01 - Design Patterns

Reusable solutions to frequent problems. High-level description of the solution (not code, not algorithm).

Pros:

- Knowledge;
- Tested solutions;
- Common language to communicate efficiently with teammates.

Cons:

- Unjustified use ("If all you have is a hammer, everything looks like a nail");
- Inefficient if not adapted to contexts;
- Most of the time are already implemented.

Sections of pattern description

- Intent: problem, solution (short);
- Motivation: problem, solution (extended);
- Structure: components and how they are related.

SLIDES PROFESSORE:

- Name;
- Problem addressed;
- Context (used to determine applicability);
- Forces (why: constraints, issues solution must address, some may conflict);
- Solution (static and dynamic relationships among pattern components: structure participants, collaboration. Solutions must solve all forces).

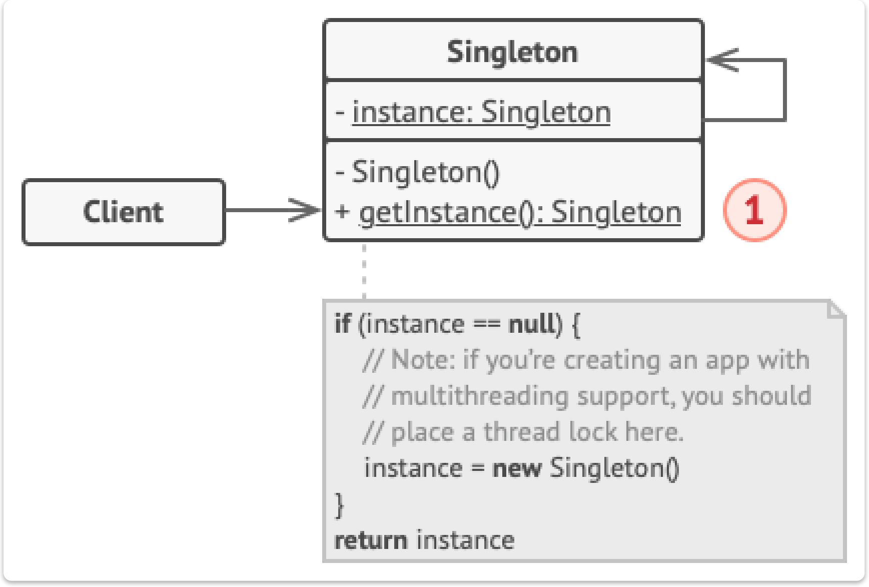
Groups

- Creational: objects creation -> flexibility and reuse existing code;
- Structural: objects assembling -> flexibility and efficiency structure;
- Behavioural: communication between objects.

Example: Singleton

One instance class, multiple pointers

Disable (make private) all object creation method except special creation method that create a new instance only the first time.



```
if (instance == null) {
  // Note: if you're creating an app with
  // multithreading support, you should
  // place a thread lock here.
  instance = new Singleton()
}
return instance
```

In Java, inherit from Singleton.

Design Patterns vs Frameworks

- More abstract;
- Smaller architectural elements (framework contains lot of design patterns, reverse is never true);
- Less specialized (frameworks work on particular application domain).