Design Pattern

Ran Liao

May 22, 2019

Adapter Design Pattern

The Adapter design pattern solves the implementation incompatibilities. It also provides a general solution to the problem of permitting communication between two objects with incompatible interfaces and a way for an object to permit access to its internal implementation without coupling clients to the structure of that internal implementation. That is, Adapter provides all the advantages of information hiding without having to actually hide the implementation details

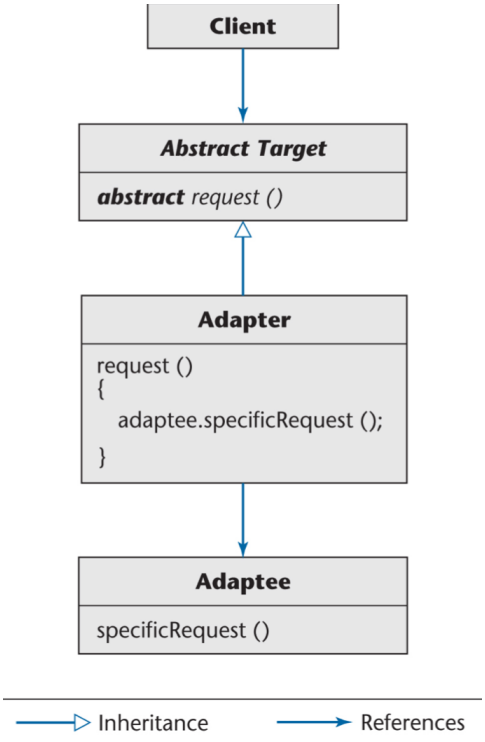

Figure 1: Adapter Design Pattern

Bridge Design Pattern

The Bridge design pattern aims to decouple an abstraction from its implementation so that the two can be changed independently of one another. Sometimes it's also called a *driver*.

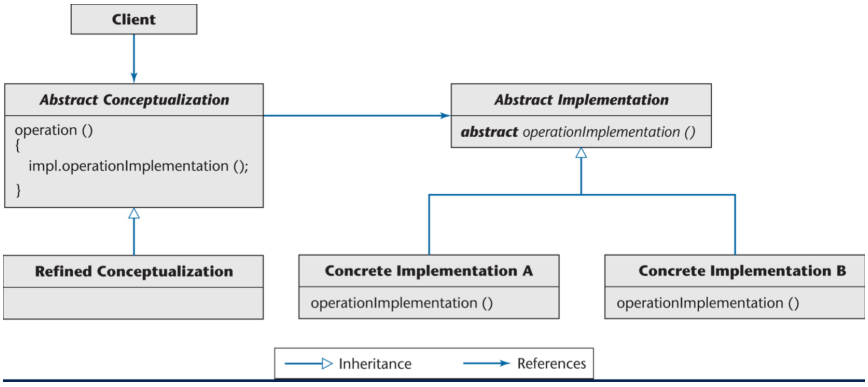

Figure 2: Bridge Design Pattern

Iterator Design Pattern

An aggregate object (or container or collection) is an object that contains other objects grouped together as a unit. An iterator (or cursor) is a programming construct that allows a programmer to traverse the elements of an aggregate object without exposing the implementation of that aggregate.

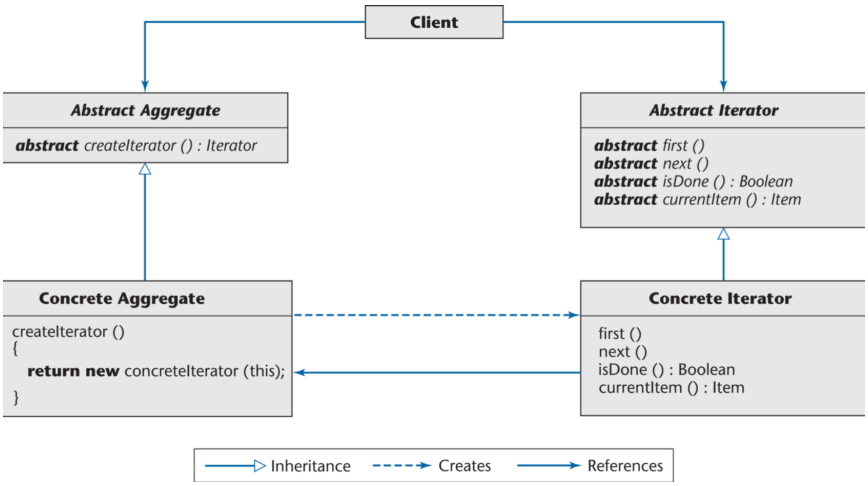

Figure 3: Iterator Design Pattern

Abstract Factory Design Pattern

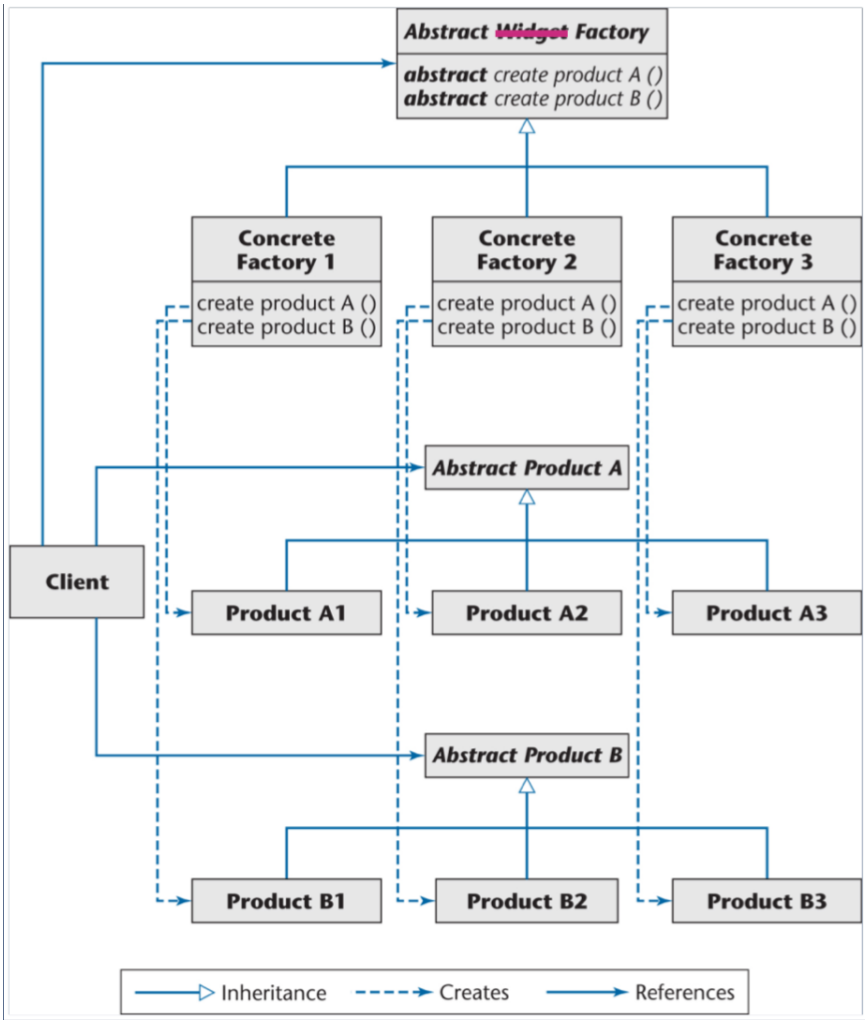

Figure 4: Abstract Factory Pattern

Categories of Design Patterns

- **Creational Design Patterns**

  Creational design patterns solve design problems by creating objects.

Creational patterns	
Abstract factory	Creates an instance of several families of classes (Section 8.6.5)
Builder	Allows the same construction process to create different representations
Factory method	Creates an instance of several possible derived classes
Prototype	A class to be cloned
Singleton	Restricts instantiation of a class to a single instance

Figure 5: Creational Design Pattern

- **Structural Design Patterns**

  Structural design patterns solve design problems by identifying a simple way to realize relationships between entities.

Structural patterns	
Adapter	Matches interfaces of different classes (Section 8.6.2)
Bridge	Decouples an abstraction from its implementation (Section 8.6.3)
Composite	A class that is a composition of similar classes
Decorator	Allows additional behavior to be dynamically added to a class
Façade	A single class that provides a simplified interface
Flyweight	Uses sharing to support large numbers of fine-grained classes efficiently
Proxy	A class functioning as an interface

Figure 6: Structural Design Pattern

- **Behavioral Design Patterns**

  Behavioral design patterns solve design problems by identifying common communication patterns.

Behavioral patterns	
Chain-of-responsibility	A way of processing a request by a chain of classes
Command	Encapsulates an action within a class
Interpreter	A way to implement specialized language elements
Iterator	Sequentially access the elements of a collection (Section 8.6.4)
Mediator	Provides a unified interface to a set of interfaces
Memento	Captures and restores an object's internal state
Observer	Allows the observation of the state of an object at run time
State	Allows an object to partially change its type at run time
Strategy	Allows an algorithm to be dynamically selected at run time
Template method	Defers implementations of an algorithm to its subclasses
Visitor	Adds new operations to a class without changing it

Figure 7: Behavioral Design Pattern