Week 11.2 — Some design things

1. Substitution principle
2. Choosing classes
3. Facade

Substitution Principle

OO-specific

Given a class with methods:

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

Suppose:

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

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

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

Substitution Principle

Suppose `Child1 extends Parent` but `Child1.f()` can cope 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 other programmers reading the code about why the limitation is there when the code is more capable.
(Can discuss in comments)

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

Consider the set of inputs the Parent precondition allows and the set of inputs the Child1 precondition allows. 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".

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.
The 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?

work in integer

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

$x > 0$	$x > -8$ weaker
$x! = 0$	$x! = 5$ neither
$(x + y) > 7$	$(x + y) > 7 \mid\mid (x! = 0)$ weaker
`x instanceof List<String>`	`x instanceof LinkedList<String>` stronger
$(x + y) > 5$	$(x > 2)\&\&(y > 3)$ (x+y)>6 stronger
$x > 0$	$(x > y)\&\&(y > 0)$ stronger
$(x > y)\&\&(y > 0)$	$x > 1$ same

Which objects?

What objects and methods should we have?

As a very rough first approximation: Look at the description of the system.

- *Nouns* / noun phrases are candidates to be objects.
- *Verbs* are candidates to be methods. (Attached to which classes?)

Example 1

At a zoo, the keepers hold shows at the various arenas. The shows display the animals demonstating natural behaviours. Exhibits display animals in a more natural environment.
Visitors are not permitted inside shows or exhibits but they are permitted to watch from outside. Carnivores are not to share exhibits with herbivores.
Visitors are not permitted on premises when the zoo is closed.
Staff continue to work.
There may be a treehouse.

Example 1

Nouns = possible classes
Verbs = possible methods
Nouns = possible attributes

At a zoo, the keepers hold shows at the various arenas. The shows display the animals demonstating natural behaviours. Exhibits display animals in a more natural environment.
Visitors are not permitted inside shows or exhibits but they are permitted to watch from outside. Carnivores are not to share exhibits with herbivores.
Visitors are not permitted on premises when the zoo is closed.
Staff continue to work.
There may be a treehouse.

Your choice of classes, methods and attributes will often come back to what your customer wants or needs.

Example 2

In a game of chess there are two players. One player controls the black pieces, the other controls white pieces. The game starts with 16 pieces of each colour on the board.

- 8 pawns
- 2 rooks
- 2 knights
- 2 bishops
- 1 queen
- 1 king

Each type of piece moves and captures differently. When a piece is captured, it is removed from the board.
Each player has a limited amount of time to make their turns.
The game ends when one player puts their opponent's king in checkmate.

Example 2

Nouns = possible classes
Verbs = possible methods
Nouns = possible attributes

In a game of chess there are two players. One player controls the black pieces, the other controls white pieces. The game starts with 16 pieces of each colour on the board.

- 8 pawns
- 2 rooks
- 2 knights
- 2 bishops
- 1 queen
- 1 king

Each type of piece moves and captures differently. When a piece is captured, it is removed from the board.
Each player has a limited amount of time to make their turns.
The game ends when one player puts their opponent's king in checkmate.

Facade

Facade

Facade is an OO software design pattern.

We use it to create a simple facade for *users* to interact with, even if the program they are using is very complex behind the scenes.

- Many classes interacting
- Subclasses with different behaviours (overloading/overriding)
- Different representations of data

From user's perspective, easy-to-use software based on a clear model of interaction.
From designer's perspective, complex and/or well-optimised code.