(4) Object-Oriented Design Principles

Java Packages

- A package is a collection of compilation units (.java files) grouped together into a folder
- Each compilation unit has a package declaration at the beginning of the file
- Each compilation unit uses packages by use of import statements

Access Modifiers

Access Modifiers in Java

- In Java, field variables and methods can have any of the four access modifiers:

  1. public
  2. protected
  3. package-private (when no access modifier is specified)
  4. private

- The outer-most classes/interfaces can have only either public or package-private

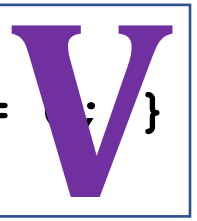
(1) public

package p

```
public class A {
    public int i;

    void x() { i = 0; }
}
```

```
public class B extends A {
    void y() { i = 0; }
}
```

```
public class C {
    void z(A a) { a.i = 0; }
}
```



package q

```
public class D extends p.A {
    void y() { i = 0; }
}
```

```
public class E {
    void z(p.A a) { a.i = 0; }
}
```

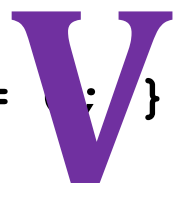
(2) protected

package p

```
public class A {
    protected int i;

    void x() { i = 0; }
}
```

```
public class B extends A {
    void y() { i = 0; }
}
```

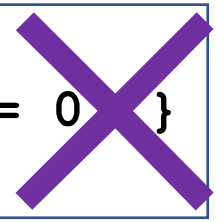
```
public class C {
    void z(A a) { a.i = 0; }
}
```



package q

```
public class D extends p.A {
    void y() { i = 0; }
}
```

```
public class E {
    void z(p.A a) { a.i = 0; }
}
```



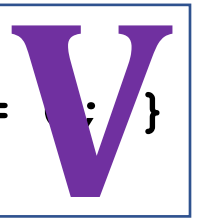
(3) package-private

package p

```
public class A {
    int i;

    void x() { i = 0; }
}
```

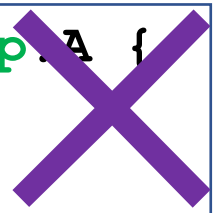
```
public class B extends A {
    void y() { i = 0; }
}
```

```
public class C {
    void z(A a) { a.i = 0; }
}
```

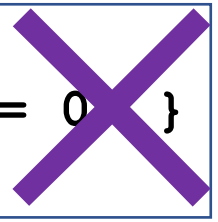


package q

```
public class D extends p.A {
    void y() { i = 0; }
}
```



```
public class E {
    void z(p.A a) { a.i = 0; }
}
```

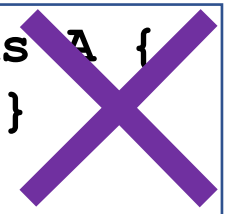


(4) private

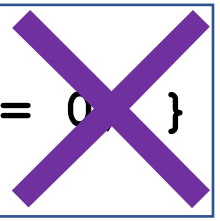
package p

```
public class A {
    private int i;

    void x() { i = 0; }
}
```

```
public class B extends A {
    void y() { i = 0; }
}
```

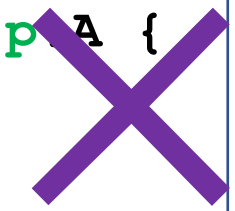


```
public class C {
    void z(A a) { a.i = 0; }
}
```

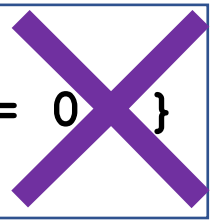


package q

```
public class D extends p.A {
    void y() { i = 0; }
}
```



```
public class E {
    void z(p.A a) { a.i = 0; }
}
```



Access Modifiers

Access Modifiers	Same Class	Other Classes in Same Package	Child Classes	Other Packages
public				
private				
protected				
package-private				

Access Modifiers

Access Modifiers	Same Class	Other Classes in Same Package	Child Classes	Other Packages
public	Y	Y	Y	Y
private	Y	N	N	N
protected	Y	Y	Y	Y (in subclasses) N (not in subclasses)
package-private	Y	Y	Y (in same package) N (not in same package)	N

Exercises

(1) public

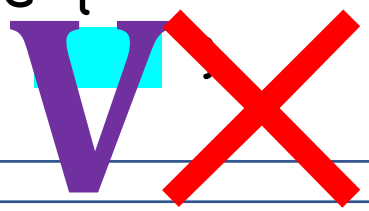
```
p.A a = new p.A();
a.i = 0;

p.B b = new p.B();
b.i = 0;

p.C c = new p.C();
c.i = 0;

q.D d = new q.D();
d.i = 0;

q.E e = new q.E();
e.i = 0;
```

package p

```
public class A {
    public int i;

    void x() {
    }
}
```

```
public class B extends A {
    void y() {
    }
}
```

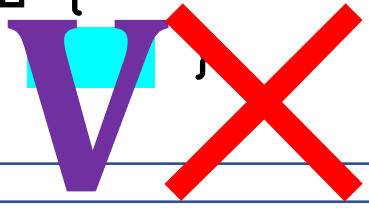
```
public class C {
    void z() {
    }
}
```



package q

```
public class D extends p.A {
    void y() {
    }
}
```

```
public class E {
    void z() {
    }
}
```



(3) package-private

```
p.A a = new p.A();
a.i = 0;

p.B b = new p.B();
b.i = 0;

p.C c = new p.C();
c.i = 0;

q.D d = new q.D();
d.i = 0;

q.E e = new q.E();
e.i = 0;
```

package p

```
public class A {
    int i;

    void x() {
    }
}
```

V D

```
public class B extends A {
    void y() {
    }
}
```

V D

```
public class C {
    void z() {
    }
}
```

V X

package q

```
public class D extends p.A {
    void y() {
    }
}
```

X X

```
public class E {
    void z() {
    }
}
```

X X

(4) private

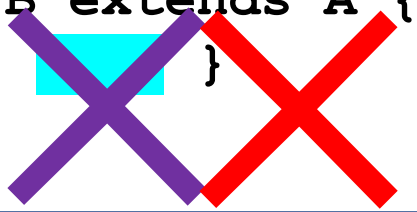
```
p.A a = new p.A();
a.i = 0;

p.B b = new p.B();
b.i = 0;

p.C c = new p.C();
c.i = 0;

q.D d = new q.D();
d.i = 0;

q.E e = new q.E();
e.i = 0;
```

package p

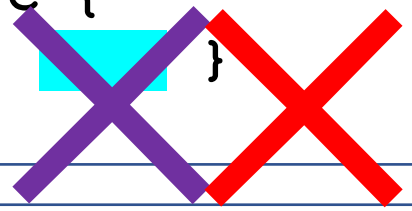
```
public class A {
    private int i;

    void x() {
}
```

VD

```
public class B extends A {
    void y() {
}
```

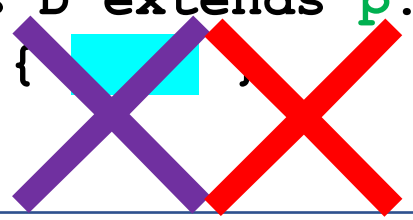


```
public class C {
    void z() {
}
```

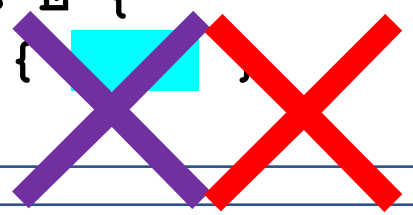


package q

```
public class D extends p.A {
    void y() {
}
```



```
public class E {
    void z() {
}
```



(2) protected

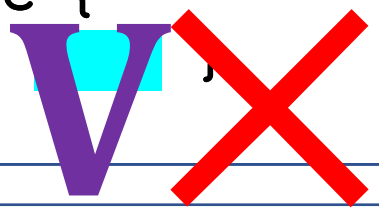
```
p.A a = new p.A();
a.i = 0;

p.B b = new p.B();
b.i = 0;

p.C c = new p.C();
c.i = 0;

q.D d = new q.D();
d.i = 0;

q.E e = new q.E();
e.i = 0;
```

package p

```
public class A {
    protected int i;

    void x() {
    }
```

```
public class B extends A {
    void y() {
    }
```

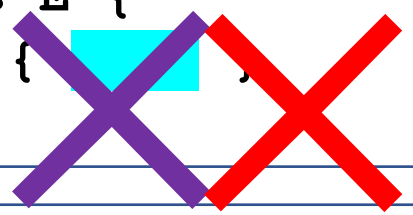
```
public class C {
    void z() {
    }
```



package q

```
public class D extends p.A {
    void y() {
    }
```

```
public class E {
    void z() {
    }
```



In other packages, the access is permitted if and only if the type is the current class or any subclass

Polymorphism

Method Signature

1. Method name
2. Parameter types in order

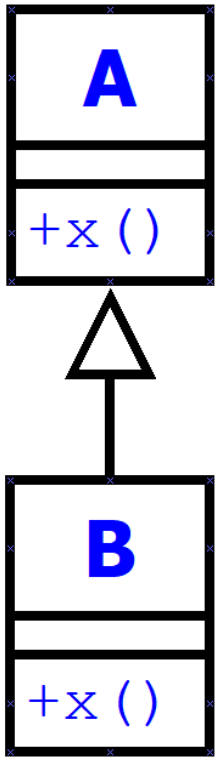
```
void m(String x) {...}
void m(String y) {...}

void   m(String x) {...}
String m(String x) {...}

void m(int i, String x) {...}
void m(int x, String i) {...}


void m(int i, String x, double d) {...}
void m(double i, String x, int d) {...}
```

different signature

Overridden/Overriding methods

The parent's method is overridden by its child's method (or the child's method is overriding its parent's method) if and only if

1. both parent and child classes have methods with same signature
2. the parent's method is visible in the child



* parent = ancestor

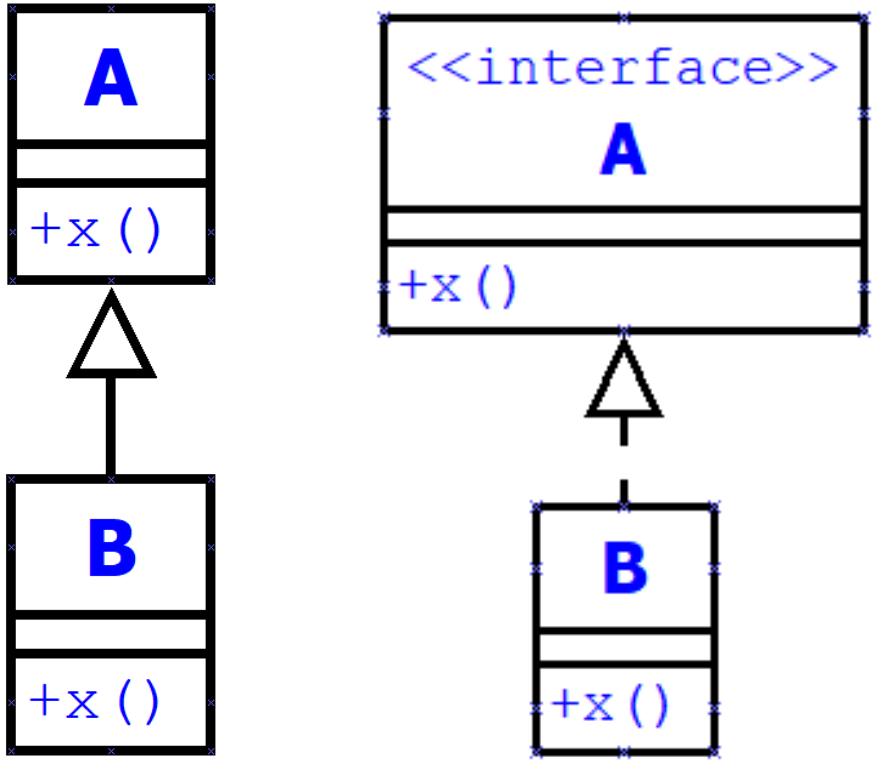
  child = descendant

** An overriding method cannot reduce the visibility of the overridden method.

Run-time Polymorphism

- Given overridden/overriding methods, Java allows to invoke the overriding method using a reference of the parent type that refers to an object of child type

```
A v = new B();
v.x();
```



Compile-time Polymorphism

- A class has more than one visible method that has same name but different parameter types in order

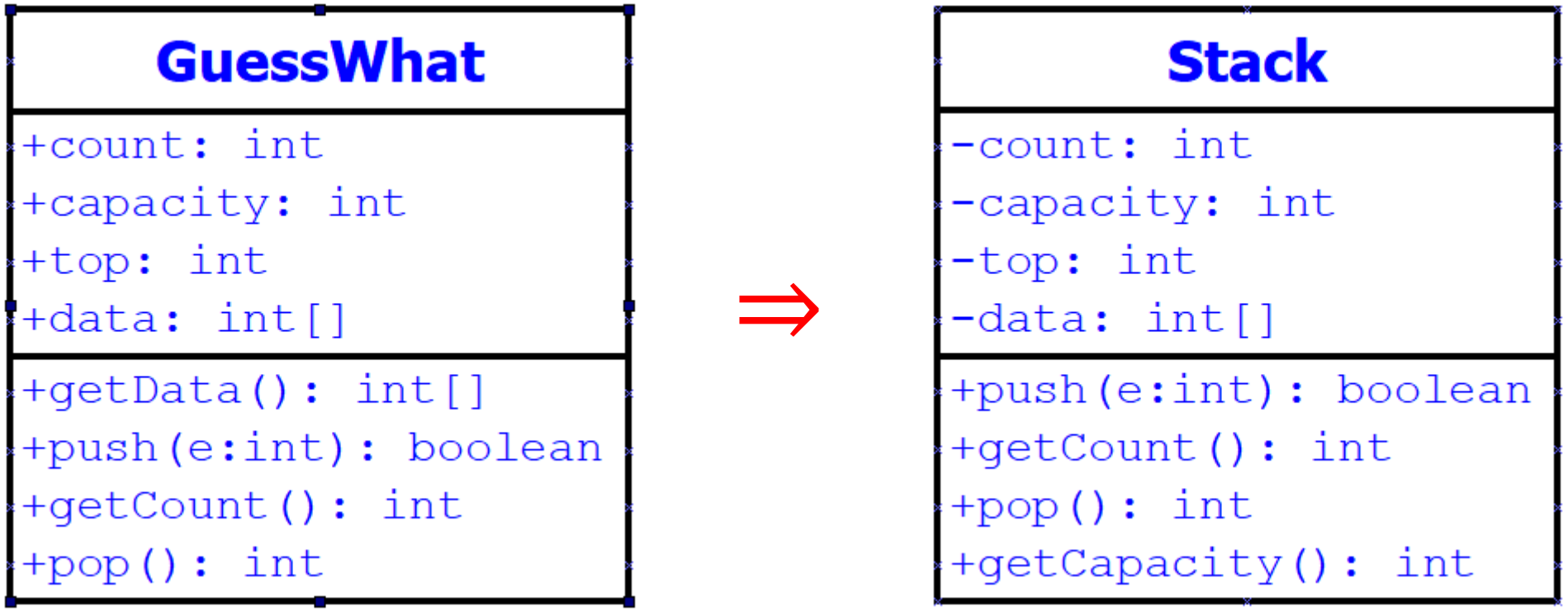
```
void m(String x) {...}
void m(String y) {...}

void   m(String x) {...}
String m(String y) {...}

void m(int i, String x) {...}
void m(int x, String i) {...}

void m(int i, String x, double d) {...}
void m(double i, String x, int d) {...}
```

compile-time Polymorphism (method overloading)

Principles of Object-Oriented Design

Information Hiding

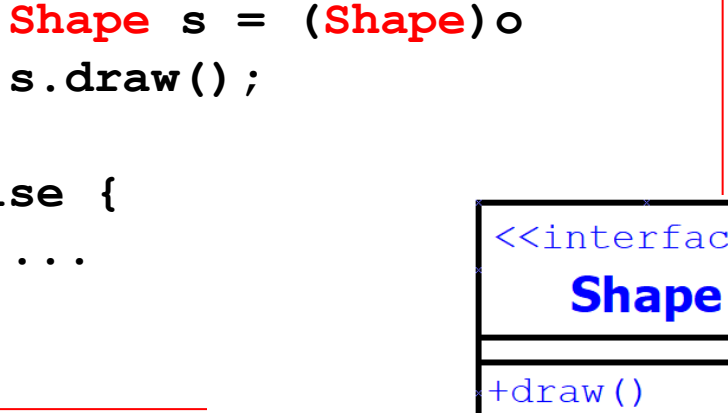
- Keep things as private as you can



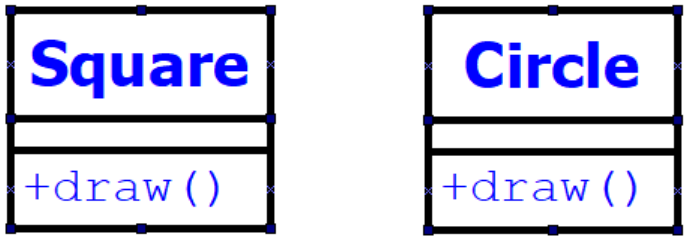
Open/Closed principle

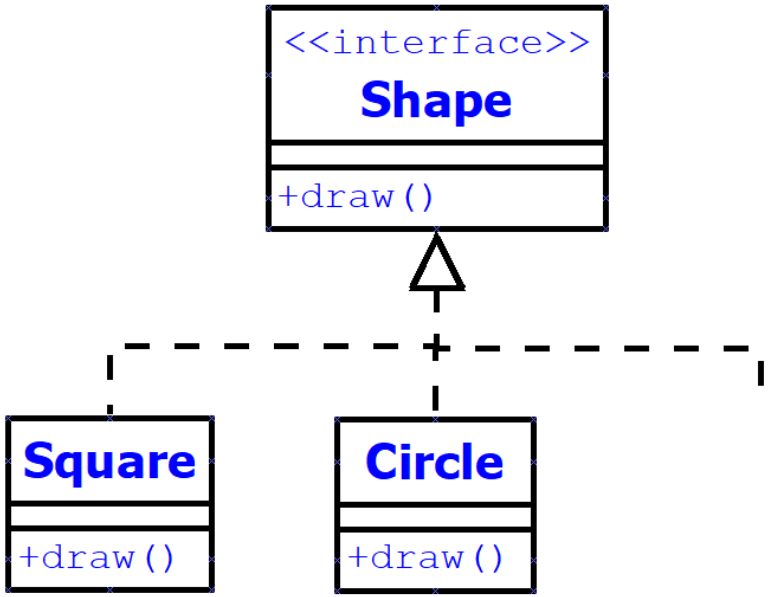
- Be open for extensions, but closed for modifications

```
static void draw(Object o) {
  if(o instanceof Square) {
    Square s = (Square)o;
    s.draw();
  }
  else if(o instanceof Circle) {
    Circle c = (Circle)o;
    c.draw();
  }
  else {
    ...
  }
}
```



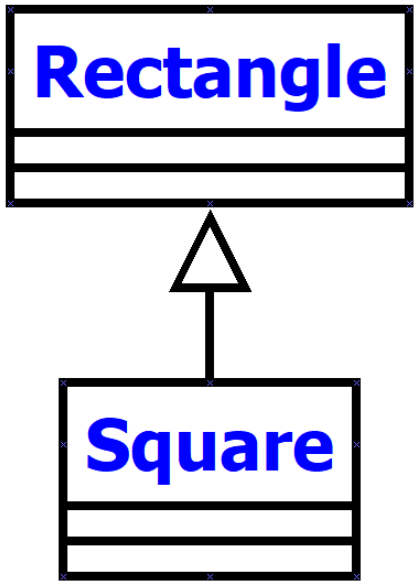
```
static void draw(Object o) {
  if(o instanceof Shape) {
    Shape s = (Shape)o
    s.draw();
  }
  else {
    ...
  }
}
```





Liskov Substitution Principle

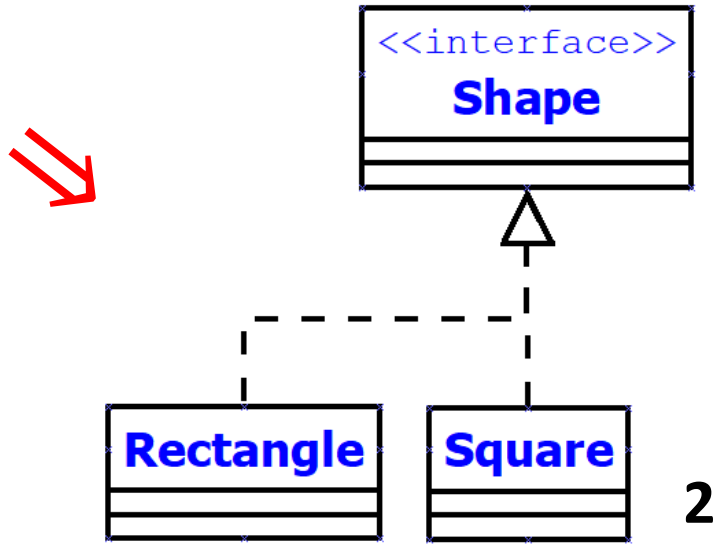
- The subtype must behave like its supertype




```
Rectangle r  = ...
r.setWidth(5);
r.setHeight(4);
if(r.getArea() == 20) {
  ...
}
```

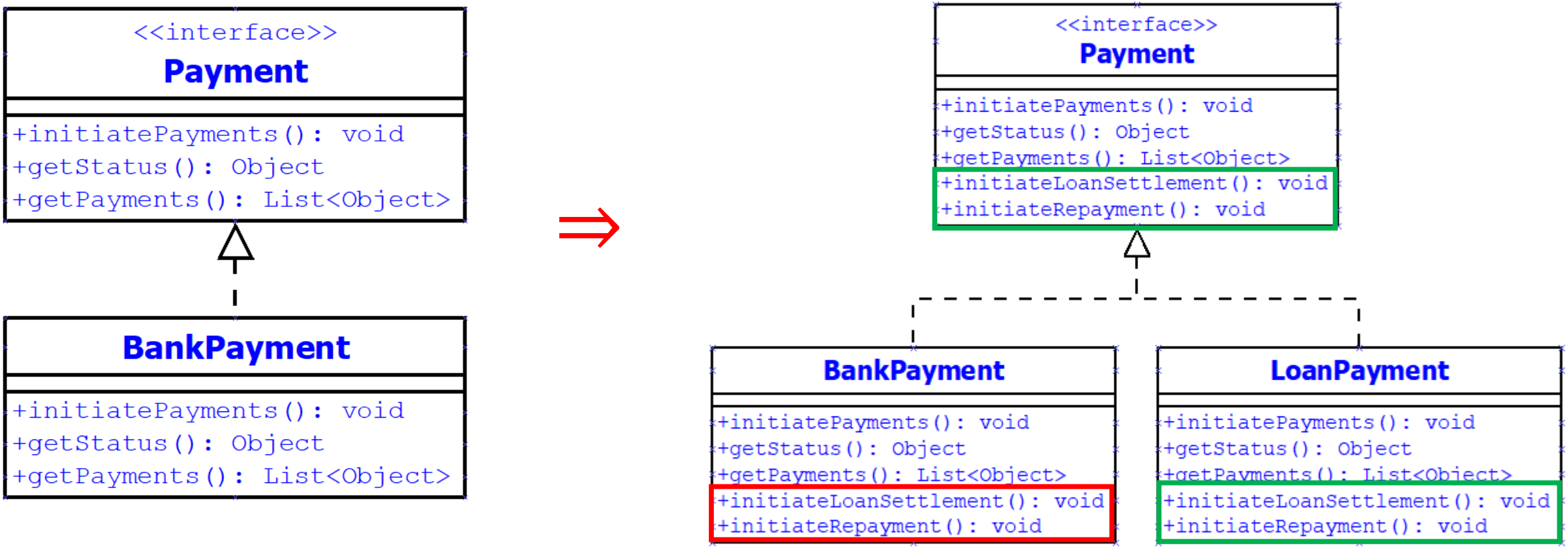
```
class Rectangle {
  protected int width;
  protected int height;

  public void setWidth(int width) {
    this.width = width;
  }

  public void setHeight(int height) {
    this.height = height;
  }

  public int getWidth(){
    return width;
  }

  public int getHeight(){
    return height;
  }

  public int getArea(){
    return this.width * this.height;
  }
}
```

```
class Square extends Rectangle {
  public void setWidth(int width){
    this.width = width;
    this.height = width;
  }

  public void setHeight(int height){
    this.width = height;
    this.height = height;
  }
}
```



Interface Segregation Principle

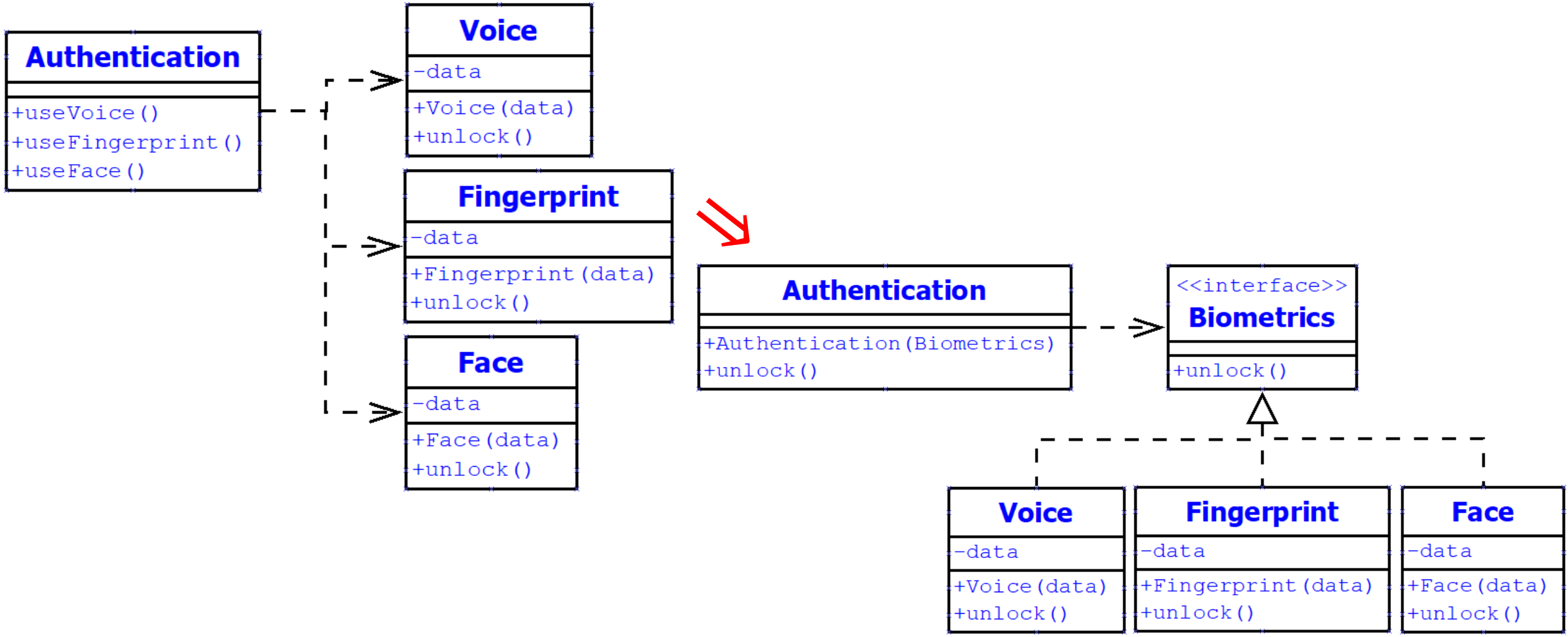
- One interface per job



Implementing unsupported operations

Dependency Inversion Principle

- High level classes should not depend on low level classes



Object-Oriented Design Principles

1. Single-responsibility principle
   Every class should have only one responsibility
2. Open–closed principle
   Be open for extension, but closed for modification
3. Liskov substitution principle
   Subtypes must be substitutable for their base types
4. Interface segregation principle
   One interface per job
5. Dependency inversion principle
   High level modules should not depend on low level modules