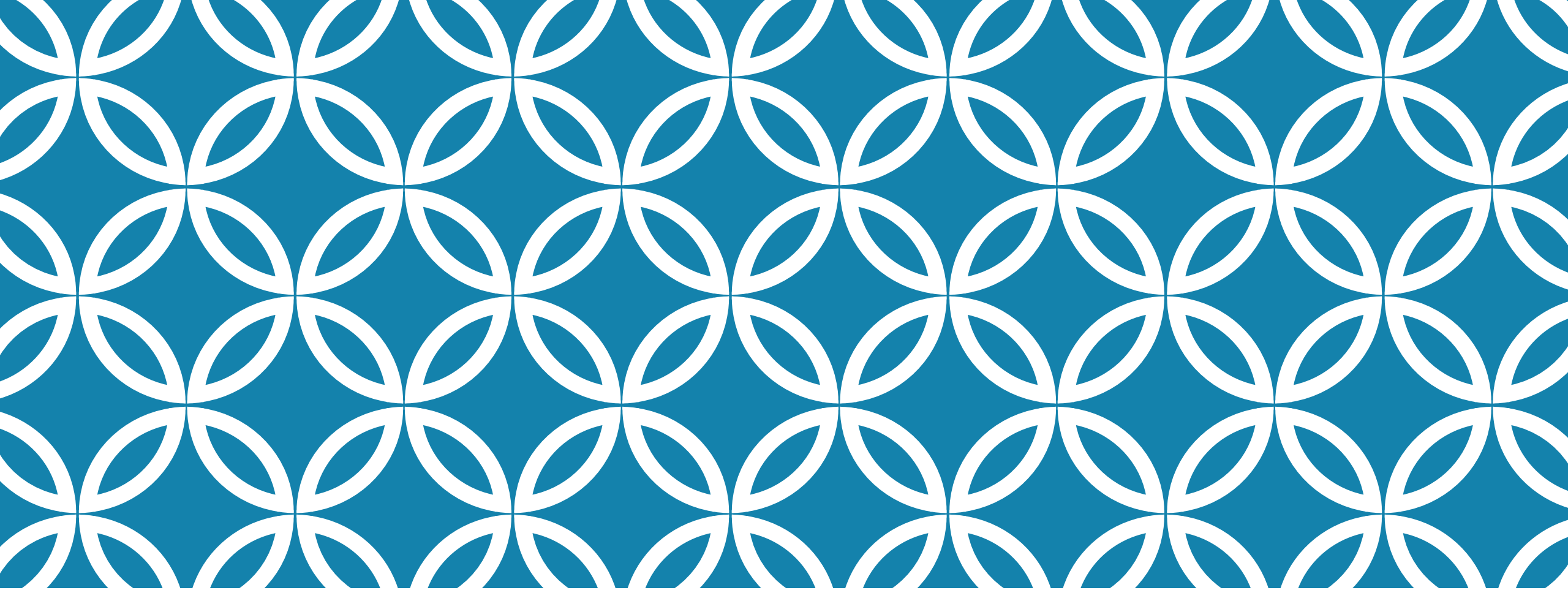
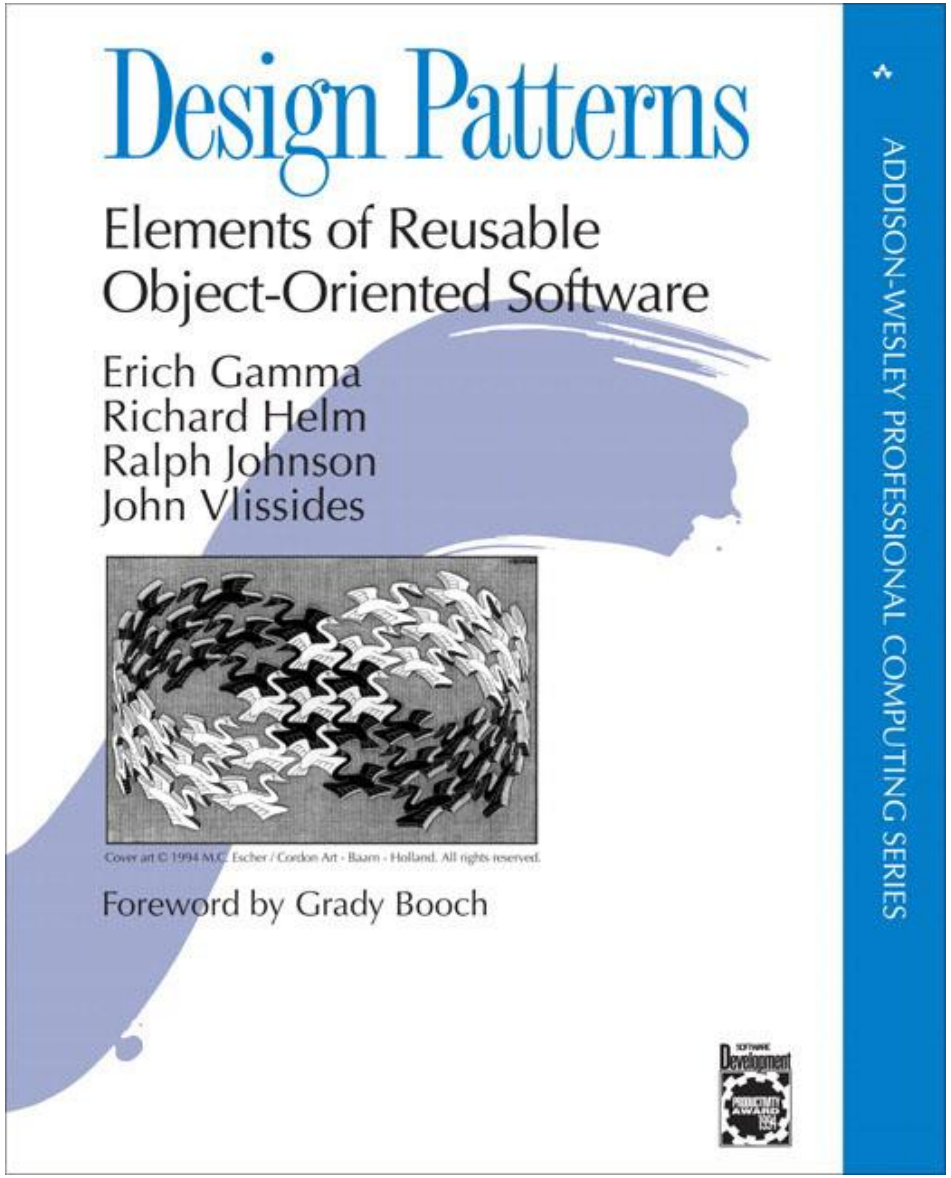


DESIGN PATTERNS IN C#

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

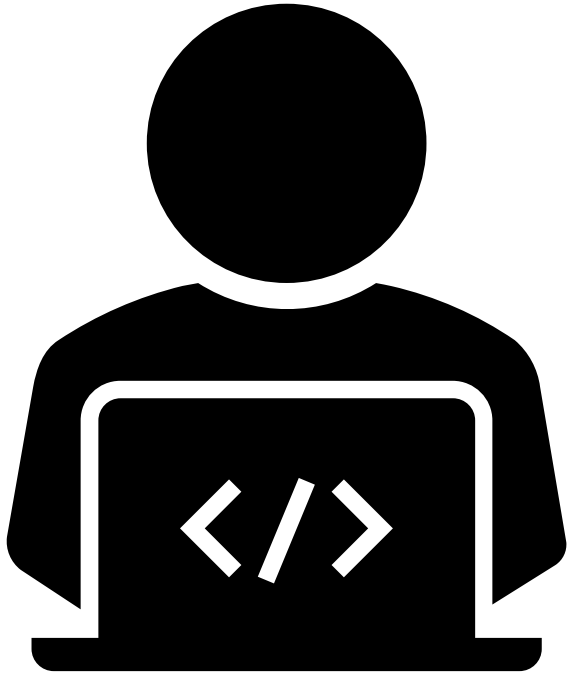
CREATIONAL DESIGN PATTERNS

- 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?

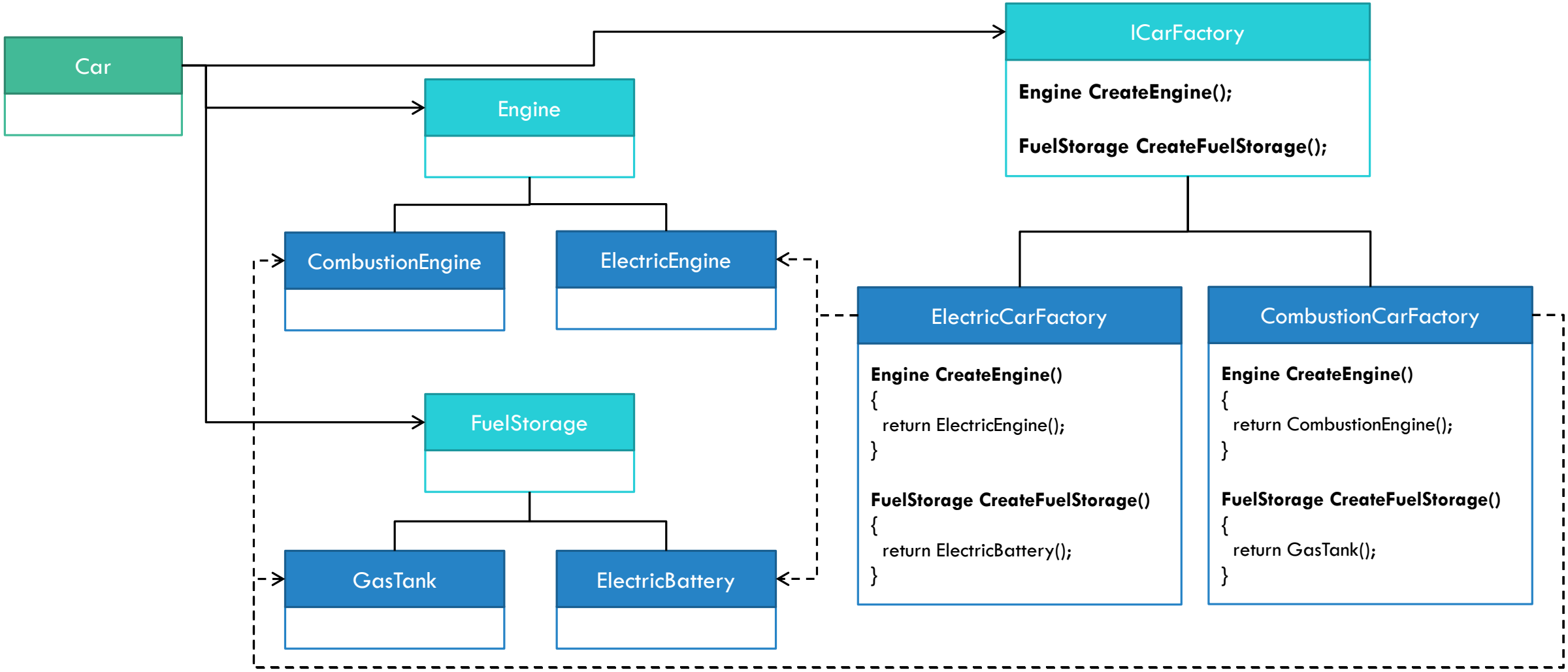
- “Provide an interface for creating families of related or dependent objects without specifying their concrete classes” (GoF)



DEMO

AbstractFactory – Cars

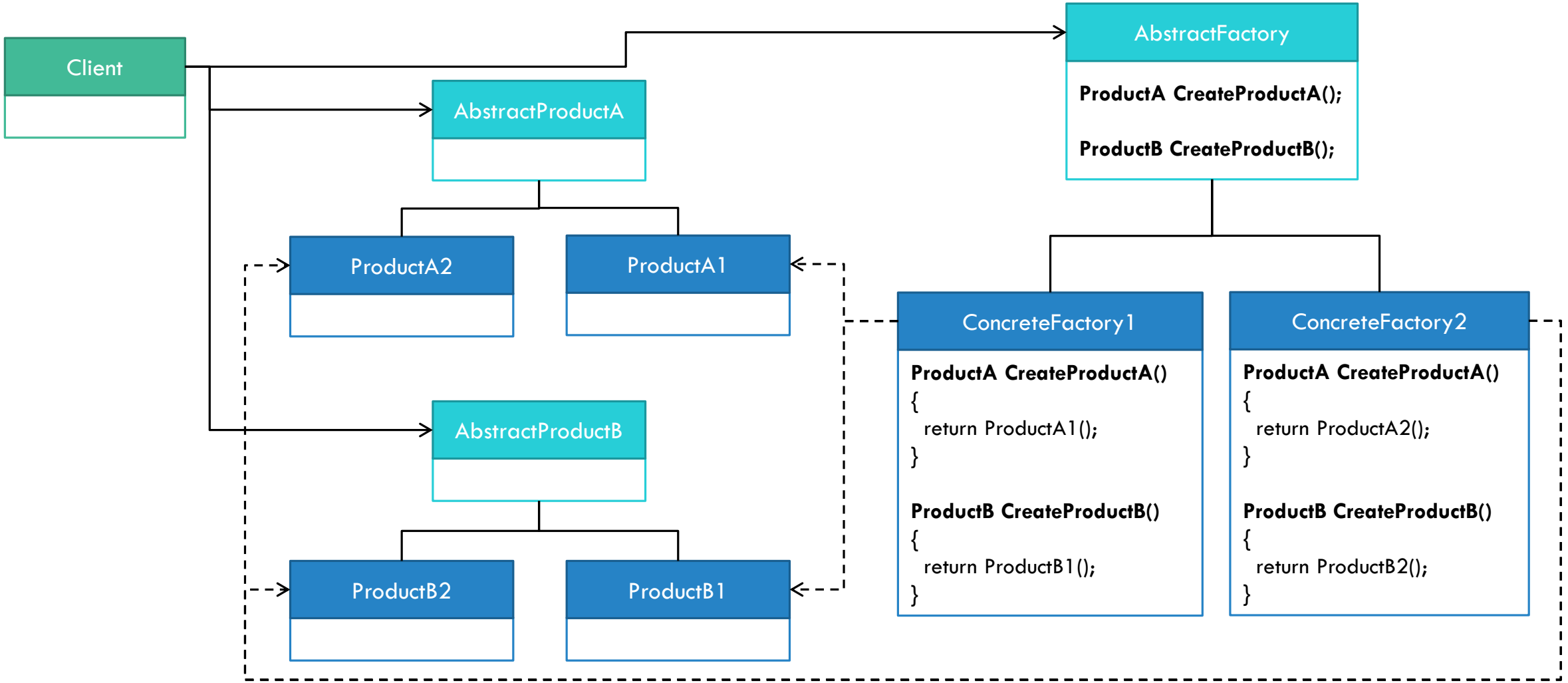
ABSTRACT FACTORY – DIAGRAM – CAR DEMO

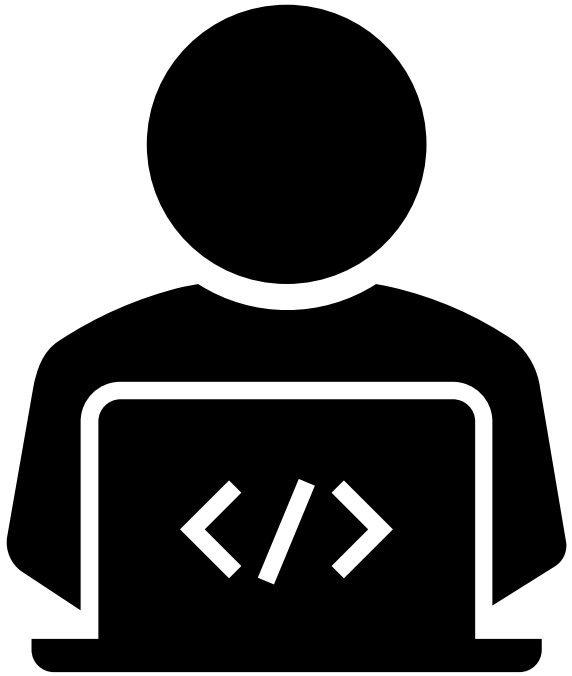


ABSTRACT FACTORY – WHEN TO USE?

- 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

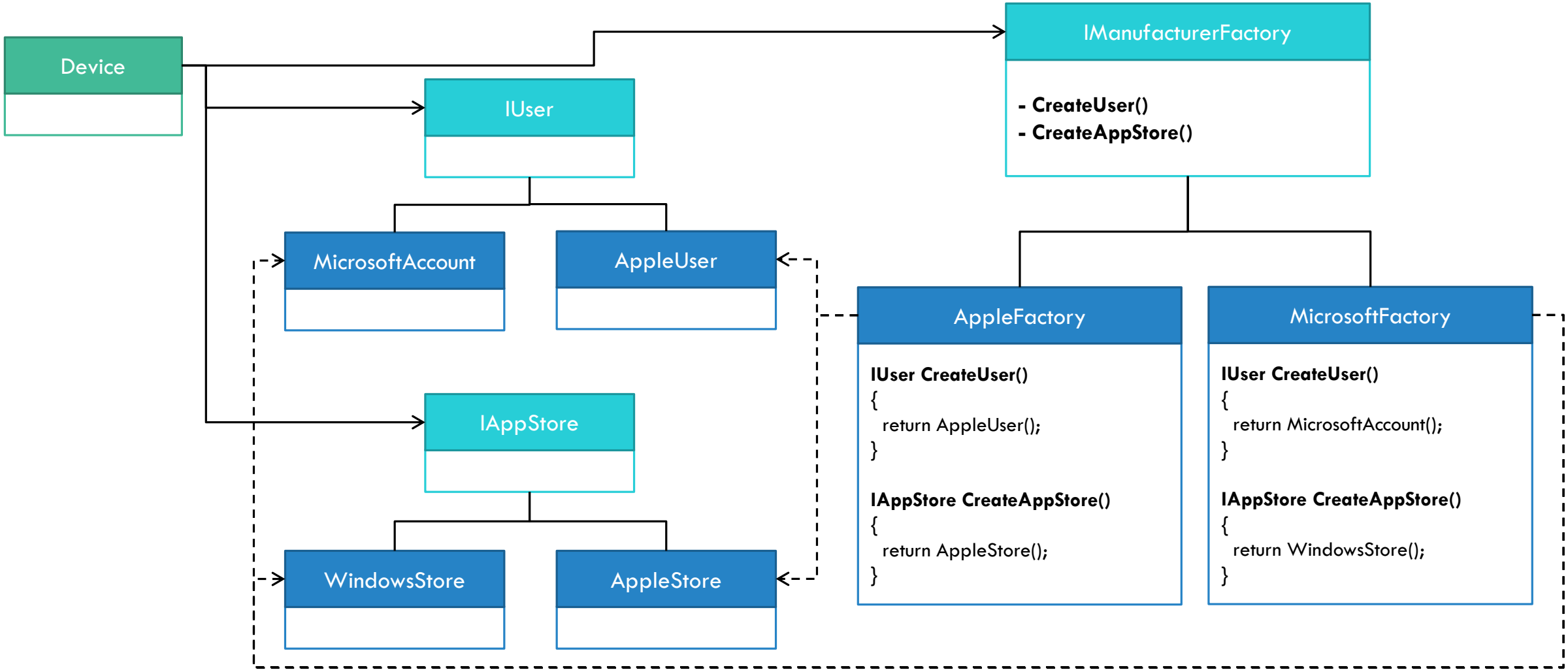




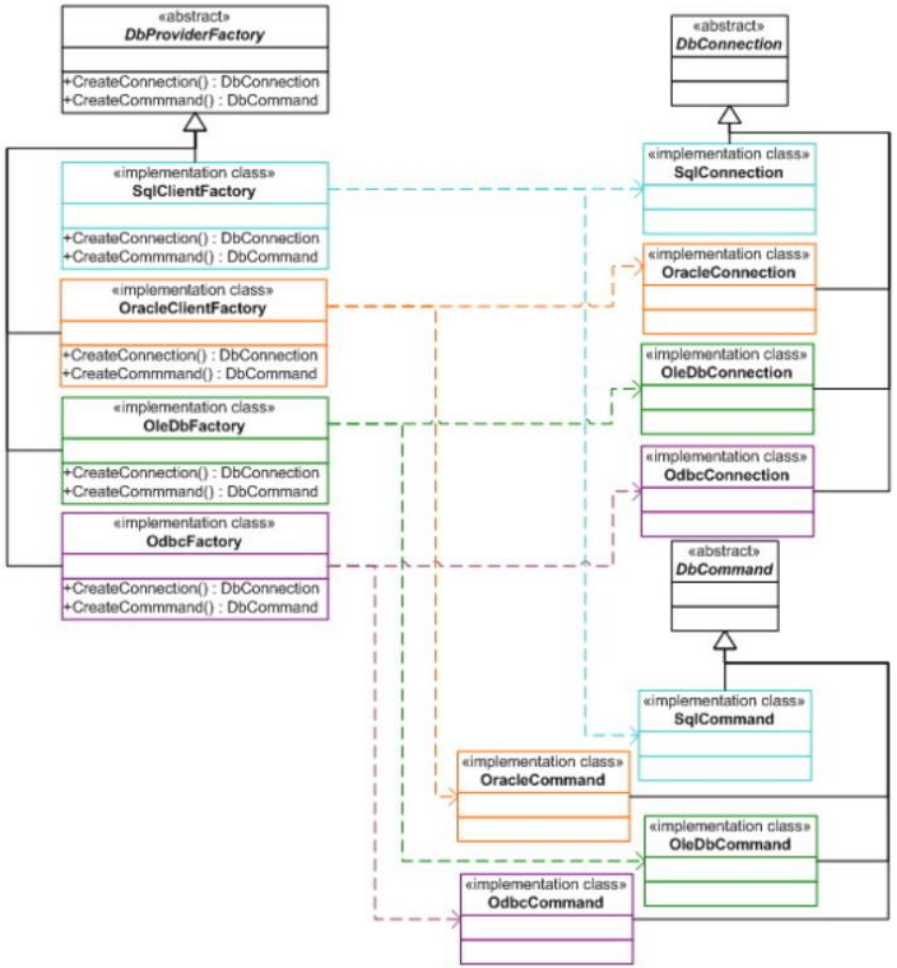
DEMO

AbstractFactory – 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
- Can take longer to implement at first

2. FACTORY METHOD

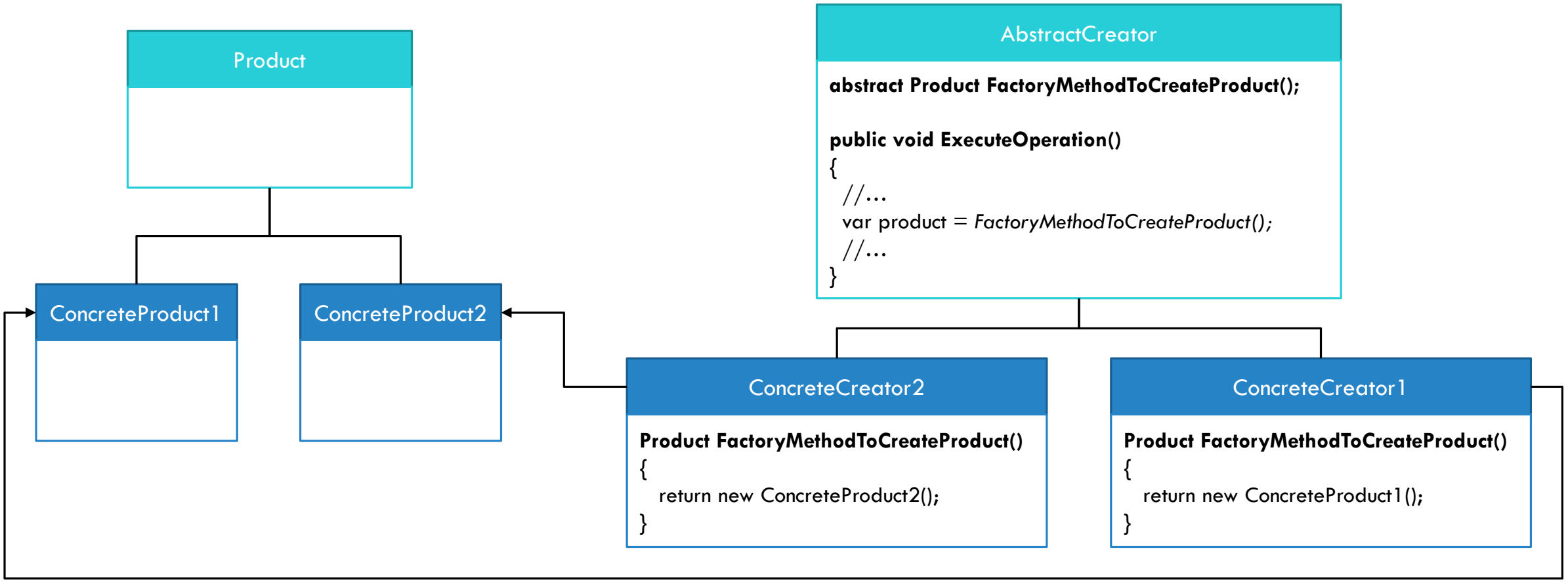
FACTORY METHOD – WHAT DOES IT DO?

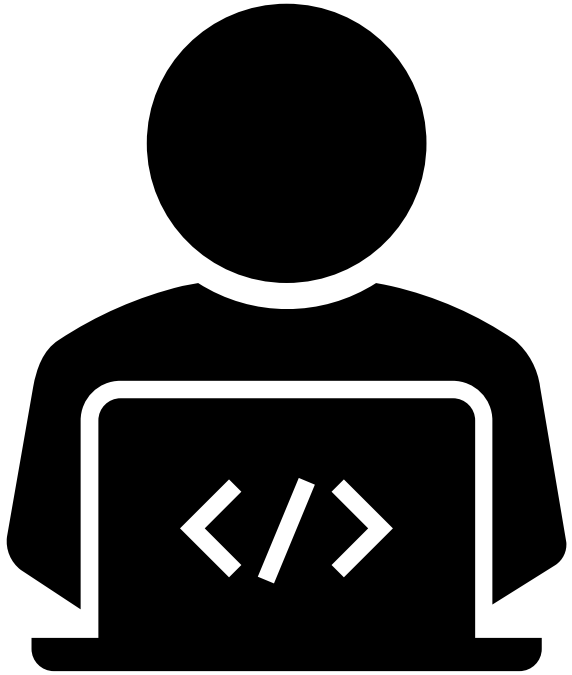
- “Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses” (GoF)

FACTORY METHOD – WHEN TO USE?

- A class can't anticipate the class of objects it must create.
- A class wants its subclasses to specify the objects it creates.
- Classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate

FACTORY METHOD – DIAGRAM

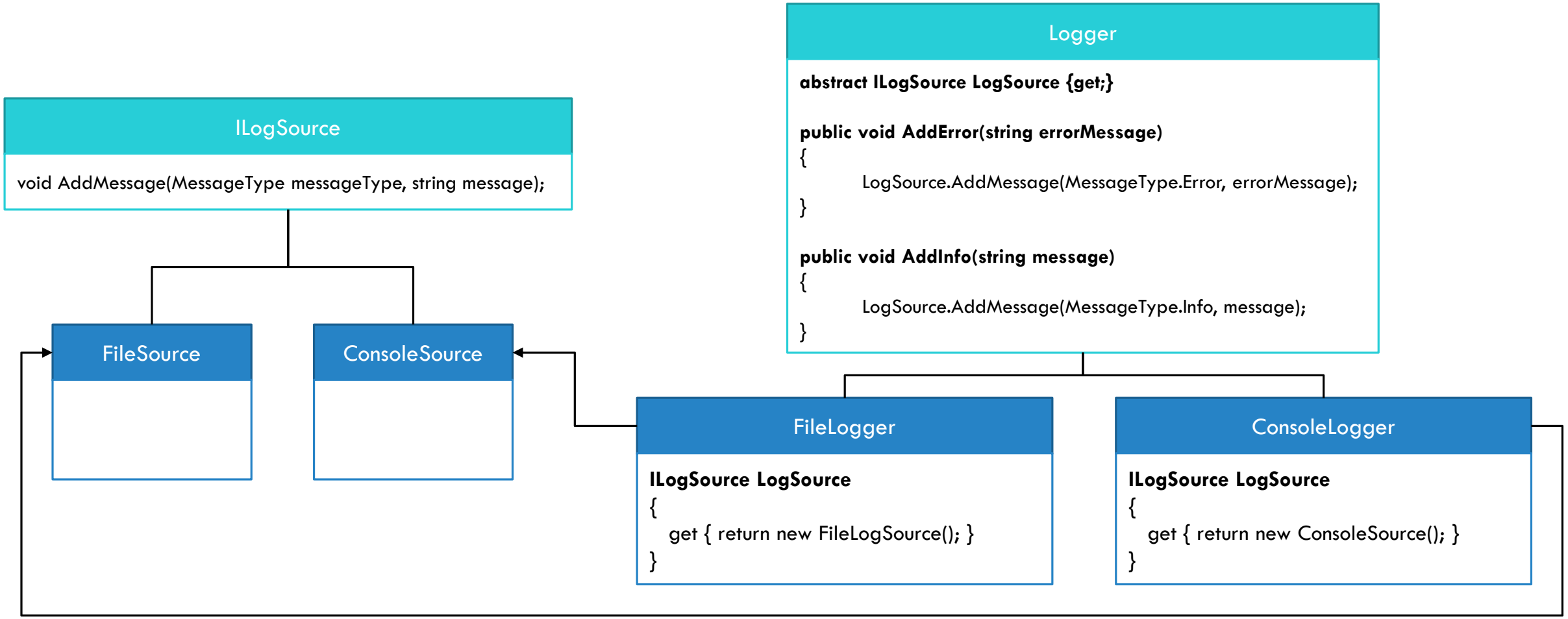




DEMO

Factory Method – Logger

FACTORY METHOD – DIAGRAM – LOGGER DEMO



FACTORY METHOD - ADVANTAGES

- Can easily add another implementation
- Connects parallel hierarchies of classes

FACTORY METHOD - VARIETY

- The FactoryMethod can return a new instance each time or it can use lazy loading, to load the instance at first request and return the same instance

```
public abstract class Logger
{
    4 references
    protected abstract LogSource LogSource { get; }

    2 references
    public void AddError(string errorMessage)
    {
        LogSource.AddMessage(MessageType.Error, errorMessage);
    }

    1 reference
    public void AddInfo(string message)
    {
        LogSource.AddMessage(MessageType.Info, message);
    }
}
```

```
public class ConsoleLogger : Logger
{
    4 references
    protected override LogSource LogSource
    {
        get
        {
            return new ConsoleLogSource();
        }
    }
}
```

```
public class FileLogger : Logger
{
    private readonly string _fullFilePath;
    private FileLogSource _fileLogSource;

    0 references
    public FileLogger(string fullFilePath)
    {
        _fullFilePath = fullFilePath;
    }

    4 references
    protected override LogSource LogSource
    {
        get
        {
            if (_fileLogSource == null)
            {
                _fileLogSource = new FileLogSource(_fullFilePath);
            }
            return _fileLogSource;
        }
    }
}
```

FACTORY METHOD - VARIETY

- The FactoryMethod can have a default implementation in the base class or not
- The Creator can create the product based on a parameter

```
public class VehicleCreator
{
    1 reference
    public IVehicle GetVehicle(int numberOfWheels)
    {
        switch (numberOfWheels)
        {
            case 1:
                return new UniCycle();
            case 2:
                return new Bicycle();
            case 3:
                return new Tricycle();
            case 4:
                return new Car();
            default:
                throw new ArgumentException("Invalid number of wheels");
        }
    }
}
```

```
var vehicleCreator = new VehicleCreator();
for (int i = 1; i <= 4; i++)
{
    var vehicle = vehicleCreator.GetVehicle(i);
    vehicle.Accelerate();
    vehicle.Stop();
}
```

FACTORY METHOD - NOTES

- Factory Method inside .NET Framework
  - Example: WebRequest
  - http://aspalliance.com/1751_Exemplifying_the_Factory_Method_Pattern_inside_the_NET_Framework.3

- More samples:
  - https://sourcemaking.com/design_patterns/factory_method
  - https://www.dofactory.com/net/factory-method-design-pattern
  - https://exceptionnotfound.net/the-daily-design-pattern-factory-method/

FACTORY METHOD VS ABSTRACT FACTORY

Abstract Factory	Factory Method
Uses **composition**	Uses **inheritance**
A class delegates the responsibility of object instantiation to another object	Relies on a subclass to handle the desired object instantiation.
Creates a family of related objects	Creates an object

- More: https://dzone.com/articles/factory-method-vs-abstract

Q&A
FACTORY METHOD

3. BUILDER

BUILDER – WHAT DOES IT DO?

- “Separate the construction of a complex object from its representation, so that the same construction process can create different representations.” (GoF)

BUILDER – WHEN TO USE?

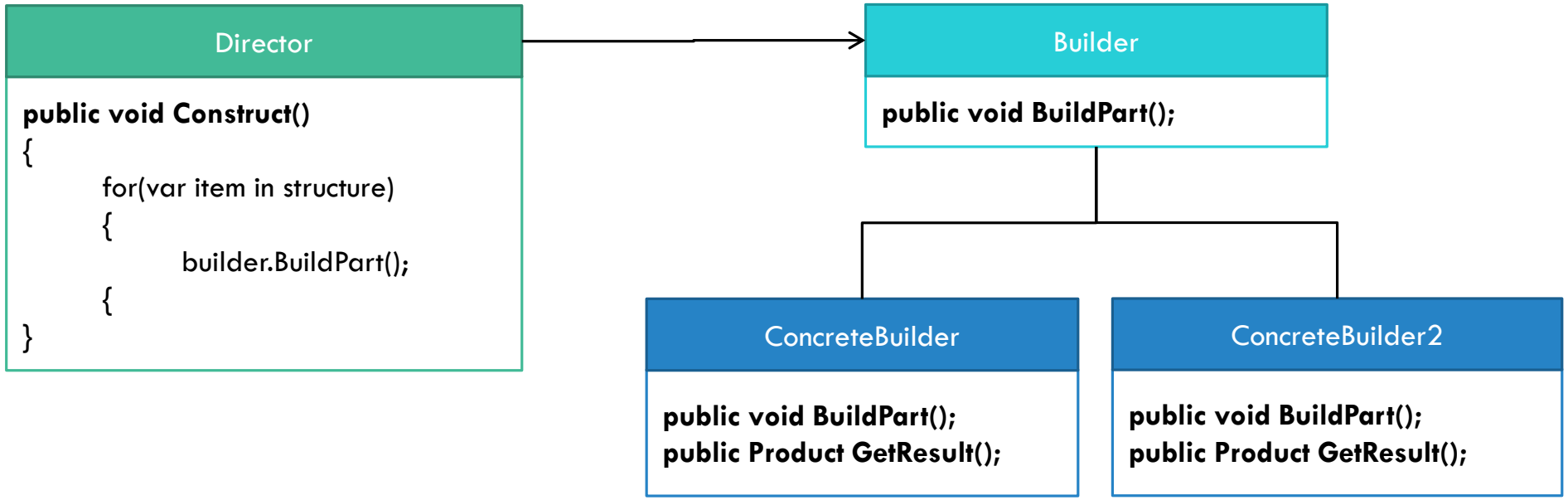
- To separate the construction of a complex object from the actual functionality

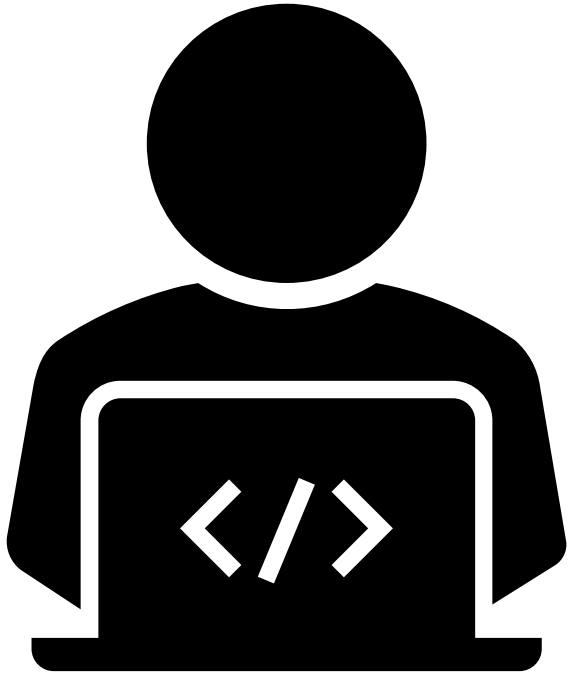
- More:
  - https://www.dofactory.com/net/builder-design-pattern

BUILDER – WHAT IS NOT

- StringBuilder is not a builder design pattern implementation
- Fluent Interface is not a builder implementation
  - Fluent Interfaces are semantic facades, they just improve readability, but don't actually enforce the construction of an object
  - https://medium.com/@sawomirkowalski/design-patterns-builder-fluent-interface-and-classic-builder-d16ad3e98f6c

BUILDER – DIAGRAM

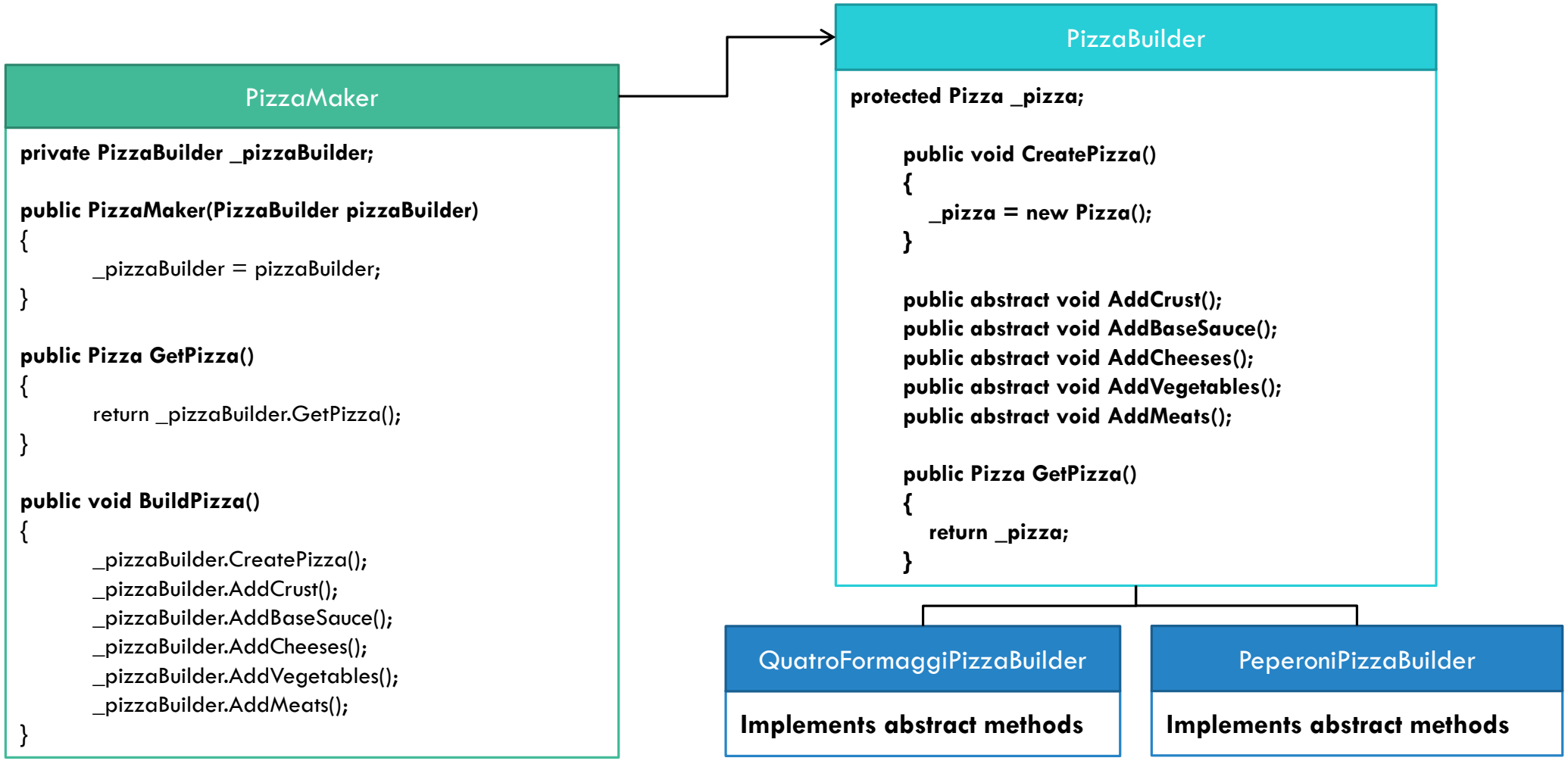




DEMO

Builder - Pizza

BUILDER – DIAGRAM – PIZZA DEMO



BUILDER – ADVANTAGES

- Hides the representation of a complex object
- Allows you to vary the internal representation and parts of an object
- Isolates the construction and the representation
  - Client's don't know about the internal representation of the products
- Enforces the steps for creating the object
  - The Director returns the object only after all the steps were done

Q&A BUILDER

4. SINGLETON

SINGLETON – WHAT DOES IT DO?

- "Ensure a class only has one instance, and provide a global point of access to it." (GoF)

SINGLETON – WHEN TO USE?

- There must be only one instance of the class
- The sole instance must be accessible to clients using a access point

- More:
  - https://www.dofactory.com/net/singleton-design-pattern
  - https://csharpindepth.com/articles/BeforeFieldInit
  - https://csharpindepth.com/articles/singleton

SINGLETON – DIAGRAM

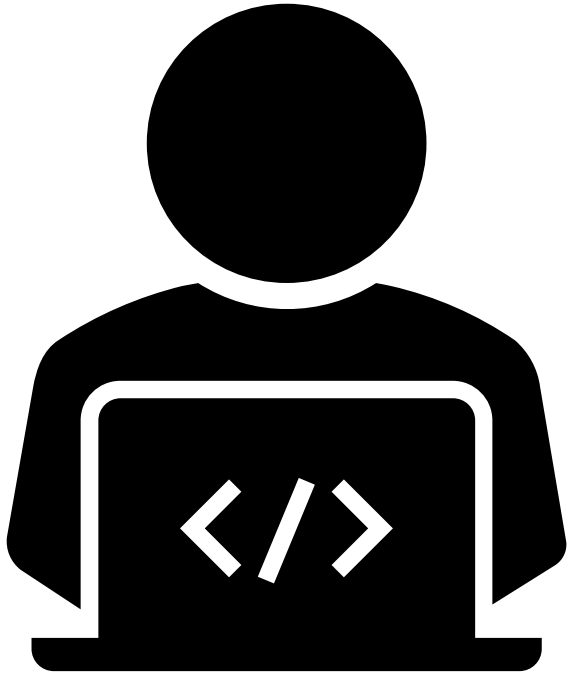
Singleton

```
private static Singleton _instance;
private string _data;

private Singleton()
{
        // initialize non-static data
        _data = string.Empty;
}

public static Singleton GetInstance()
{
        if (_instance==null)
        {
                _instance = new Singleton();
        }
        return _instance;
}

public void SingletonOperation()
{
        // do something with _data
}
```

Seal Singleton class

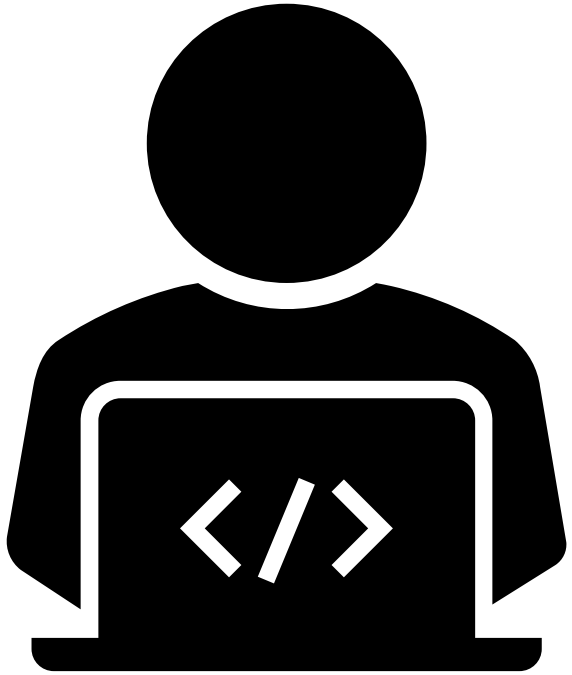
Each Singleton should have:
- private static instance of it's own type = _instance
- public static access point = GetInstance()
- private constructor

Each Singleton can have:
- Internal non-static data



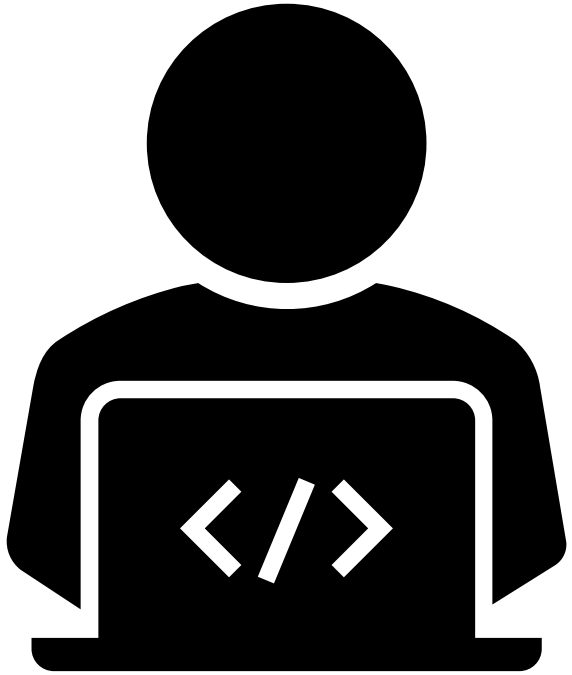
DEMO

Singleton - Logger



DEMO

Singleton - LoggerDerived



DEMO

Singleton – ThreadSafeLogger

SINGLETON - ADVANTAGES

- Control over the access point
- Alternative to global variables -> global accessible
- Can use lazy loading to create the sole instance
- Can be easily adapted for multiple instances as well

SINGLETON - DISADVANTAGES

- Can be an antipattern
  - It's still "global"
  - Tendency to have multiple responsibilities
  - Tight coupling between collaborating classes
- Default implementation is not thread safe
  - Should not use in multi thread environments (e.g. ASP.NET)
- Difficult to test
  - If you call GetInstance() inside the method, instead of sending it as parameter - Static cannot be mocked
  - Try to inject the dependency as much as you can ☺
  - Use a IoC container
- Can have multiple instances, not only one
  - Make it sealed + private constructor ☺
- More:
  - https://blogs.msdn.microsoft.com/scottdensmore/2004/05/25/why-singletons-are-evil/

Q&A
SINGLETON

5. PROTOTYPE

PROTOTYPE – WHAT DOES IT DO?

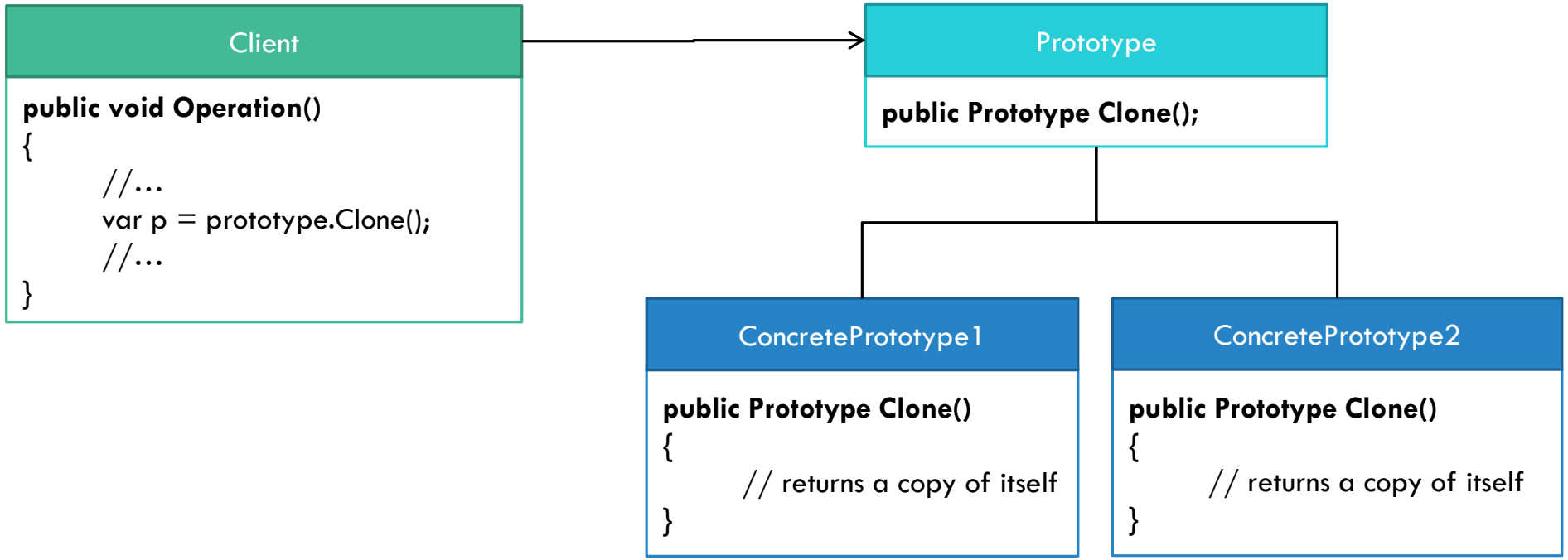
- “Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.” (GoF)

PROTOTYPE – WHEN TO USE?

- To clone one or multiple objects that are created after a costly operation, like
  - Database call
  - Apply algorithm
  - Get response from webpage
- The cloned objects are very similar to the prototype object
- It's faster to clone than to create a new object
- When the classes to instantiate are specified at runtime

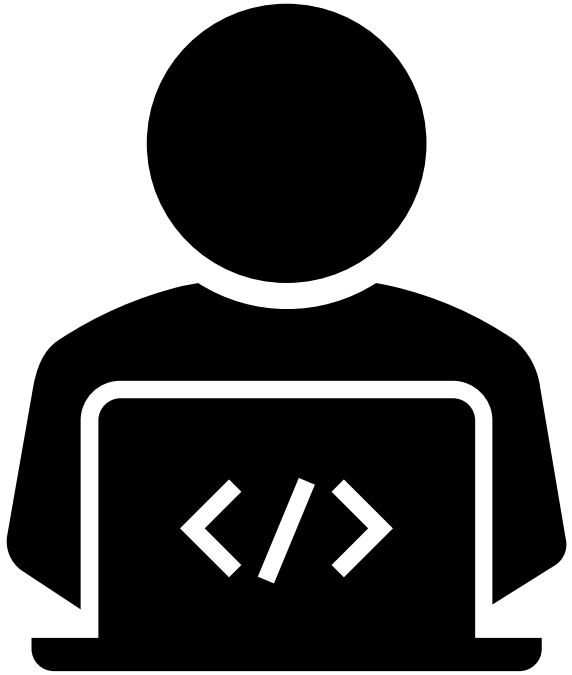
- More:
  - https://www.dofactory.com/net/prototype-design-pattern

PROTOTYPE – DIAGRAM



PROTOTYPE – IN .NET FRAMEWORK

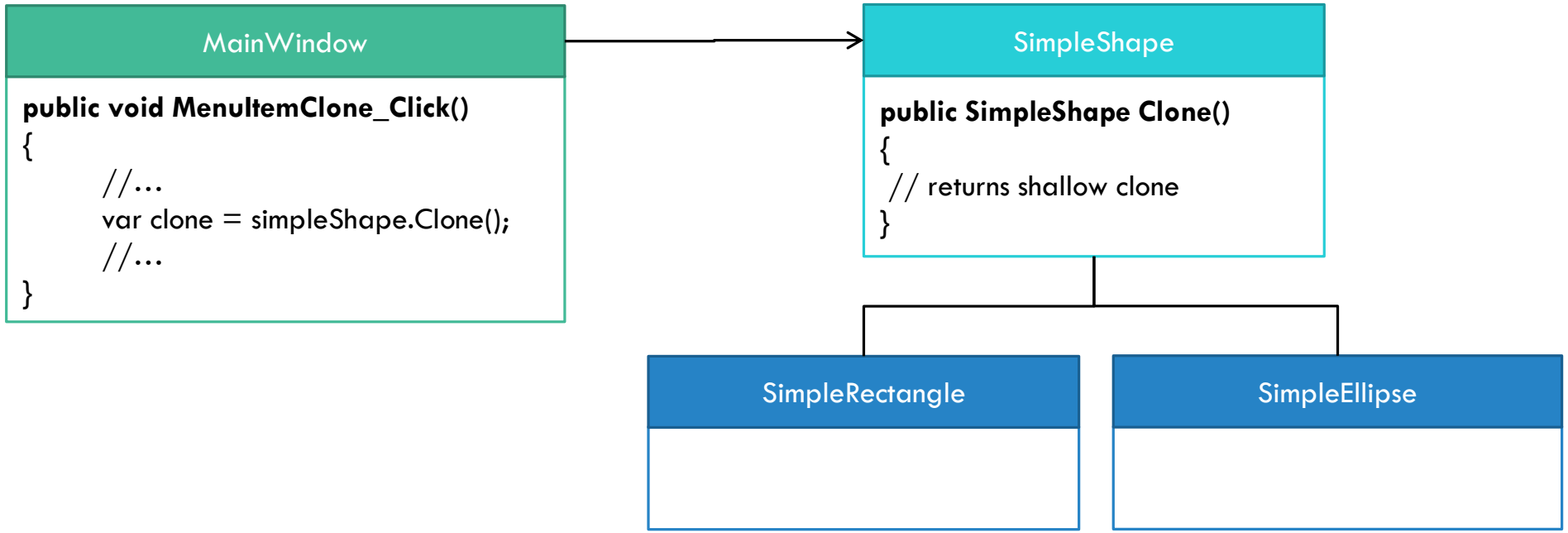
- You can use ICloneable



DEMO

Prototype - Shapes

PROTOTYPE – DIAGRAM – SHAPES DEMO

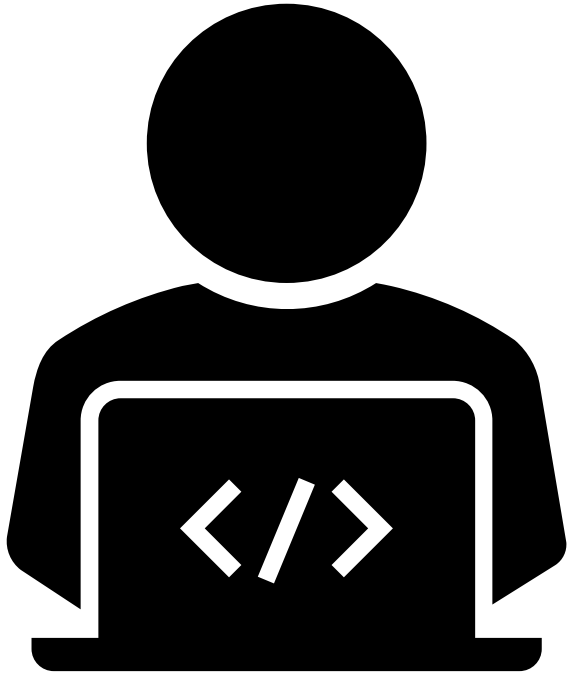


PROTOTYPE

Shallow clone	Deep clone
- Makes copies of value-type members - Uses the same reference-type members	- Duplicates everything (value or reference types)
- MemberwhiseClone()	- If class is Serializable, you can use MemoryStream and BinaryFormatter

```
public object Clone()
{
    return MemberwiseClone();
}
```

```
public static T DeepClone<T>(T obj)
{
    using (var ms = new MemoryStream())
    {
        var formatter = new BinaryFormatter();
        formatter.Serialize(ms, obj);
        ms.Position = 0;

        return (T)formatter.Deserialize(ms);
    }
}
```



DEMO

Prototype - ShallowAndDeepClone

PROTOTYPE – ADVANTAGES

- Create at runtime a object by cloning an already existing one, not by creating a new one
- Can reduce the number of classes in the system
- The concrete classes are hidden from the client
- Greater flexibility in combining the objects of the system, at runtime

PROTOTYPE – DISADVANTAGES

- You might need a PrototypeManager to register all available prototype
- Cloning when you have circular references
  - Shallow clone vs Deep clone
- If you want the clone to have different values when it is initialized, you might need to add an Initialize method

Q&A
PROTOTYPE