

Creational Pattern: Prototype



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com

Coming Up



Describing the prototype pattern

Structure of the prototype pattern

Implementation

- Real-life sample: cloning an Employee and Manager, deriving from Person

Deep copy versus shallow copy

IClonable interface



Coming Up



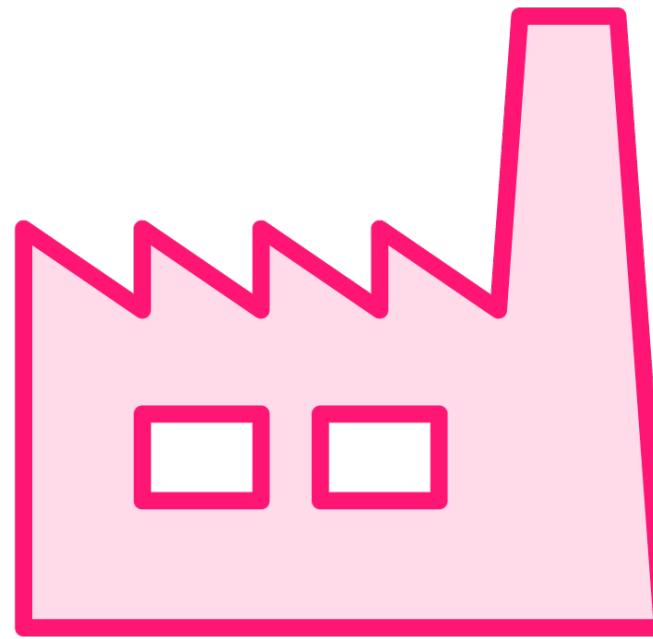
Use cases for this pattern

Pattern consequences

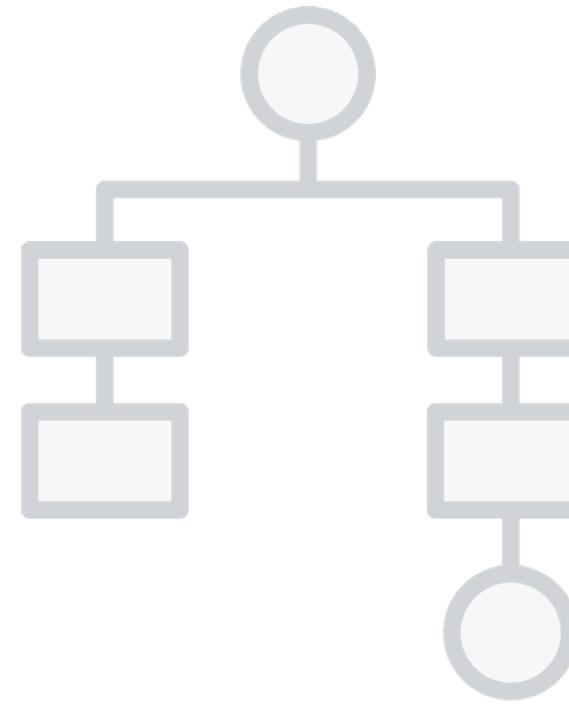
Related patterns



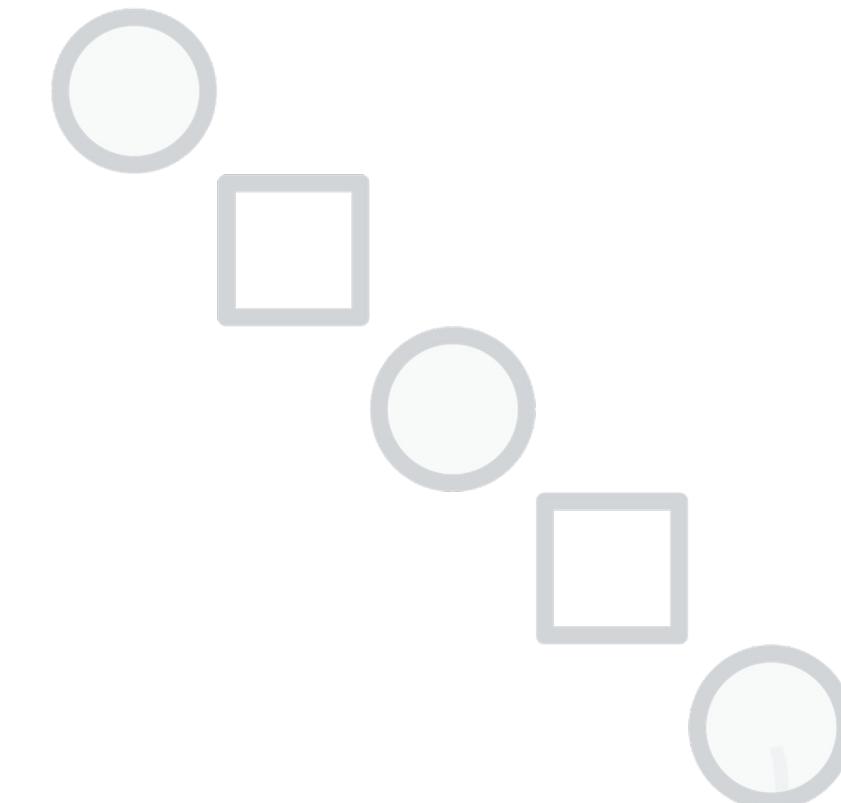
Describing the Prototype Pattern



Creational



Structural



Behavioral



Prototype

The intent of this pattern is to specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype



```
var existingManager = new Manager("Cindy");
var existingEmployee = new Employee("Kevin", existingManager);

var newEmployee = new Employee();
newEmployee.Name = existingEmployee.Name;
newEmployee.Manager = existingEmployee.Manager;
```

Describing the Prototype Pattern



```
var existingManager = new Manager("Cindy");
var existingEmployee = new Employee("Kevin", existingManager);

var newEmployee = new Employee();
newEmployee.Name = existingEmployee.Name;
newEmployee.Manager = existingEmployee.Manager;
```

Describing the Prototype Pattern



```
var existingManager = new Manager("Cindy");
var existingEmployee = new Employee("Kevin", existingManager);

var newEmployee = new Employee();
newEmployee.Name = existingEmployee.Name;
newEmployee.Manager = existingEmployee.Manager;
```

Describing the Prototype Pattern

Requires intrinsic knowledge of concrete classes, and how to create them



Describing the Prototype Pattern

Manager

Employee



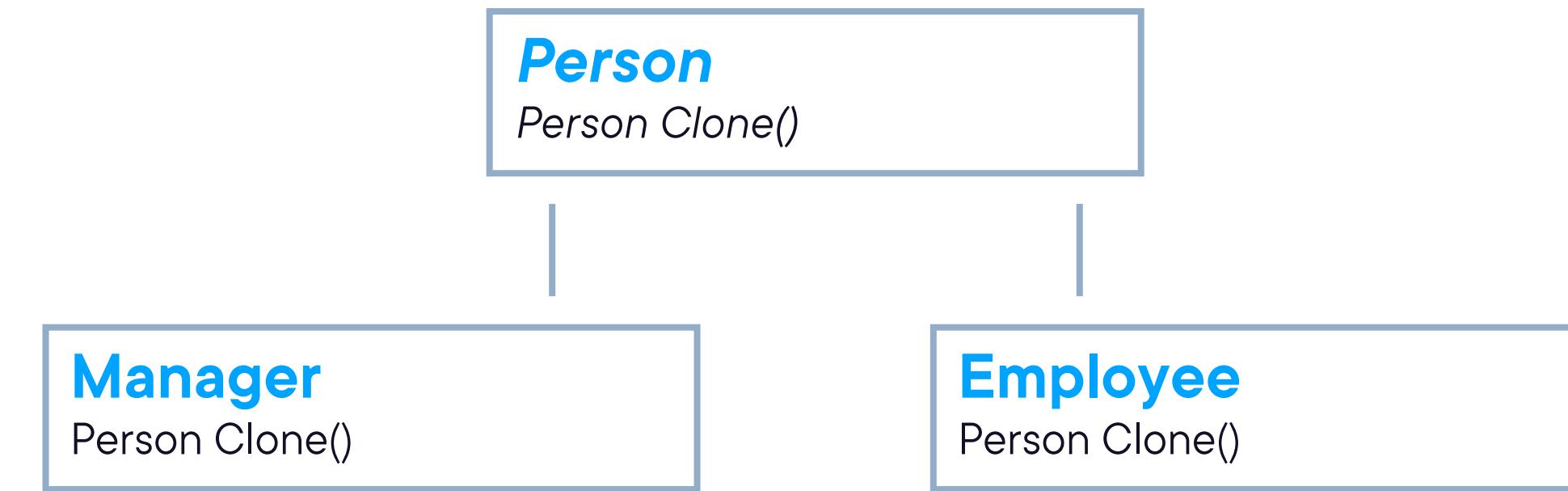
Describing the Prototype Pattern

Manager
Clone()

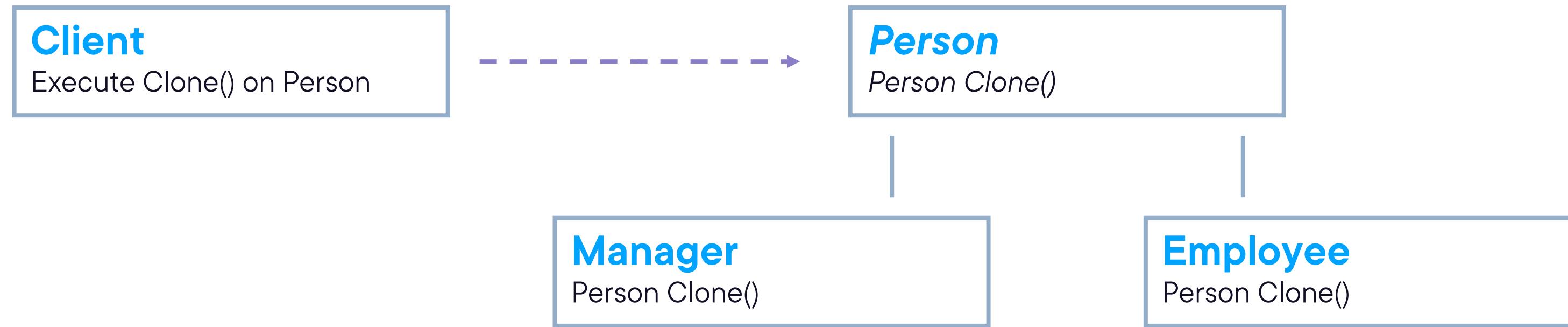
Employee
Clone()



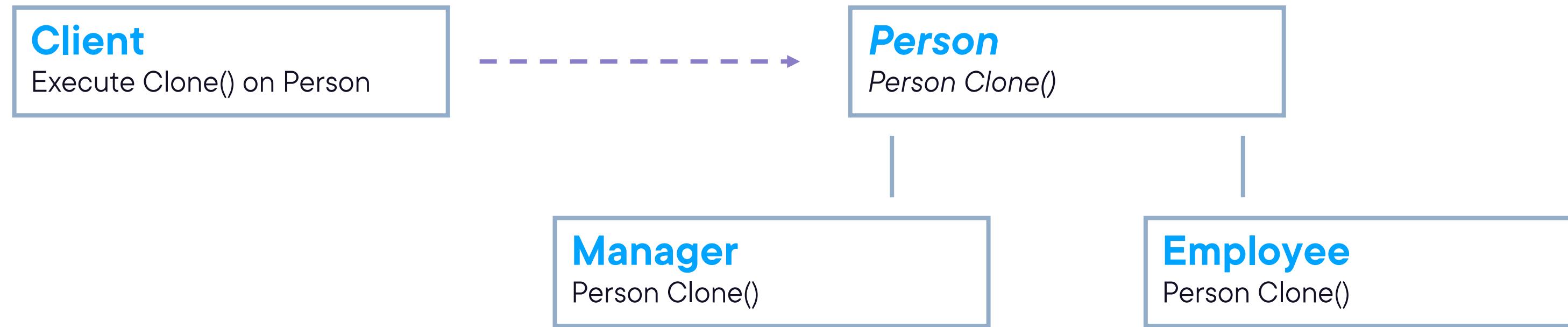
Describing the Prototype Pattern



Describing the Prototype Pattern



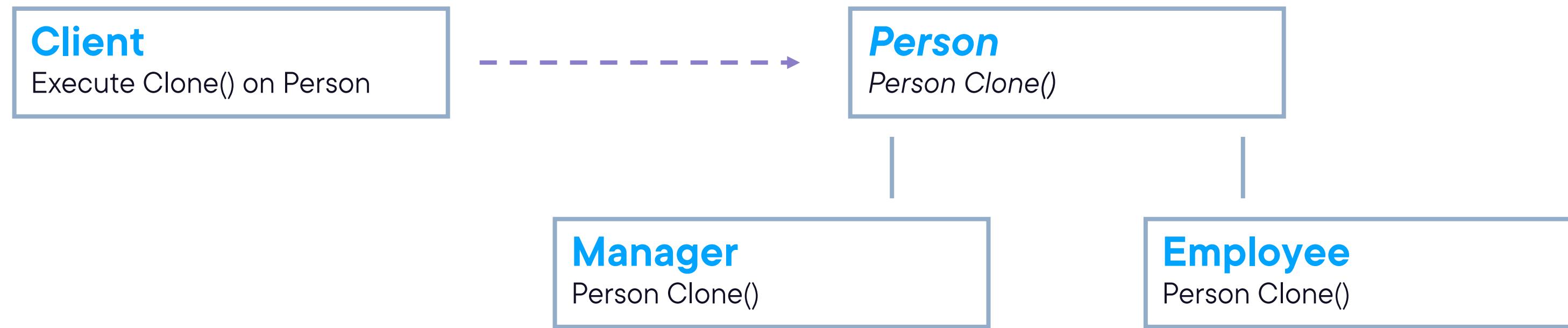
Prototype Pattern Structure



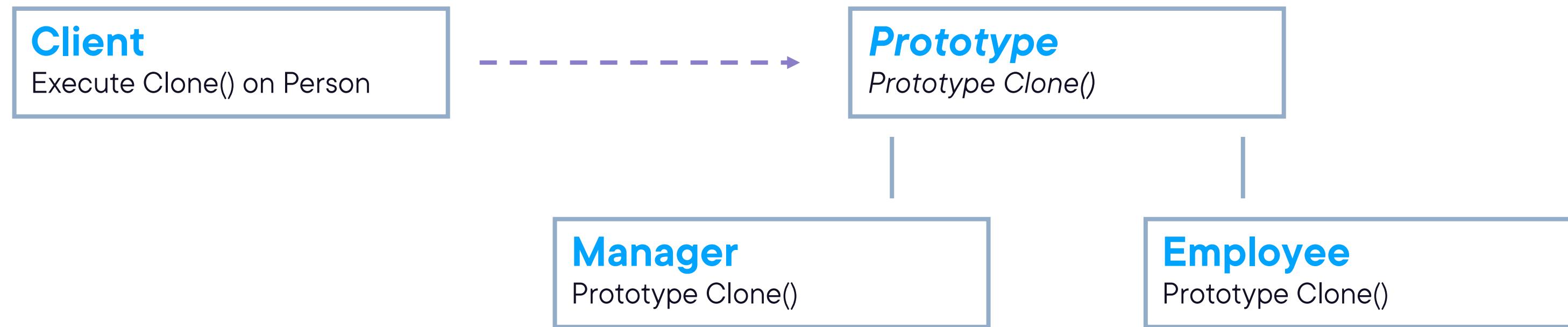
Prototype declares an interface for cloning itself



Prototype Pattern Structure



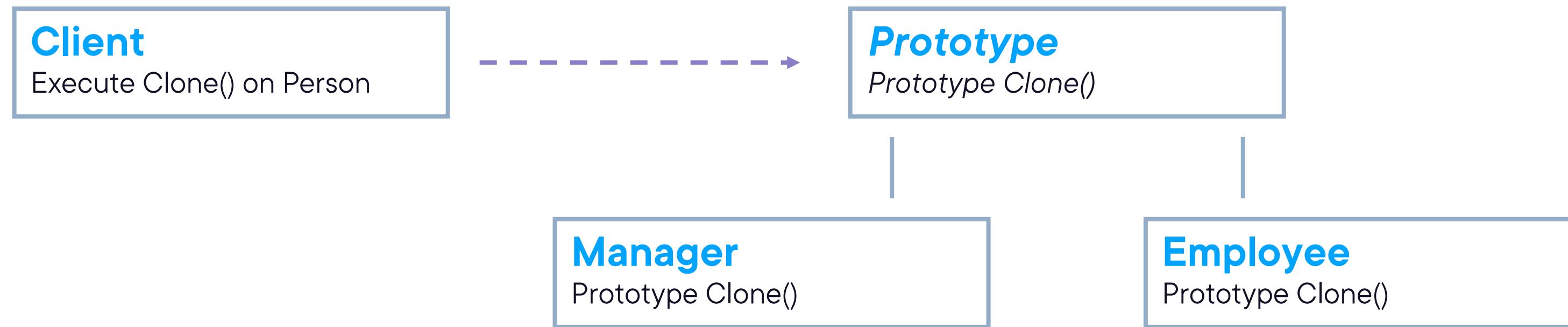
Prototype Pattern Structure



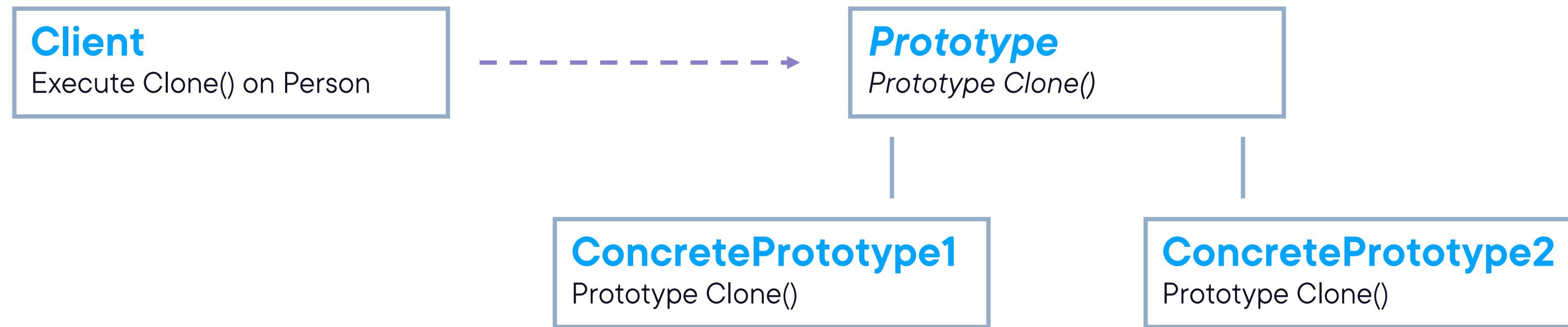
**ConcretePrototype
implements an operation for
cloning itself**



Prototype Pattern Structure



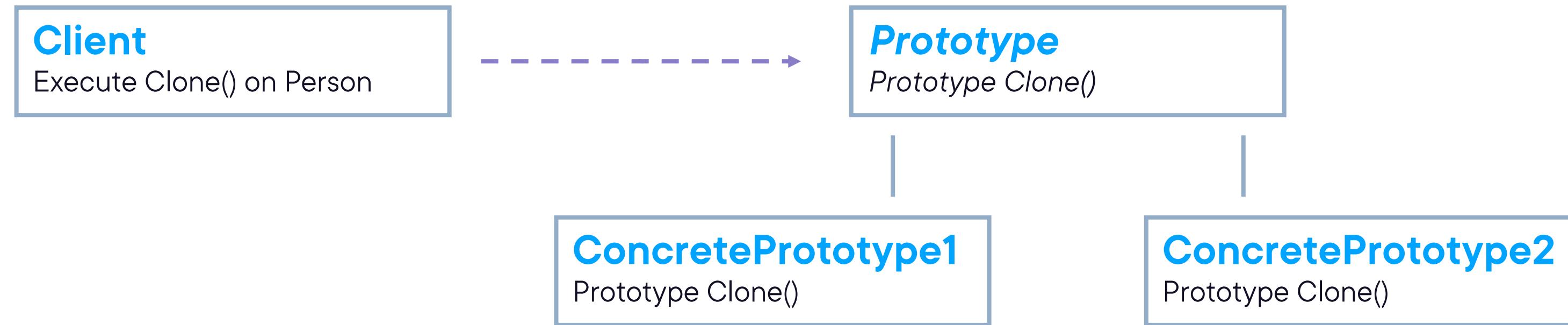
Prototype Pattern Structure



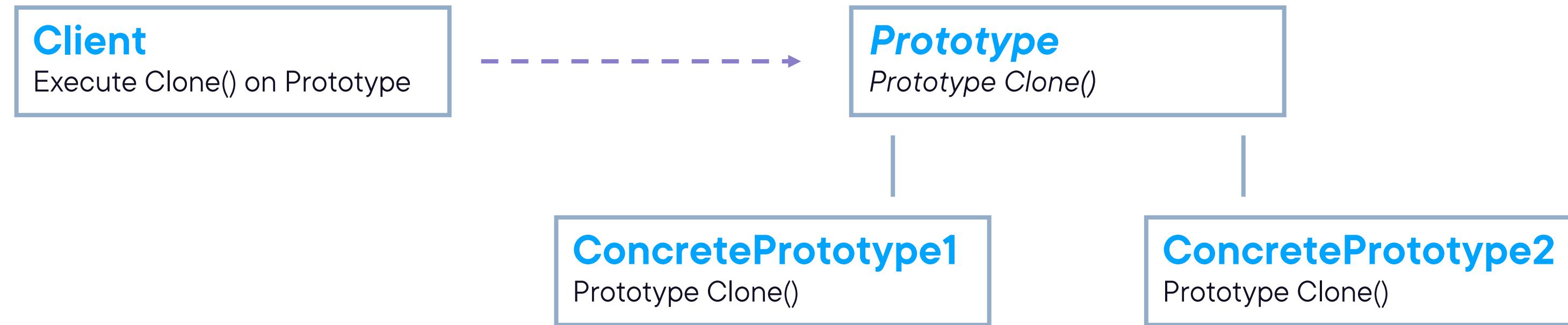
**Client creates a new
object by asking a
Prototype to clone itself**



Prototype Pattern Structure



Prototype Pattern Structure



Demo



Implementing the prototype pattern



Shallow Copy vs. Deep Copy

Shallow copy

vs

Deep copy

Copy of primitive type values

**Complex type values will be
shared across clones**

**Copy of primitive type values and
complex type values**



Demo



Supporting deep copies

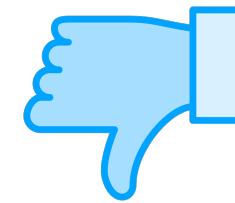


What About the ICloneable Interface?

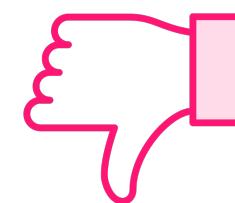
ICloneable enables us to provide a customized implementation that creates a copy of an existing object



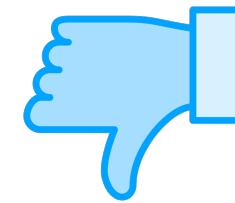
What About the ICloneable Interface?



It does not specify whether the cloning operation performs a deep copy, a shallow copy, or something in between



It doesn't require all property values of the original instance to be copied to the new instance



It returns an object, which means the client could need an additional cast



Use Cases for the Prototype Pattern



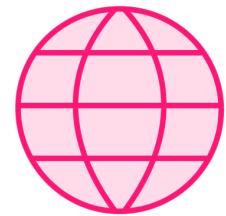
When a system should be independent of how its objects are created, and to avoid building a set of factories that mimics the class hierarchy



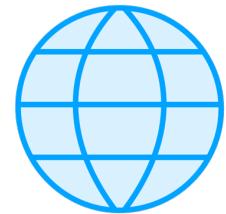
When a system should be independent of how its objects are created, and when instances of a class can have one of only a few different combinations of states



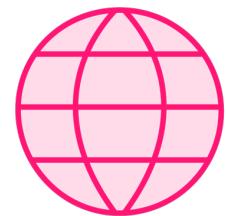
Use Cases for the Prototype Pattern



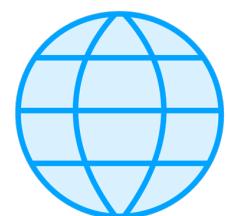
Document cloning in word processors



Managing configuration instances



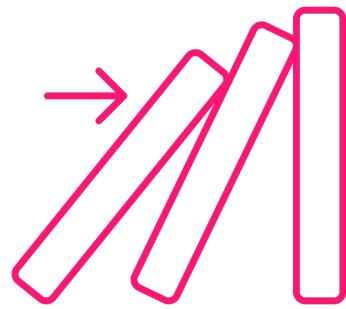
Creating reusable UI components



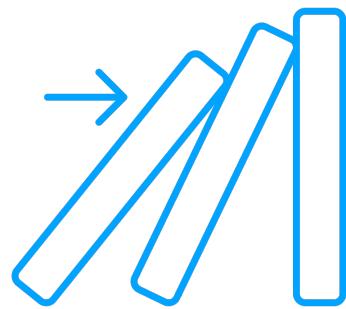
Creating characters in game development



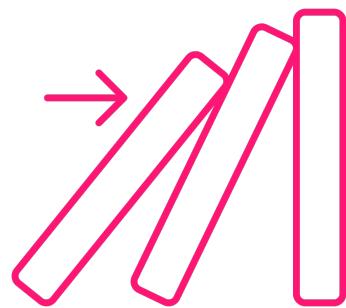
Pattern Consequences



Prototype hides the `ConcreteProduct` classes from the client, which reduces what the client needs to know



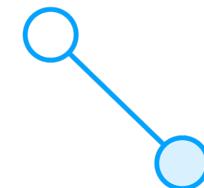
Reduced subclassing



Each implementation of the prototype base class must implement its own `clone` method

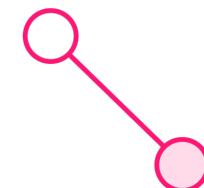


Related Patterns



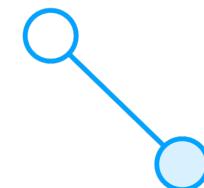
Abstract factory

A factory might store a set of prototypes from which it clones when a new instance is requested



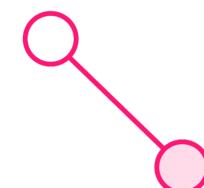
Factory method

Factory method is based on inheritance, but doesn't require an initialization step



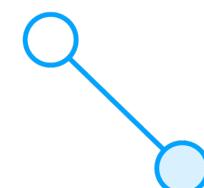
Singleton

Prototype can be implemented as a singleton



Composite

Can use prototype for convenient object creation



Decorator

Can use prototype for convenient object creation



Summary



Intent of the prototype pattern:

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



Summary



Implementation:

- Subclasses of the Prototype implement the `Clone()` method
- Clients work on the Prototype



Summary



A shallow copy is a copy of primitive type values, while a deep copy is a copy of primitive type values and complex type values



Up Next:

Structural Pattern: Adapter

