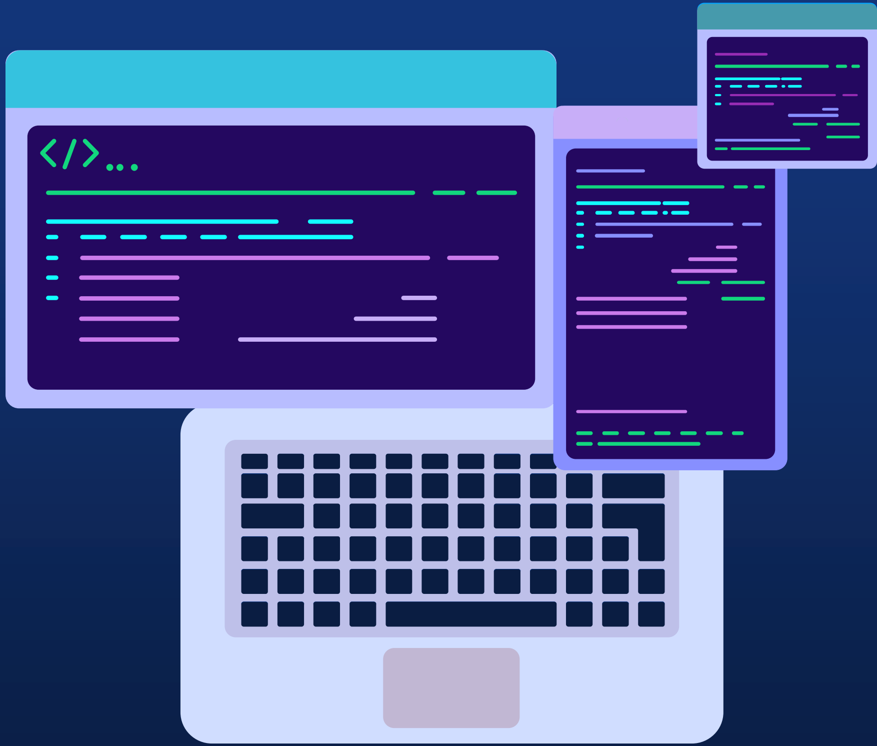
Software Design and Important concepts



Mentor: Einar Rocha

CONTENT

01
OOP Pillars

Inheritance, Polymorphism
Encapsulation, Abstraction

02
Clean Code

Meaningful Names,
Functions, Unit test
Code Smells…

03
SOLID

Single Responsiblity
Open closed
Liskov Substitution
Interface Segregation
Dependency Inversion

04
Design patterns

Singleton, Factory Method
Strategy, Observer
Builder…

04

Design patterns

Agenda

Introduction	What are?, Why?, How to select?
Abstract Factory	Example...
Factory Method	Example...

What are Design Patterns

- ✓ Set of solutions already written by some of the advanced and experienced developers
- ✓ Patterns are not complete code, but it can use as a template which can be applied to a problem

Elements

- Pattern name
- The problem
- The solution
- The results and consequences

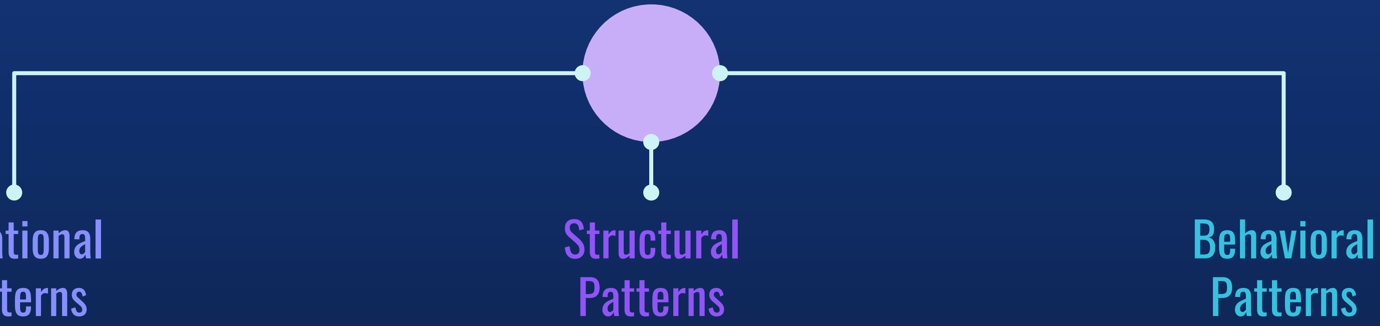
Why use them

- Flexibility
- Reusability
- Shared Vocabulary
- Capture best practices

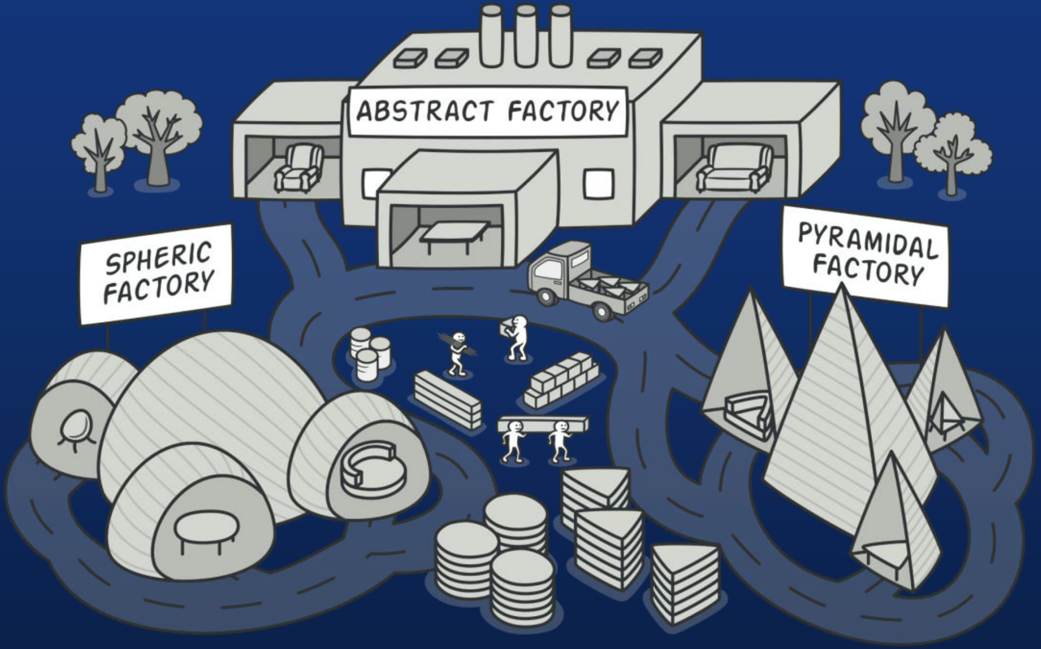
How to select and use one

- ✓ Good knowledge of each one
- ✓ Identify the kind of design problem you are facing.

Categorization of patterns



Abstract Factory



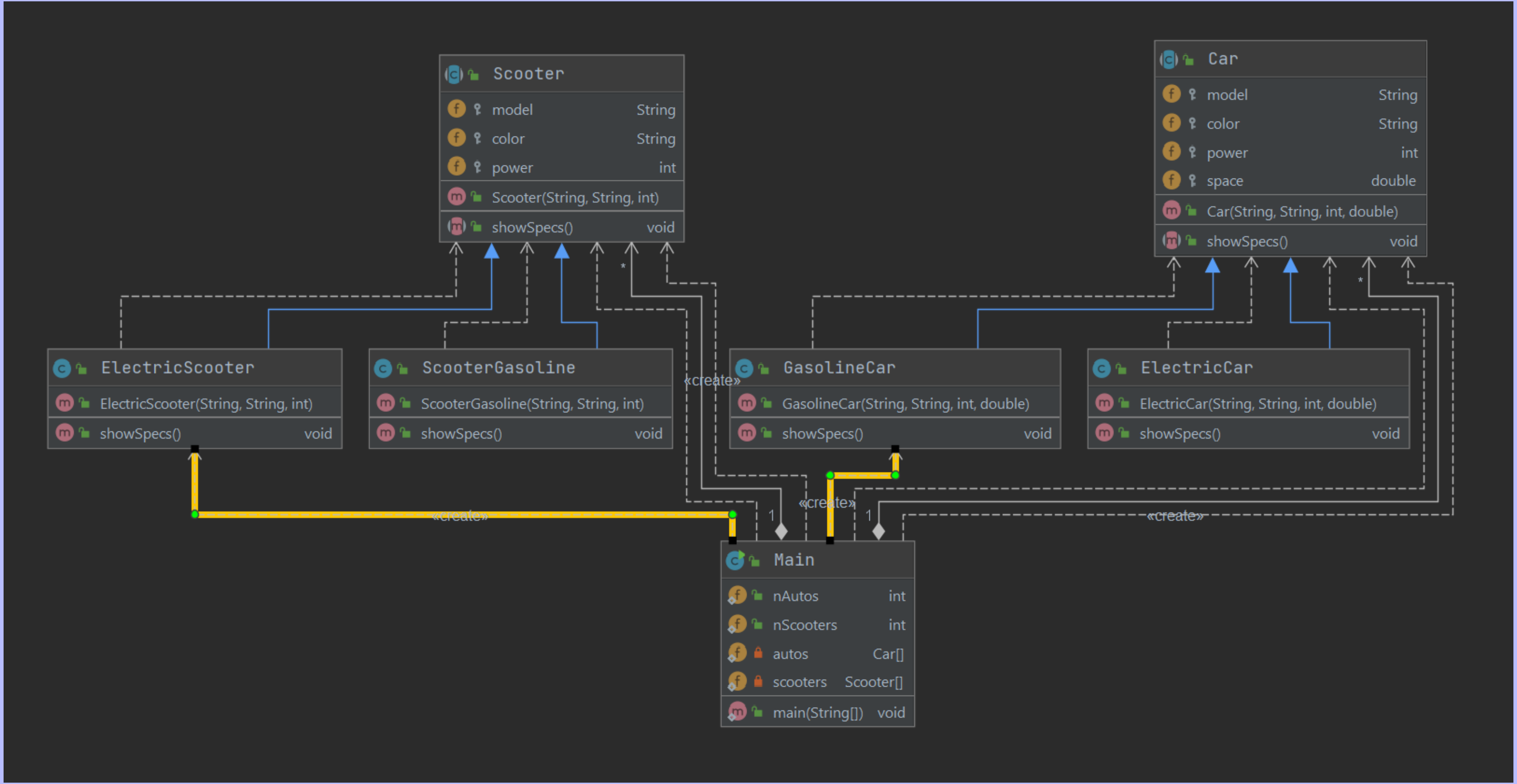
When to use the Abstract Factory

- A system should be configured with one of multiple families of products.

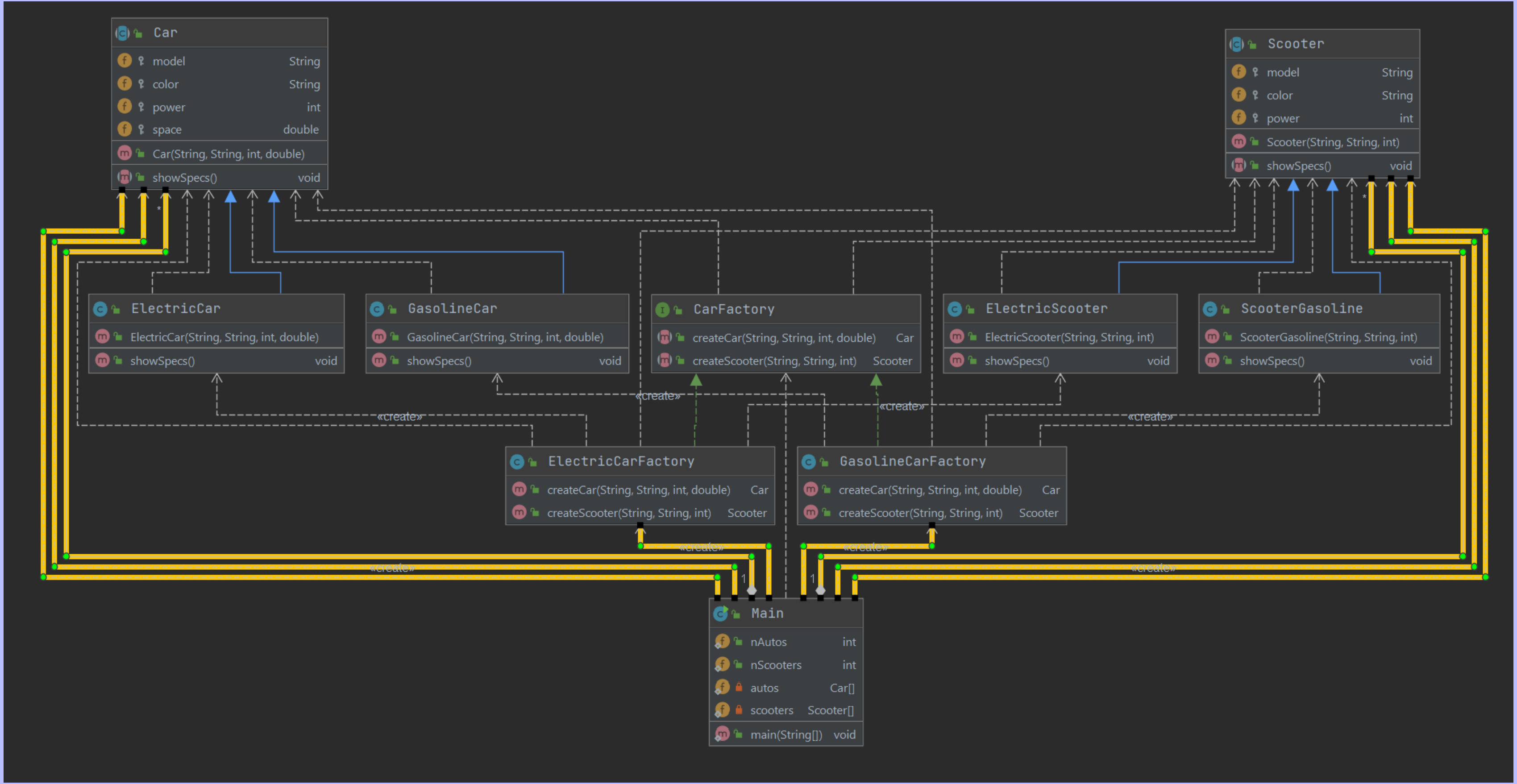
- A family of related product objects is designed to be used together

- You want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

Abstract Factory



Abstract Factory


Car
model String
color String
power int
space double
Car(String, String, int, double)
showSpecs() void
Scooter
model String
color String
power int
Scooter(String, String, int)
showSpecs() void
ElectricCar
ElectricCar(String, String, int, double)
showSpecs() void
GasolineCar
GasolineCar(String, String, int, double)
showSpecs() void
CarFactory
createCar(String, String, int, double) Car
createScooter(String, String, int) Scooter
ElectricScooter
ElectricScooter(String, String, int)
showSpecs() void
ScooterGasoline
ScooterGasoline(String, String, int)
showSpecs() void
ElectricCarFactory
createCar(String, String, int, double) Car
createScooter(String, String, int) Scooter
GasolineCarFactory
createCar(String, String, int, double) Car
createScooter(String, String, int) Scooter
Main
nAutos int
nScooters int
autos Car[]
scooters Scooter[]
main(String[]) void
«create»
«create»
«create»
«create»
«create»
«create»
«create»
«create»
1
1
*
*

CONS

The code may become more complicated than it should be, since a lot of new interfaces and classes are introduced along with the pattern.

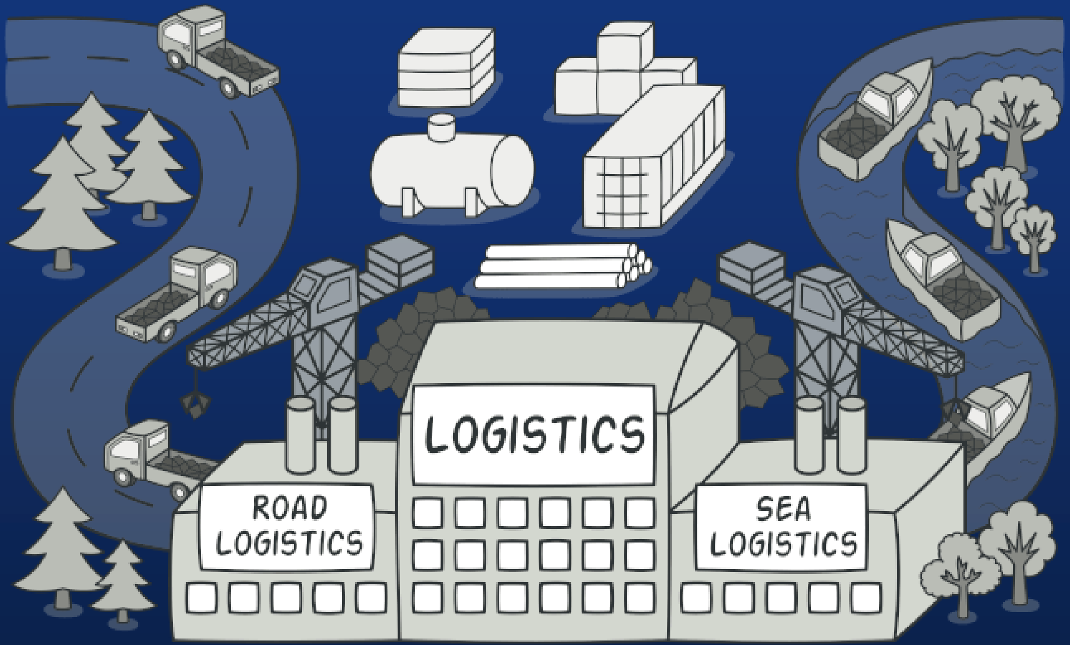
PROS

You avoid tight coupling between concrete classes and client code.

Single Responsibility Principle.

Open/Closed Principle.

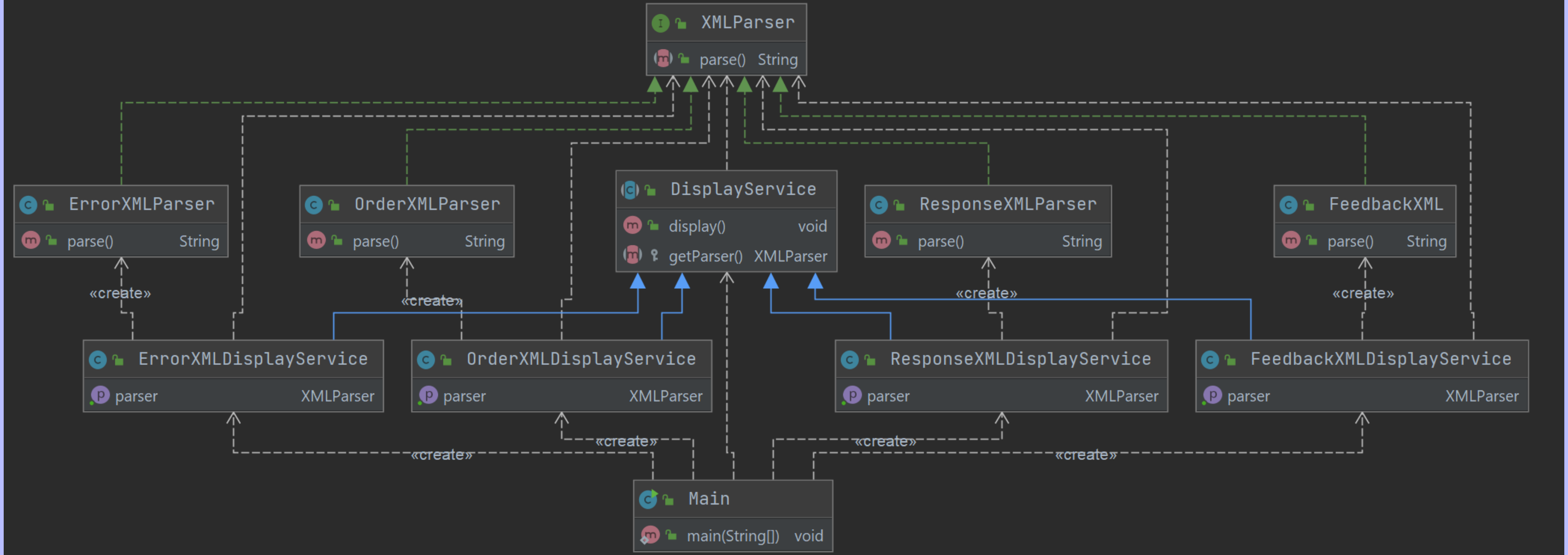
Factory Method



When to use the Abstract Factory

- Use the Factory Method when you don’t know beforehand the exact types and dependencies of the objects your code should work with.

Factory Method



CONS

The code may become more complicated than it should be, since a lot of new interfaces and classes are introduced along with the pattern.

PROS

You avoid tight coupling between the creator and the concrete products.

Single Responsibility Principle.

Open/Closed Principle.