OODP– Session 5a

Next week: Reading week

Session times

PT group 1 – Monday	18:00-21:00	room: Malet 403
PT group 2 – Thursday	18:00-21:00	room: Malet 407
FT - Tuesday	13:30-17:00	room: Malet 404

Email: oded@dcs.bbk.ac.uk

Web Page: http://www.dcs.bbk.ac.uk/~oded

Visiting Hours: Tuesday 17:00 to 19:00

Previously on OODP

- GRASP design patterns
  - **Information Expert**
  - **Creator**
  - **Low Coupling**
  - **High Cohesion**
  - **Controller**

(don’t think in patterns, think patterns)

Classification (GoF – Gang of Four)

- Creational Patterns
  - Abstract object instantiation
- Structural Patterns
  - Compose and organise objects into larger structures
- Behavioural Patterns
  - Algorithms, interactions and control flow between objects.

Previously on OODF

- List of advanced patterns we already discussed
  - **Strategy** (behavioural)

- List of advanced patterns for today
  - **Adapter** (structural)
  - **Decorator** (structural)
  - **Factory Method** (Creational)
  - **Abstract Factory** (Creational)
  - **Singleton** (Creational)

Adapter

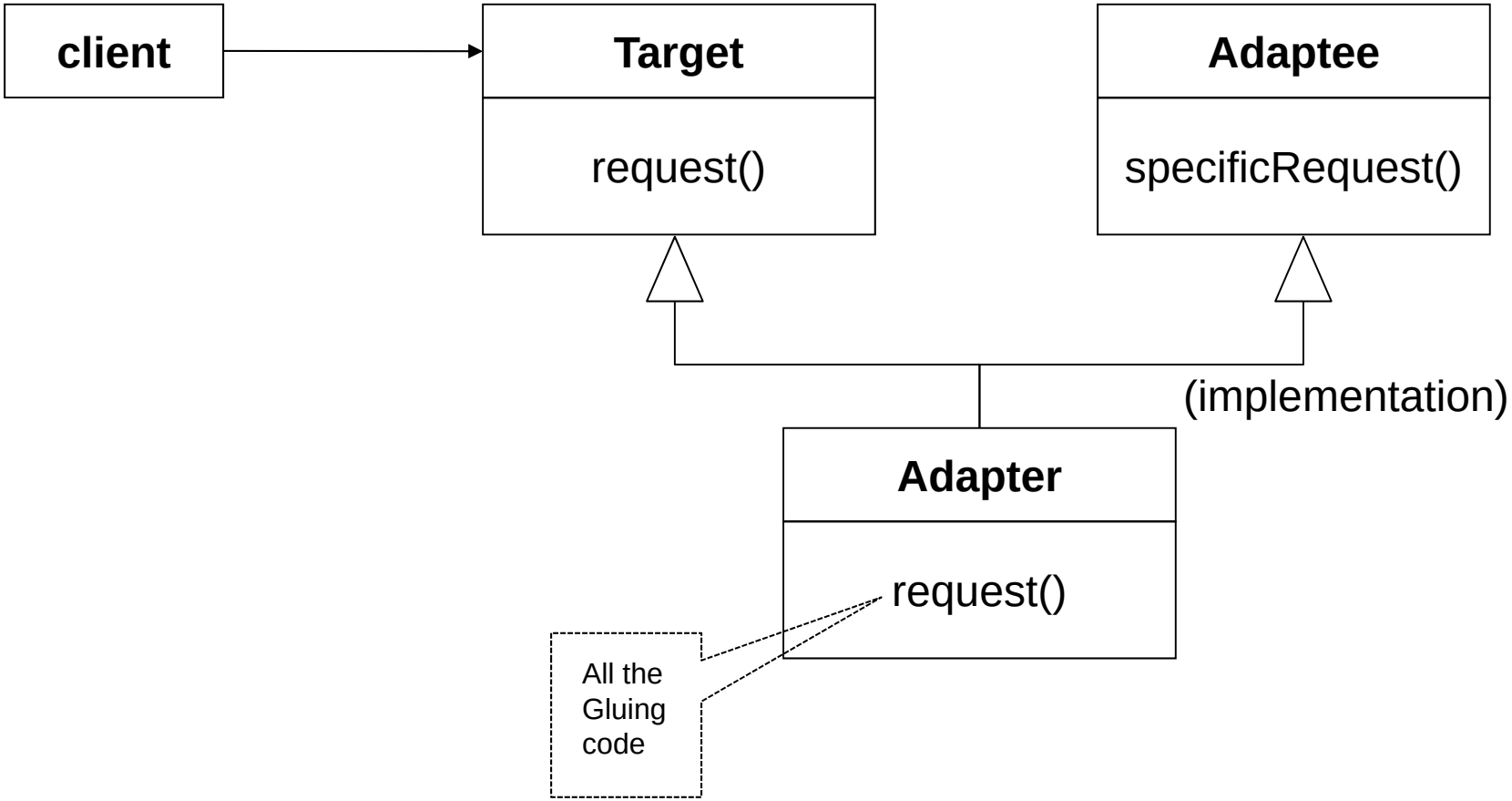
Adapter

Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?

Solution: Convert the original interface of a component into another interface through an intermediate adapter object

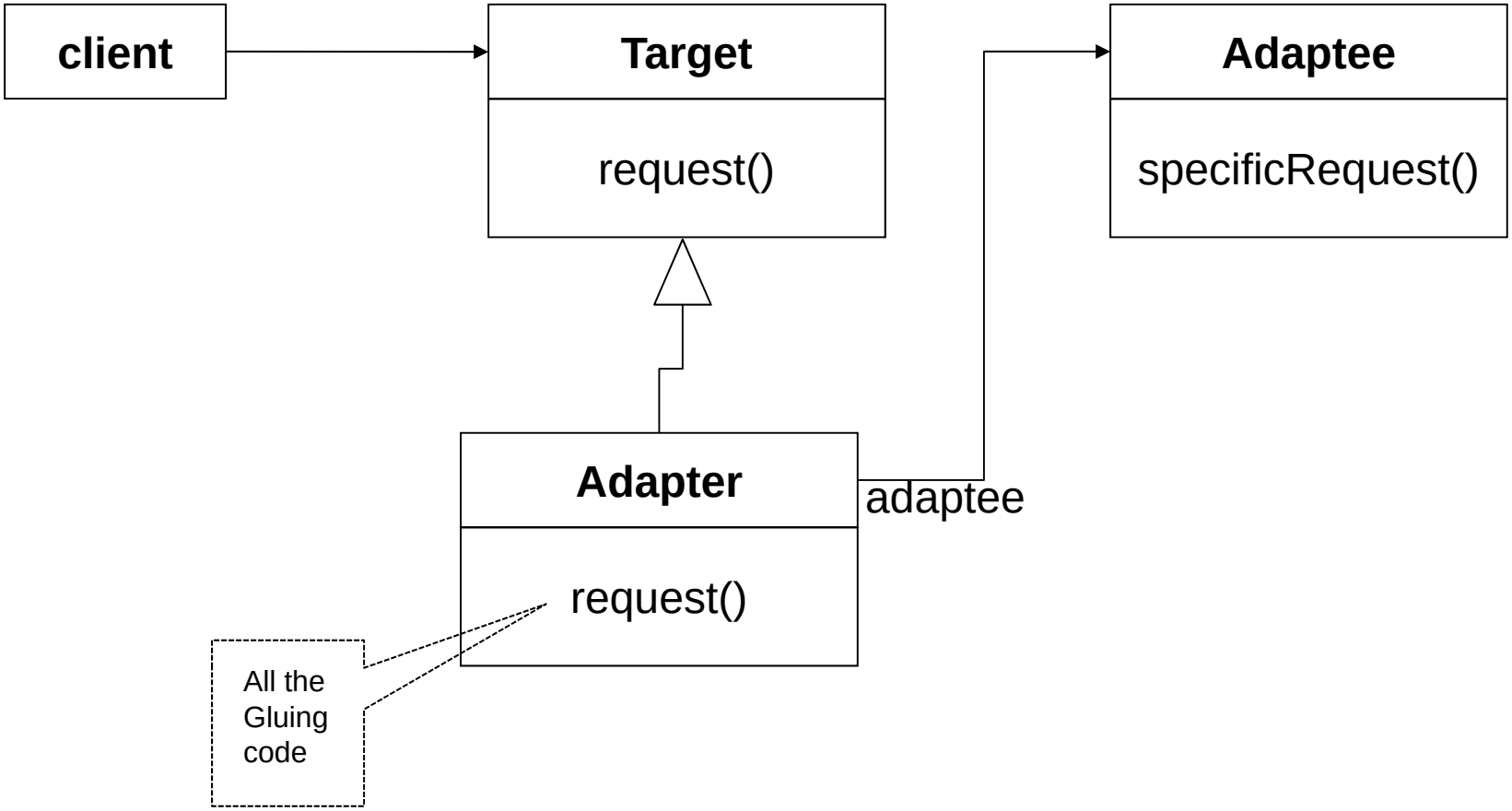
Class Adapter

How: Inheritance



Object Adapter

How: Inheritance + delegation



Discussion

Why Use?

- Want to use other people code
- Not just use other peoples code but override some of it

Class Adapter advantages:

- Simpler

Class Adapter Issues

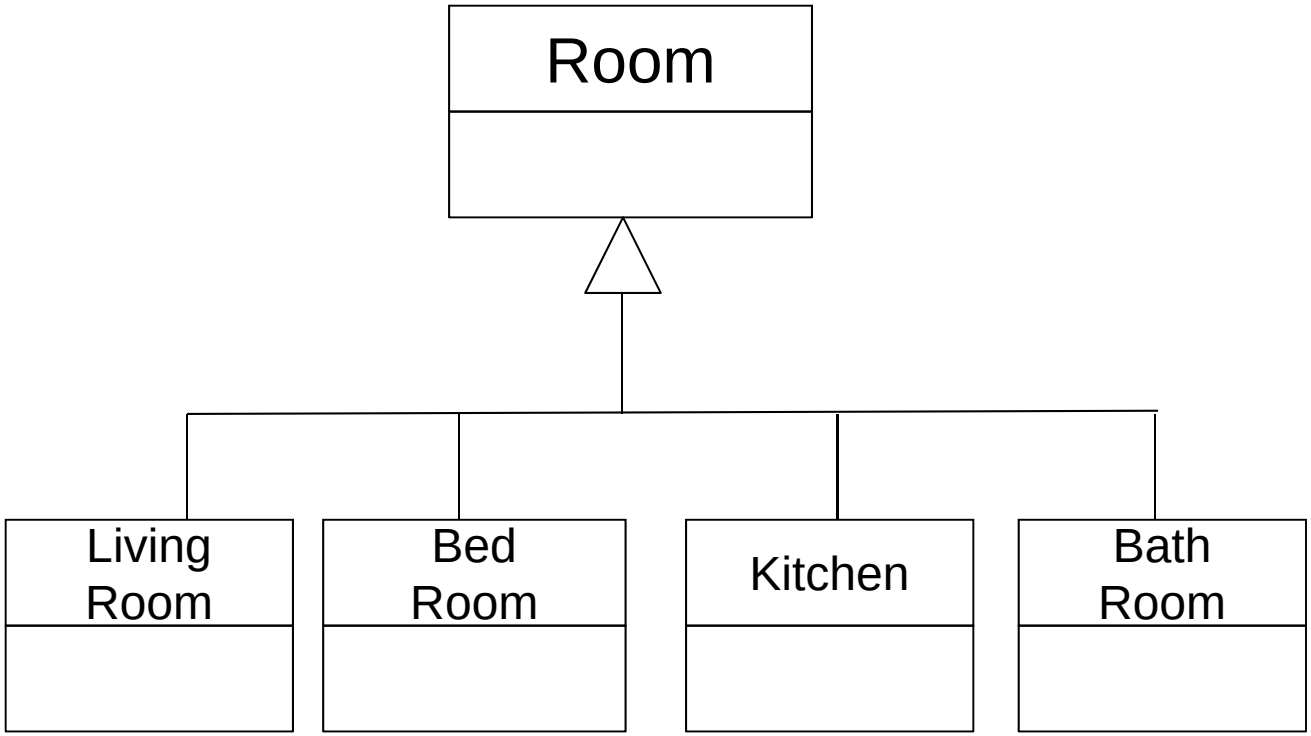
- Works only for one specific class and not its subclasses

Object Adapter Issues

- Harder to override adaptee behaviour

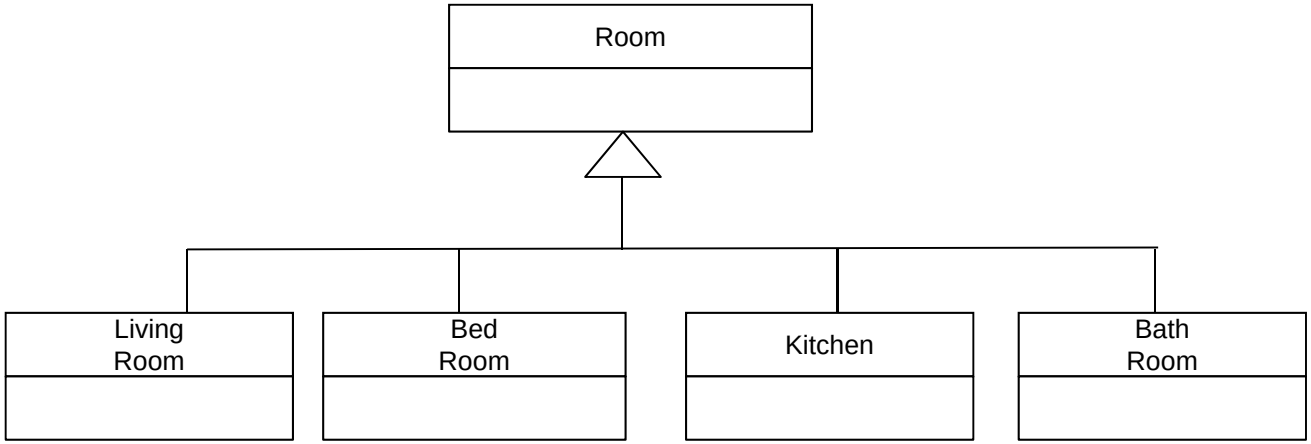
Decorator

House, (partial) Domain Model



Problem: How can we add responsibilities to rooms?

Room, Add Responsibilities



Naïve idea: Just add the required methods to all classes

What if?

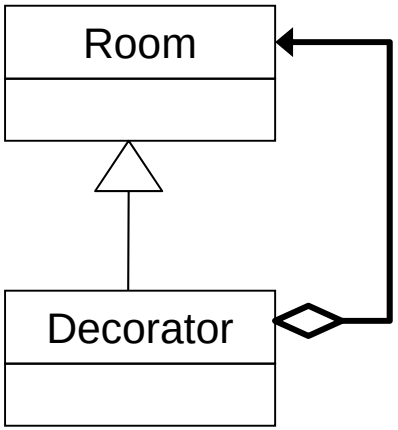
- Someone else's classes and you don't want to change them
- Don't want to add responsibilities to all subclasses
- Want the decision of adding the functionality to be in run time

Airplane Game, Decorator

Not so naïve anymore

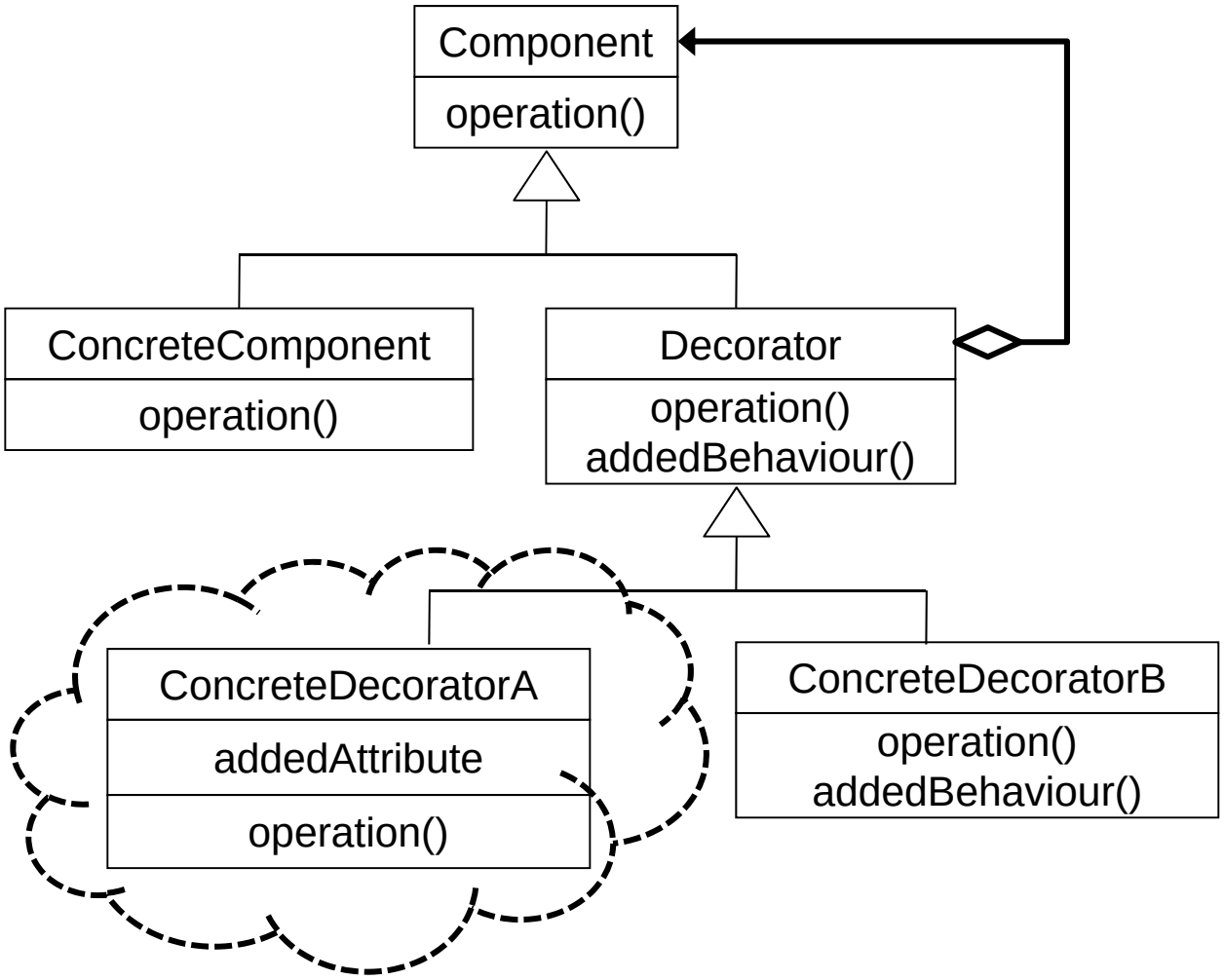
What needs to happen? What can we do?

- New class with the same interface (Decorator)
- Encapsulation



Decorator Pattern

Problem: How can we add responsibilities to airplanes?



Decorator Pattern

Pros

- Classes remain light weight
- More flexibility

Cons

- More classes
- Decorator and components aren’t identical

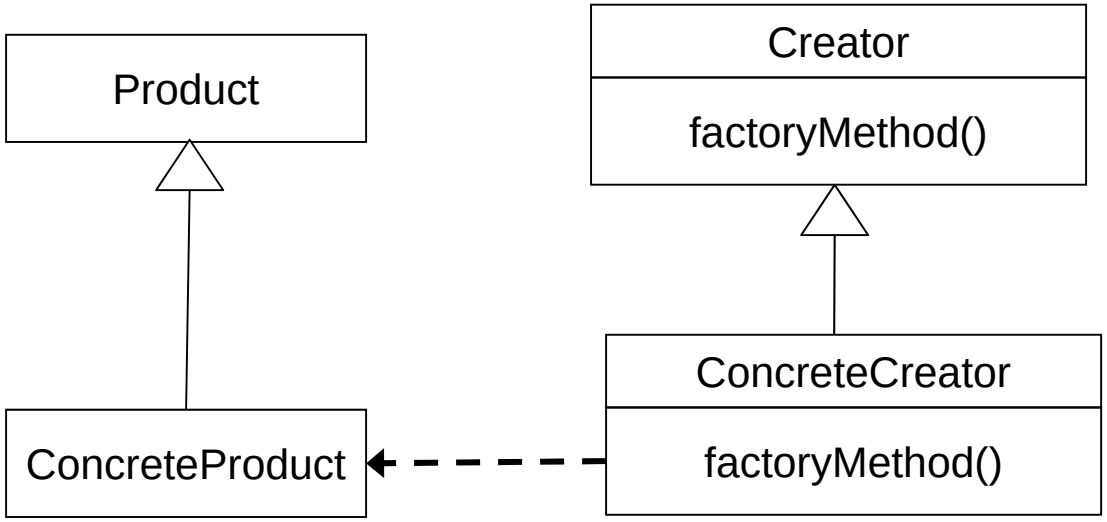
Factory Method

Square subclasses in the Monopoly Game

- In the monopoly game there are different classes of squares.
- Say it only has a single abstract class as an attribute.
- Regretfully the actual choice of class depends on the subclass of square

Factory Method

- **Idea**: Instead of using new inside the constructor, use a dedicated method (factoryMethod)
- Override this method in subclasses
- Method may also get a parameter as input

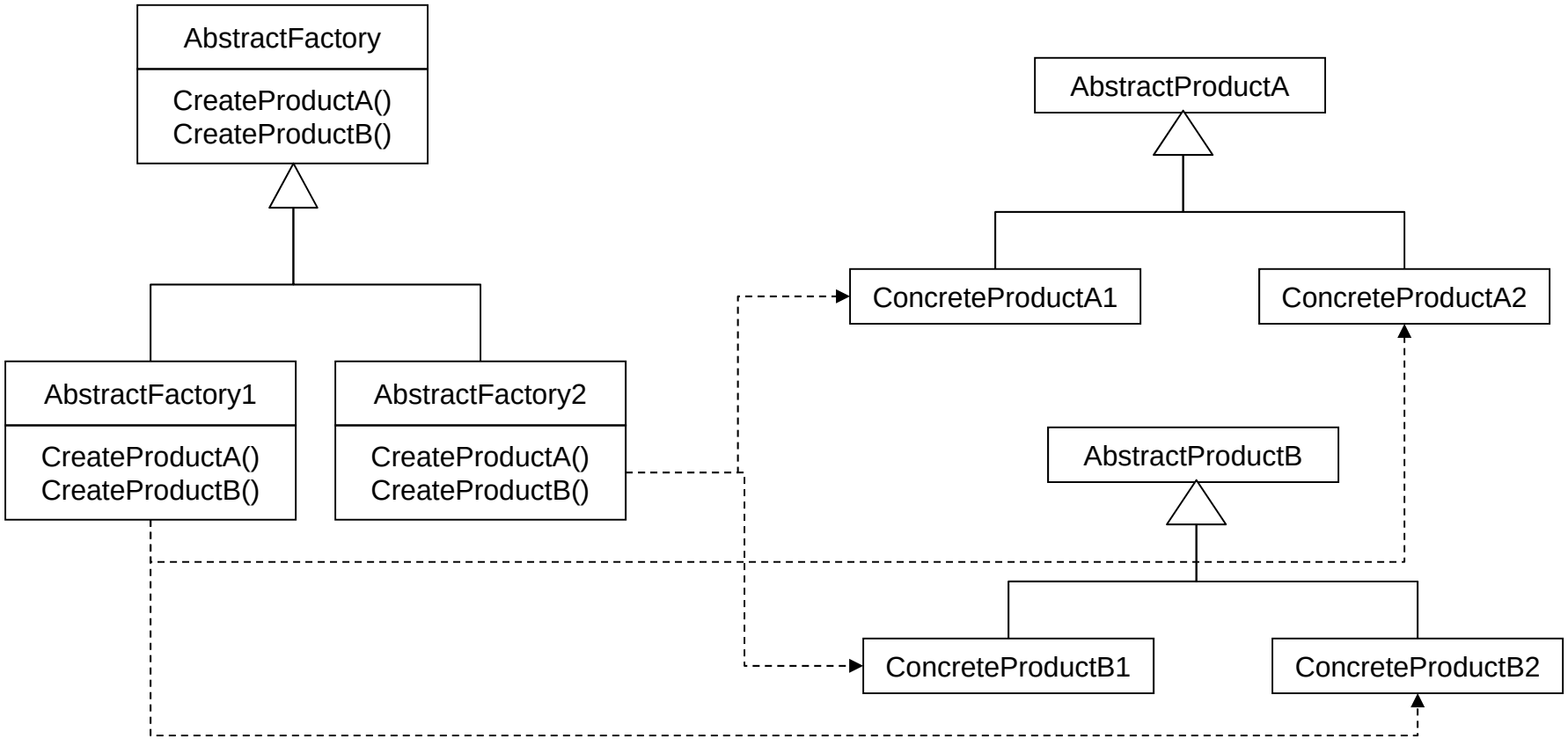


Abstract Factory

Monopoly Game GUI

- It has become fashionable to let the user control the look-and-feel of GUI's.
- Why not have this in our monopoly game.
- Problem: the look and feel of the monopoly game depends on many different objects.
- Idea – different objects come from different factories

Abstract Factory



Doesn’t this add a lot of coupling?

Singleton

Singleton

Structure:

Intent:

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

Singleton - Example

```
Public class PrintSpooler {
      // a prototype for a spooler class,
      // such that only one instance can ever exist
   private static PrintSpooler _spooler;
   private PrintSpooler() { /* private constructor */ }
   public static synchronized getSpooler() {
      if (_spooler == null) _spooler = new PrintSpooler();
      return _spooler;
   }
}
```

```
class Main {   // example of use
   public static void main(String args[]) {
   PrintSpooler spl =PrintSpooler.getSpooler();
   spl.print ("Printing data");
   }
}
```

Back to GRASP

More GRASP patterns

- **Information Expert, Creator, Low Coupling, High Cohesion, Controller**

- **Polymorphism**
- **Indirection**
- **Pure Fabrication**
- **Protected Variations**

Polymorphism

Pattern Name: Polymorphism

- **Problem It Solves:** How to handle alternatives based on type? How to create pluggable software components?
- **Solution:** When related alternatives or behaviour vary by type, assign responsibilities for the behaviour using polymorphism

Alternatives based on type – replacing code by classes and types

Pluggable software components – viewing components in client server relationships, how can you replace one server component with another without affecting the client

Where have we seen Polymorphism

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Indirection

Pattern Name: Indirection

- **Problem It Solves:** Where to assign a responsibility, to avoid direct coupling between two or more things
- **Solution:** Assign the responsibility to an intermediate object

Where have we seen Indirection

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Pure Fabrication

Pattern Name: Pure Fabrication

- **Problem It Solves:** What object should have the responsibilities, when you don't want to violate high cohesion and low coupling, or other goals, but solutions offered by Expert (for example) are not appropriate.

- **Solution:** Assign a highly cohesive set of responsibilities to an artificial class that does not represent a problem domain concept (such a class is a fabrication of the imagination).

Where have we seen Pure Fabrication

- GoF Design Patterns:
  - Strategy
  - Abstract Factory

Protected Variation

Pattern Name: Protected Variation

- **Problem It Solves:** How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?
- **Solution**: Identify points of predicted variation or instability; assign responsibilities to create a stable interface around them.

Where have we seen Protected Variation

- GoF Design Patterns:
  - Strategy
  - Decorator
  - Abstract Factory

  ...