Creational Patterns

Agenda

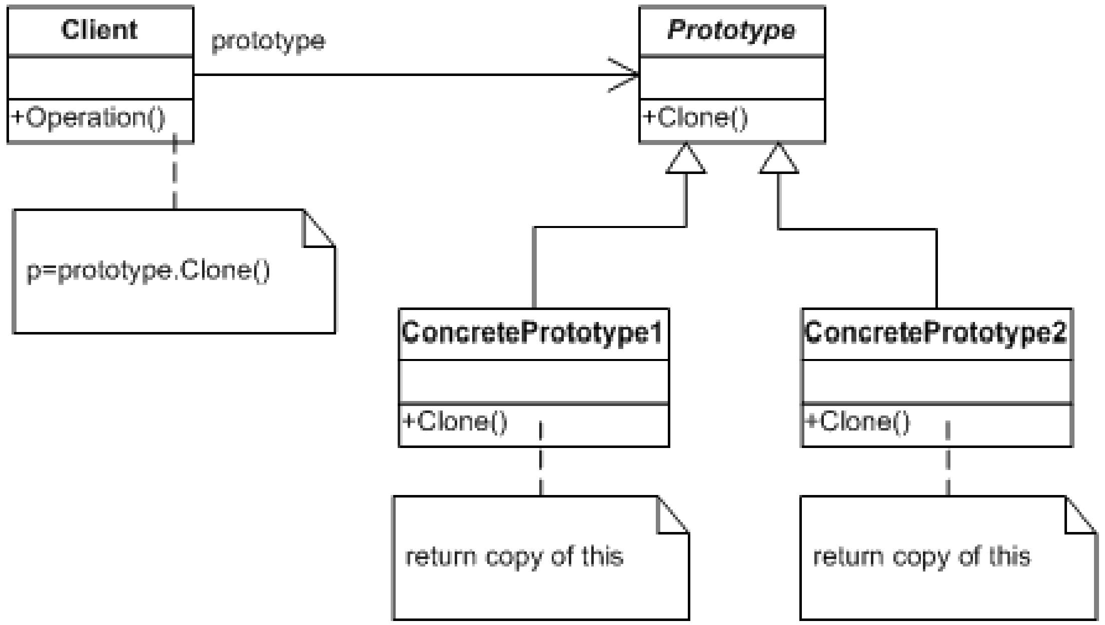
- Prototype
- Singleton
  - Simple - Not Thread-Safe
  - Simple Thread-Safety
  - Thread Safety with Double Checking
  - Thread-Safe, No Lock, Half-Lazy
  - Fully-Lazy and Safe Instantiation
  - Generic Solution
- Factory
  - Static Factory Class (Simple Factory)
  - Polymorphic Factory Class (Factory Object)
  - Factory Method
  - Abstract Factory
- Builder

Prototype

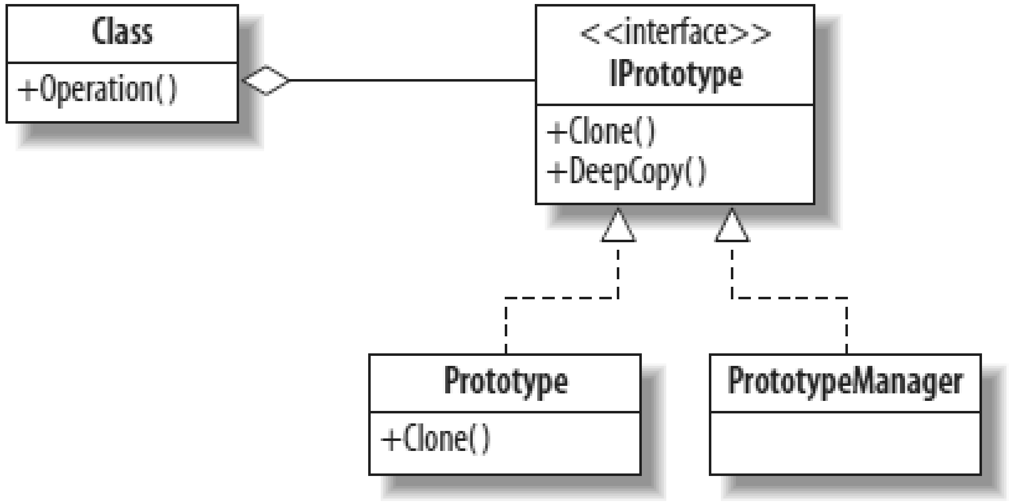
- Definition:
  Specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Participant
  - Prototype – A class that provides a Clone() and a DeepClone()
  - ConcretePrototype – A concrete type that can be prototyped (implements a method for cloning and deep-cloning itself)
  - PrototypesRepository – Holds the prototypes
  - Client – Uses the prototypes and create new objects by cloning prototypes
- Motivation
  - The Prototype pattern co-opts one instance of a class and uses it as a “breeder” for all future instances.
  - Prototypes are useful when object initialization is expensive and you anticipate few variations on the initialization parameters. In this context, the Prototype pattern can avoid expensive “creation from scratch,” supporting cheap cloning of a preinitialized prototype.
  - However, cloning via serialization is not particularly cheap itself, so it is worth considering shallow cloning if you are absolutely sure you have a flat, single-level data structure.

Prototype – Class Diagrams

- Gang-Of-4

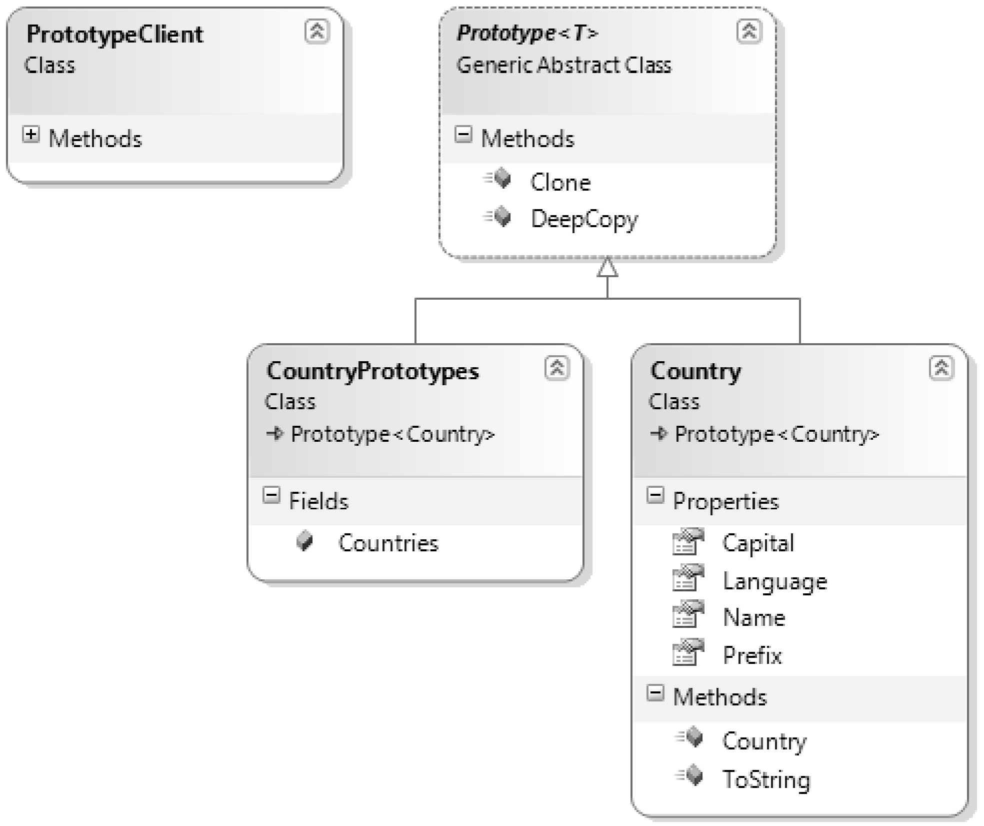


- Improvement



Prototype – Using C# 3.0

- Generics
- Object.MemberwiseClone()
- Serialization
- Example Oriented Features:
  - Automatic Properties
  - Type Initializers



Factories

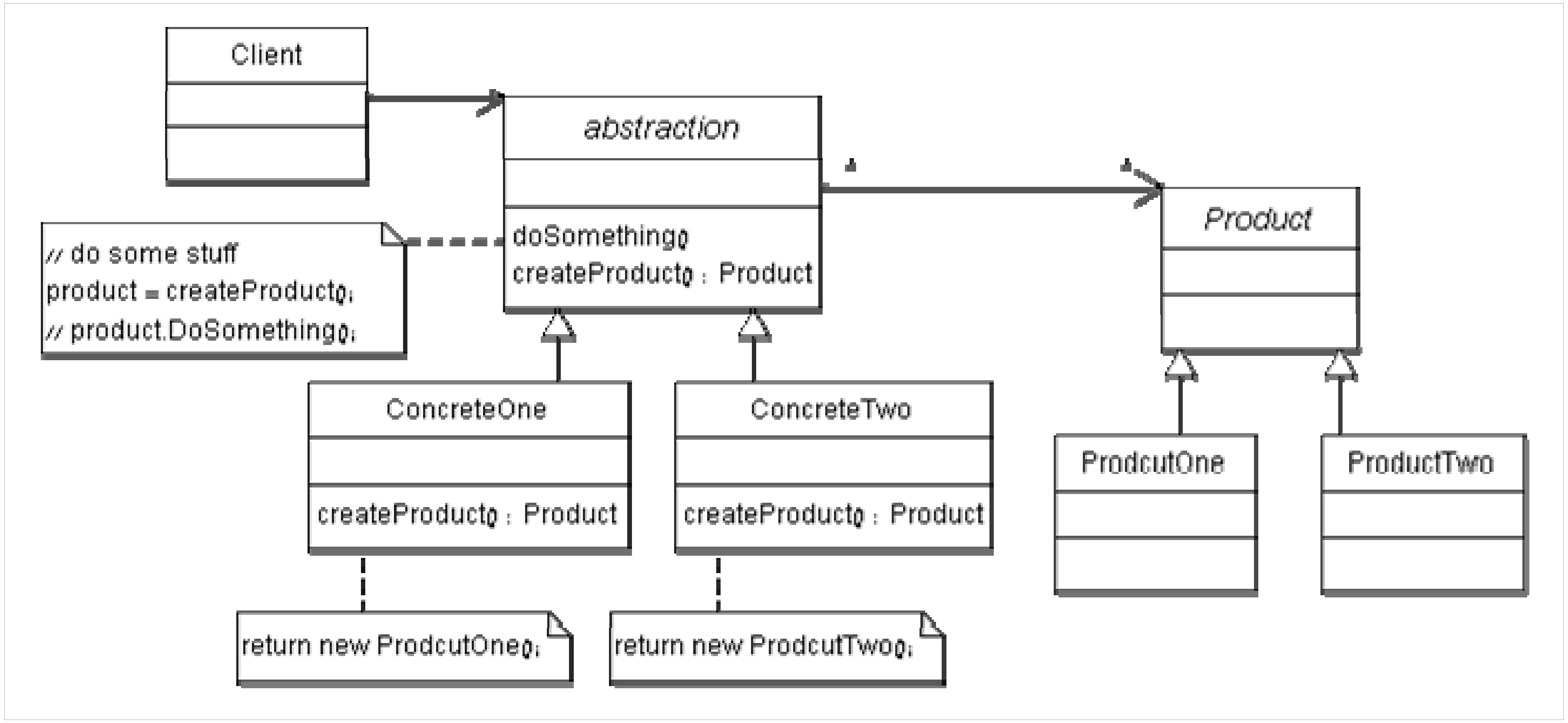
- **Factory Methods**
  - Static Factory Class
    A class that has a static method that creates an instance of one of the derived classes
    - Another Version: A Static Factory Method
      in the abstract base product class
  - Factory Method
    An abstract class that does all the job except creating a product instance which is done in an abstract method.
    Derived classes implements this method for concrete creation.
- **Abstract Factories**
  - A Polymorphic Factory Class (Factory Object)
    A base factory which can be derived to specific factory
    (very similar to Abstract Factory).
  - An abstract factory class
    serves as a factory to a verity of objects of a family of products

Factory Method

- **Participants**
  - Product (Page)
    Defines the interface of objects the factory method creates
  - ConcreteProduct (SkillsPage, EducationPage)
    Implements the Product interface
  - Creator (Document)
    Declares the factory method, which returns an object of type Product.
  - ConcreteCreator (Report, Resume)
    Overrides the factory method to return an instance of a ConcreteProduct.

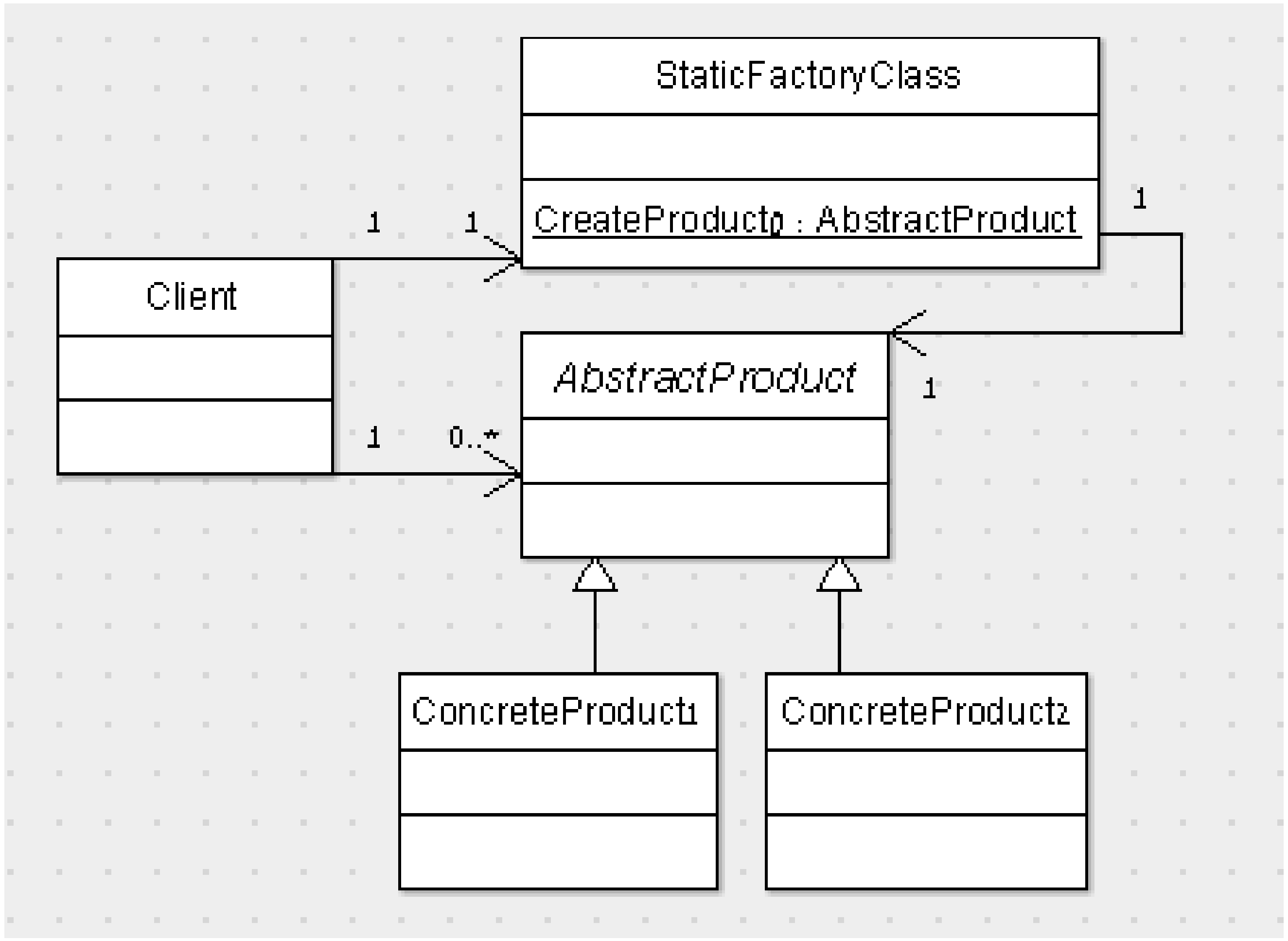
Factory Methods

- **Factory Method - Class Diagram**



Factory Methods

- **Static Factory Class - Class Diagram**



Builder

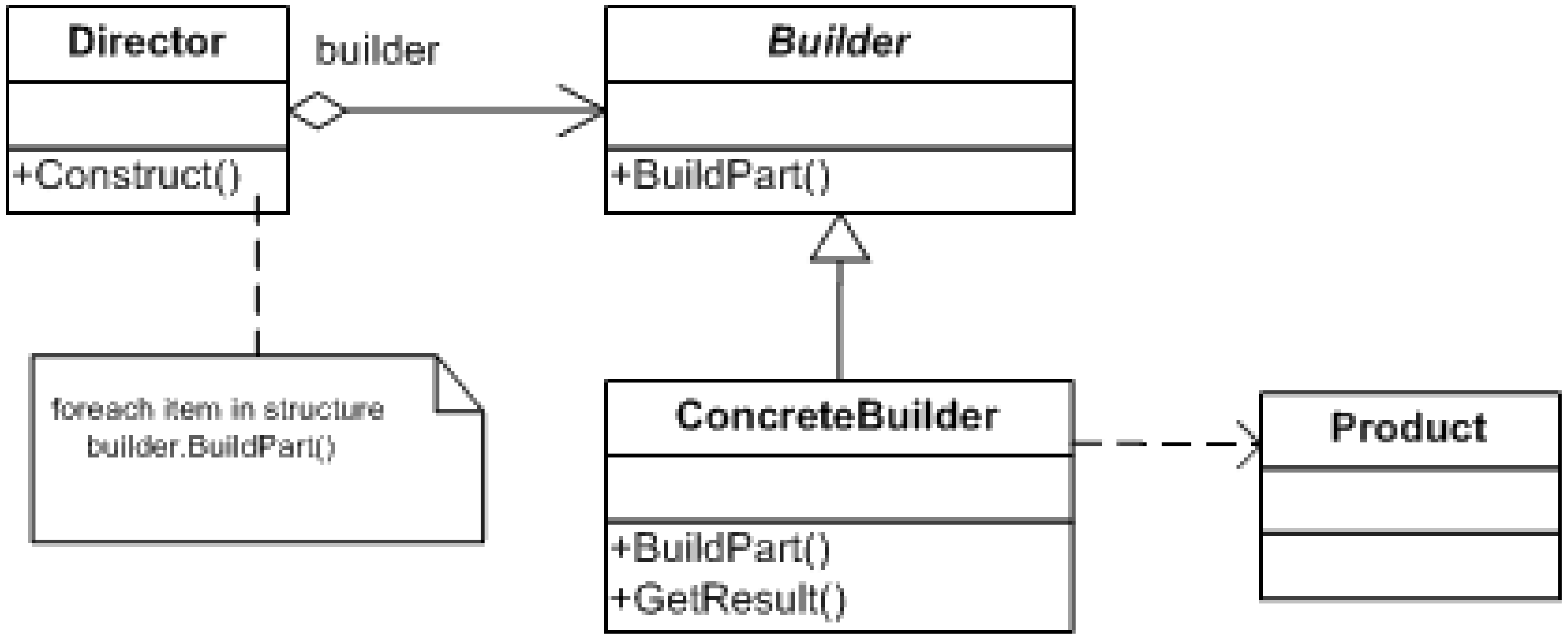
- **Motivation and Principles**
  - Separate the construction of a complex object from its representation.
    [GoF, p97]
  - Parse a complex representation, create one of several targets.
  - Affords finer control over the construction process.
  - Unlike creational patterns that construct products in one shot, the **Builder** pattern constructs the product step by step under the control of the "composer"..
  - Non-Software example:
    McDonald’s (composing meals)

Builder

- **Participants**
  - **Director/Composer** (CarShop)
    Constructs an object using the Builder interface
  - **Builder / Creator** (CarBuilder)
    Specifies an abstract interface for creating parts of a Product
  - **Concrete Builder** / Creator
    (HondaBuilder, ToyotaBuilder, MazdaBuilder)
    - Constructs and assembles parts of the product by implementing the Builder interface
    - Defines and keeps track of the representation it creates
  - **Product** (Car)
    - Represents the complex object under construction. ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled
    - includes classes that define the constituent parts, including interfaces for assembling the parts into the final result

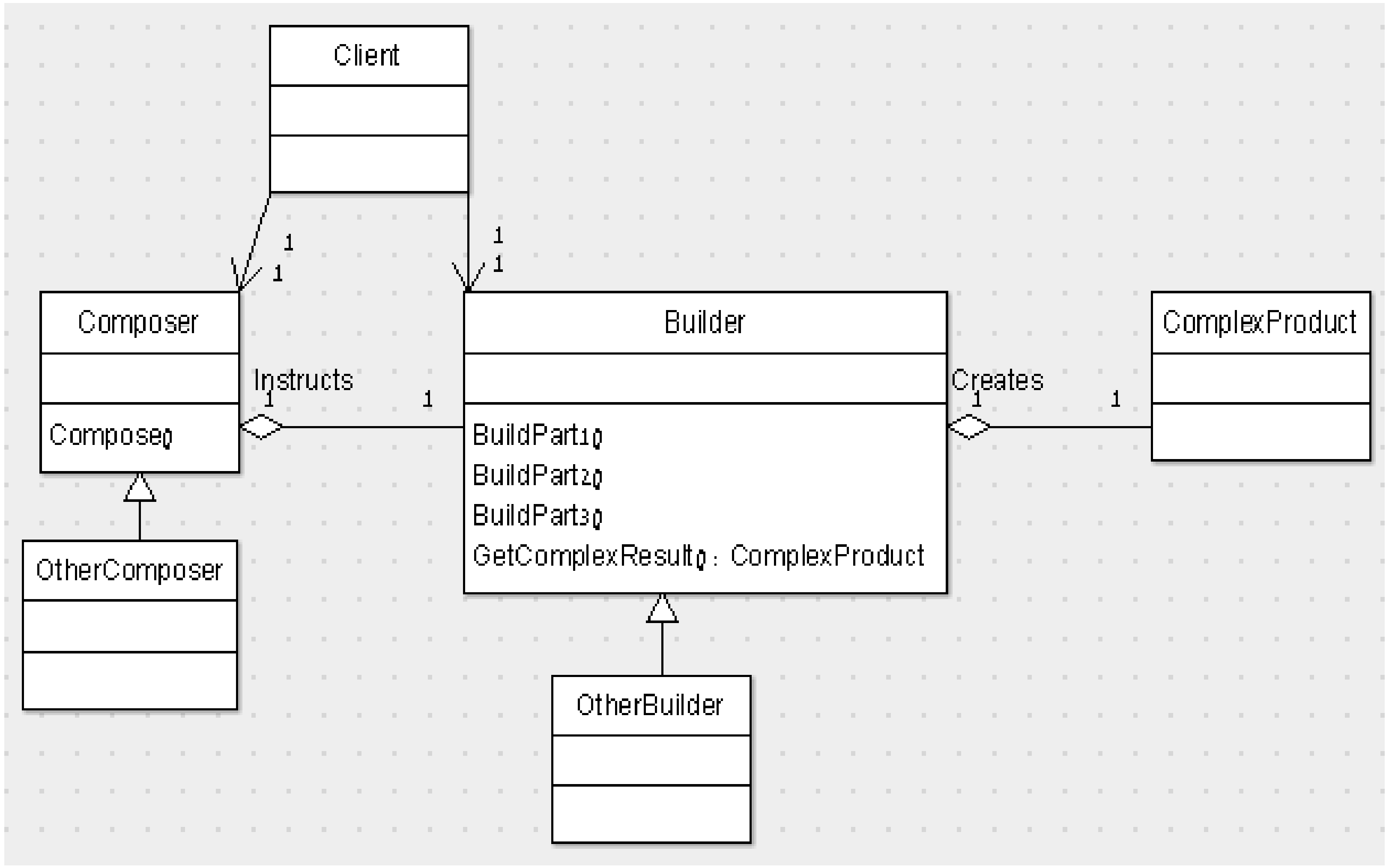
Builder

- **Class Diagram - Original**



Builder

Class Diagram - Alternative



Creational Patterns - Summary

- The **Factory** pattern is used to choose and return an instance of a class from a number of similar classes, based on data you provide to the factory.
- The **Abstract Factory** pattern is used to return one of several groups of classes. In some cases, it actually returns a Factory for that group of classes.
- The **Builder** pattern assembles a number of objects to make a new object, based on the data with which it is presented. Frequently, the choice of which way the objects are assembled is achieved using a Factory.
- The **Prototype** pattern copies or clones an existing class, rather than creating a new instance, when creating new instances is more expensive.
- The **Singleton** pattern ensures that there is one and only one instance of an object and that it is possible to obtain global access to that one instance.