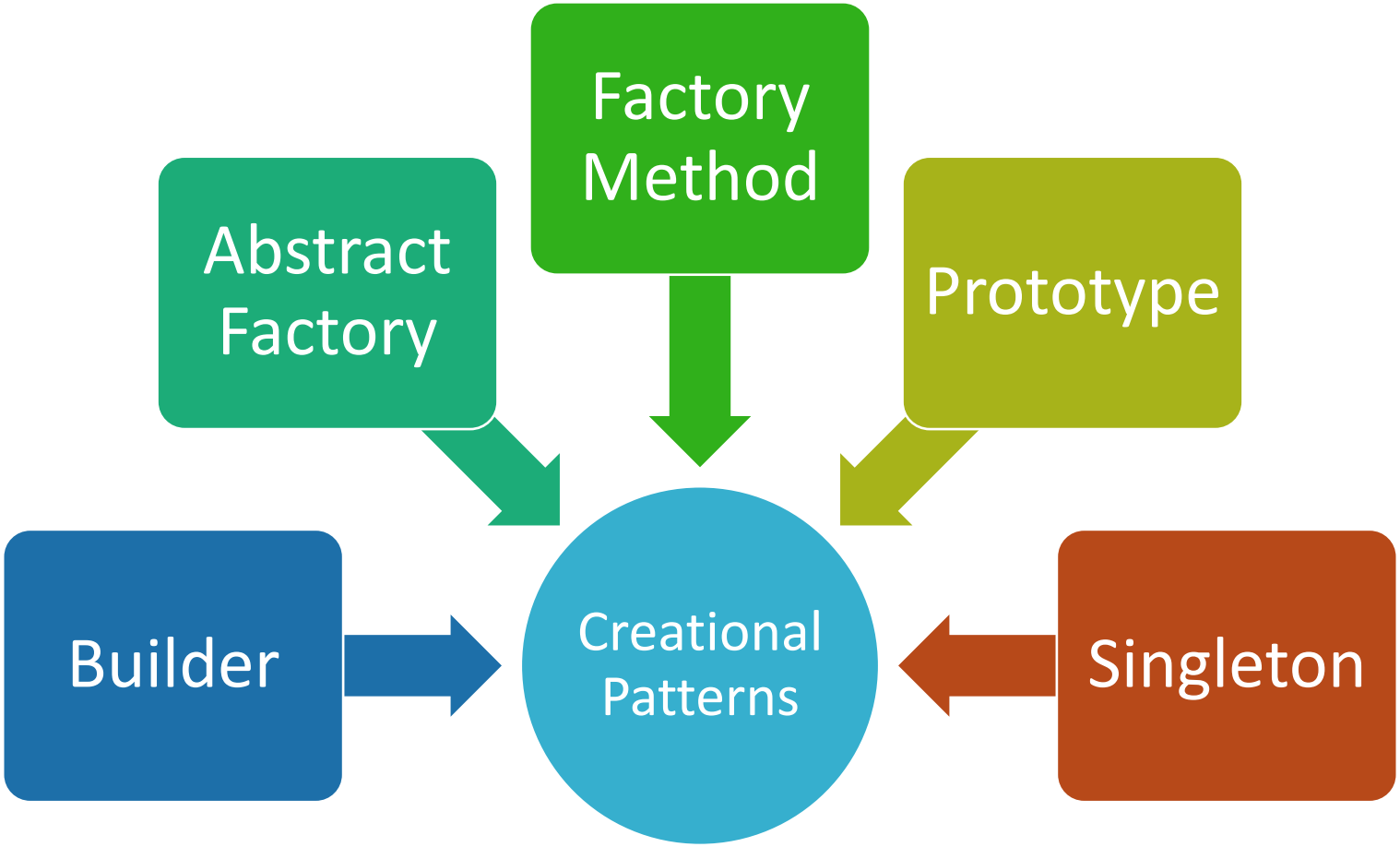
Design Patterns

Creational Patterns

Creational Patterns



Builder

Builder

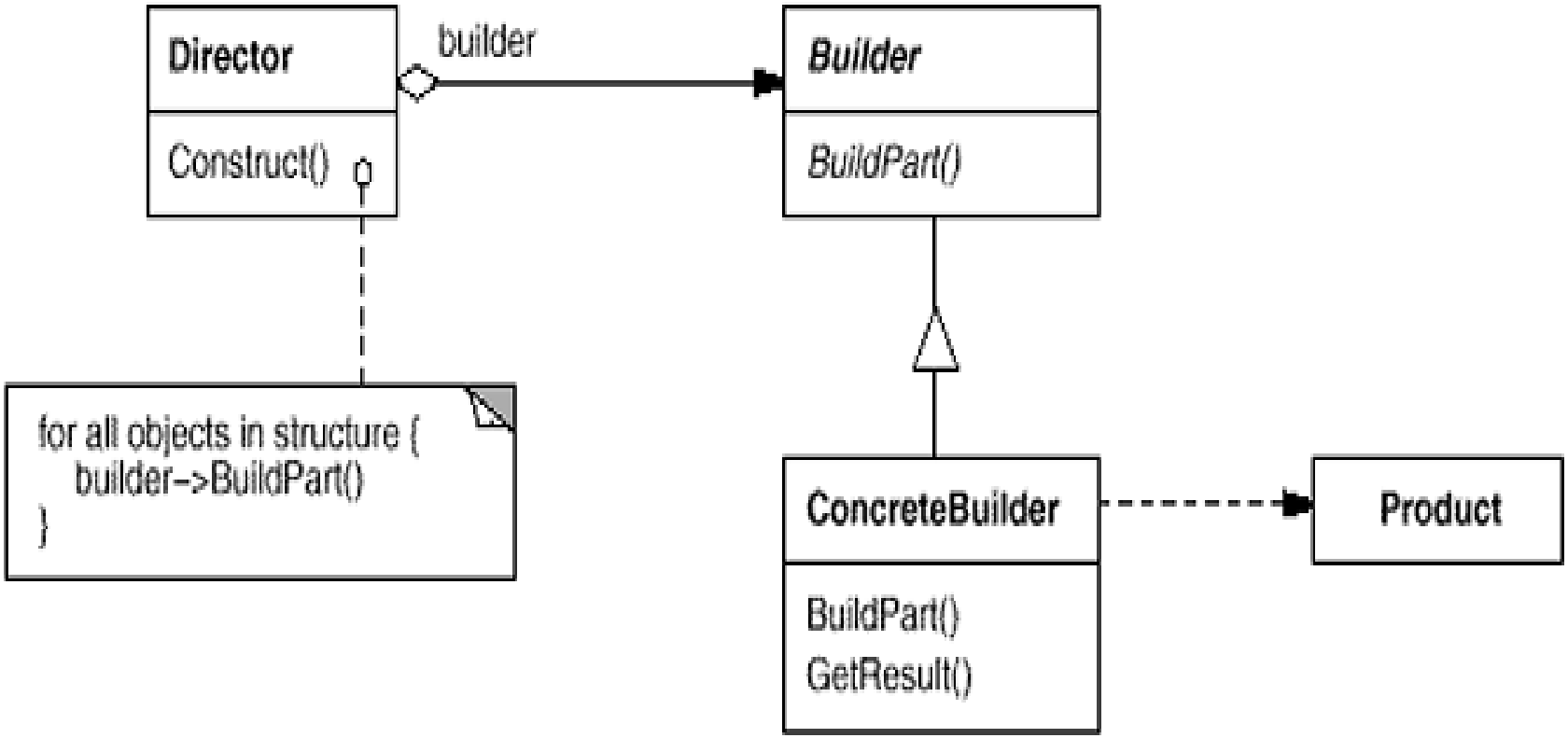
Intent

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Applicability

- the algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.
- the construction process must allow different representations for the object that's constructed

Builder - Structure



Builder - Participants

Builder	• Specifies an abstract interface for creating parts of a Product object
ConcreteBuilder	• constructs and assembles parts of the product by implementing the Builder interface • defines and keeps track of the representation it creates • provides an interface for retrieving the product
Director	• constructs an object using the Builder interface
Product	• represents the complex object under construction • includes classes that define the constituent parts

Builder - Consequences

Collaborations

- The client creates the Director object and configures it with the desired Builder object.
- Director notifies the builder whenever a part of the product should be built.
- Builder handles requests from the director and adds parts to the product.
- The client retrieves the product from the builder.

Consequences

- Lets you vary products internal representation
- Isolates code for construction and representation
- Gives finer control over construction process

Demo

Abstract Factory

Abstract Factory

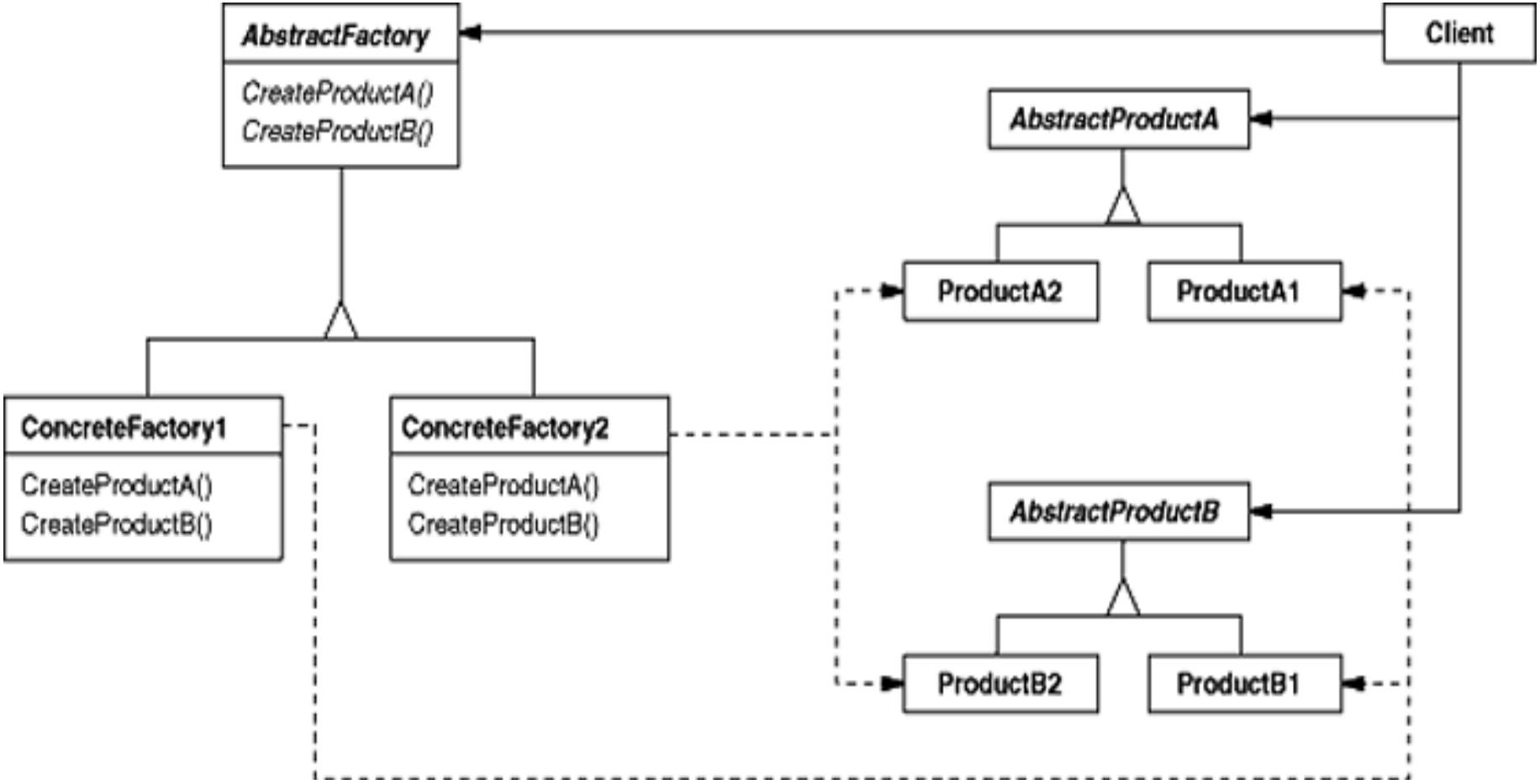
Intent

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes
- Also known as **“Kit”**

Applicability

- A system should be independent of how its products are created, composed, and represented
- A system should be configured with one of multiple families of products
- A family of related product objects is designed to be used together, and you need to enforce this constraint
- You want to provide a class library of products, and you want to reveal just their interfaces, not their implementations

Abstract Factory - Structure



Abstract Factory - Participants

AbstractFactory	• Declares an interface for operations that create abstract product objects
ConcreteFactory	• Implements the operations to create concrete product objects
AbstractProduct	• Declares an interface for a type of product object
ConcreteProduct	• Defines a product object to be created by the corresponding concrete factory • Implements the AbstractProduct interface
Client	• Uses only interfaces declared by AbstractFactory and AbstractProduct classes

Abstract Factory

Consequences

- Isolates concrete classes
- Makes exchanging product families easy
- Promotes consistency among products
- Supporting new kinds of products difficult.

Implementation – useful techniques

- Factories as Singleton
- Parameterization as a way of controlling interface size
- Configuration with Prototypes

Demo

Factory Method

Factory Method

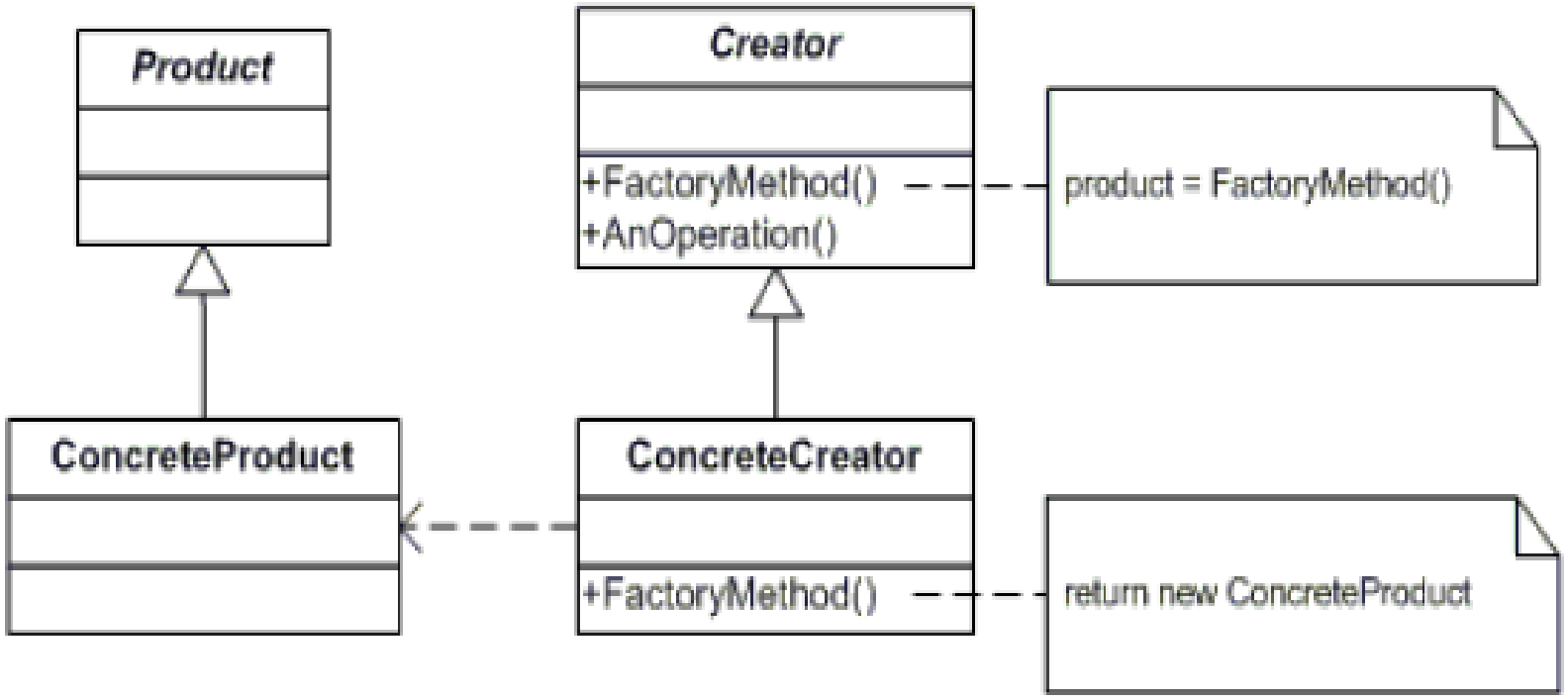
Intent

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Also known as **“Virtual Constructor”**

Applicability

- A class can’t anticipate the 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 - Structure



Factory Method - Participants

Product	• Defines the interface of objects the factory method creates.
ConcreteProduct	• Implements the Product interface
Creator	• Declares the factory method, which returns an object of type Product. • May call the factory method to create a Product object.
ConcreteCreator	• Overrides the factory method to return an instance of a ConcreteProduct

Factory Method

Consequences

- Provides hooks for subclasses
- Connects parallel class hierarchies

Implementation

- Creator class is can be abstract or concrete
- Parameterized factory method
- Language-specific variants and issues
- Naming Conventions

Demo

Singleton

Singleton

Intent

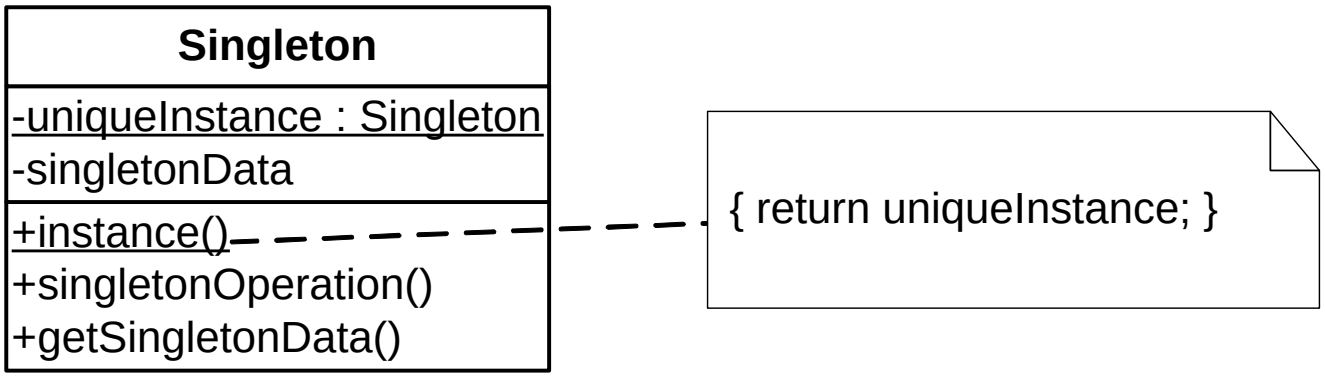
- Ensure a class only ever has one instance, and provide a global point of access to it.

Applicability

- When there must be exactly one instance of a class, and it must be accessible from a well-known access point
- When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code

Singleton (cont'd)

- Structure



Singleton - Participants

Singleton

- Defines an Instance operation that lets clients access its unique instance.
- Instance is a class operation
- May be responsible for creating its own unique instance

Singleton

Consequences

- Reduces name space pollution
- Controlled access to sole instance
- Allow extension by subclassing
- Implementation may be less efficient than a global
- Concurrency pitfalls

Implementation

- Ensuring a unique instance
- Static instance operation

Demo

Prototype

Prototype

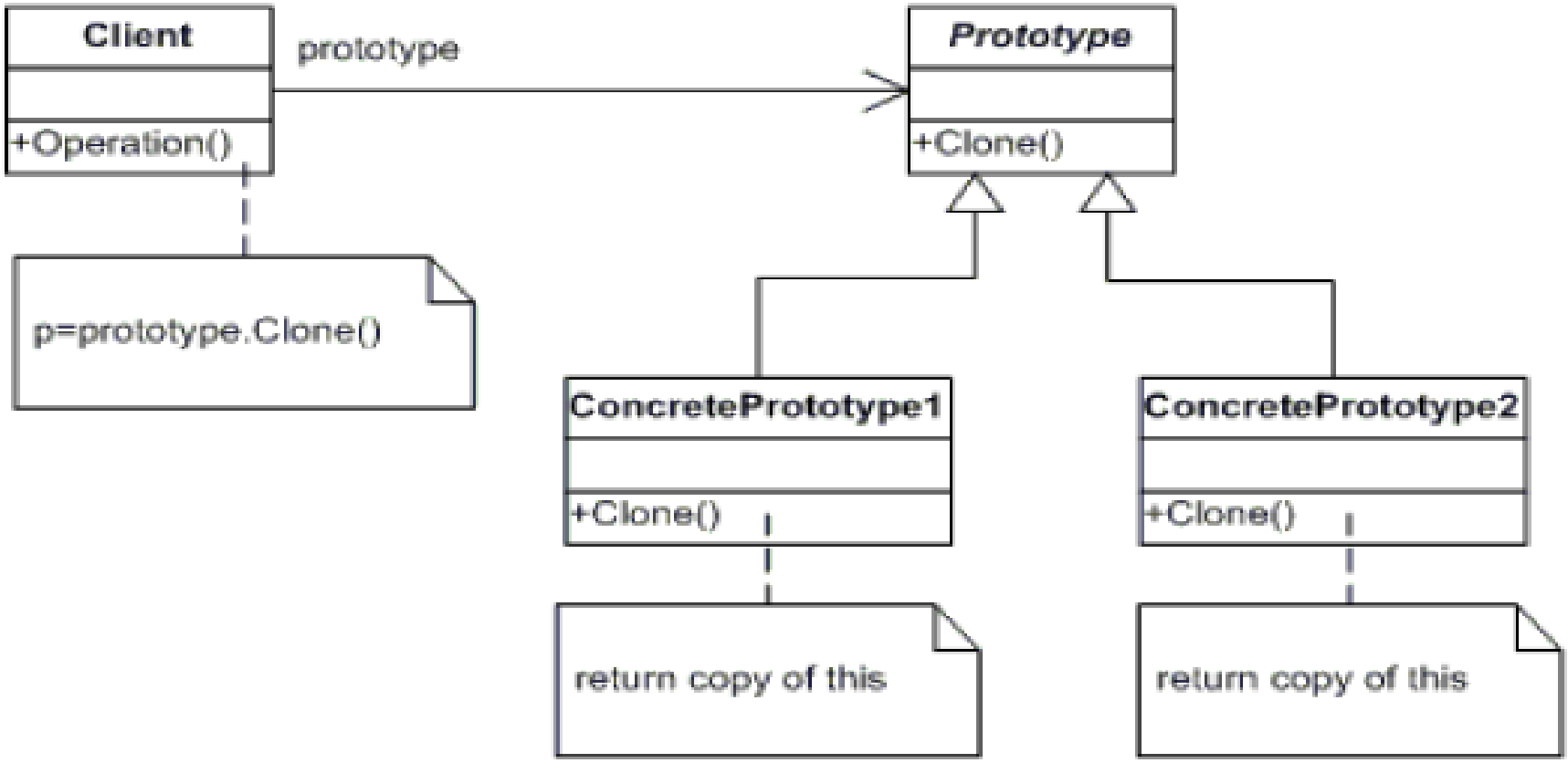
Intent

- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

Applicability

- When the classes to instantiate are specified at run-time, for example, by dynamic loading
- To avoid building a hierarchy of factories that parallels the class hierarchy of products
- When instances of a class can have one of only a few different combinations of state

Prototype - Structure



Prototype - Participants

Prototype	• Declares an interface for cloning itself.
ConcretePrototype	• implements an operation for cloning itself
Client	• Creates a new object by asking a prototype to clone itself

Prototype

Consequences

- Adding and removing prototypes at run-time
- Specifying new objects by varying values
- Specifying new objects by varying structures
- Reduced subclassing
- Configuring an application with classes dynamically.

Implementation

- Client code must provide an existing prototype object.
- Using a manager prototype
- Implementing the clone operation

Demo