



# Unidade Curricular

## “Padrões e Desenho de Software”

### #05 – Creational Patterns (1)

---

António José Ribeiro Neves

[an@ua.pt](mailto:an@ua.pt)

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



universidade  
de aveiro



IEETA





# Outline

---

## Creational Design Patterns

---

## Factory Pattern

---

## Practical work

---

## Abstract Factory Pattern

## Creational



Factory



Singleton



Builder

## Structural



Adapter



Decorator



Facade

## Behavioural

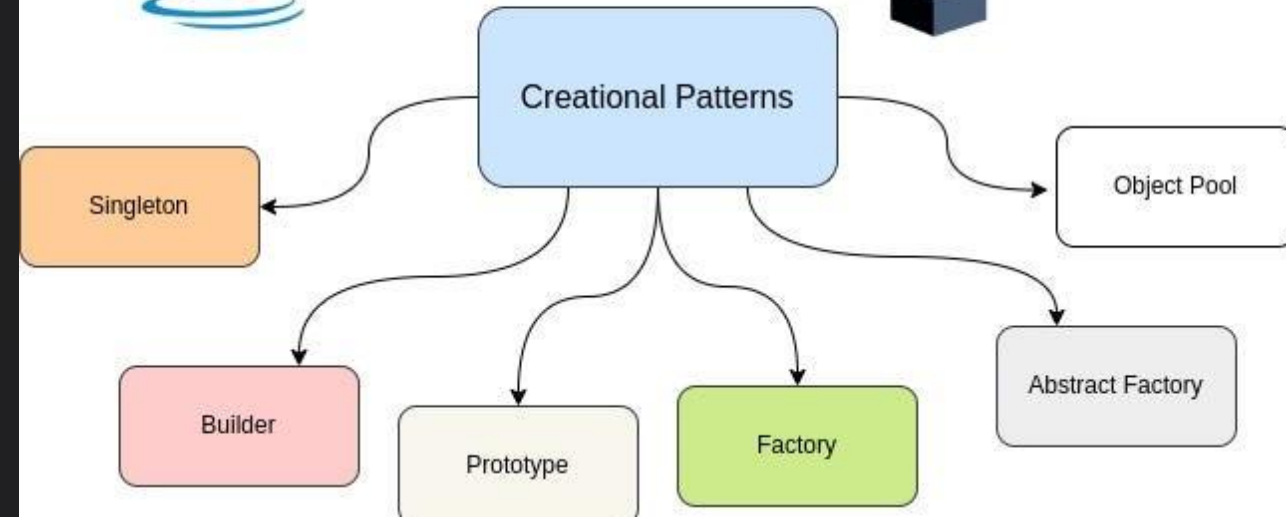


Strategy



Observer

# Creational Design Patterns



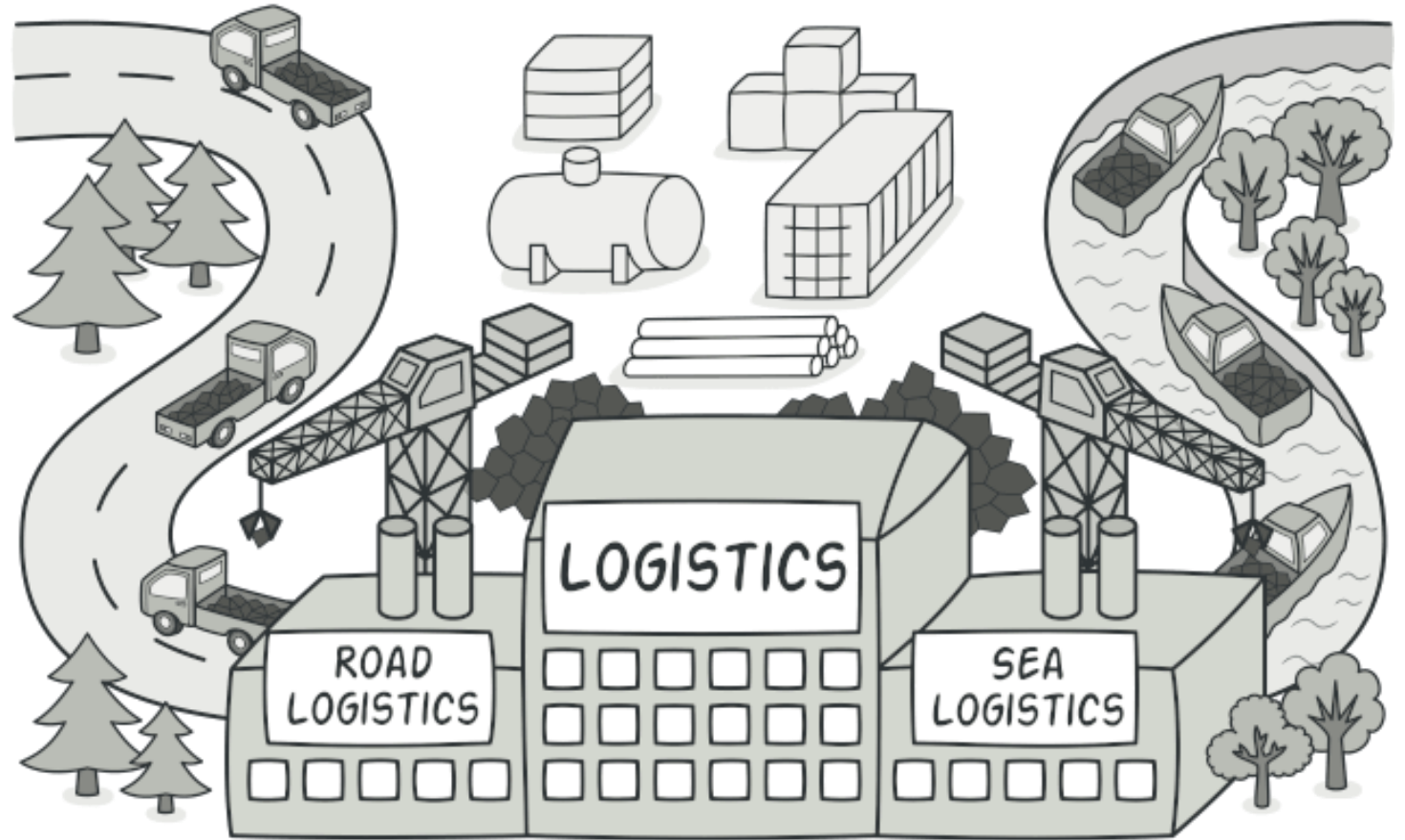
- **20 minutes** to explore the Factory Design Pattern and answer the questions in the link:

<https://forms.gle/SQP6aANqZcxrVE6DA>

# Factory Pattern Overview

---

- Definition: The Factory pattern is a creational design pattern that provides an interface for creating objects without specifying their concrete classes.
- Purpose: Encapsulates object creation logic, promotes loose coupling, and simplifies object instantiation.



# Problem Statement

---

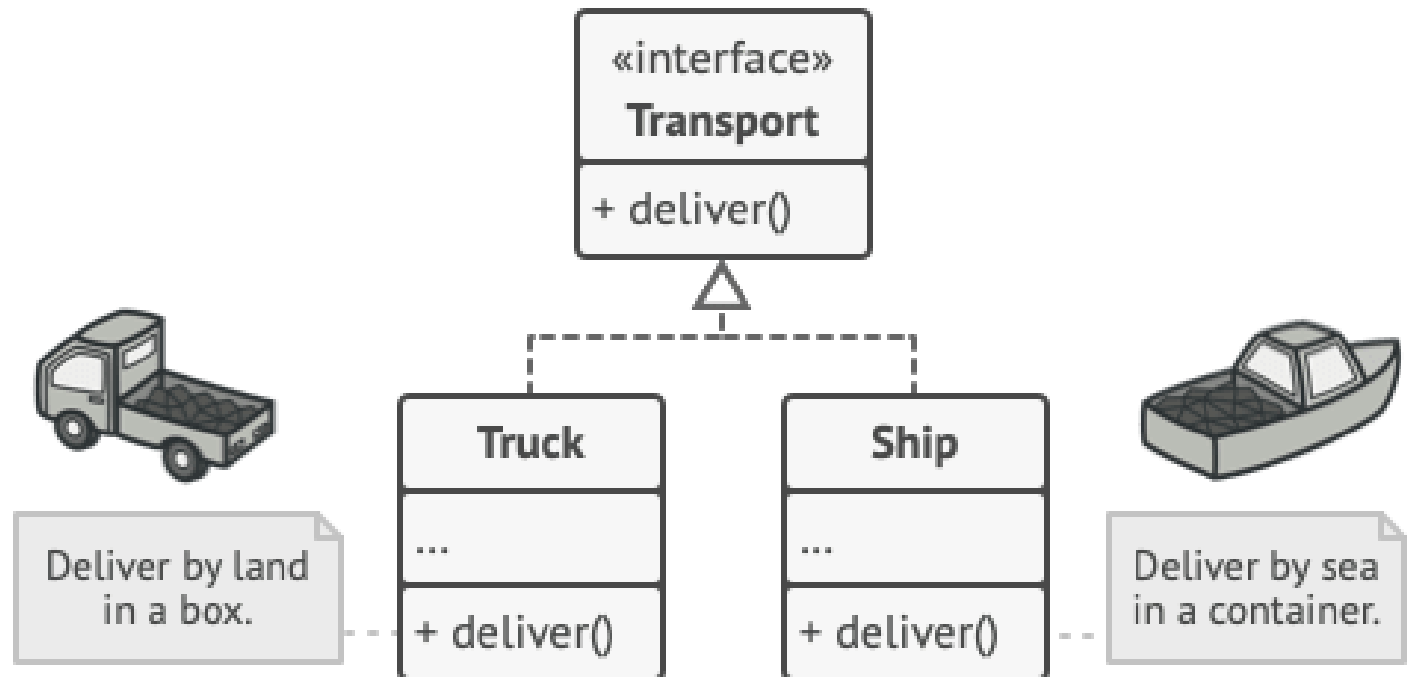
- Difficulty in managing object creation logic directly within client code.
- Tight coupling between client code and concrete classes.
- The new operator considered harmful.



# Solution: Factory Pattern

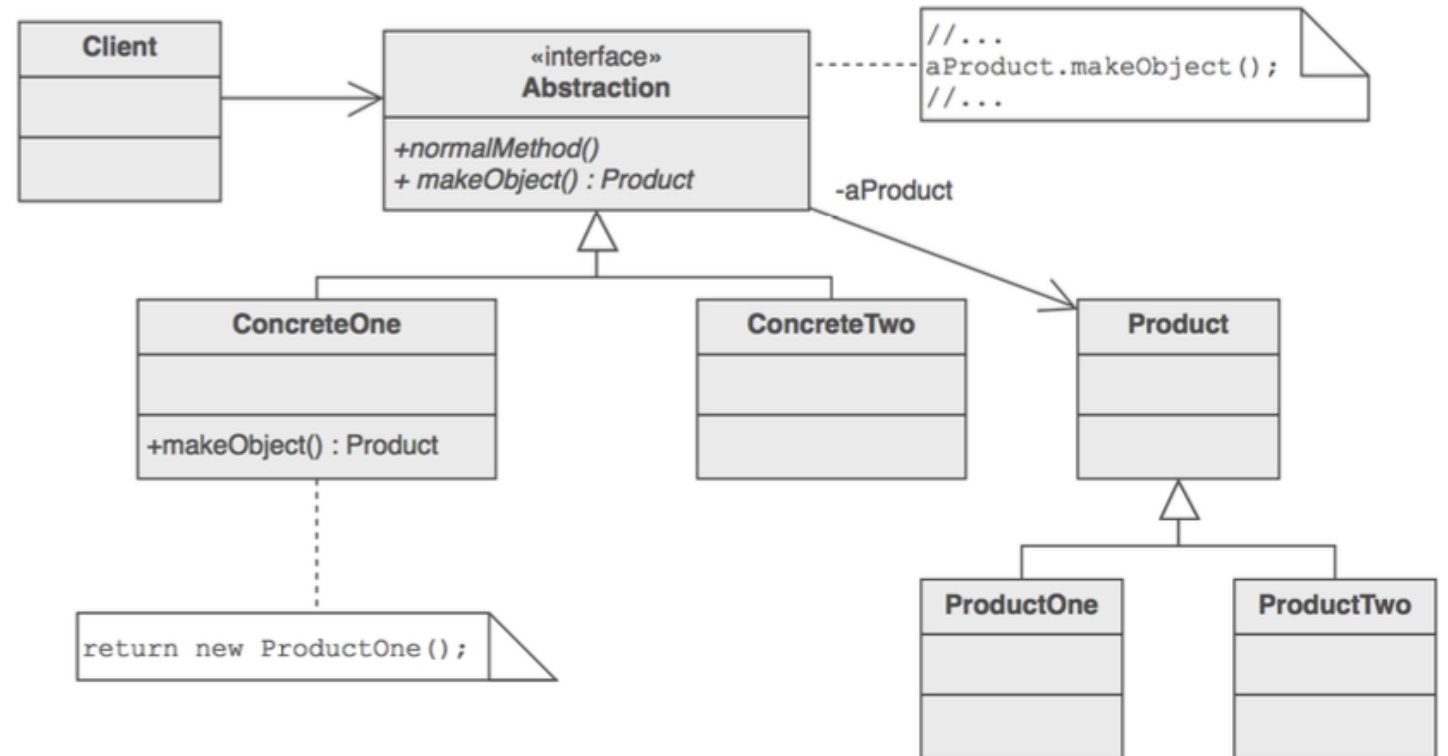
---

- Provides a separate factory class responsible for object creation.
- Clients interact with the factory interface to create objects without knowledge of their concrete classes.

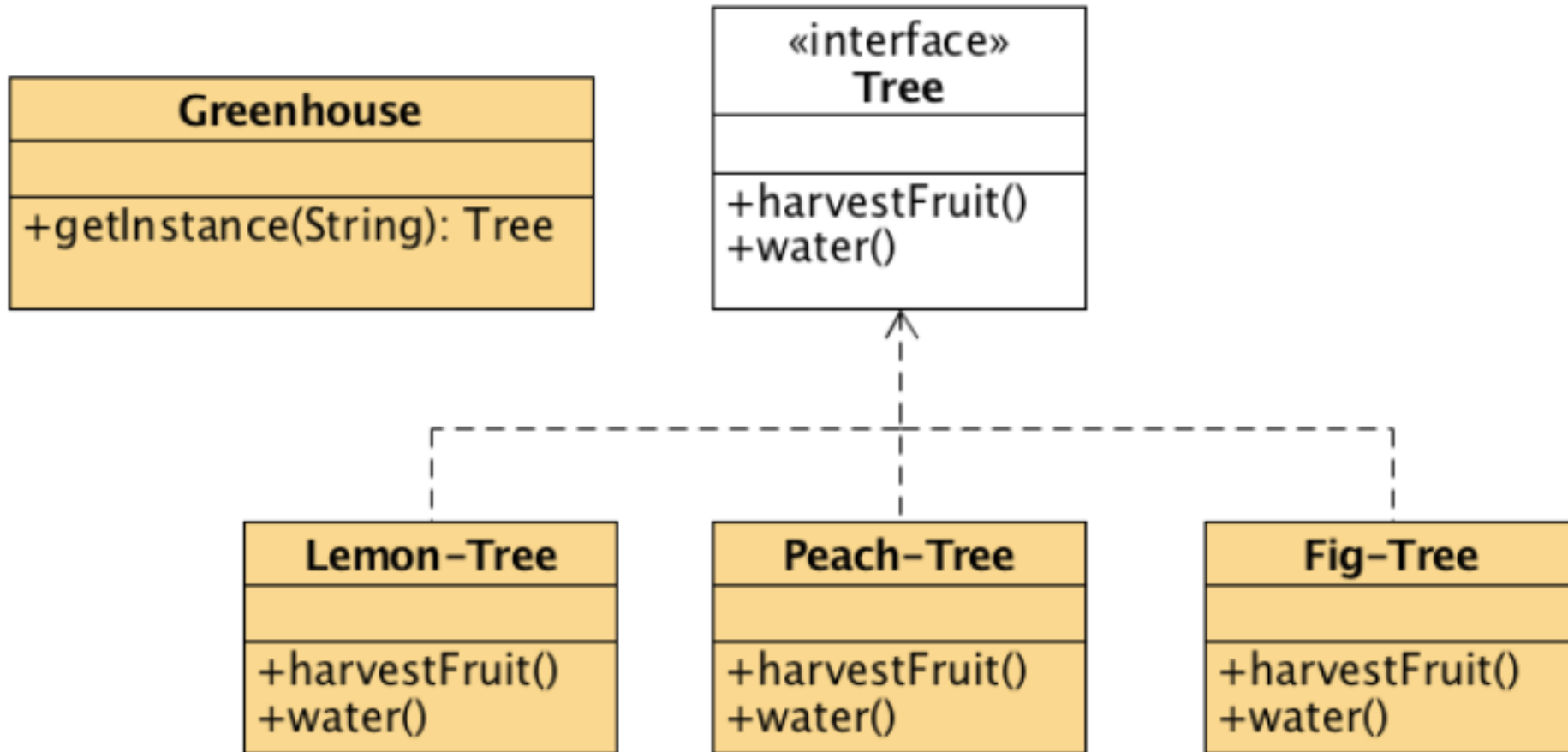


# Structure of Factory Pattern

- Factory Interface: Defines methods for creating objects.
- Concrete Factory Classes: Implement the factory interface and provide specific implementations for object creation.
- Client: Requests objects from the factory interface.



# Example



# Benefits of Factory Pattern

---

- Encapsulates object creation logic, promoting code reuse and maintainability.
- Reduces coupling between client code and concrete classes, facilitating easier changes and enhancements.
- Supports the open/closed principle by allowing new object types to be added without modifying existing code.
- Real-World Applications:
  - GUI frameworks use the Factory pattern for creating UI components
  - Game development frameworks use the Factory pattern for creating game objects and entities
  - JDK itself (where `getInstance()` method is present – ex. `Calendar`)

## Exercise POO (last year)

Implement a simple Shape factory that can create different types of shapes, such as:

- Circles (with a radius)
- Rectangles (two dimensions)
- Triangle (three sides)
- Methods for the area and perimeter

- **30 minutes** to solve this problem and submit the code in the elearning:

`https://elearning.ua.pt/mod/assign/view.php?id=1406268`

Let's take a short break  
**10 Minutes**

You are free to go grab  
a coffee, water, etc.

But... 10 minutes **is 10 minutes** (600 seconds, **not 601 seconds!**)



## Creational



Factory



Singleton



Builder

## Structural



Adapter



Decorator



Facade

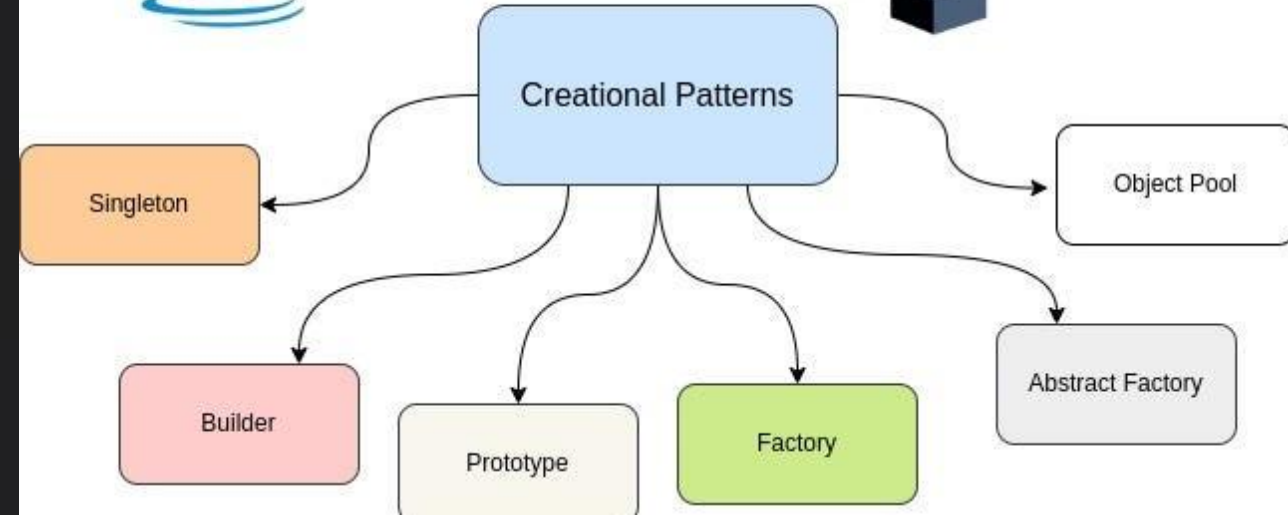
## Behavioural



Strategy



Observer

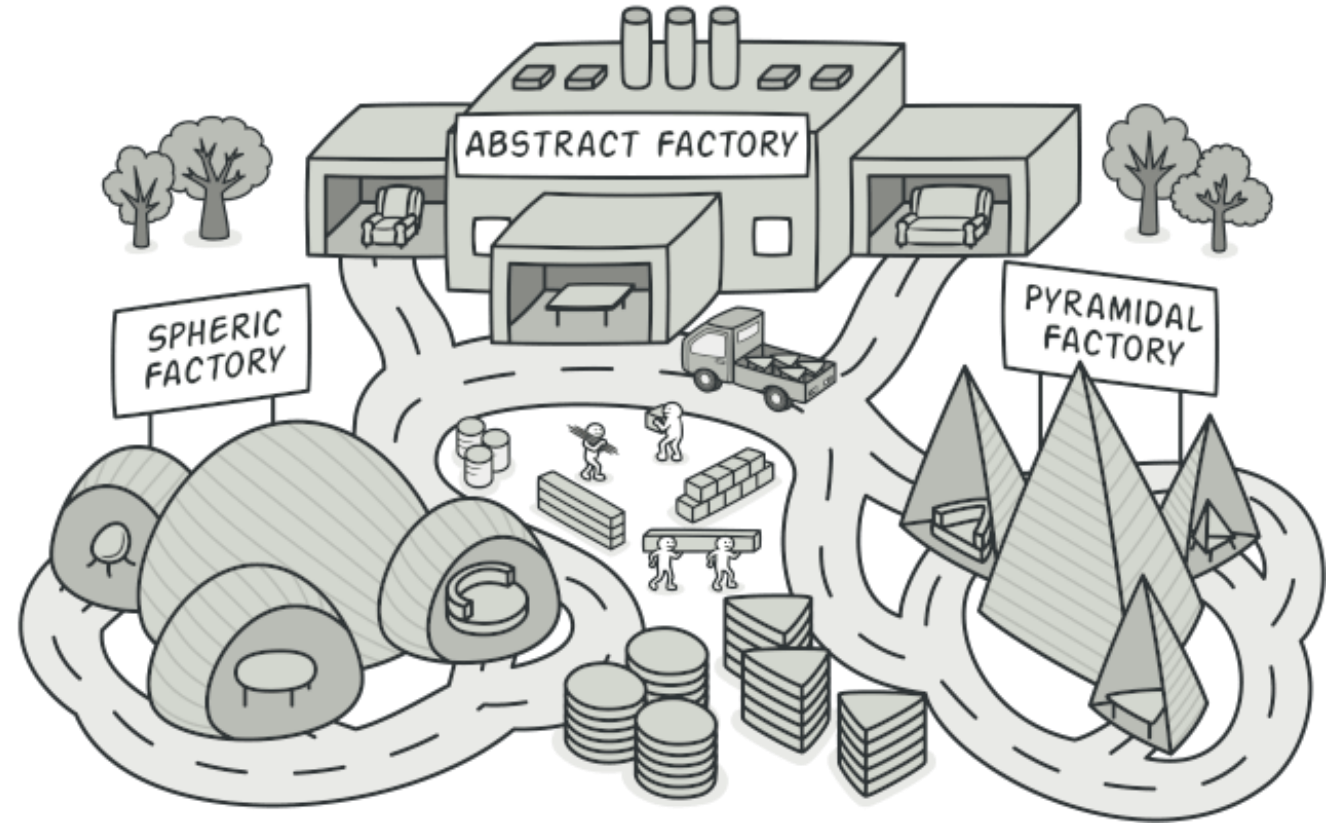


# Creational Design Patterns

- **20 minutes** to explore the Abstract Factory Design Pattern and answer the questions in the link:

<https://forms.gle/sFapkipBS6merdUt7>

# Abstract Factory Pattern Overview

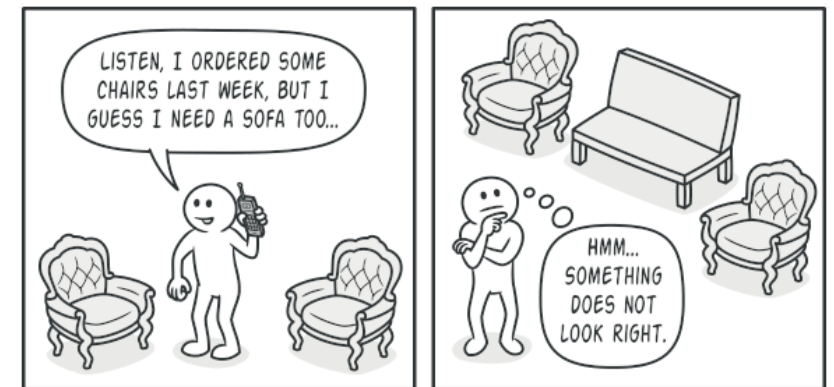
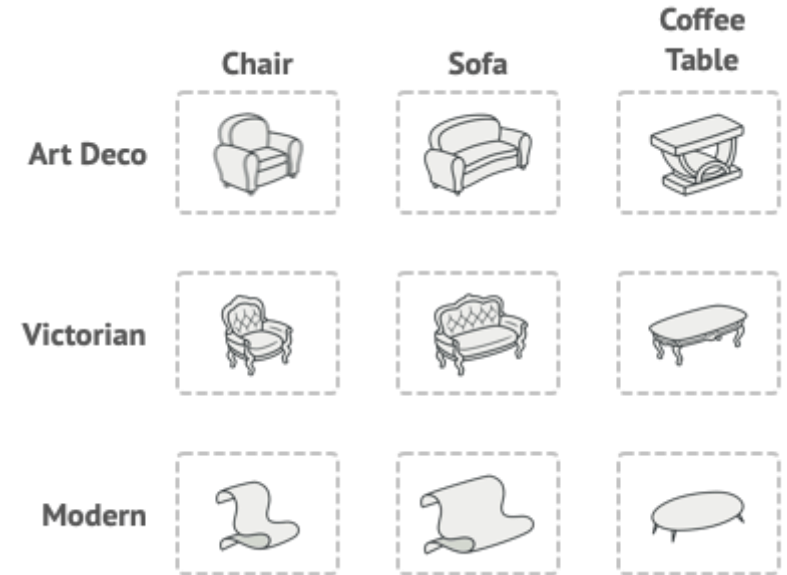


- Definition: The Abstract Factory pattern is a creational design pattern that provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- Purpose: Encapsulates object creation logic, promotes loose coupling, and ensures compatibility between related object families.

# Problem Statement

---

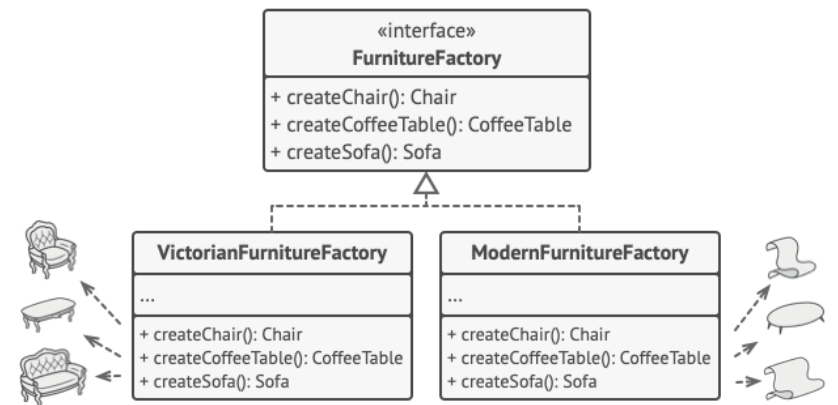
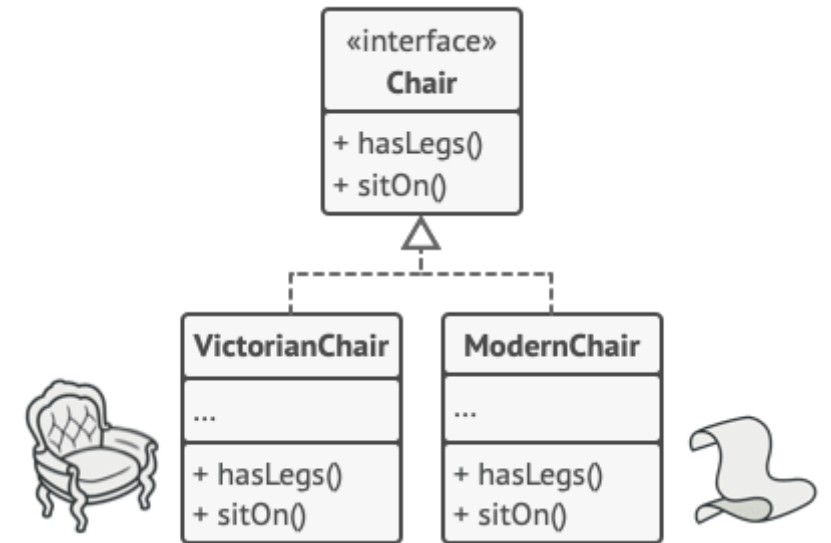
- Difficulty in managing object creation logic for families of related objects.
- Tight coupling between client code and concrete classes of object families.



# Solution: Abstract Factory Pattern

---

- Provides an abstract factory interface for creating families of related objects.
- Concrete factory classes implement the abstract factory interface to create specific families of objects.
- Clients interact with the abstract factory interface to create objects without knowledge of their concrete classes.



# Structure of Abstract Factory Pattern

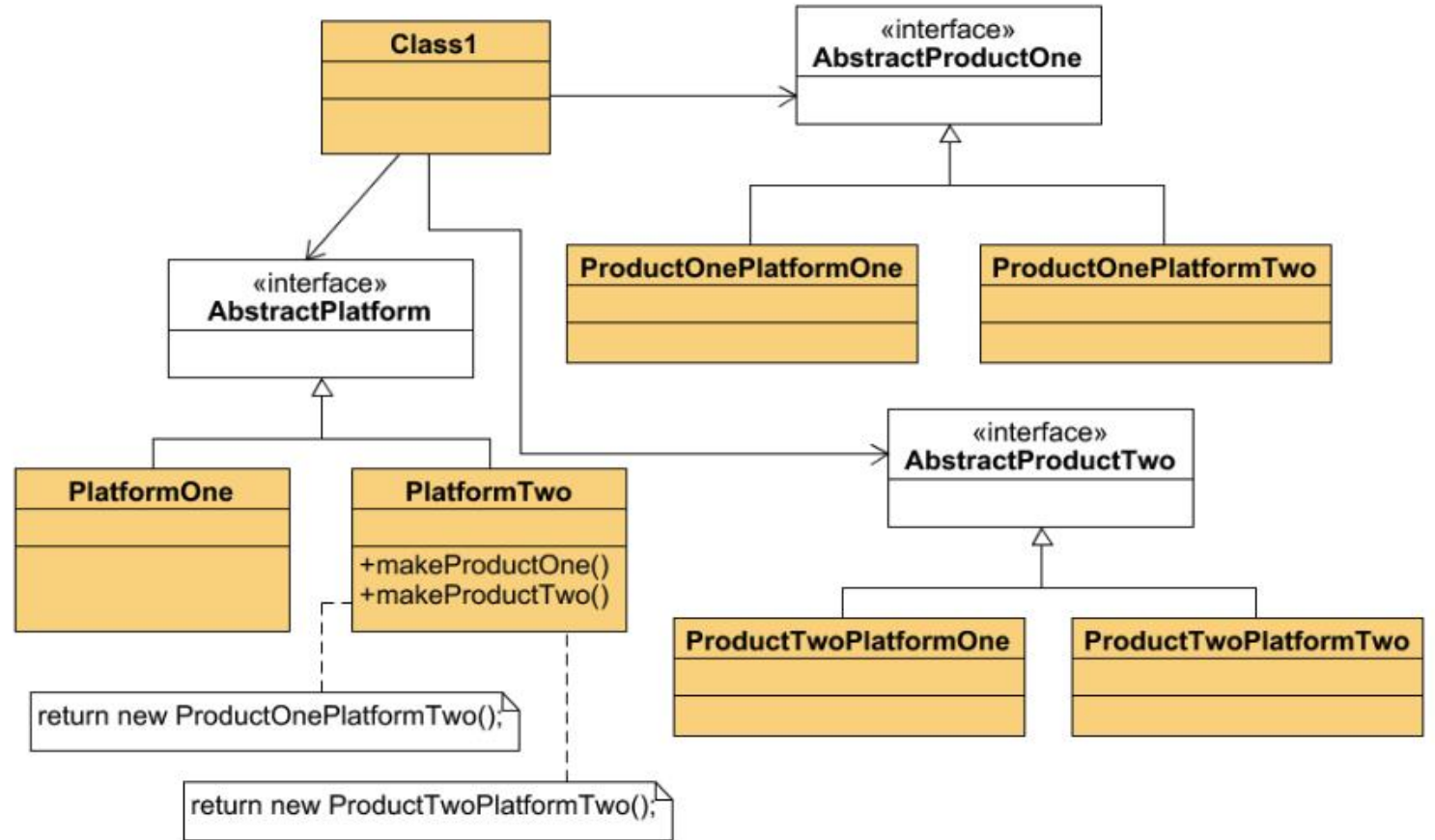
---

- Abstract Factory Interface: Defines methods for creating families of related objects.
- Concrete Factory Classes: Implement the abstract factory interface and provide specific implementations for creating families of objects.
- Client: Requests objects from the abstract factory interface.

```
public class TestJuiceFactory {  
    Run | Debug  
    public static void main(String[] args) {  
        JuiceFactory metalFactory = new MetalJuiceFactory();  
        JuiceFactory plasticFactory = new PlasticJuiceFactory();  
        JuiceFactory glassFactory = new GlassJuiceFactory();  
  
        OrangeJuice metalOrangeJuice = metalFactory.makeOrangeJuice();  
        metalOrangeJuice.drink();  
  
        AppleJuice plasticAppleJuice = plasticFactory.makeAppleJuice();  
        plasticAppleJuice.drink();  
  
        LemonJuice glassLemonJuice = glassFactory.makeLemonJuice();  
        glassLemonJuice.drink();  
    }  
}
```

```
Drinking Orange Juice from a Metal Container  
Drinking Apple Juice from a Plastic Container  
Drinking Lemon Juice from a Glass Container
```

# Example



# Benefits of Abstract Factory Pattern

---

- Encapsulates object creation logic for families of related objects, promoting code reuse and maintainability.
- Reduces coupling between client code and concrete classes of object families, facilitating easier changes and enhancements.
- Ensures compatibility between related object families by providing a common interface for object creation.
- Examples:
  - GUI frameworks use the Abstract Factory pattern for creating platform-specific UI components.
  - Database libraries use the Abstract Factory pattern for creating database connections and queries specific to different database vendors.