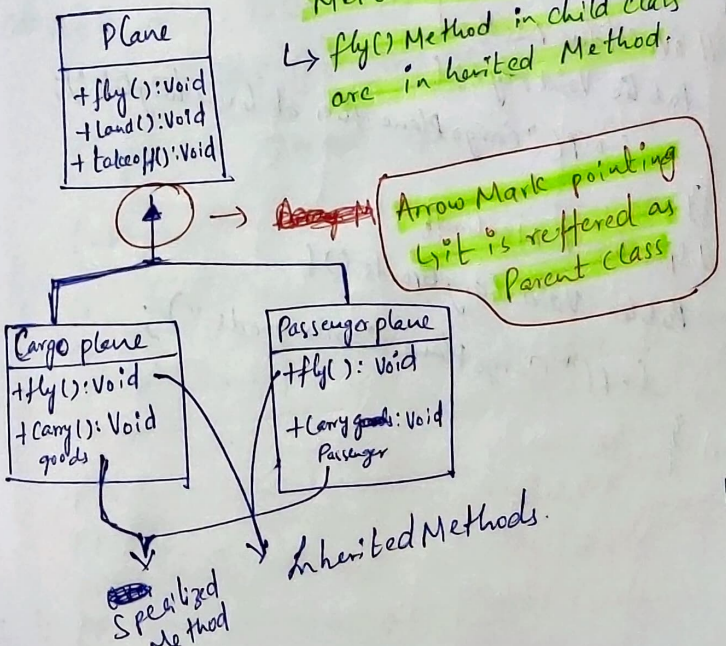
⇒

```
class plane
{
  Private int Cost;
  Private String Color;
  Public Void fly()
  {
  }
  Protected Void Land()
  {
  }
}
```

Code

②

→ Carrygoods() & CaryPassangers() Methods are specialized Methods.
↳ fly() Method in child class are inherited Method.



Code :-

```
Class plane
{
  Public Void takeoff(){
    S.o.p("plane taking off");
  }
  Public Void fly(){
    S.o.p("plane is flying");
  }
  Public Void landing(){
    S.o.p("plane is landing");
  }
}
Class Cargoplane Extends Plane
{
  // overidden Method
  Public Void fly(){
    S.o.p("Cargo Plane flies at lower height");
  }
  // Specilized Methods
  Public Void CarryGoods(){
    S.o.p("Cargo plane carries goods");
  }
}
```

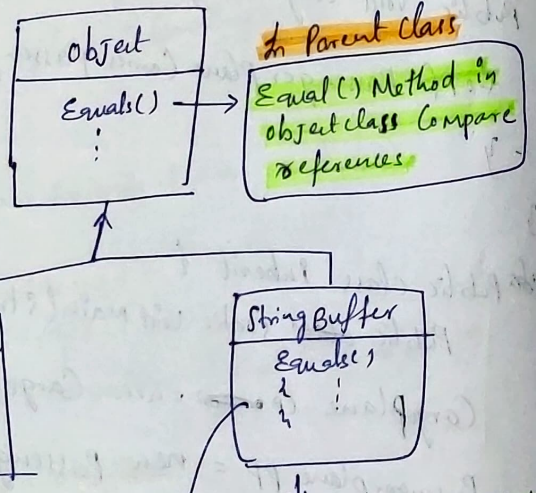
```
Class Passengerplane Extends Plane
{
  // overidden Method
  Public Void fly(){
    S.o.p("Passenger plane flies at Medium height");
  }
  // Specilized Method.
  Public Void CarryPassengers(){
    S.o.p("Passenger plane Carries passengers");
  }
}
Public class Inherit {
  Public static Void main(String[] args){
    Cargoplane cp = new Cargoplane();
    Passengerplane PP = new Passengerplane();
    cp.takeoff();      // Inherited Method
    cp.fly();          // overridden Method
    cp.carryGoods();   // Specialized Methods
    cp.landing();      // Inherited Method.
    PP.takeoff();      // Inherited Method
    PP.fly();          // overridden Method
    PP.carryPassenger(); // Specialized Method
    PP.landing         // Inherited Method.
  }
}
```

⑧ Important Points

→ object class is parent of all classes in Java.

→ Equals(), toString() etc are present in object Class.

→ In built classes are inheriting from object Classes (Parent class).



① here Equals() Method used to compare object Content

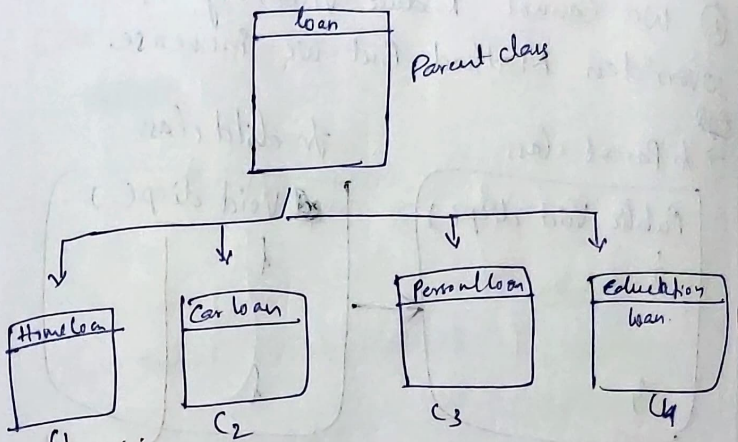
② As per the need Equals() Method is modified.

① here Equals() Method used to compare references.

② here it is inherited Method.

→ Equals() Method in String and StringBuffer acts different.

Home Work



→ Parent → child → inherited → overridden → Specialized.

① Rules to override method :

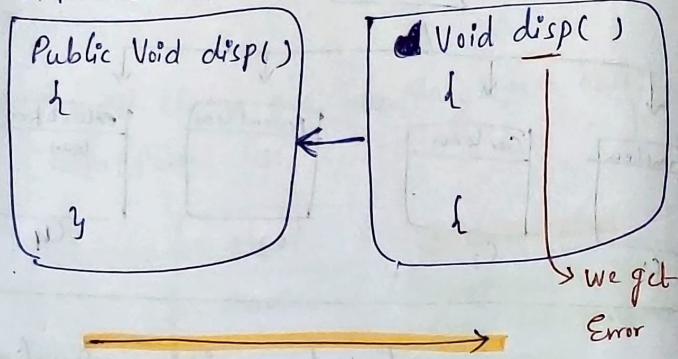
② Base Class

⑨ Rules to override Method :-

① We Cannot Reduce Visibility of overriden Method But we increase.

Eg①

→ In Parent class



→ Method Visibility is Reduce.

Eg② Code ① We Cannot Reduce Visibility of overriden Method

```
Class Parent {
    Public Void disp()
    {
        s.o.p("Parent class");
    }
}
Class Child Extends Parent
{
    Void disp()
    {
    }
}
```

→ here it is default.

we get Error.
Method visibility Cannot be Reduced

Code ② We Can increase Visibility of overriden Method

```
Class Parent {
    Void disp()
    {
        s.o.p("Parent class");
    }
}
Class Child Extends Parent
{
    Public Void disp()
    {
    }
}
```

→ here it is default

⇒ here Visibility of Overriden Method is increasing

→ Many inbuilt Methods are public.

↳ In order to override Method we have increase it's visibility.

Rule (2) :- Return-type of overridden Method Must be same as that of overriding Method in parent.

Code :-

```
Class Parent {                    → overriding Method
    Public int add(){             → here return type of add Method is int.
        S.o.p("Parent");
        return 10+2;
    }
}
Class child Extends Parent
{
    // overriden Method
    // here return type of add Method is Void
    Public Void add()             → Return-type Cannot be Changed it should be same as retun type of overiding Method in Parent class
    {
      S.o.p("child");
    }
```

Rule-③ :- Return type of overriden Method in child class can be different as that of Parent class if it is Co-Varient return type (return type is-A relationship (or Parent-child relationship).

Code :- check this code in github (inherit 13)

```
Class Interest {

}
Class Interest PersonalLoan Extends Interest
{

}

Class Loans {
  //overriding Method
  // here return type is Interest
  Public Interest interest()
                 → Return type
  {
    Interest it = new Interest()
    return it;
  }
}
Class PersonalLoan Extends Loans
{
  //overridden Method.
  Public Interest PersonalLoan interest()
                 → Return type
  {
    Interest PersonalLoan ipl = new
                    Interest PersonalLoan();
    return ipl;
  }
}
```

→ here is-A relationship (or Parent-child relationship) is there. that's the reason different return type is allowed for overriding and overriden Method.

→ We Call it as Co-Varient return-type.

→ In return type there is Parent child relationship then return types can be different.

Rule-④ :- Parameters of overriden Method Must be same as that of Parent else it will be Considered as Specialized Method Considering Method overloading.

Code :-

```
Class Parent {
  Public int add(int a, int b)
  {
    return a+b;
  }
}
class child Extends Parent
{
  Public int add (int a)
  {
    return a;
  }
}
```

→ here Name of Methods are same but different parameters
→ it is Called Method overloading
→ This Method is Called as Specialized Method

→ check this Code in github (inherit 14)

→ Super() Method

↳ it is written inside Constructor.

↳ it will call parent class Constructor.

→ Super _ keyword.

(10) Super keyword :-

the Super keyword in Java is a reference Variable which is used to refer immediate Parent class object.

Usage of Java Super Keyword

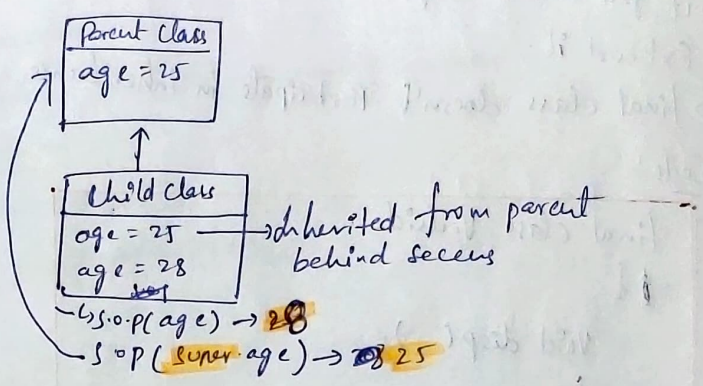
① Super can be used to refer immediate Parent class instance Variable.

② Super can be invoke immediate Parent class Method

Code :-

```
Class Parent {
    int age = 25;
}
class child Extends parent{
 int age = 28;
 Void disp() {
 S.o.p(age);          → 1128
 S.o.p(super.age);    → 1125
 }
Public class inherit {
    Public static Void Main(String() args){
        Child c = new child();
        c.disp
}
```

→ here Super keyword refered to immediate Parent class instance Variable.



(11) final Keyword :-

the final keyword in Java is used to restrict the Java user. final can be

① Variable
② Method
③ class

class keyword → to Create class.
new keyword → to Create object.

→ final keyword applied for Class, Method and Variable.

(i) Java final class :-

→ if you make any class as final, you cannot Extend it.

→ final class doesn't Participate in inheritance

Code :-

```
final class Vehicle
{
    Void disp( )
    {
      S.o.p("Vehicle");
    }
}
Class Car Extends Vehicle
{

}
Public class Inherit {
    Public static Void main(string[] args){

      Car c = new Car();
       c.disp;
    }
}
```

↳ we get Error
↳ final class doesn't Participate in inheritance

(ii) Java final Method

① if you Make any Method as final, you Cannot override it..

② final Method participates in inheritance.

Code :-

```
Class Vehicle {
   final Void disp( ) {
      S.o.P("Vechicle");
   }
}
Class Car Extends Vehicle
{
  Void disp()
  {
    S.o.P("Car");
  }
}
Public class Inherit {
      Public Static Void main(string( ) arg) {

        Car c = new Car( );
         c.disp( );
      }
}
```

we get Error (final method get inherited but we Cannot override)

(iii) Java final Variable

→ if you make any variable as final, you Cannot change the value of final variable. (it will be Constant).

Code :-

```
Class Parent {
    final int i=10;
    Void disp() {
        i=20;
        S.o.p(i); // output: 10
        S.o.p("Parent");
    }
}
Class child Extends Parent
{

}
Public class inherit {
    Public Static Void main(String() args)
    {
        Car c = new Car();
        c.disp();
    }
}
```

here i will be 10 not 20

Java final Keyword

① Stop Value change
② Stop Method overriding
③ Stop inheritance.