Builder Pattern

Lecture 16

Builder Pattern

- The builder pattern is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming.
- The intent of the Builder design pattern is to separate the construction of a complex object from its representation.

Problem

- The Builder design pattern solves problems like:
  - How can a class (the same construction process) create different representations of a complex object?
  - How can a class that includes creating a complex object be simplified?

Solution

- The Builder design pattern describes how to solve such problems:
- Encapsulate creating and assembling the parts of a complex object in a separate Builder object.
- A class delegates object creation to a Builder object instead of creating the objects directly.

Intent

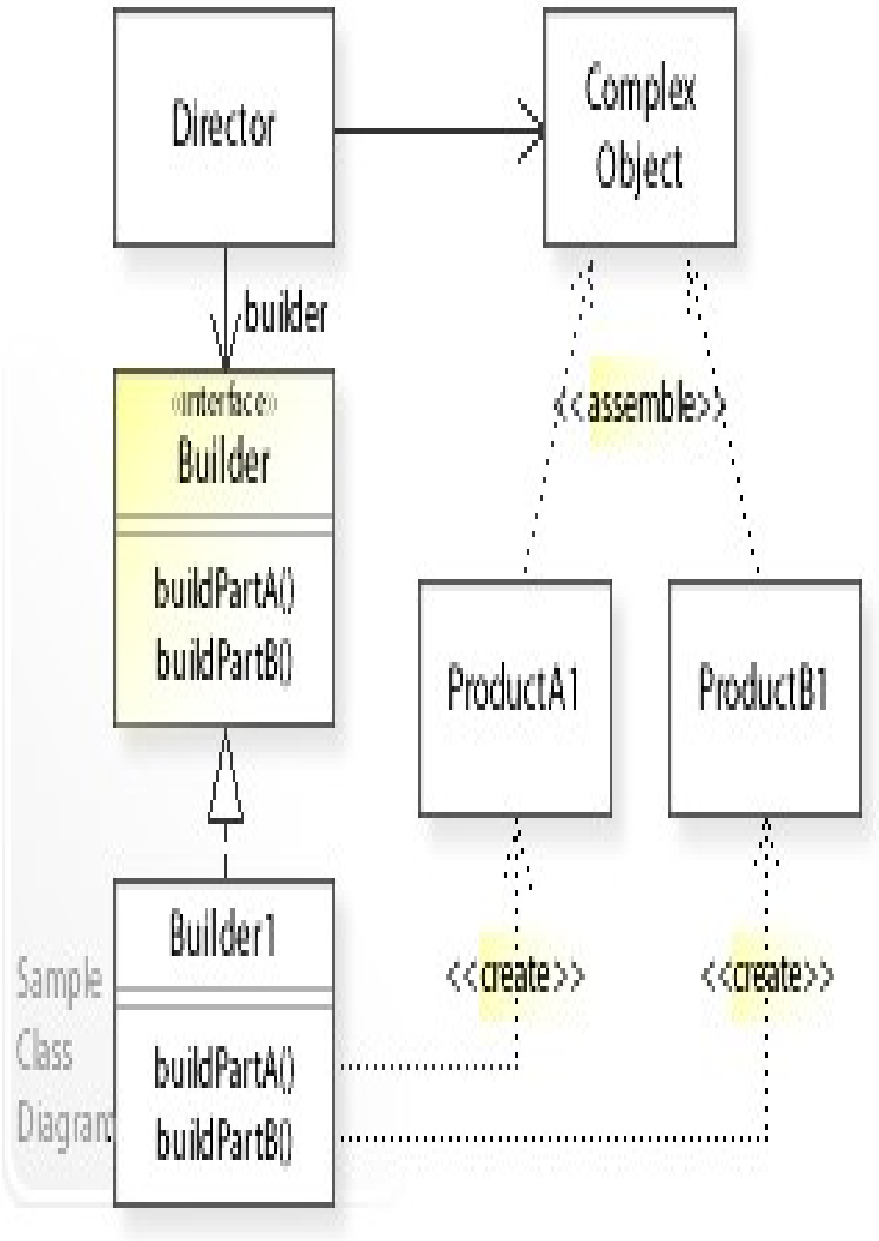
- The intent of the Builder design pattern is to separate the construction of a complex object from its representation. By doing so the same construction process can create different representations.

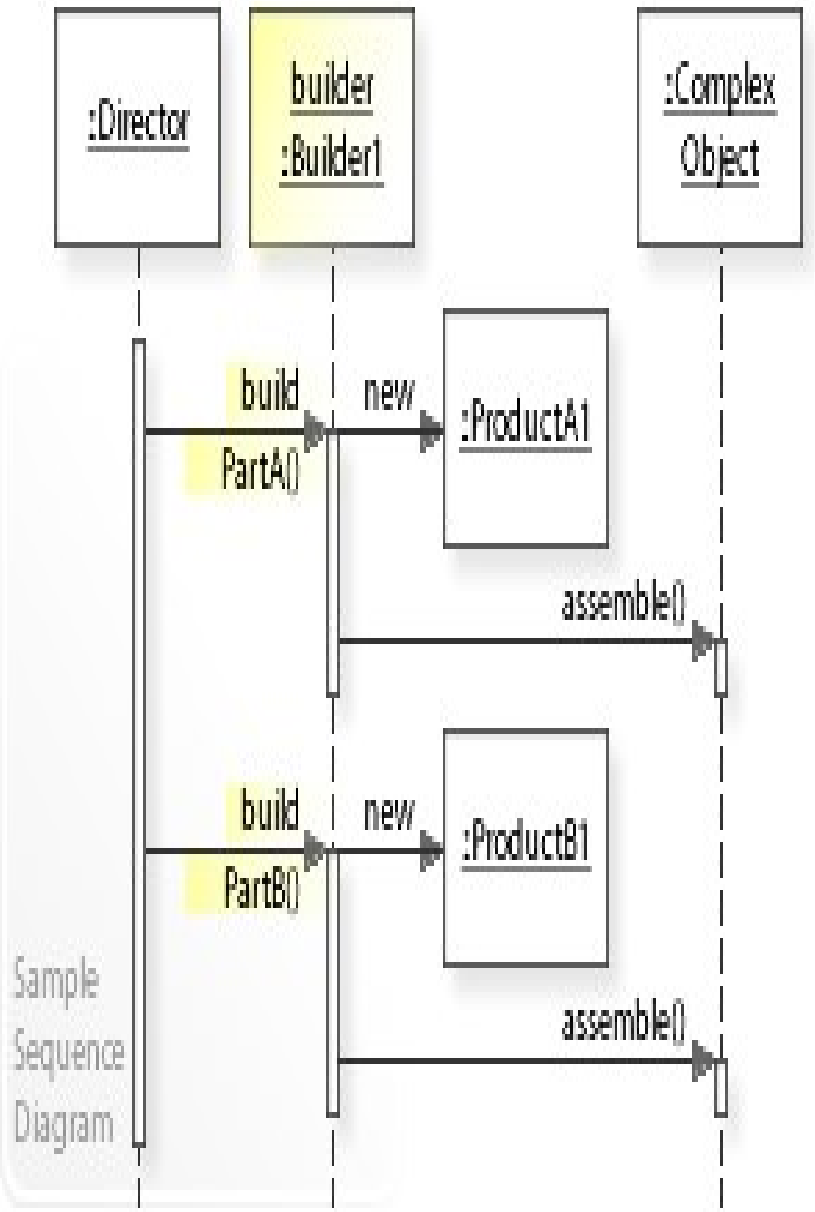
Advantages

- Advantages of the Builder pattern include:
  - Allows you to vary a product's internal representation.
  - Encapsulates code for construction and representation.
  - Provides control over steps of construction process.

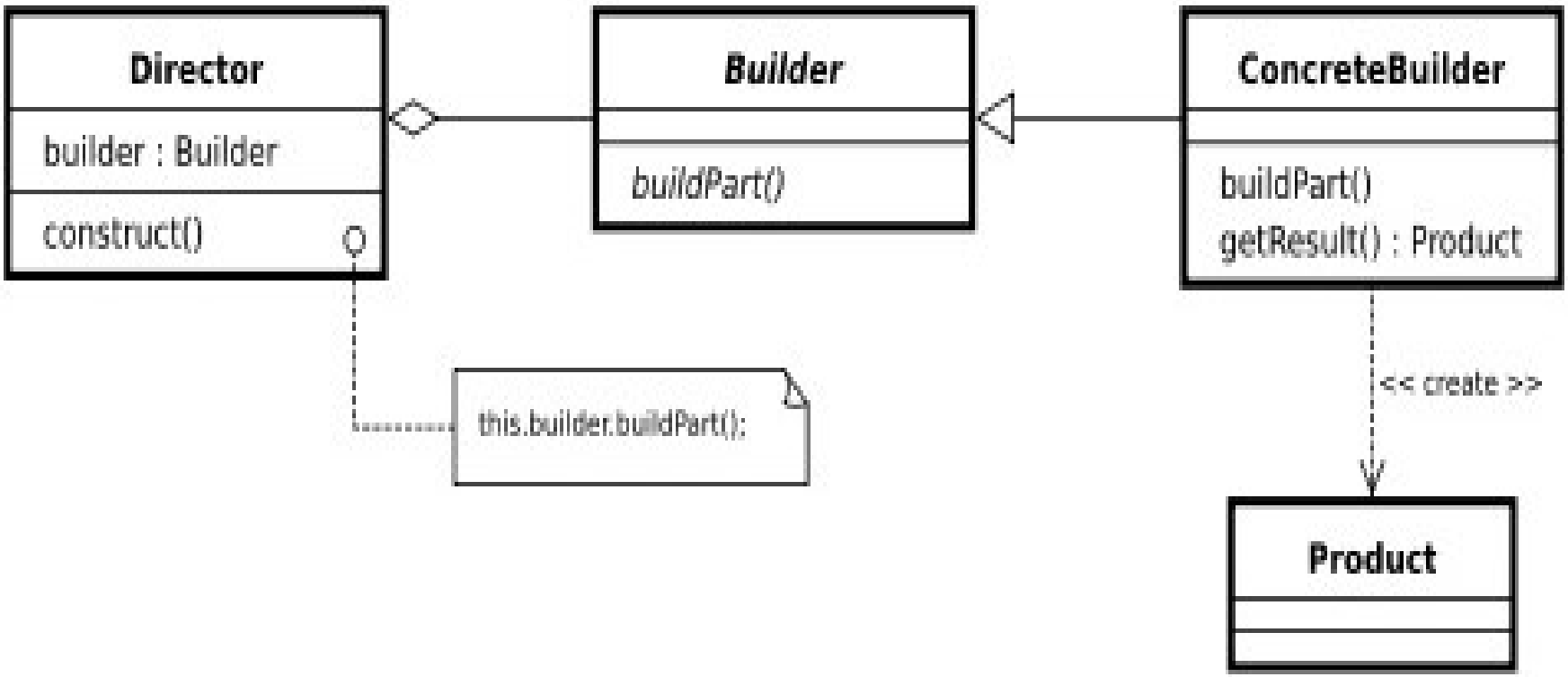
Disadvantages

- Disadvantages of the Builder pattern include:
  - Requires creating a separate ConcreteBuilder for each different type of product.
  - Requires the builder classes to be mutable.
  - Dependency injection may be less supported.


Director
builder
«interface»
Builder
buildPartA()
buildPartB()
Builder1
buildPartA()
buildPartB()
Complex
Object
«assemble»
ProductA1
ProductB1
«create»
«create»
Sample
Class
Diagram


:Director
builder
:Builder1
:Complex
Object
build
PartA()
new
:ProductA1
assemble()
build
PartB()
new
:ProductB1
assemble()
Sample
Sequence
Diagram

Class diagram



Represents a product created by the builder

```
public class Car
{
    public string Make { get; set; }
    public string Model { get; set; }
    public int NumDoors { get; set; }
    public string Colour { get; set; }

    public Car(string make, string model, string colour, int
     numDoors)
    {
        Make = make;        Model = model;
        Colour = colour;        NumDoors = numDoors;
    }
}
```

```
/// The builder abstraction
public interface ICarBuilder
{
    string Colour { get; set; }
    int NumDoors { get; set; }

    Car GetResult();
}
```

```
/// Concrete builder implementation
public class FerrariBuilder : ICarBuilder
{
    public string Colour { get; set; }
    public int NumDoors { get; set; }

    public Car GetResult()
    {
        return NumDoors == 2 ? new Car("Ferrari",
    "488 Spider", Colour, NumDoors) : null;
    }
}
```

```
/// The director
public class SportsCarBuildDirector
{
    private ICarBuilder _builder;
    public SportsCarBuildDirector(ICarBuilder builder)
    {
        _builder = builder;
    }

    public void Construct()
    {
        _builder.Colour = "Red";
        _builder.NumDoors = 2;
    }
}
```

```
public class Client
{
    public void DoSomethingWithCars()
    {

        var builder = new FerrariBuilder();
        var director = new
    SportsCarBuildDirector(builder);

        director.Construct();
        Car myRaceCar = builder.GetResult();
    }
}
```

References

- https://en.wikipedia.org/wiki/Builder_pattern