Design Defects and Restructuring

LECTURE 09

SAT, NOV 7, 2020

Prototype

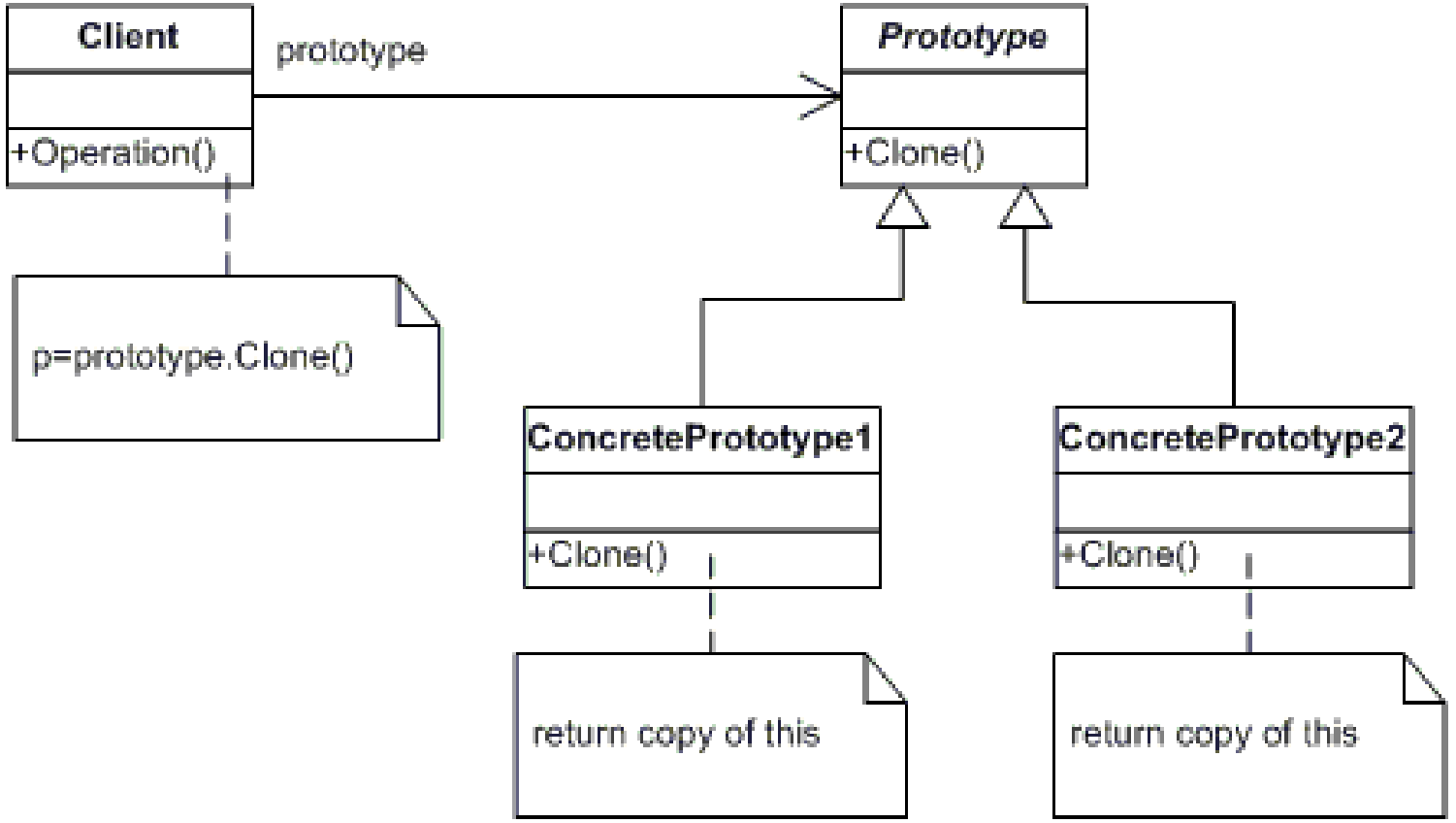
Intent

- 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; or
- To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or
- When instances of a class can have one of only a few different combinations of state
  - It may be more convenient to install a corresponding number of prototypes and clone them rather than instantiating the class manually, each time with the appropriate state

Prototype


Client
prototype
+Operation()
p=prototype.Clone()
Prototype
+Clone()
ConcretePrototype1
+Clone()
return copy of this
ConcretePrototype2
+Clone()
return copy of this

Singleton

Intent

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

Applicability

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

Singleton


Singleton
-instance : Singleton
-Singleton()
+Instance() : Singleton

Structural Patterns

Adapter

Bridge

Composite

Decorator

Façade

Flyweight

Proxy

Adapter

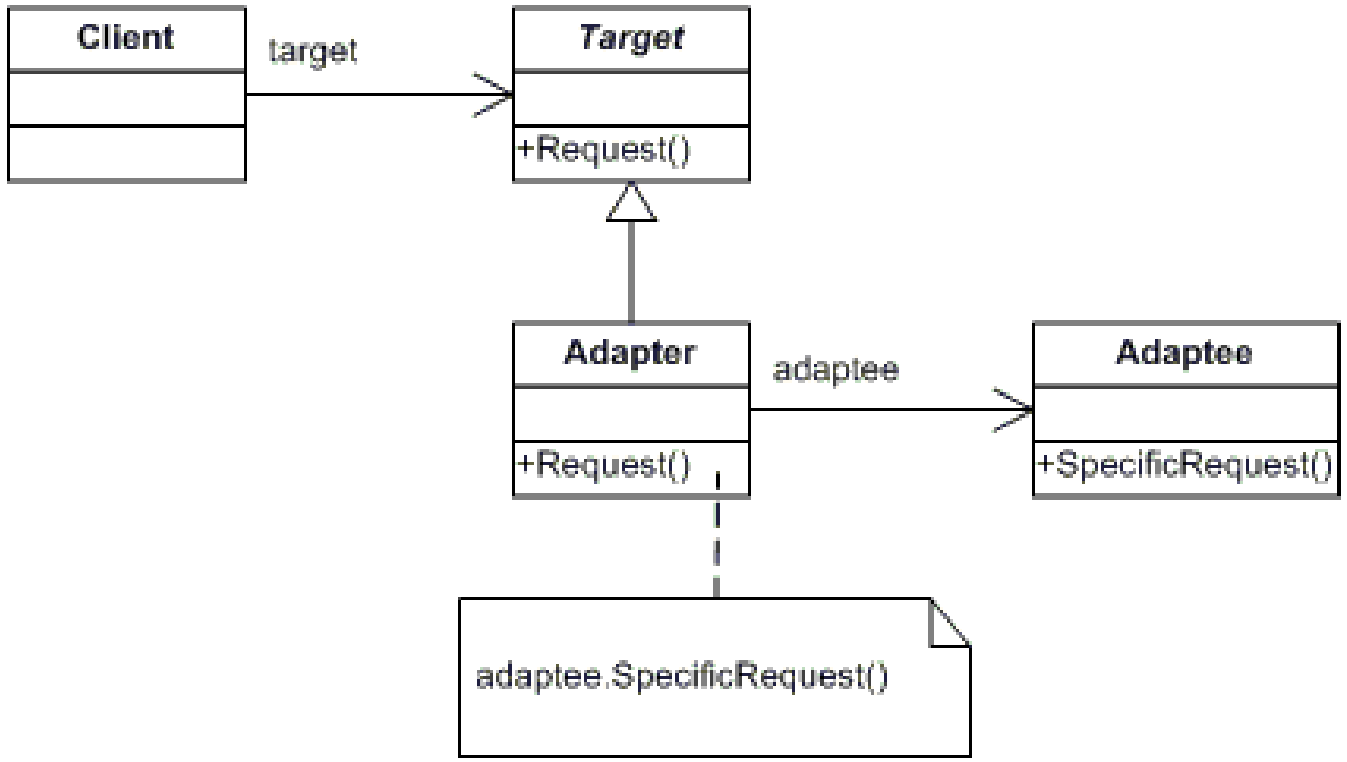
Intent

- Convert the interface of a class into another interface clients expect
- Adapter lets classes work together that could not otherwise because of incompatible interfaces

Applicability

- You want to use an existing class, and its interface does not match the one you need
- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that do not necessarily have compatible interfaces
- You need to use several existing subclasses, but it is impractical to adapt their interface by sub-classing every one
  - An object adapter can adapt the interface of its parent class

Adapter


Client
target
Target
+Request()
Adapter
+Request()
adaptee
Adaptee
+SpecificRequest()
adaptee.SpecificRequest()

Bridge

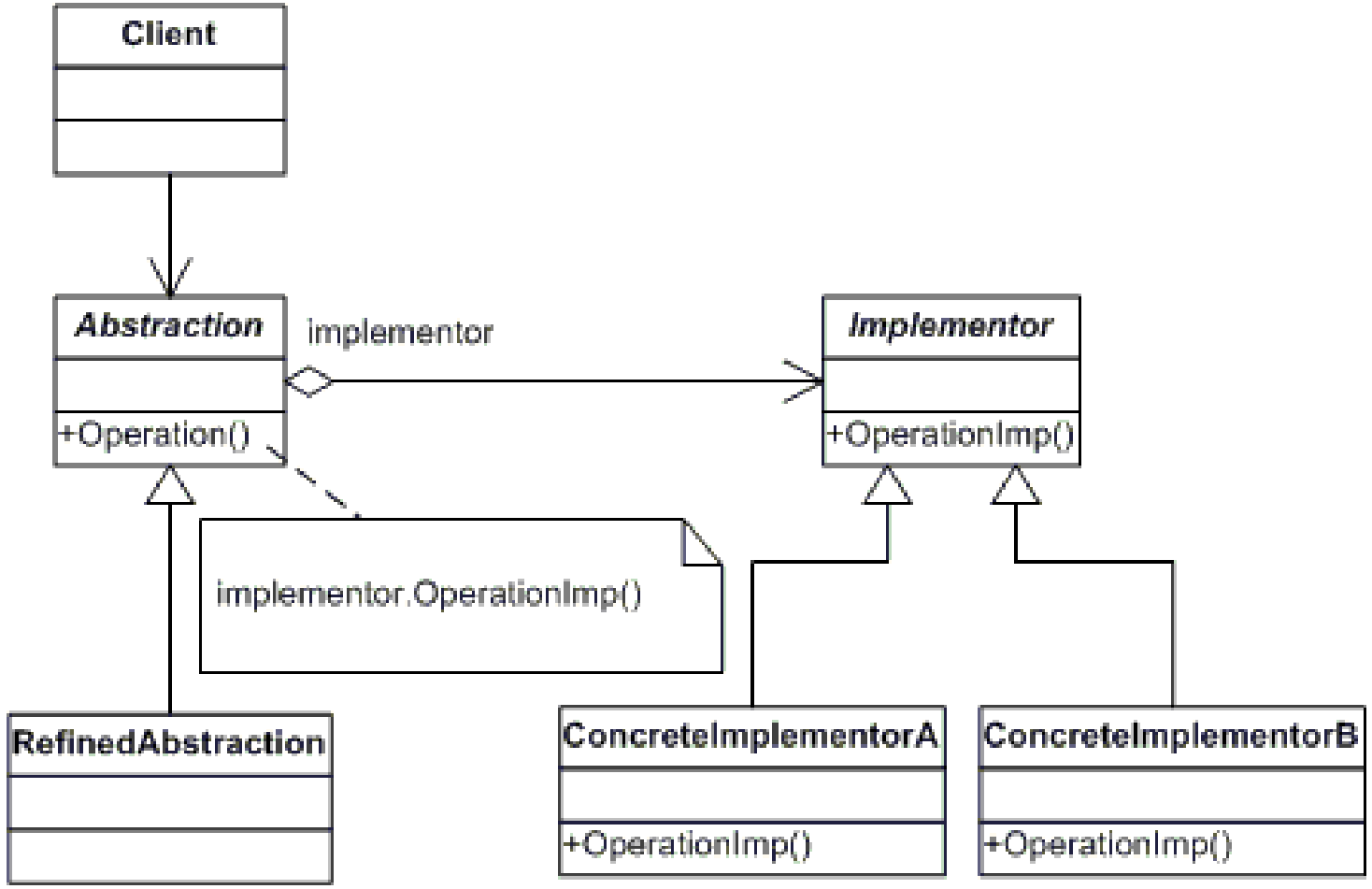
Intent

- Decouple an abstraction from its implementation so that the two can vary independently

Applicability

- You want to avoid a permanent binding between an abstraction and its implementation
- Both the abstractions and their implementations should be extensible by sub-classing
- Changes in the implementation of an abstraction should have no impact on clients

Bridge


Client
Abstraction
implementor
Implementor
+Operation()
+OperationImp()
implementor.OperationImp()
RefinedAbstraction
ConcreteImplementorA
+OperationImp()
ConcreteImplementorB
+OperationImp()

Composite

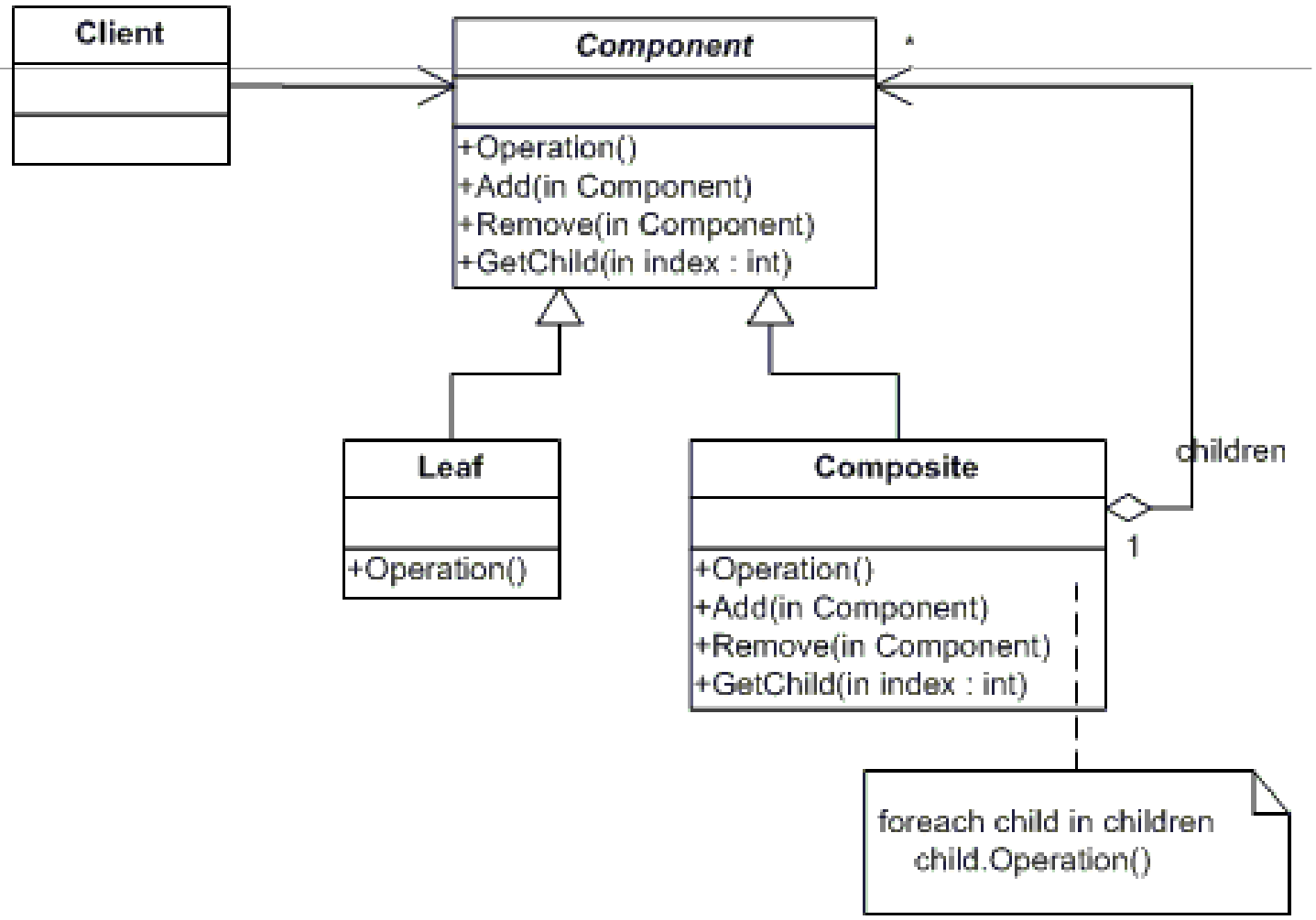
Intent

- Compose objects into tree structures to represent part-whole hierarchies
- Composite lets clients treat individual objects and compositions of objects uniformly

Applicability

- You want to represent part-whole hierarchies of objects
- You want clients to be able to ignore the difference between compositions of objects and individual objects
  - Clients will treat all objects in the composite structure uniformly

Composite


Client
Component
*
+Operation()
+Add(in Component)
+Remove(in Component)
+GetChild(in index : int)
Leaf
+Operation()
Composite
+Operation()
+Add(in Component)
+Remove(in Component)
+GetChild(in index : int)
children
1
foreach child in children
child.Operation()