SENG5802: Software Engineering II - Software Design
Design Patterns 1

Outline

Outline

Domain model details

Good domain models have transparent, predictable structure

- Entities, Values, and Services — Evans
- Archetypes mapped in well-understood way — Coad

Entities

If it's a persistent object with possibly varying attributes,

it's an Entity (or reference object).

Values

If it's a description of an Entity, or if it's a data holder or message,

it's a Value.

Services

If it's a set of operations not obviously associated with any particular Entity or Value and (ideally) has no state,

it's a Service

Domain Services

What sorts of services do we expect to see in the domain model?

Coad's archetypes

- Party, Place, Thing – things (often entities) that can play a role in the domain or system
- Moment-interval – activities or events that are located in time or extend over an interval.
- Role – a way that something (usually a PPT) participates in a M–I.
- Description – a value or set of values that label a thing or set of things (non-identifying)

Coad Discussion

Coad claims that the addition of color "increases the amount of content we can express."

Does color make the diagrams more expressive?

What if you could add or remove colors, or change their meaning?

Modeling associations

Bidirectional associations

- Don't work so well in code
- Make sure you fully understand domain and application requirements
- Use implicit associations for the less-traveled route

Assigning Responsibility

- Begin with the clear operations on Entities and Values

- Operation on multiple objects? Sounds like a service.

Exercise

Work out a more detailed domain model for Battleship

Goals:

- Understand the game
- Agree on vocabulary
- Find Entity and Value objects
- Find Service objects

Outline

Capturing Change and Risk

- Good management anticipates change and estimates risk
- Good management attempts to minimize impact
- Good design incorporates decisions that minimize impact
- Analysis paralysis is a big issue

Risk vs Change Case

- Primarily a semantic differences
- Change case: new use cases which require new (or vastly modified) requirements
- Risk: alterations outside of project control that impact how (or, even, if) current requirements can be completed.

Measuring impact and likelihood

- In general, assign both to change cases and risks.

- Two categories of scale: ordinal and ratio.
  - Ordinal: High, medium, low
  - Ratio: $ (impact) or % (probability)

Discussion: Risk/Change Case

Is High Impact Medium Probability or Medium Impact High Probability the higher priority? H-L, L-H? H-L, M-M?

What is your opinion on using these ordinal categorizations versus the numerical ratio measurements of impact and likelihood?

Outline

Patterns

- Elements of reusable design
- Vocabulary to talk about design
- Another abstraction

Perspective

- Stu Halloway says

  Design patterns are the enemy of agility. They introduce repetition and accidental variation into your codebase.

- What does he mean by "repetition and accidental variation"?

- What is his recommended alternative?

Outline

Wrapper Patterns

- **Façade** and **Adapter** are basic modularization patterns.
- A Façade is a "window" into a potentially complex module or subsystem
- Adapter "wraps" existing code to the interface requirements of another design.

Façade

"A showy misrepresentation intended to conceal something unpleasant."
– WordNet

GOF says it "provides a unified interface to a set of interfaces in a subsystem."

- Raises abstraction by eliminating dependencies on subsystem details.
- Key element of layered architectures

Façade Difficulties

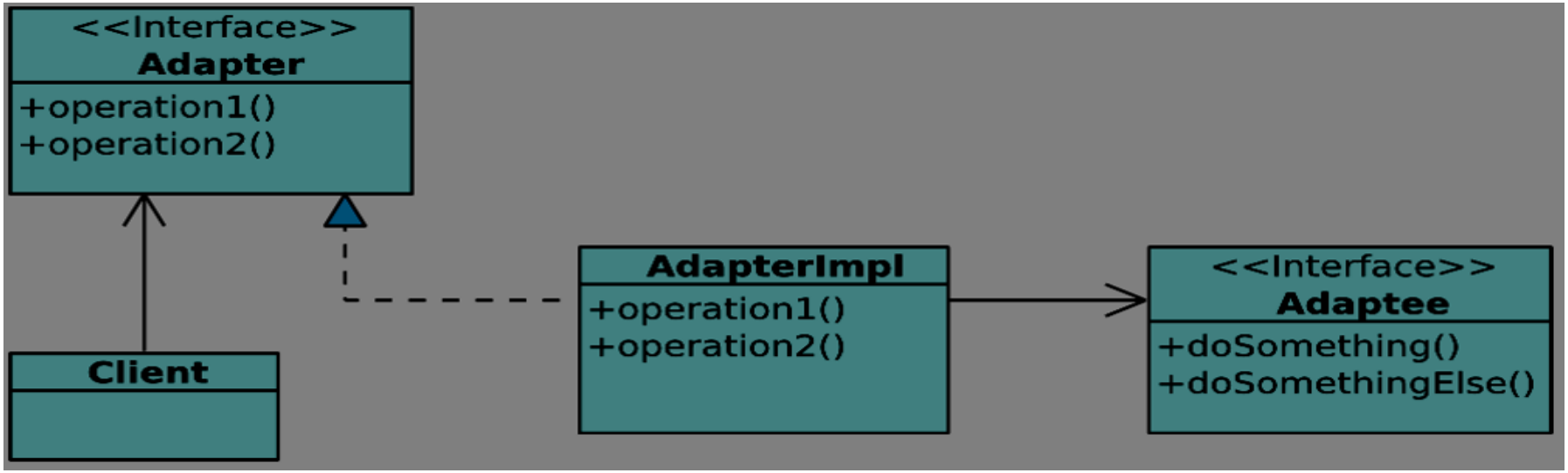
Façades can grow out of control and become quite baroque.

Façades often violate the Law of Demeter – forcing you to talk to strangers.

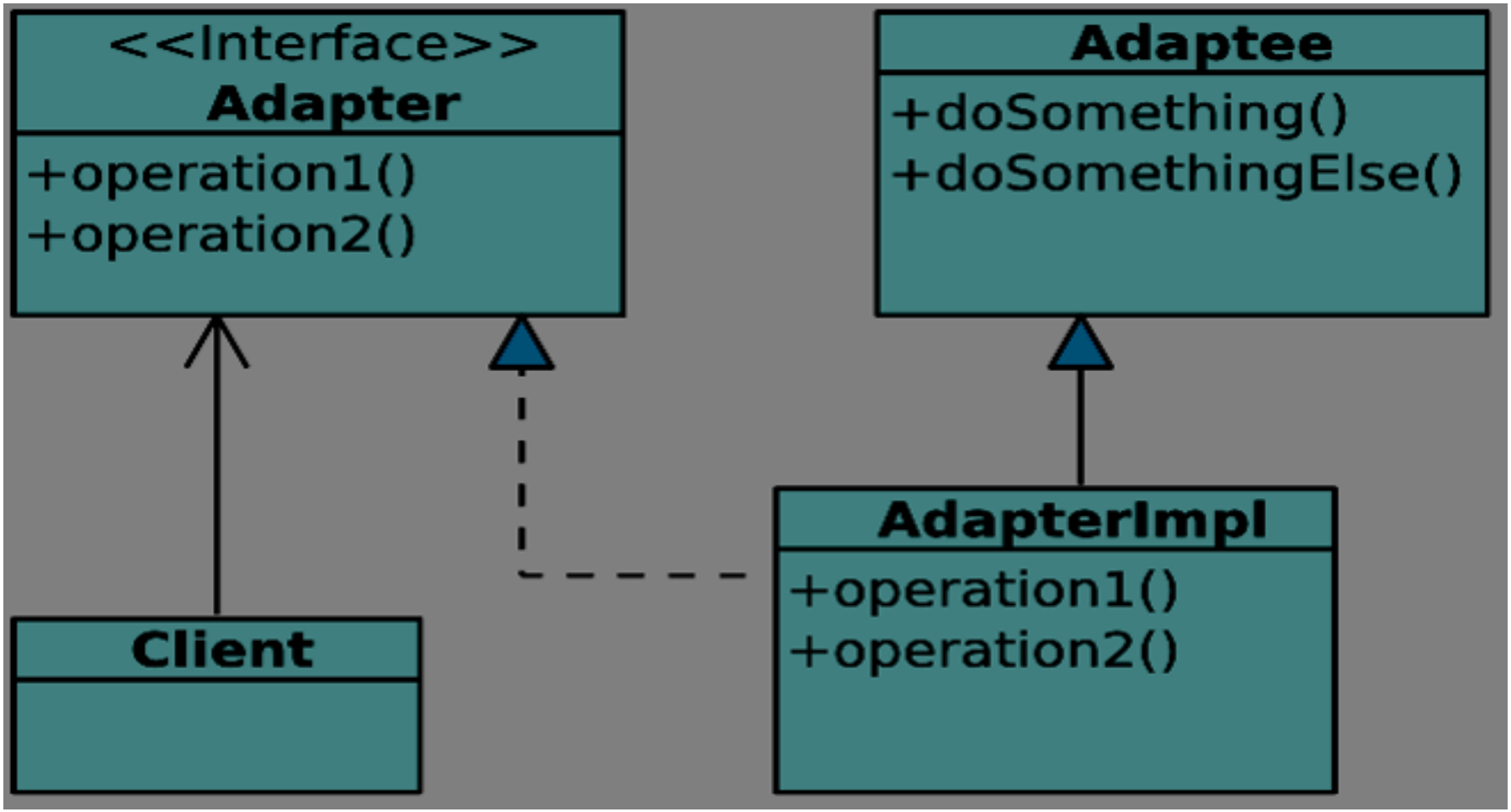
Adapter

GOF: Convert the interface of a class into another interface that clients expect

Object Adapter



Class Adapter



Adapter Trade-Offs

- Object Adapter
- Class Adapter

Outline

Strategy

Define a family of algorithms,
encapsulate each one,
and make them interchangeable

Gang of Four

Strategy Theory

- Client
- Context
- Strategy (Abstraction)
- Strategy (Concrete)

Context needs differing versions of some algorithm

Choice may be out-of-scope of Context

Example: Logger

Logger can be configured to produce several kinds of output

Selecting Strategies

How do we select concrete target?

If selection is out-of-scope for context?

Strategy Factory

Client (or Context) requests Strategy instances from StrategyFactory, passing properties uninterpreted.

Factory itself is global, responsible for interpreting attributes

Strategy Example

- Strategy separates operation based on circumstances
- Factory encapsulates complexity of selecting specific Strategy implementation
- Client is purely ignorant of types and selection criteria for strategies.