Design Patterns

Instructors **Battista Biggio**, **Angelo Sotgiu** and **Leonardo Regano**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

Design Patterns

"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." [1]

The idea was adopted by software engineering **Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (GoF) - Design Patterns: Elements of Reusable Object-Oriented Software (1994)**

---

[1] Christopher Alexander, "A Pattern Language", 1977. https://en.wikipedia.org/wiki/Christopher_Alexander

Basic concept of a pattern: **isolate changes in your code**.

Basic concept of a pattern: **isolate changes in your code**.

- Add a layer of abstraction to isolate particular details.
  In this way, you can separate **things that change from things that stay the same**.

Basic concept of a pattern: **isolate changes in your code**.

- Add a layer of abstraction to isolate particular details.
  In this way, you can separate **things that change from things that stay the same**.
- Once you find some part of your program that is likely to change, prevent the changes from propagating through your code.

A pattern has four essential elements:

A pattern has four essential elements:

1. The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two. It makes it easier to think about designs and to communicate them and their trade-offs to others.

A pattern has four essential elements:

1. The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
   It makes it easier to think about designs and to communicate them and their trade-offs to others.
2. The **problem** describes when to apply the pattern.
   It explains the problem and its context.

A pattern has four essential elements:

1. The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
   It makes it easier to think about designs and to communicate them and their trade-offs to others.
2. The **problem** describes when to apply the pattern.
   It explains the problem and its context.
3. The **solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations.
   The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations.
   Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects in our case) solves it.

A pattern has four essential elements:

1. The **pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two.
   It makes it easier to think about designs and to communicate them and their trade-offs to others.
2. The **problem** describes when to apply the pattern.
   It explains the problem and its context.
3. The **solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations.
   The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations.
   Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects in our case) solves it.
4. The **consequences** are the results and trade-offs of applying the pattern.

Categories of patterns

- Architectural patterns
- Design patterns
- Language level patterns. This is the lowest level of the pattern-categories, also known as idioms

Idioms

```
# repeat n times
n = 5
i = 0
while i < n:
  i += 1
  print("DONE")

####################

# stop when input == 0
for _ in iter(int, 1):
  v = int(input())
  if v == 0:
    break
```

```
# repeat n times
n = 5
for _ in range(n):
  print("DONE")

####################

# stop when input == 0
flag = True
print("0 -> exit")
while (flag):
  v = int(input())
  flag = v != 0
```

Classifying Design Patterns

The Design Patterns book [Gamma et al.] discusses 23 different patterns, classified under three purposes.

Classifying Design Patterns

The Design Patterns book [Gamma et al.] discusses 23 different patterns, classified under three purposes.

- **Creational**: how an object can be created. This often involves isolating the details of object creation, so your code doesn't change when you add a new type of object.

Classifying Design Patterns

The Design Patterns book [Gamma et al.] discusses 23 different patterns, classified under three purposes.

- **Creational**: how an object can be created. This often involves isolating the details of object creation, so your code doesn't change when you add a new type of object.
- **Structural**: designing objects to satisfy particular project constraints. These work with how objects are connected with other objects to ensure that changes in the system don’t require changes to those connections.

Classifying Design Patterns

The Design Patterns book [Gamma et al.] discusses 23 different patterns, classified under three purposes.

- **Creational**: how an object can be created. This often involves isolating the details of object creation, so your code doesn't change when you add a new type of object.
- **Structural**: designing objects to satisfy particular project constraints. These work with how objects are connected with other objects to ensure that changes in the system don’t require changes to those connections.
- **Behavioral**: objects that handle particular types of actions within a program. It is a way to encapsulate algorithms that you want to perform.