CSSE2002/7023

Semester 2, 2021

Programming in the Large

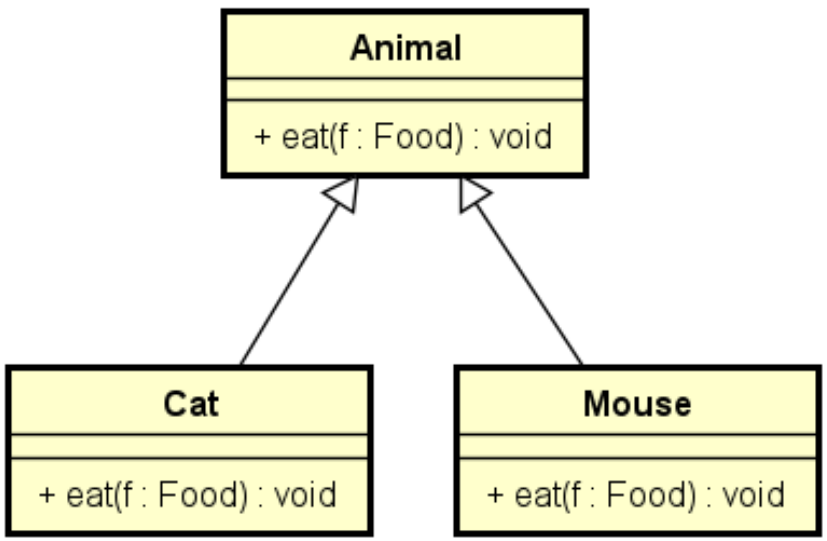
Week 5.2: Substitution Principle

In this Session

- Liskov Substitution Principle
- Specifications and Inheritance

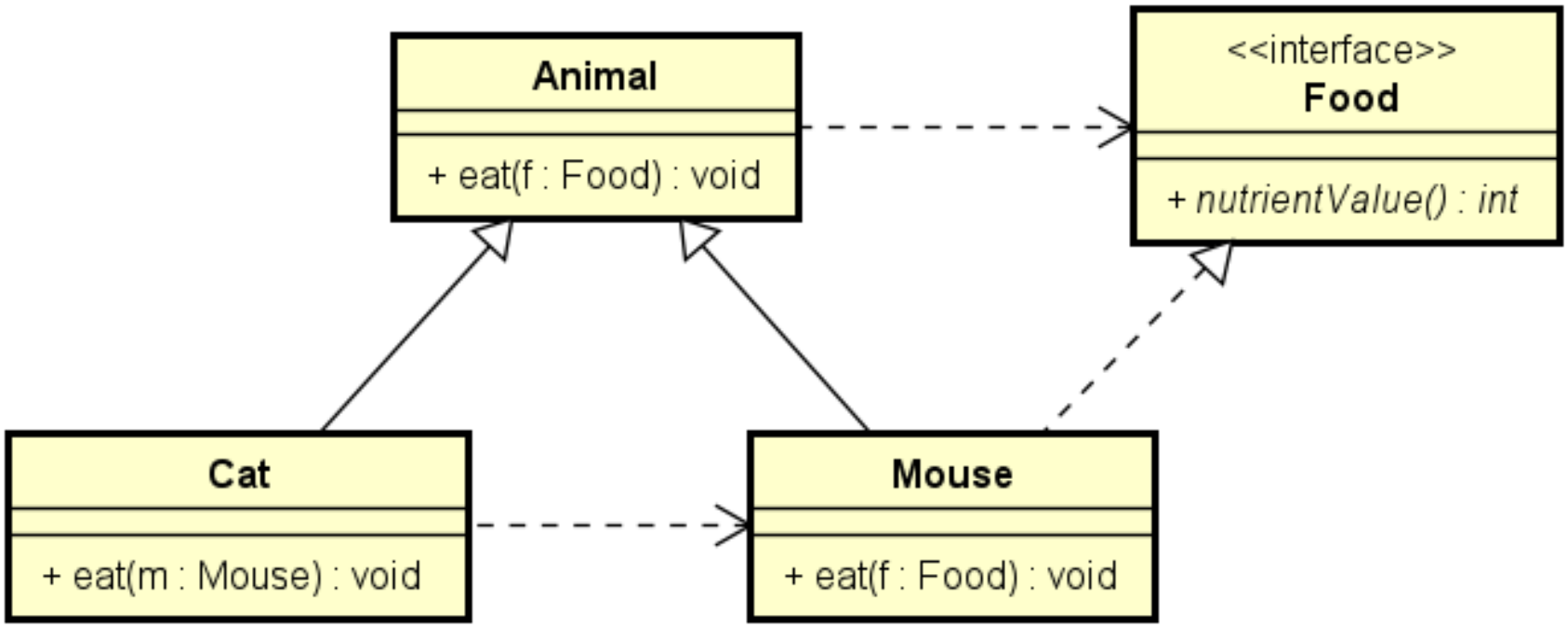
Liskov Substitution Principle

An object of a subclass type can be used wherever an object of a superclass type is expected.




```
Food f = new Food();
List<Animal> l = new ArrayList<>();
l.add(new Cat());
l.add(new Mouse());
for (Animal a : l) {
    a.eat(f);
}
```

Valid Substitution?



Substitution Principle

```
class Parent {
    ...
    char f(int x);
}
```

Suppose:

- precondition (`@require`) for `f` is:
  `x > 0`
- postcondition (`@ensure`) for `f` is:
  `\result > 'A' && \result < 'Z'`

What happens if we override `f` in a subclass of `Parent`?

Even when we change the implementation, we should still follow the contract commitments of the original version.

Substitution Principle

Suppose `Child1 extends Parent`, but `Child1.f()` can deal with negative numbers as well.

What to specify as the precondition of `Child1.f()`?

1. Leave it as `x > 0`?
2. Change it to `x != 0`?

Possible problem with #1:
Confusion for readers about why the limitation is there when the code is more capable.

Substitution Principle – Preconditions

`x > 0` vs `x != 0`?

Consider the set of inputs the Parent precondition allows (`x > 0`), and the set of inputs the Child1 precondition allows (`x != 0`).

Everything in the first set is also in the second set.

The second precondition is "weaker" or "less strict" than the first.

It does not reject any states that the first one would allow.

Remember: having a precondition does not promise that the code will not work if it is not met. It only indicates "we make no promises".

Substitution Principle – Postcondition

What if `Child1.f()` only returned 'K', 'L', or 'M'.

Would

```
\result > 'J' && \result < 'N'
```

be an acceptable postcondition?

Every result still fits within the original postcondition, so yes.

New postcondition is "stronger", in that it is more restrictive.

Substitution Principle

When behaviour is redefined in subclasses:

- Preconditions can be made **weaker**, but not *stronger*.
- Postconditions can be made **stronger**, but not *weaker*.

Stronger, Weaker or Neither?

Is the second condition stronger than the first, weaker than the first, or neither?

`x > 0`	`x > -8`
`x != 0`	`x != 5`
`(x + y) > 7`	`(x + y) > 7 || (x != 0)`
`x instanceof List<String>`	`x instanceof LinkedList<String>`
`(x != 0) && (x + 7) < 5`	`(x + 7) < 4`
`(x + y) > 5`	`(x > 2) && (y > 3)`
`x > 0`	`(x > 0) && (x > 3)`
`(x > y) && (y > 0)`	`x > 1`