Unidade Curricular

“Padrões e Desenho de Software”

#07 – Structural Patterns (1)

António José Ribeiro Neves

an@ua.pt

https://www.ua.pt/pt/uc/12275

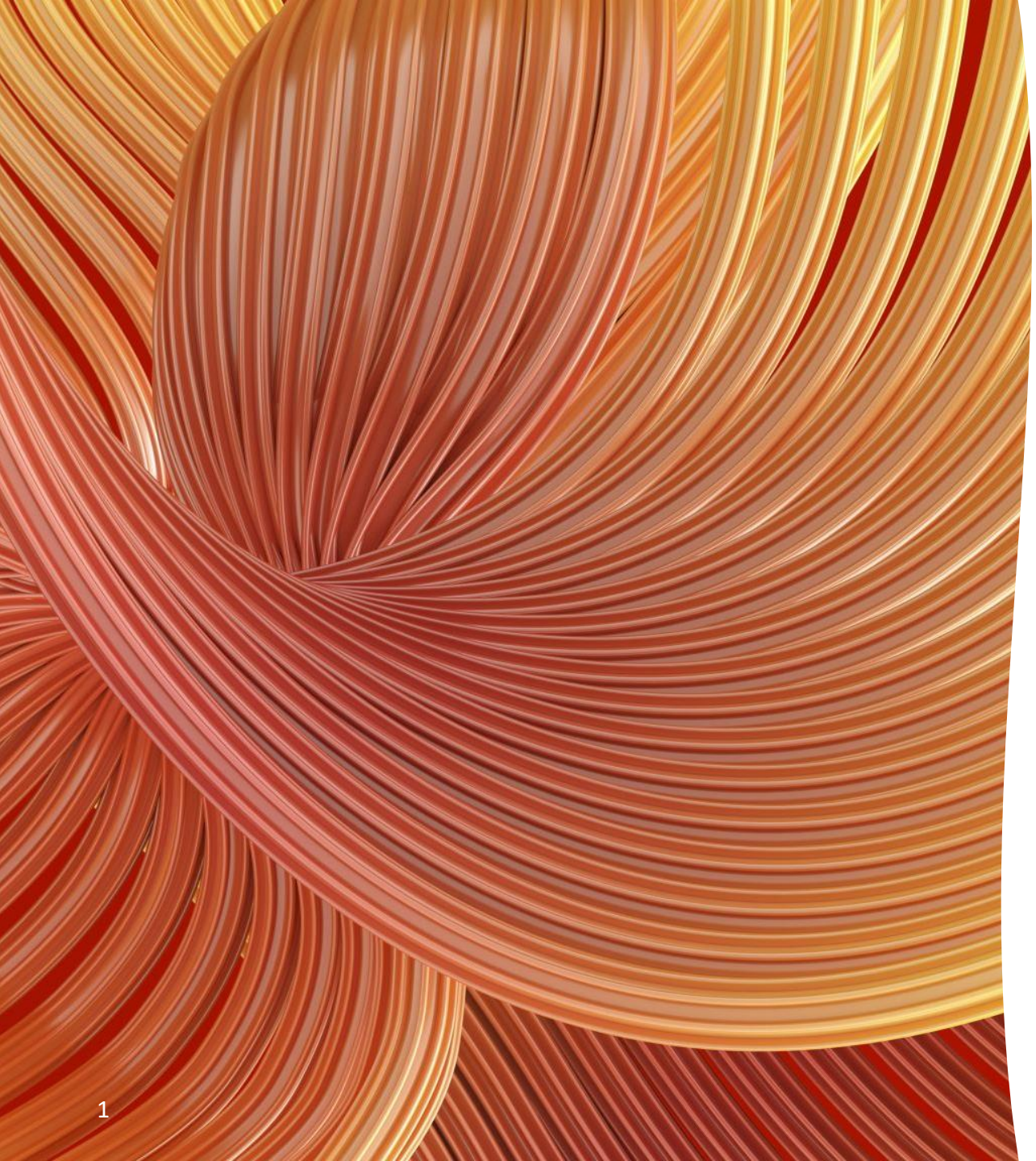
universidade de aveiro

ECIU university

deti

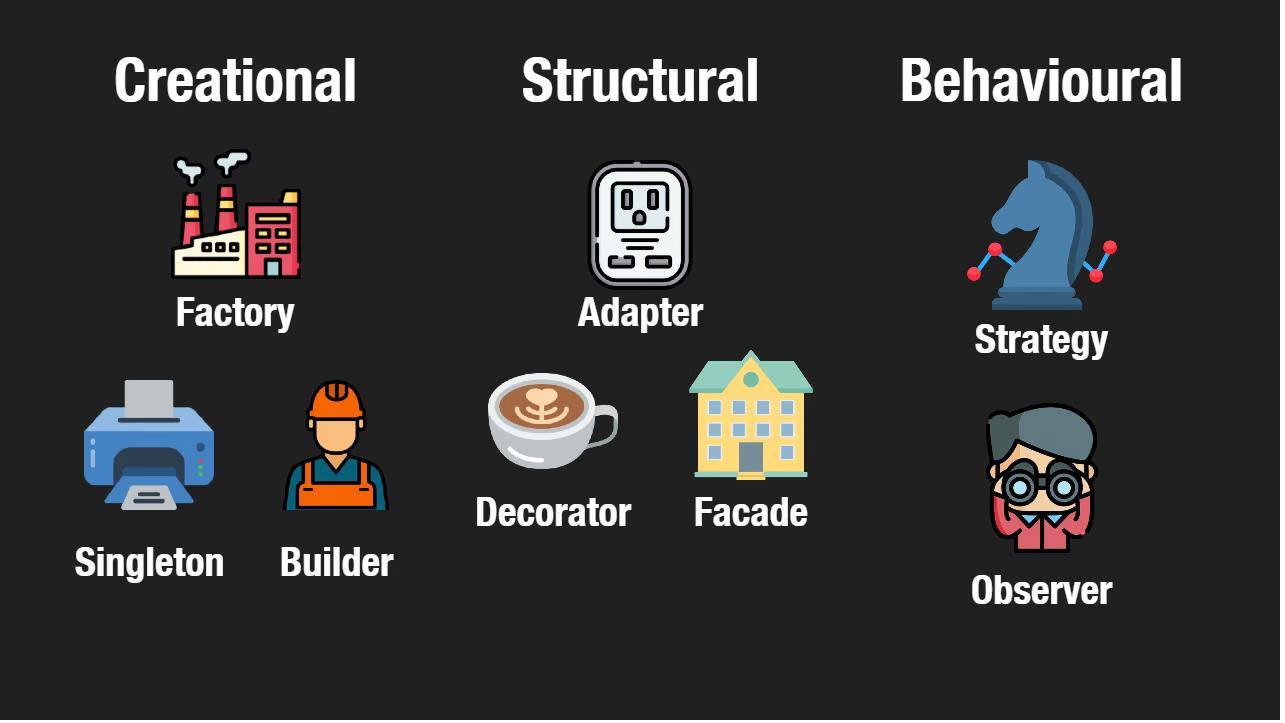
IEETA

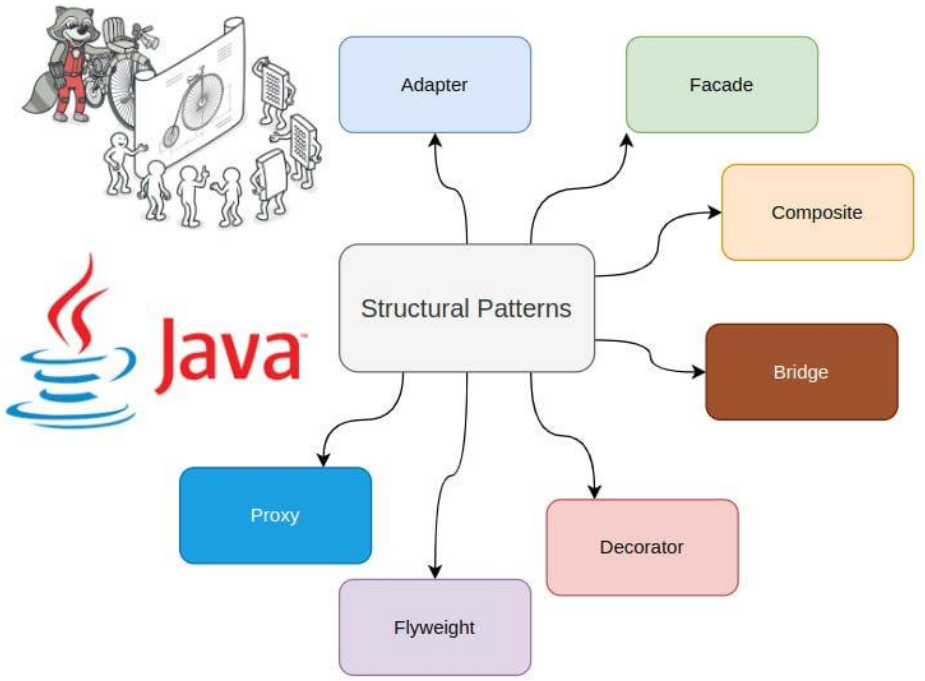
LASI



Outline

- Builder demonstration
- Quiz
- Adapter Pattern
- Bridge Pattern
- Presentations





Structural Design Patterns

- **40 minutes** to explore the the Adapter OR the Bridge design patterns.
- Join a group with at most 10 students. Elect the leader! She/He will present later the work.
- This is a field work, You can go outside and find a comfortable place to discuss the topic.
- You have 10 minutes to present your findings later – you can talk and use the white board.
- Deliver the result in a digital file (text, presentation, …)!

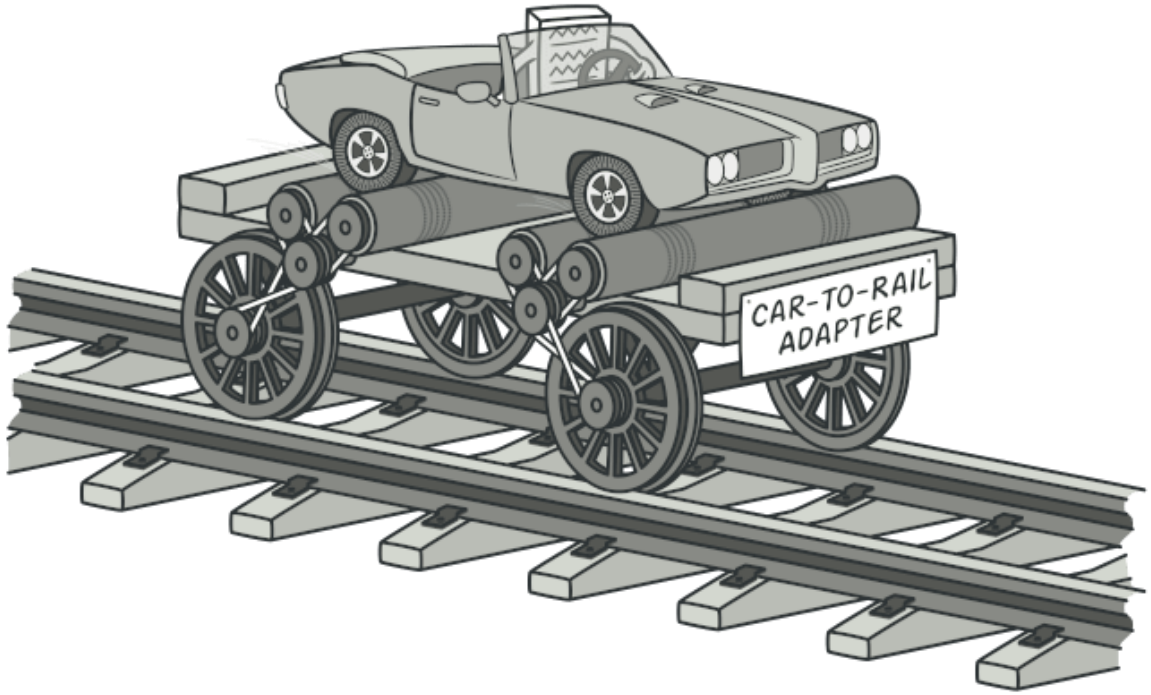
Time for the colleagues ☺

Creational vs Structural

Aspect	Creational Patterns	Structural Patterns
Focus	How objects are **created**	How classes and objects are **organized and combined**
Main goal	Control the **instantiation process** and hide complexity	Build **flexible structures** and enable collaboration between components
When used	Before the object exists	After the object exists
Typical benefit	Encapsulation of creation logic	Reusability and low coupling between parts
Mental model	“How the cake is baked”	“How the cakes are arranged in the showcase”

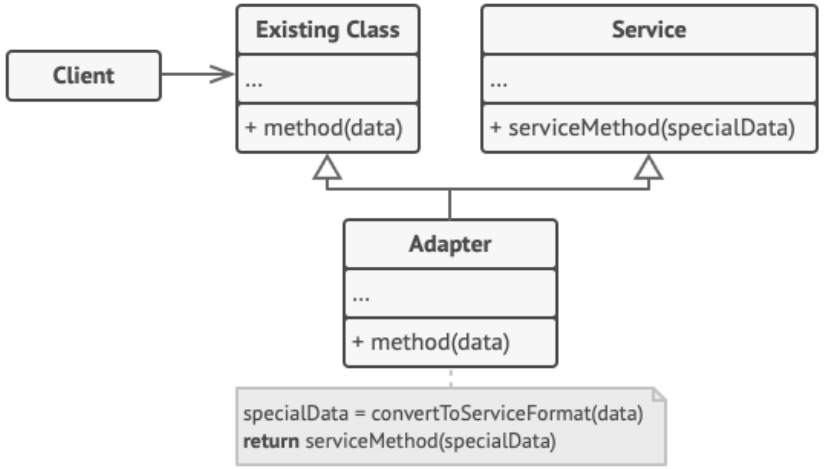
Adapter Design Pattern

- Allows objects with incompatible interfaces to collaborate
- A special object that converts the interface of one object so that another object can understand it
- An adapter wraps one of the objects to hide the complexity of conversion happening behind the scenes
- Adapters can not only convert data into various formats but can also help objects with different interfaces collaborate

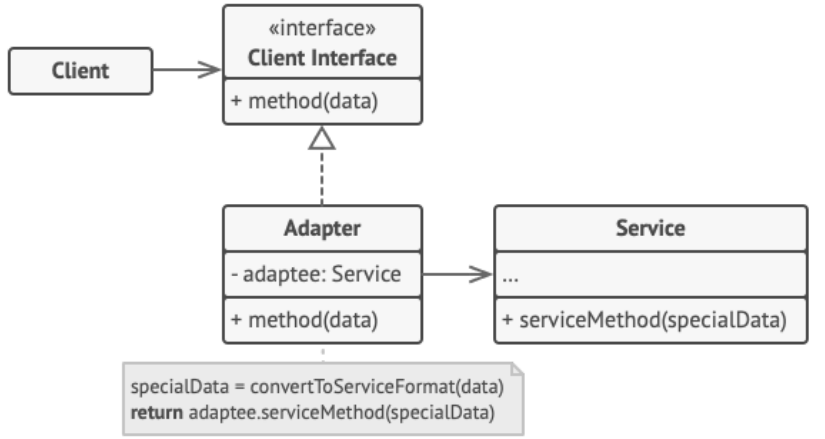


Adapter Design Pattern

- Object adapter
    - This implementation uses the object composition principle: the adapter implements the interface of one object and wraps the other one. It can be implemented in all popular programming languages.

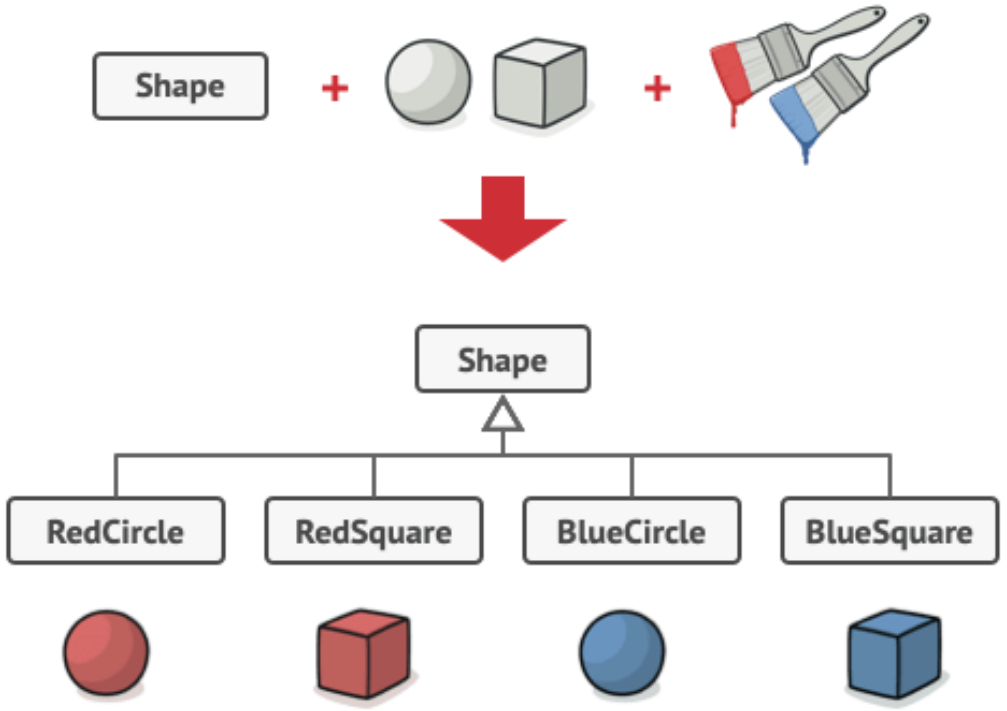


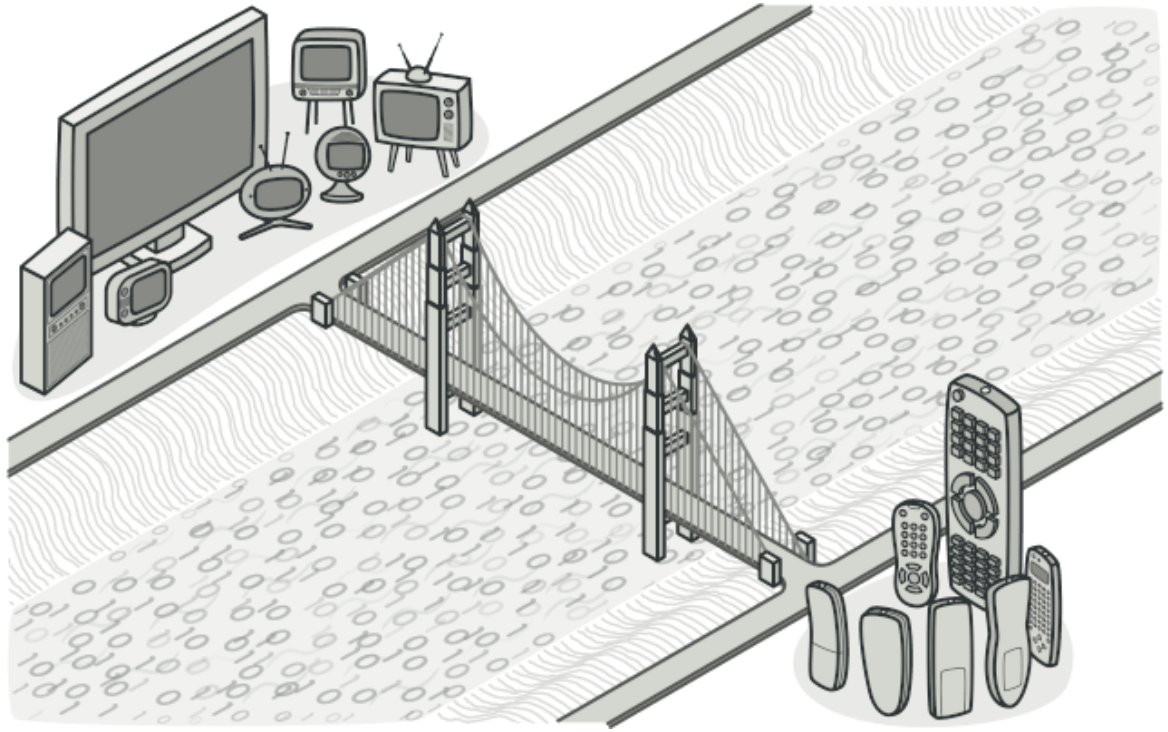
- Class adapter
    - This implementation uses inheritance: the adapter inherits interfaces from both objects at the same time. Note that this approach can only be implemented in programming languages that support multiple inheritance, such as C++.



Bridge Design Pattern

- Lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other





Bridge Design Pattern

Adding new shape types and colors to the hierarchy will grow it exponentially.

To add a triangle shape you'd need to introduce two subclasses, one for each color. And after that, adding a new color would require creating three subclasses, one for each shape type.

This problem occurs because we're trying to extend the shape classes in two independent dimensions: by form and by color. That's a very common issue with class inheritance.

The Bridge pattern attempts to solve this problem by switching from inheritance to the object composition.

You extract one of the dimensions into a separate class hierarchy, so that the original classes will reference an object of the new hierarchy, instead of having all of its state and behaviors within one class.