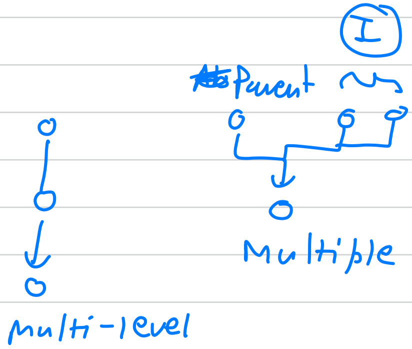
Agenda

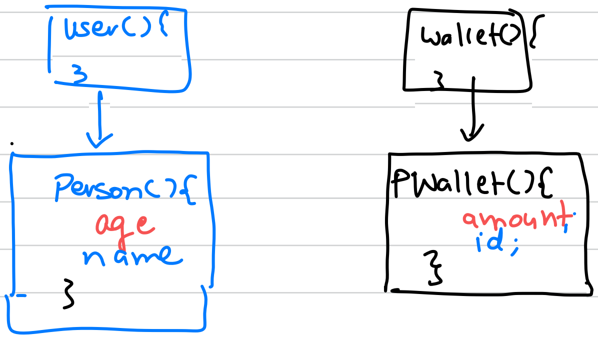
- Interfaces.
  - ↳ Code Demo
  - ↳ Advantages (Multiple Inheritance)
- Abstract classes
  - ↳ code
  - ↳ Theory

Real examples



Interface

- Java (Abstract Type)
- collection of methods



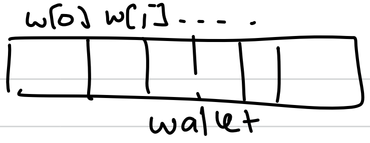
```
class Person {
    age;
    name,                          Person
    bool comparePerson(          p2 )
             ret  ~~s~~.age - p2.age
    }

}
```

sort (... <Person> ...) acc to age
         list

sort (.... <P-wallets> ...) acc to amount
         list

```
class wallet {
        amount,
        id,
     bool comparewallet (w2){
          ret  ~~#~~ amt -
                 w2.amt,
     }
}
```



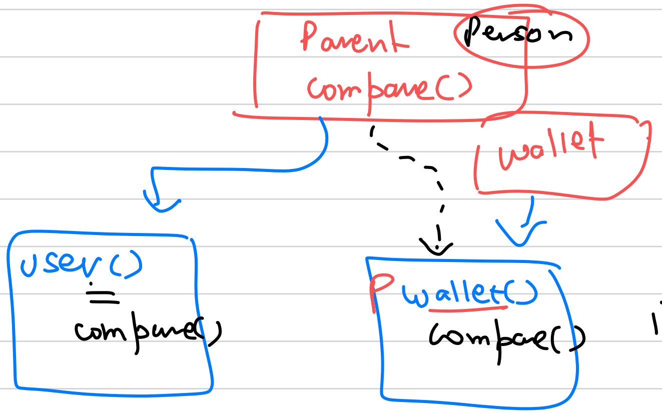
```
Sort (        ) {
  ⋮
    if ( w[i].compareTo ~~Wallet~~ (w[j]) ){
           swap(w(i), w(j));
    }
  ⋮
  ⋮
}

Sort (        ) {
  ⋮
    if ( w[i].compareTo ~~Person~~ (w[j]) ){
           swap(w(i), w(j));
    }
  ⋮
  ⋮
}
```

```
interface
    Comparable {

        compareTo(){
            =
        }
    }
```

if both classes have a compare ~~men~~ sort doesn't need to modified.



user is A Person
~~wallet is A Person.~~

inheritance.

Contract. your class must implement following methods given in the interface.

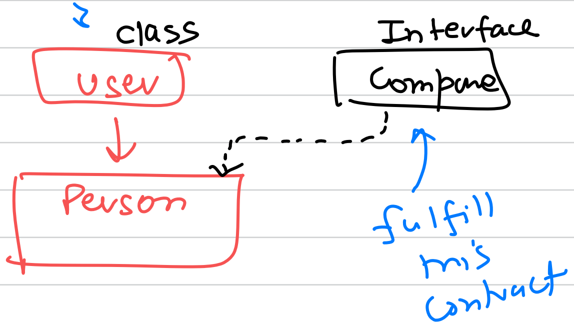
```
public interface Compare {
        :
        :
        bool   compare( );
        :
```

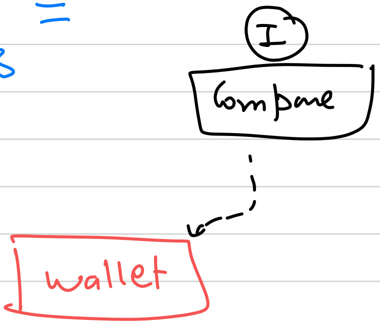
} one or more methods.

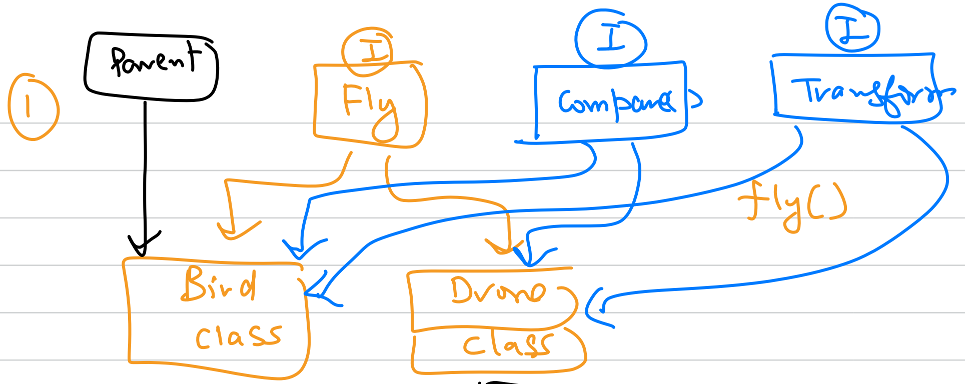
3

class Person extends User implement Compare {
   compare ( ) {
      =
   }
}

class Wallet implement compare {
   compare ( ) {
      =
   }
}



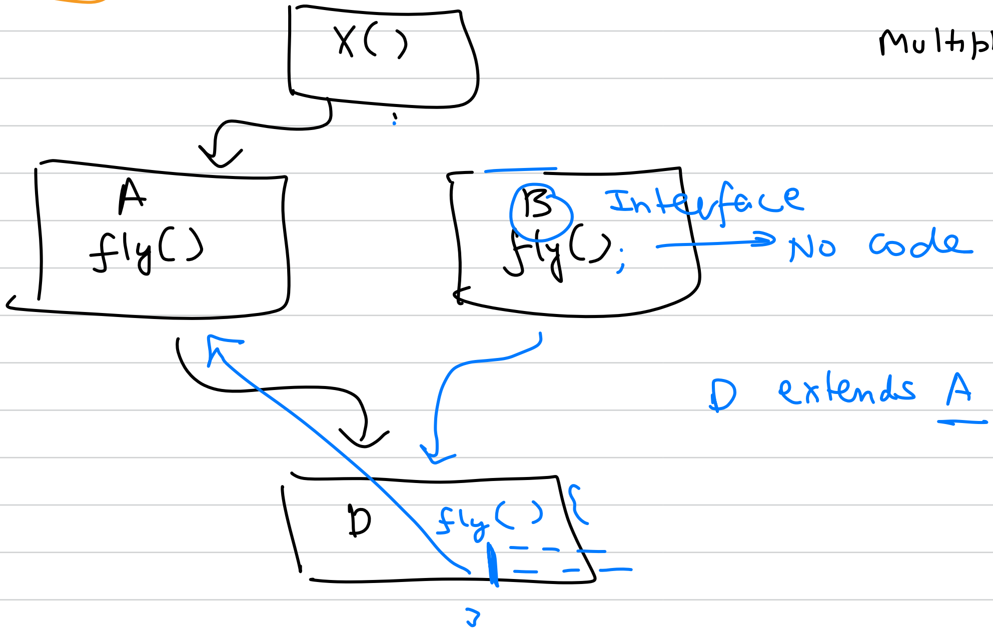




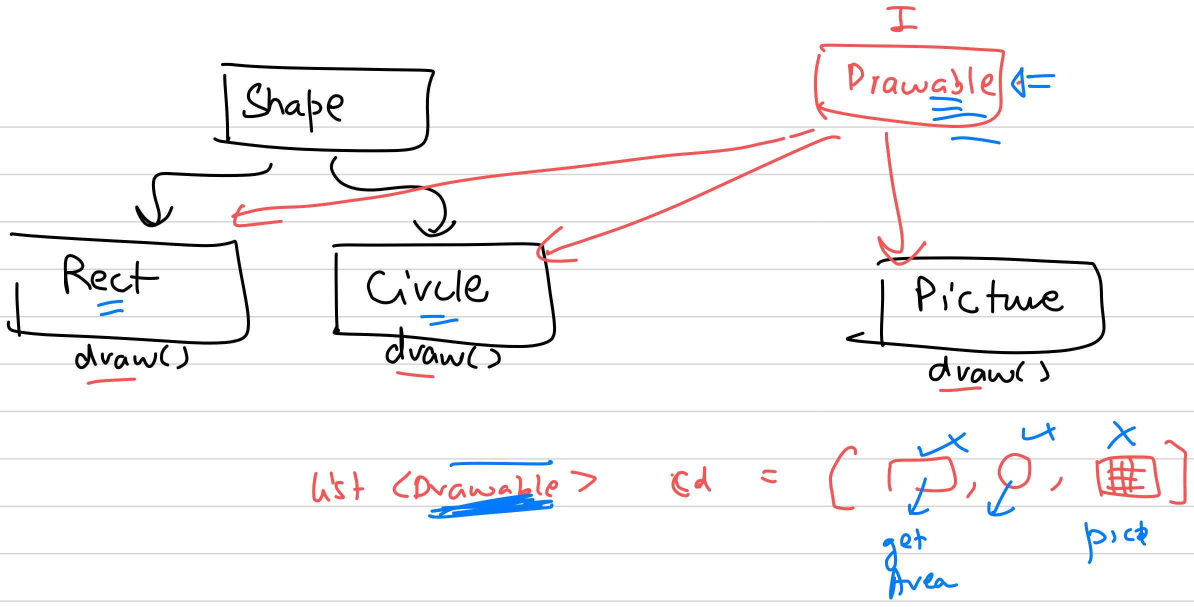
Multiple I implemented.

Multiple inheritance solved

Only one parent



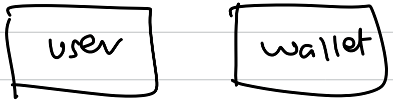
D extends A implement B


I
Drawable
Shape
Rect
draw()
Circle
draw()
Picture
draw()
list <Drawable> d = [, ,]
get Area
pict

d. getArea()

Interfaces :

1. Abstraction
2. Multiple inheritance
3. Loose coupling
4. Define a common ~~for~~ unrelated class
5. Polymorphism



ABSTRACT Classes

⇒ you want to design $\overset{\text{a class}}{\wedge}$ but you might not want to implement now.

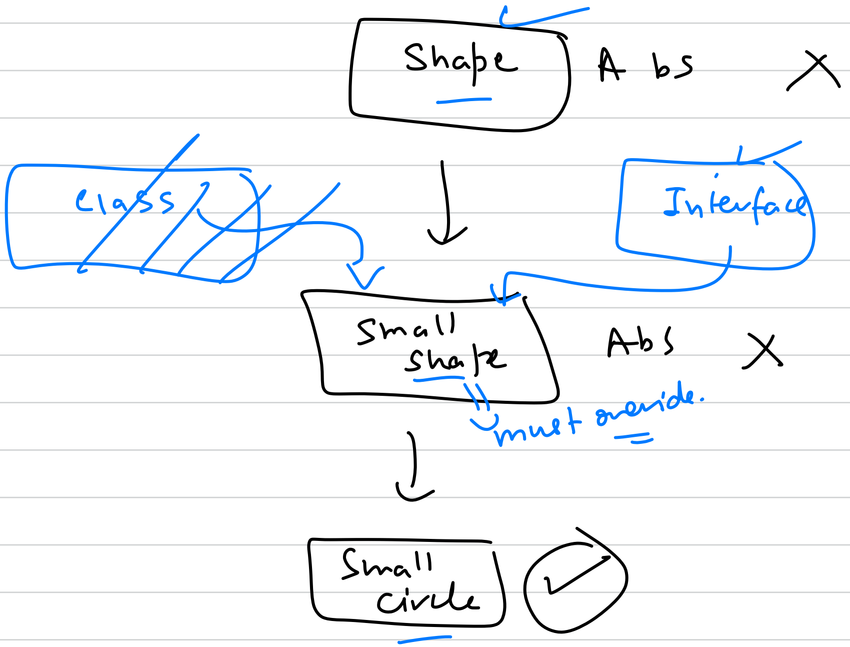
⇒ Postpone your implementation level details later to some other class

When you don't want provide implementation for a method you can make method and class As abstract.

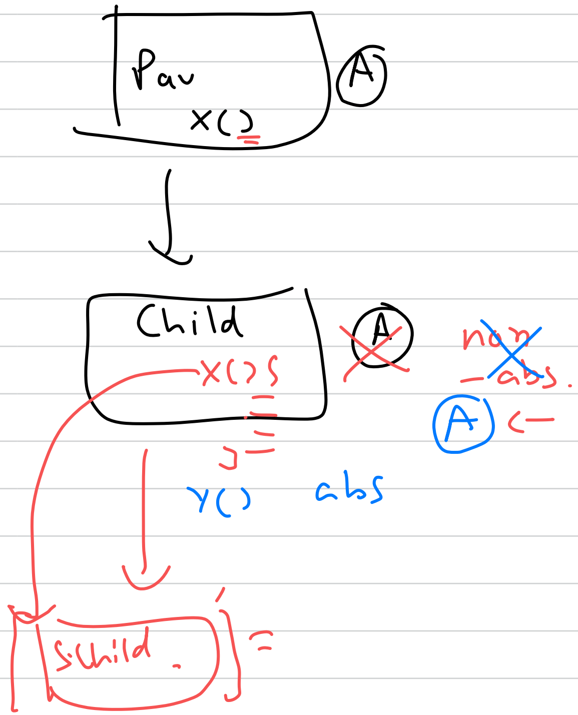
An abstract class can have both abstract and non-abstract methods.

You can't create objects of an abstract class, but you can still achieve polymorphic behaviour .

```
Shape r = new Rectangle();
Shape c = new Circle();
```



Shape
A bs
X
class
Interface
Small Shape
Abs
X
must override.
Small circle
✓



Par
x()
Ⓐ
Child
x() {
}
Ⓐ
non -abs.
Ⓐ
y()
abs
S.Child