

CS 520

Theory and Practice of Software Engineering
Fall 2026

Object Oriented Design Principles

1

Today

- OO design principles
 - Information hiding (and encapsulation)
 - Polymorphism
 - Open/closed principle
 - Inheritance in Java
 - The diamond of death
 - Liskov substitution principle
 - Composition/aggregation over inheritance
- On your own exercise:
Code review and (re)design of an MVC application

2

OO design principles

- **Information hiding (and encapsulation)**
- Polymorphism
- Open/closed principle
- Inheritance in Java
- The diamond of death
- Liskov substitution principle
- Composition/aggregation over inheritance

3

Information hiding

MyClass
+ nElem : int + capacity : int + top : int + elems : int[] + canResize : bool
+ resize(s:int):void + push(e:int):void + capacityLeft():int + getNumElem():int + pop():int + getElems():int[]

```
public class MyClass {
    public int nElem;
    public int capacity;
    public int top;
    public int[] elems;
    public boolean canResize;
    ...
    public void resize(int s){...}
    public void push(int e){...}
    public int capacityLeft(){...}
    public int getNumElem(){...}
    public int pop(){...}
    public int[] getElems(){...}
}
```

4

Information hiding

MyClass
+ nElem : int + capacity : int + top : int + elems : int[] + canResize : bool
+ resize(s:int):void + push(e:int):void + capacityLeft():int + getNumElem():int + pop():int + getElems():int[]

```
public class MyClass {
    public int nElem;
    public int capacity;
    public int top;
    public int[] elems;
    public boolean canResize;
    ...
    public void resize(int s){...}
    public void push(int e){...}
    public int capacityLeft(){...}
    public int getNumElem(){...}
    public int pop(){...}
    public int[] getElems(){...}
}
```

What does MyClass do?

5

Information hiding

Stack
+ nElem : int + capacity : int + top : int + elems : int[] + canResize : bool
+ resize(s:int):void + push(e:int):void + capacityLeft():int + getNumElem():int + pop():int + getElems():int[]

```
public class Stack {
    public int nElem;
    public int capacity;
    public int top;
    public int[] elems;
    public boolean canResize;
    ...
    public void resize(int s){...}
    public void push(int e){...}
    public int capacityLeft(){...}
    public int getNumElem(){...}
    public int pop(){...}
    public int[] getElems(){...}
}
```

Anything that could be improved in this implementation?

6

Information hiding

Stack
+ nElem : int
+ capacity : int
+ top : int
+ elems : int[]
+ canResize : bool
+ resize(s:int):void
+ push(e:int):void
+ capacityLeft():int
+ getNumElem():int
+ pop():int
+ getElems():int[]

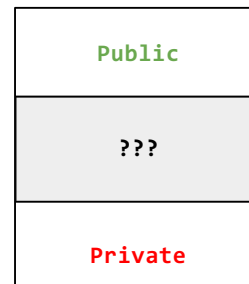
Stack
- elems : int[]
...
+ push(e:int):void
+ pop():int
...

Information hiding:

- Reveal as little information about internals as possible.
- Separate public interface from implementation details.
- Reduce complexity.

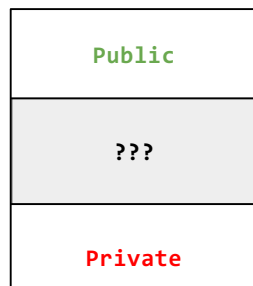
7

Information hiding vs. visibility



8

Information hiding vs. visibility



- Protected, package-private, or friend-accessible (C++).
- Not part of the public API.
- Implementation detail that a subclass/friend may rely on.

9

OO design principles

- Information hiding (and encapsulation)
- **Polymorphism**
- Open/closed principle
- Inheritance in Java
- The diamond of death
- Liskov substitution principle
- Composition/aggregation over inheritance

10

A little refresher: what is Polymorphism?



11

A little refresher: what is Polymorphism?

An object's ability to provide different behaviors.

Types of polymorphism

- Ad-hoc polymorphism (e.g., operator overloading)
 - `a + b` ⇒ `String` vs. `int`, `double`, etc.
- Subtype polymorphism (e.g., method overriding)
 - `Object obj = ...; obj.toString();` ⇒ `toString()` can be overridden in subclasses and therefore provide a different behavior.
- Parametric polymorphism (e.g., Java generics)
 - `class LinkedList<E> { void add(E) {...} E get(int index) {...}` ⇒ A `LinkedList` can store elements regardless of their type but still provide full type safety.

<https://www.destroyallsoftware.com/talks/wat>

12

A little refresher: what is Polymorphism?

An object's ability to provide different behaviors.

Types of polymorphism

- Subtype polymorphism (e.g., method overriding)
 - `Object obj = ...; ⇒ toString()` can be overridden in subclasses
`obj.toString();` and therefore provide a different behavior.

Subtype polymorphism is essential to many OO design principles.

13

OO design principles

- Information hiding (and encapsulation)
- Polymorphism
- **Open/closed principle**
- Inheritance in Java
- The diamond of death
- Liskov substitution principle
- Composition/aggregation over inheritance

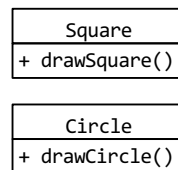
14

Open/closed principle

Software entities (classes, components, etc.) should be:

- **open** for extensions
- **closed** for modifications

```
public static void draw(Object o) {
    if (o instanceof Square) {
        drawSquare((Square) o)
    } else if (o instanceof Circle) {
        drawCircle((Circle) o);
    } else {
        ...
    }
}
```



Good or bad design?

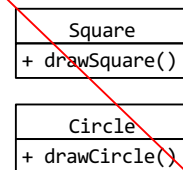
15

Open/closed principle

Software entities (classes, components, etc.) should be:

- **open** for extensions
- **closed** for modifications

```
public static void draw(Object o) {
    if (o instanceof Square) {
        drawSquare((Square) o)
    } else if (o instanceof Circle) {
        drawCircle((Circle) o);
    } else {
        ...
    }
}
```



Violates the open/closed principle!

16

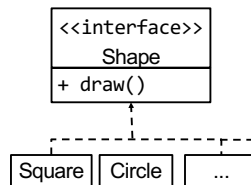
Open/closed principle

Software entities (classes, components, etc.) should be:

- **open** for extensions
- **closed** for modifications

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

```
public static void draw(Shape s) {
    s.draw();
}
```



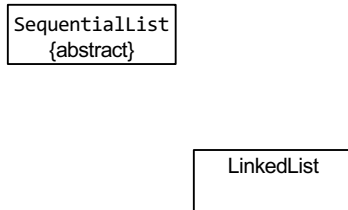
17

OO design principles

- Information hiding (and encapsulation)
- Polymorphism
- Open/closed principle
- **Inheritance in Java**
- The diamond of death
- Liskov substitution principle
- Composition/aggregation over inheritance

18

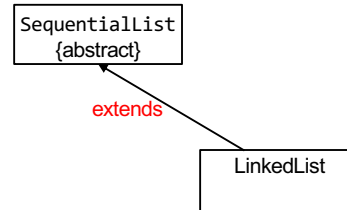
Inheritance: (abstract) classes and interfaces



19

Inheritance: (abstract) classes and interfaces

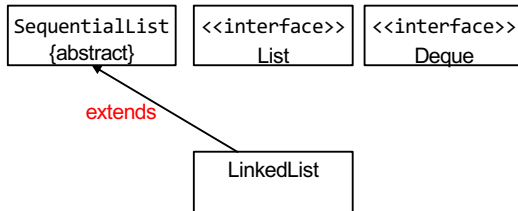
LinkedList extends SequentialList



20

Inheritance: (abstract) classes and interfaces

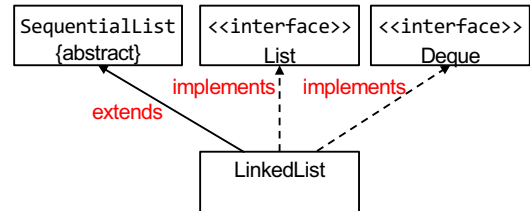
LinkedList extends SequentialList



21

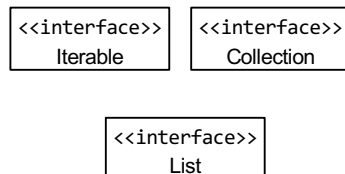
Inheritance: (abstract) classes and interfaces

LinkedList extends SequentialList implements List, Deque



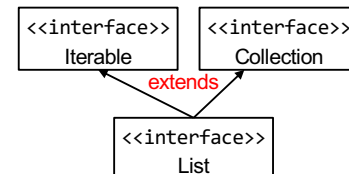
22

Inheritance: (abstract) classes and interfaces



23

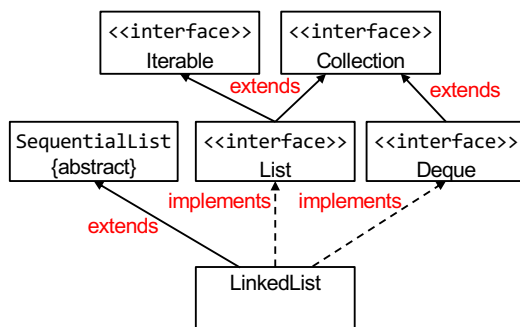
Inheritance: (abstract) classes and interfaces



List extends Iterable, Collection

24

Inheritance: (abstract) classes and interfaces



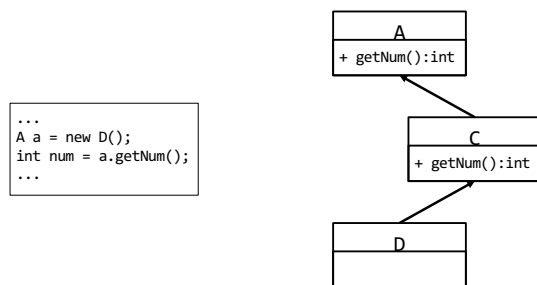
25

OO design principles

- Information hiding (and encapsulation)
- Polymorphism
- Open/closed principle
- Inheritance in Java
- **The diamond of death**
- Liskov substitution principle
- Composition/aggregation over inheritance

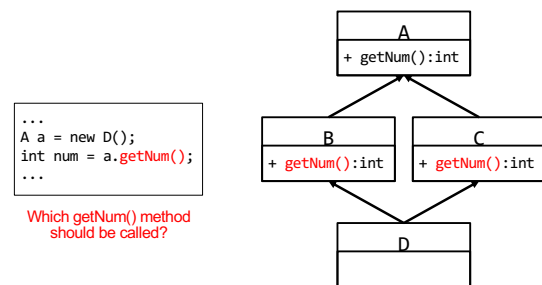
26

The “diamond of death”: the problem



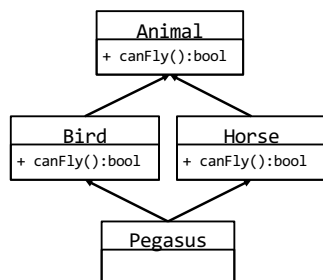
27

The “diamond of death”: the problem



28

The “diamond of death”: concrete example



29

OO design principles

- Information hiding (and encapsulation)
- Polymorphism
- Open/closed principle
- Inheritance in Java
- The diamond of death
- **Liskov substitution principle**
- Composition/aggregation over inheritance

30

Design principles: Liskov substitution principle

Motivating example

We know that a square is a special kind of a rectangle. So, which of the following OO designs makes sense?



31

Design principles: Liskov substitution principle

Subtype requirement

Let object x be of type $T1$ and object y be of type $T2$. Further, let $T2$ be a subtype of $T1$ ($T2 \leq T1$). Any provable property about objects of type $T1$ should be true for objects of type $T2$.



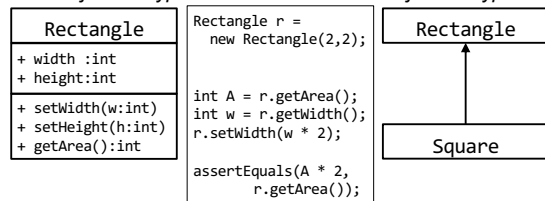
Is the subtype requirement fulfilled?

32

Design principles: Liskov substitution principle

Subtype requirement

Let object x be of type $T1$ and object y be of type $T2$. Further, let $T2$ be a subtype of $T1$ ($T2 \leq T1$). Any provable property about objects of type $T1$ should be true for objects of type $T2$.

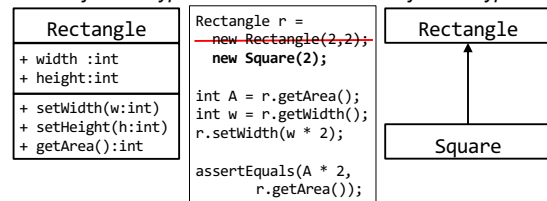


33

Design principles: Liskov substitution principle

Subtype requirement

Let object x be of type $T1$ and object y be of type $T2$. Further, let $T2$ be a subtype of $T1$ ($T2 \leq T1$). Any provable property about objects of type $T1$ should be true for objects of type $T2$.

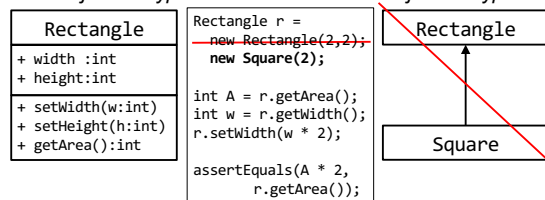


34

Design principles: Liskov substitution principle

Subtype requirement

Let object x be of type $T1$ and object y be of type $T2$. Further, let $T2$ be a subtype of $T1$ ($T2 \leq T1$). Any provable property about objects of type $T1$ should be true for objects of type $T2$.



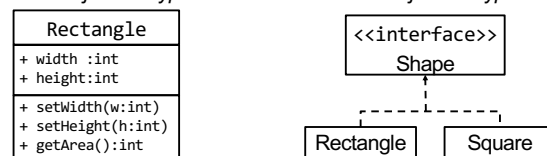
Violates the Liskov substitution principle!

35

Design principles: Liskov substitution principle

Subtype requirement

Let object x be of type $T1$ and object y be of type $T2$. Further, let $T2$ be a subtype of $T1$ ($T2 \leq T1$). Any provable property about objects of type $T1$ should be true for objects of type $T2$.



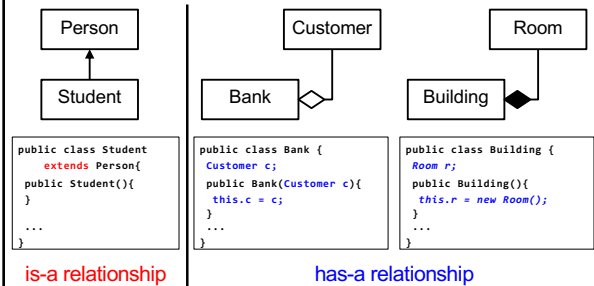
36

OO design principles

- Information hiding (and encapsulation)
- Polymorphism
- Open/closed principle
- Inheritance in Java
- The diamond of death
- Liskov substitution principle
- **Composition/aggregation over inheritance**

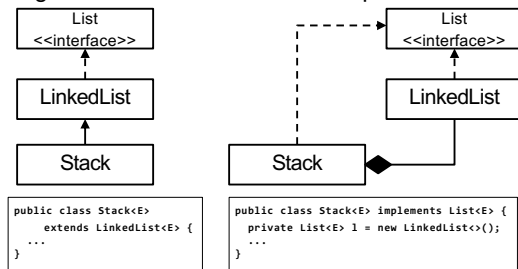
37

Inheritance vs. (Aggregation vs. Composition)



38

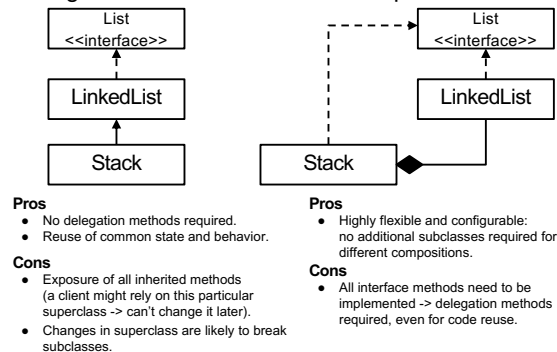
Design choice: inheritance or composition?



Hmm, both designs seem valid – what are pros and cons?

39

Design choice: inheritance or composition?



Composition/aggregation over inheritance allows more flexibility.

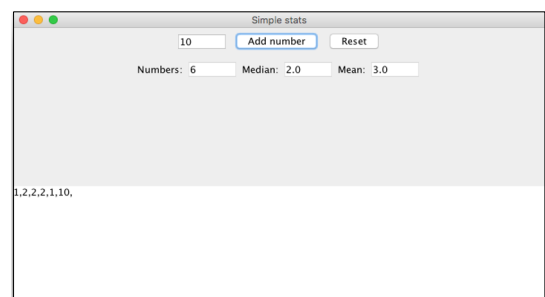
40

Today

- OO design principles
 - Information hiding (and encapsulation)
 - Polymorphism
 - Open/closed principle
 - Inheritance in Java
 - The diamond of death
 - Liskov substitution principle
 - Composition/aggregation over inheritance
- On your own exercise:
 - Code review and (re)design of an MVC application

41

review the code of the following application and practice identifying design flaws



Source code available on the course web site

42

OO design principles: summary

- Information hiding (and encapsulation)
- Open/closed principle
- Liskov substitution principle
- Composition/aggregation over inheritance