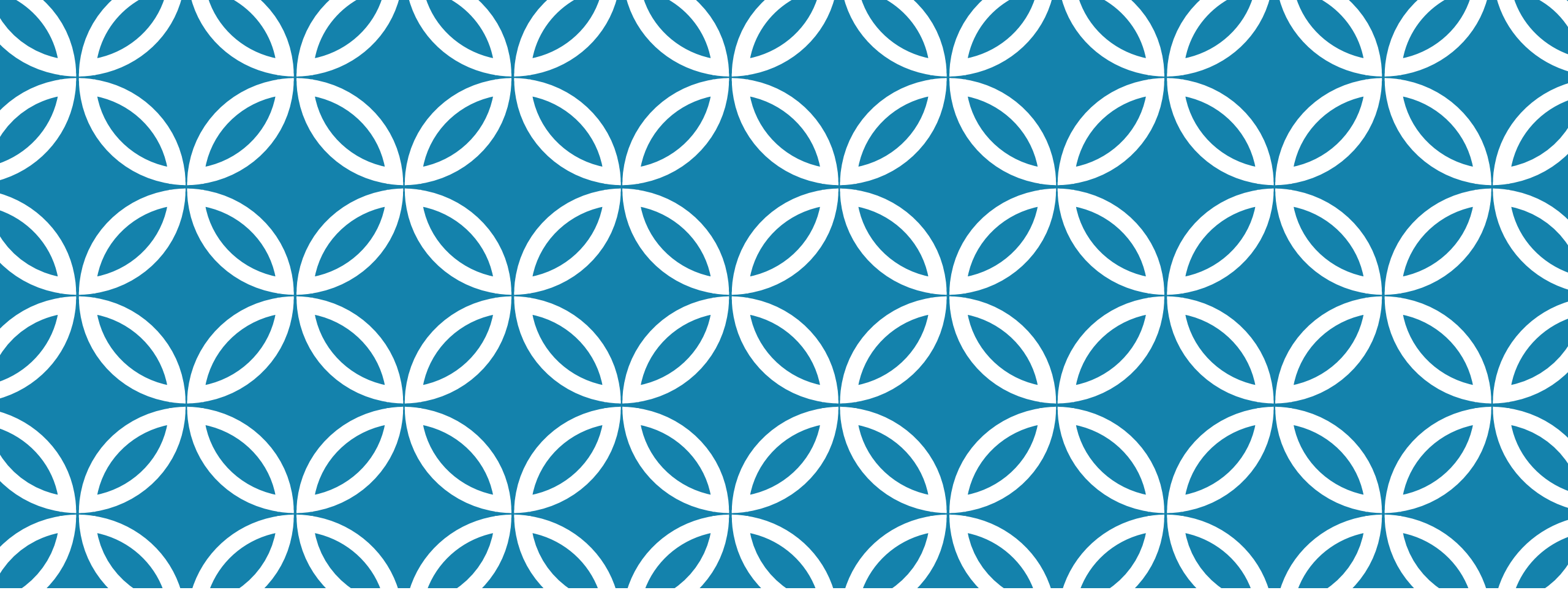
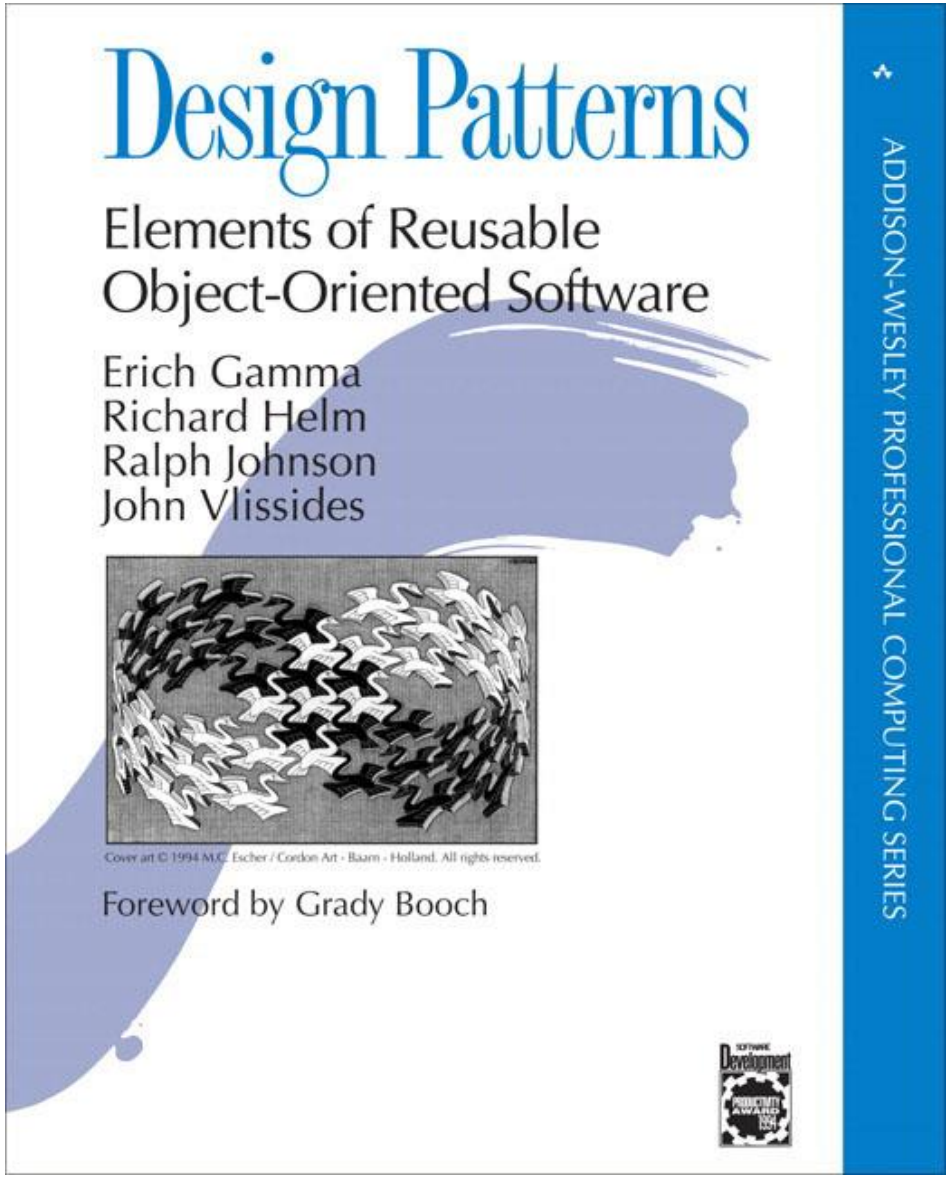


DESIGN PATTERNS IN C#

Pentalog - 2019
Trainer: Nadia Comanici

DESIGN PATTERNS – THE BOOK

- Published in 1994
- Gang of Four (GoF) = the authors
- You might need to read it twice ☺



WHAT ARE DESIGN PATTERNS?

- A design pattern is a recommended “recipe” to use in case of a certain problem
- Design patterns are:
  - independent of the programming language
  - simple, elegant & object-oriented solutions to a problem
  - not the first solution you would try (intuitively), because they were developed and evolved in time, to offer more flexibility and reusability
  - generally accepted by developers and used in programming

WHY USE THEM?

- Proven solutions, that work
- No need to reinvent the wheel, just use the well-known solution for your problem
- Common vocabulary for developers, easier to communicate and understand the needed solution
- Offer flexibility and reusability of code
- Make future changes more easier
- Object-oriented solutions

SO WHICH ARE THEY?

Scope	Creational	Structural	Behavioral
Class - relationships between classes (static + compile time)	Factory Method	Adapter	Interpreter
			Template Method
Object - relationship between objects (dynamic + runtime)	Abstract Factory	Bridge	Chain of Responsibility
	Builder	Composite	Command
	Prototype	Decorator	Iterator
	Singleton	Façade	Mediator
		Flyweight	Memento
		Proxy	Observer
			State
			Strategy
			Visitor

CREATIONAL DESIGN PATTERNS

CREATIONAL DESIGN PATTERNS

- They encapsulate knowledge about which concrete class the system is using
- The hide how instances of these classes are created and put together
- You have flexibility over the structure and functionality

1. ABSTRACT FACTORY

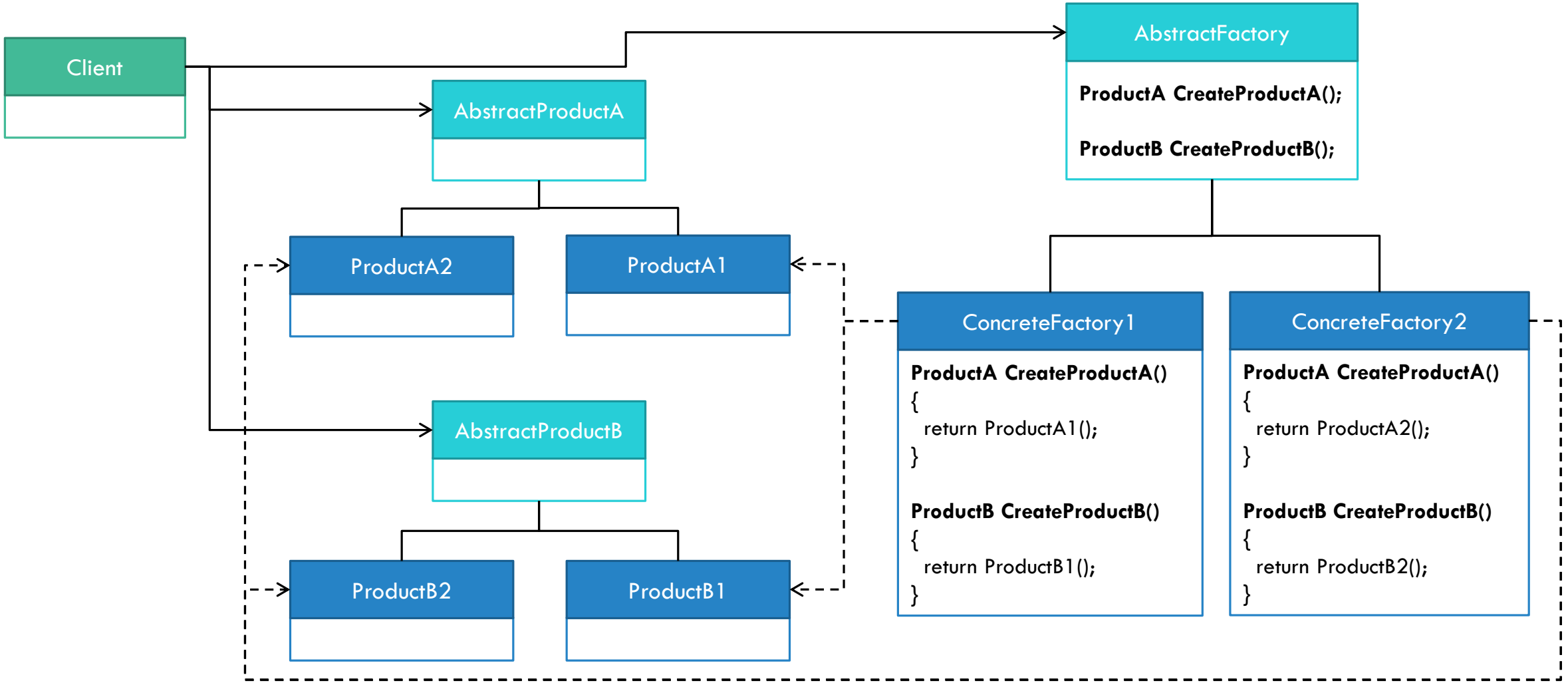
ABSTRACT FACTORY – WHAT DOES IT DO?

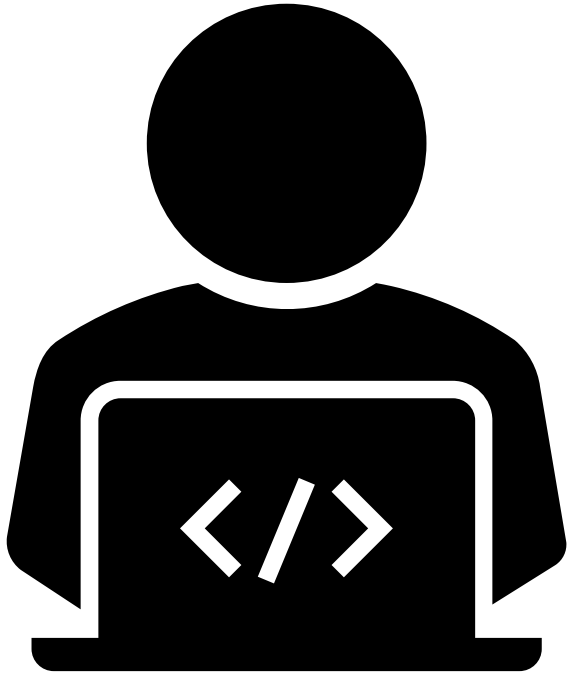
- “Provide an interface for creating families of related or dependent objects without specifying their concrete classes” (GoF)

ABSTRACT FACTORY – WHEN TO USE?

- For a system that should use one of multiple families of objects
- A family of objects or a combination of objects are designed to work together and you should enforce this constraint
- The system just needs to use the objects, without knowing how they are created, stored or represented internally
- The system uses only the interface, not the implementation

ABSTRACT FACTORY – DIAGRAM

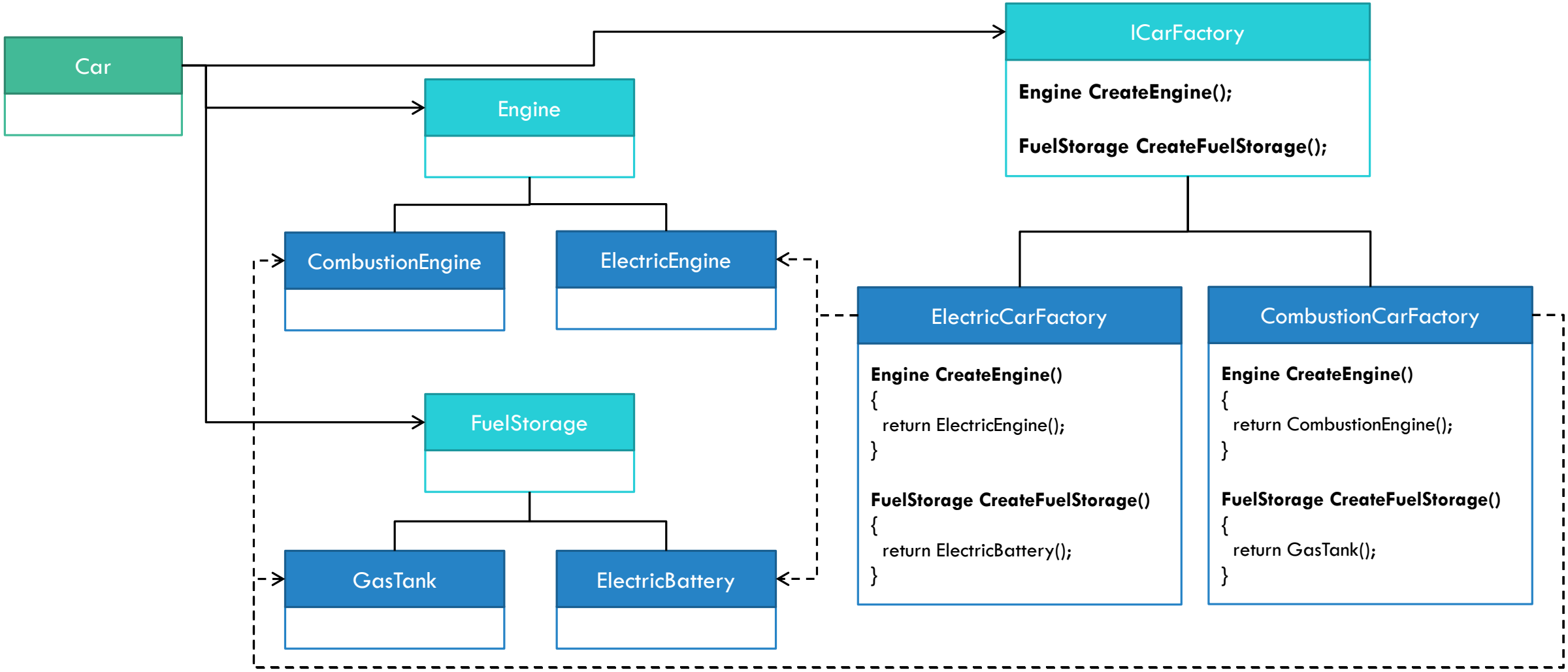


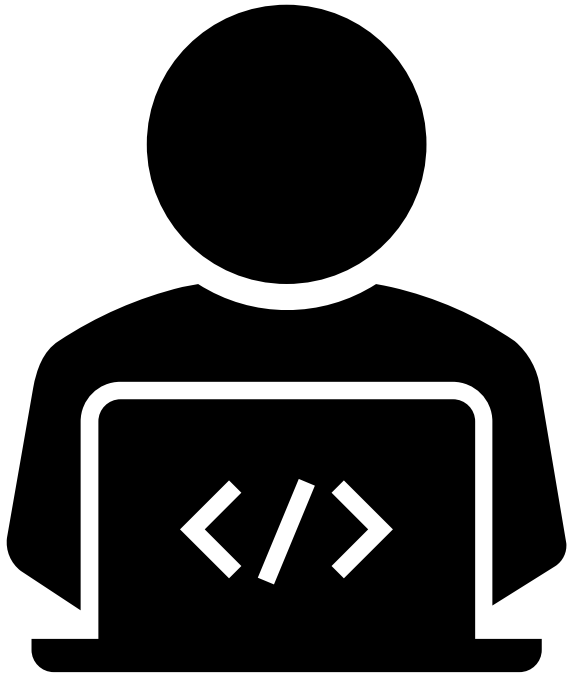


DEMO

AbstractFactory – Cars

ABSTRACT FACTORY – DIAGRAM – CAR DEMO

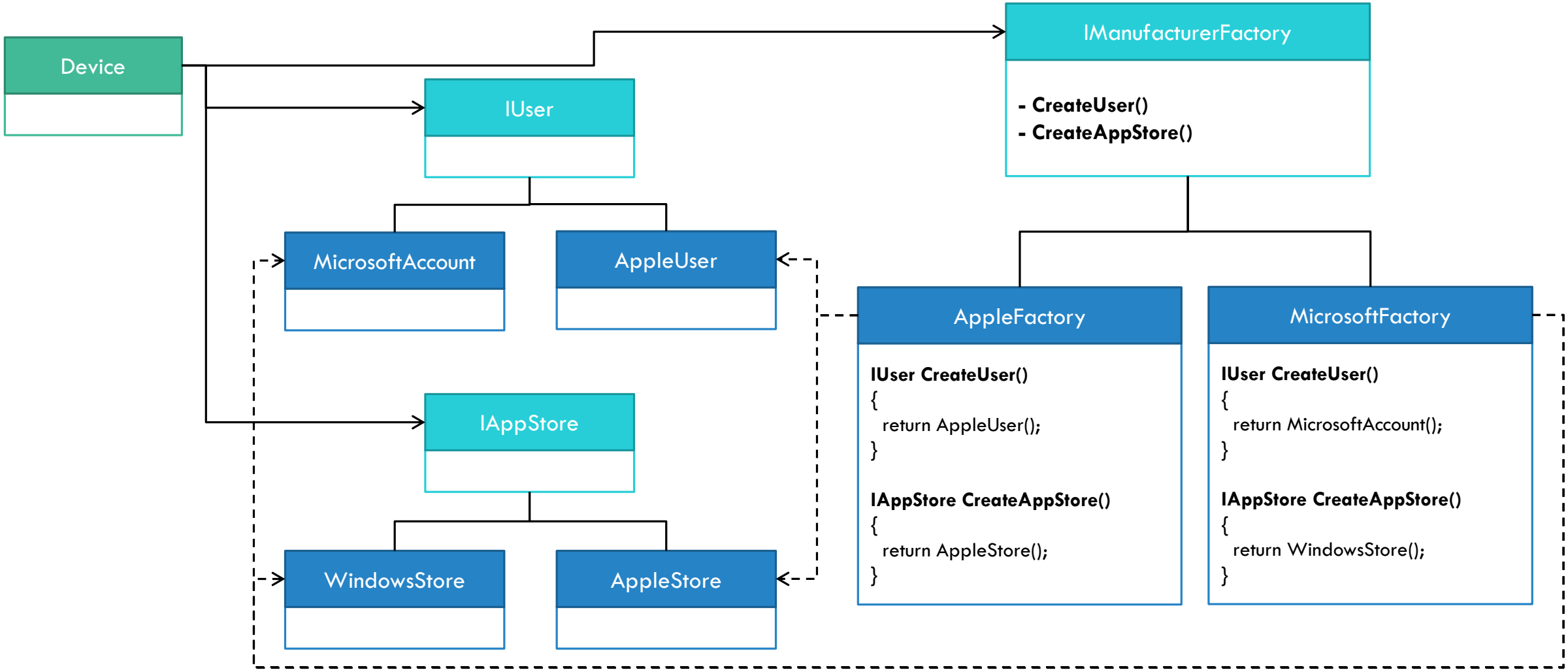




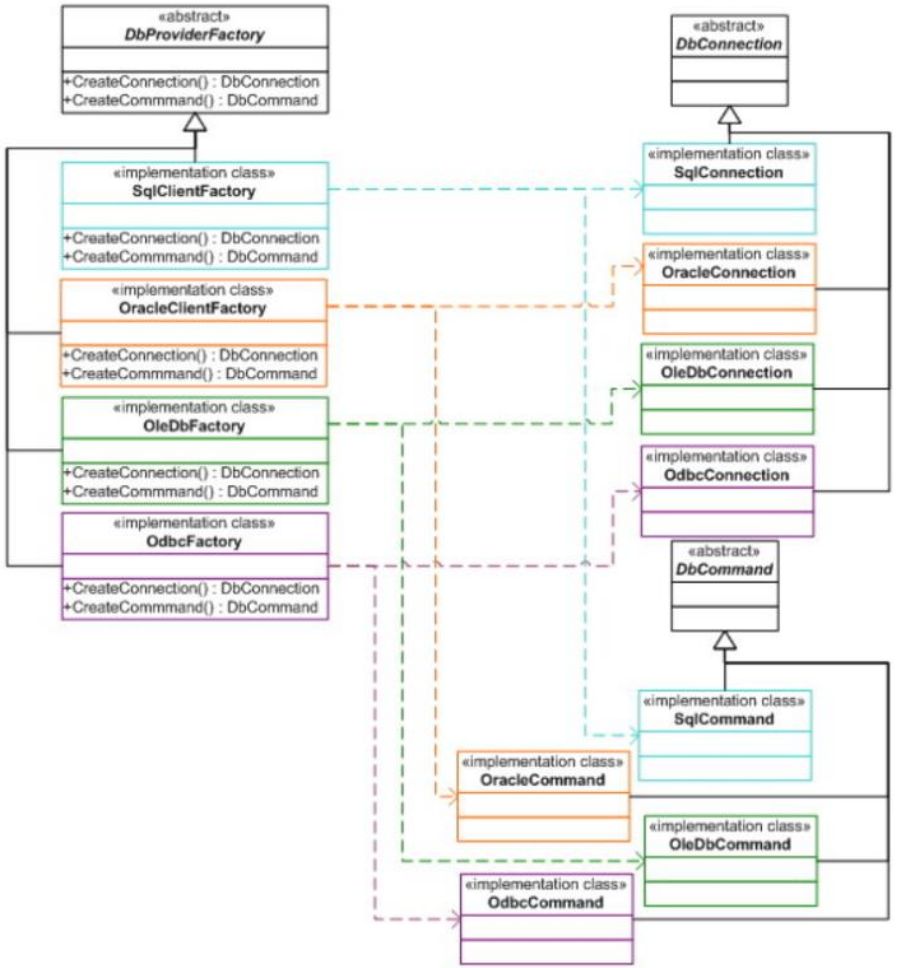
DEMO

AbstractFactory – Operating Systems

ABSTRACT FACTORY – DIAGRAM – OS DEMO



ABSTRACT FACTORY – USAGE – ADO.NET



ABSTRACT FACTORY – NOTES

- Abstract Factory in .NET Framework:
  - https://visualstudiomagazine.com/articles/2011/01/27/the-factory-pattern-in-net-part-3.aspx
  - Examples: ADO.NET, WindsorCastle, nHibernate

- More examples:
  - https://www.dofactory.com/net/abstract-factory-design-pattern
  - http://www.exceptionlesscode.com/abstract-factory-pattern-with-examples/

ABSTRACT FACTORY – ADVANTAGES

- Easy to create families of classes that should work only together (and enforce this constraint)
- Easy to replace one family with another
- The concrete classes are hidden from the client
- It enables architectures like Dependency Injection

ABSTRACT FACTORY – DISADVANTAGES

- Adding a new object to the family of objects means adding a method in the abstract interface and this will have to be implemented in all concrete factories
- The client cannot do subclass-specific operations