Object Oriented Programming - Lecture 12

Diana Borza - diana.borza@cs.ubbcluj.ro

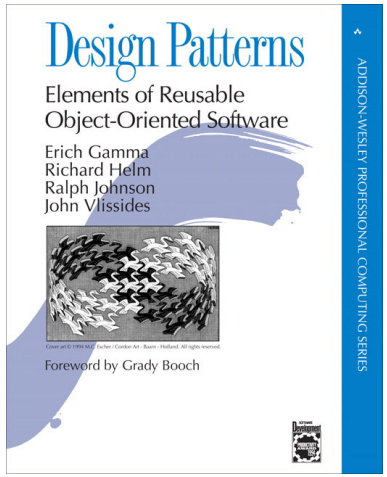
May 2021

Content

- Design patterns

Design patterns - Gang of Four (GOF)

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* (**1994**).



Design patterns - why? I

- When designing something new (a building, a novel, a computer program), designers make certain decisions.
- Experienced designers (architects, writers, software architects) know not to solve a problem from first principles, but to reuse good solutions that have worked in the past.
- Patterns are like templates that can be applied in many different situations.
- Software design patterns are recurring descriptions of classes and communicating objects that are customized to solve a general design problem in a particular context.

Design patterns - why? II

- They are general, flexible, reusable solutions to commonly occurring problems within a given context in software design.
- Object-oriented design patterns show relationships and interactions between classes or objects.
- Christopher Alexander: "*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*".

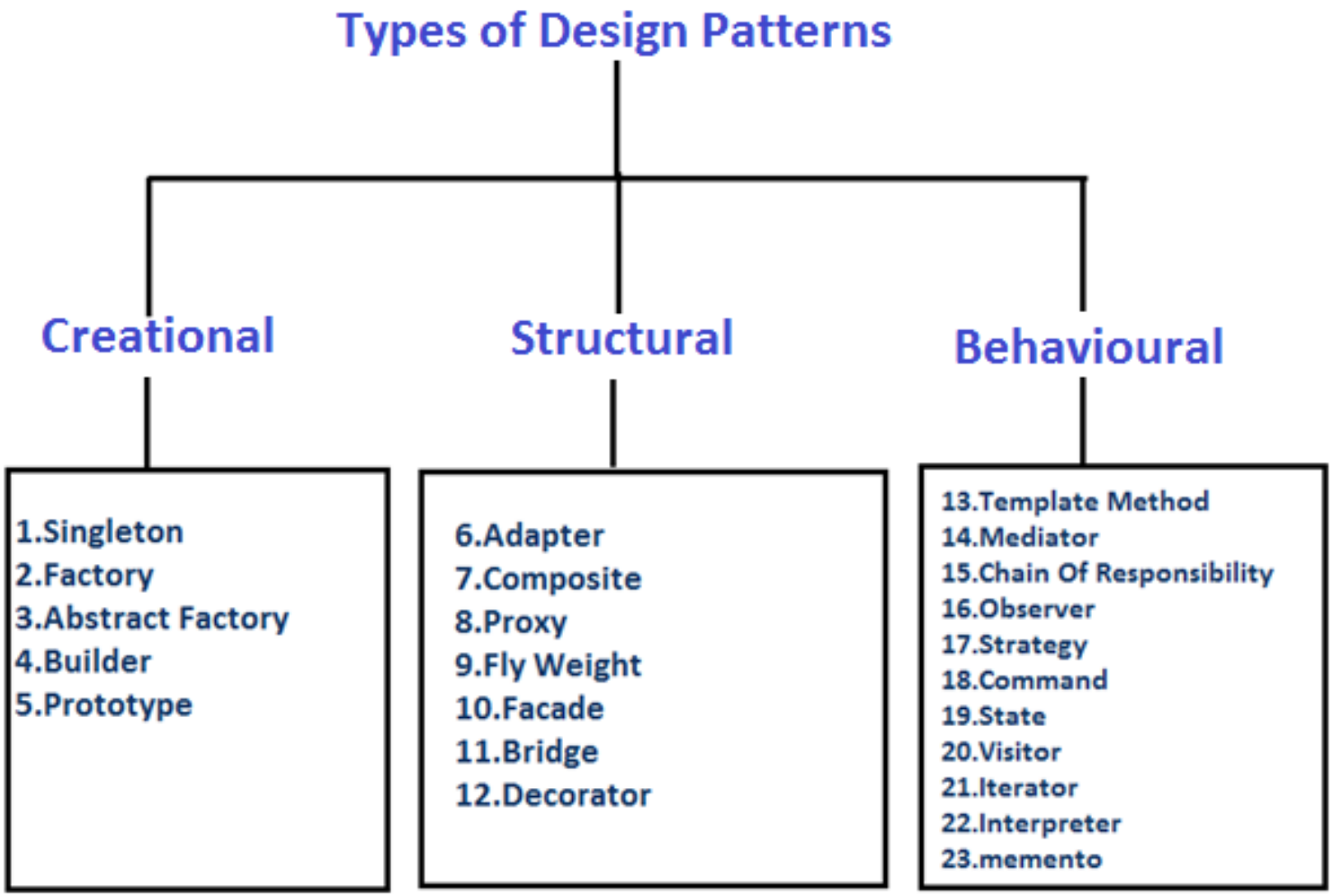
Essential elements of a design pattern I

- **Pattern name**
  - a word or two which describe the design problem, its solution and consequences;
  - it is a part of the software developer vocabulary; "*Finding good names has been one of the hardest parts of developing our catalog*." (GOF)
- **Problem**
  - describes when to apply the pattern;
  - explains the problem and its context;
  - it might include a list of conditions that must be met before it makes sense to apply the pattern.

Essential elements of a design pattern II

- **Solution**
  - describes the elements that make up the design, their relationships, responsibilities and collaborations;
  - provides an abstract description of a design problem and how general arrangement of elements (classes and objects) solves it.
- **Consequences**
  - describe the results and trade-offs (space and time trade-offs) of applying the pattern;
  - may address language and implementation issues as well;
  - they include the pattern's impact on a system's flexibility, extensibility, or portability.

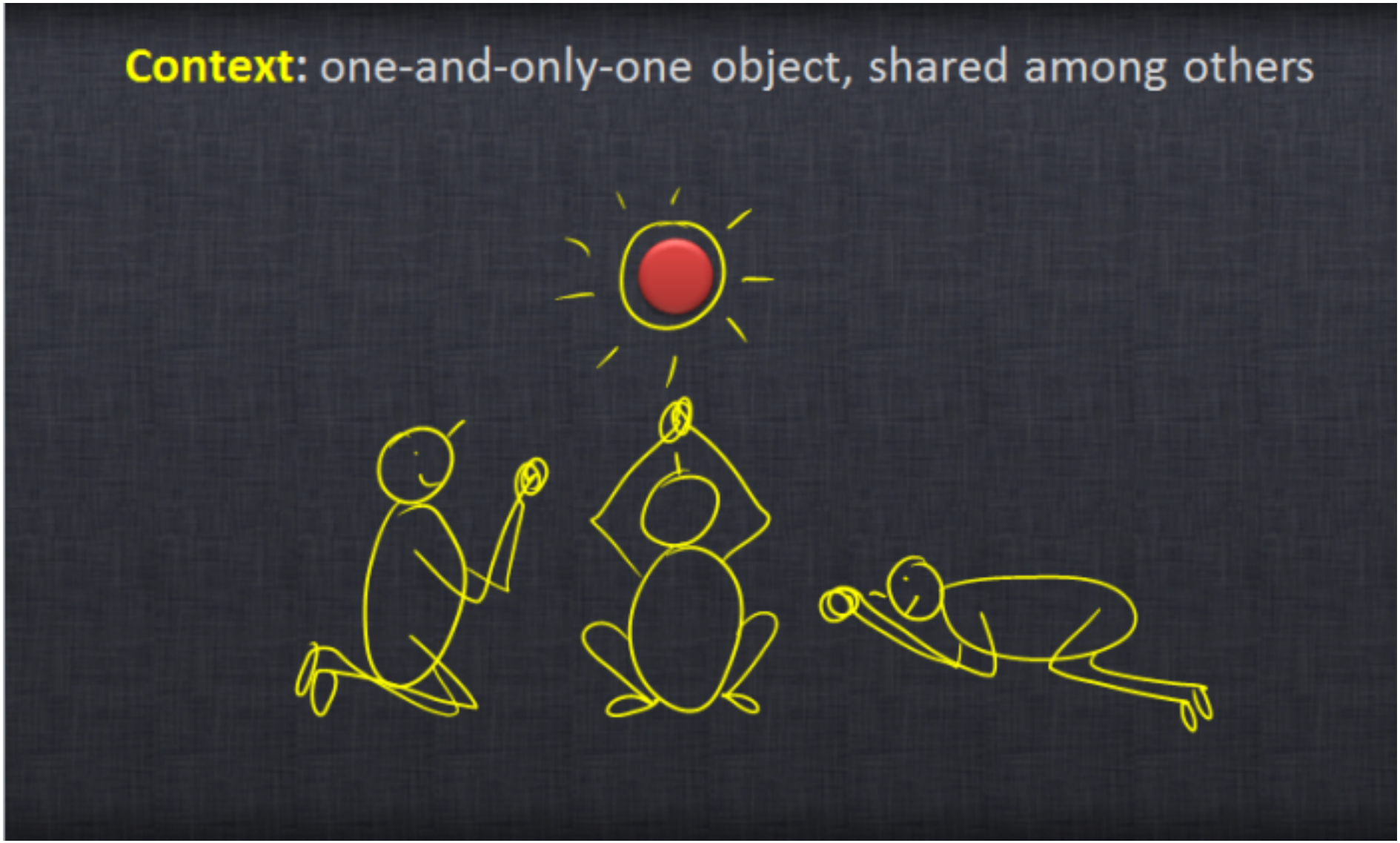
Design patterns taxonomy I



Design patterns taxonomy II

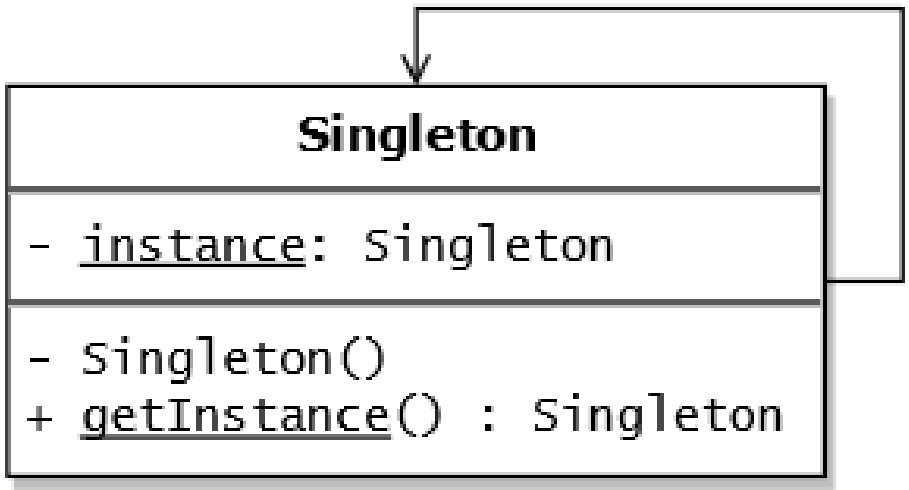
- **Creational design patterns** are all about class instantiation or object creation.
- **Structural design patterns** are about organizing different classes and objects to form larger structures and provide new functionality.
- **Behavioral patterns** are about identifying common communication patterns between objects and realize these patterns.

Singleton design pattern I



Singleton I

- **Problems** solved by singleton:
  1. How can it be ensured that a class has only one instance?
  2. How can the sole instance of a class be accessed easily?
  3. How can a class control its instantiation?
  4. How can the number of instances of a class be restricted?



Singleton II

The key idea in this pattern is to make the class itself responsible for controlling its instantiation (that it is instantiated only once).

- **Solution**
  - Ensure that only one instance of the singleton class ever exists;
  - Provide global access to that instance.
- **Implementation**
  - declaring all constructors of the class to be private;
  - providing a static method that returns a reference to the instance.
- **Consequences**
  - singletons are often preferred to global variables because: they do not pollute the global namespace with unnecessary variables.
  - allow lazy allocation and initialization, whereas global variables in many languages will always consume resources.

Static variables - revisited I

- Static member functions can be used to work with static member variables in the class.
- We use the static keyword to mark variables and functions as static.
- An object of the class is not required to call them.
- Unlike normal member variables, static member variables are shared by all objects of the class.
- Member variables of a class can be made static by using the static keyword.
- Static members exist even if no objects of the class have been instantiated!
  - they are created when the program starts, and destroyed when the program ends.

Static functions - revisited II

- Static members as belonging to the class itself, not to the objects of the class.
- We must explicitly define the static member outside of the class, in the global scope!
  - Do not put the static member definition in a header file!!
- Because static member functions are not attached to an object, they **DON'T have access** to **have NO this pointer**.
- Static member functions can directly access other static members (variables or functions), but not non-static members.
  - non-static members must belong to a class object, and static member functions have no class object to work with.

Singleton sometimes considered an anti-pattern

Issues with singleton:

- introduce global state to the application;
- you cannot completely isolate classes dependent on singletons;
- introduce tight coupling and the class retrieves the instance on its own;
- make unit testing very hard.

Homework

Write a Date class that has the instance variables year, month, and day. It is a requirement that **for one and the same date only one object** (instance) should exist in the data structure.

For example, every object that has a date variable of 10 June 2020 refers to the same object.

The object of e.g. June 10 2020 has variables with the values year = 2020, month = 6 and day = 10.

Inspire yourself from the singleton design pattern implementation.