Design Patterns

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Outline

- Why study design patterns
- Key OO principles
- Patterns by Pictures, UML

Why study design patterns

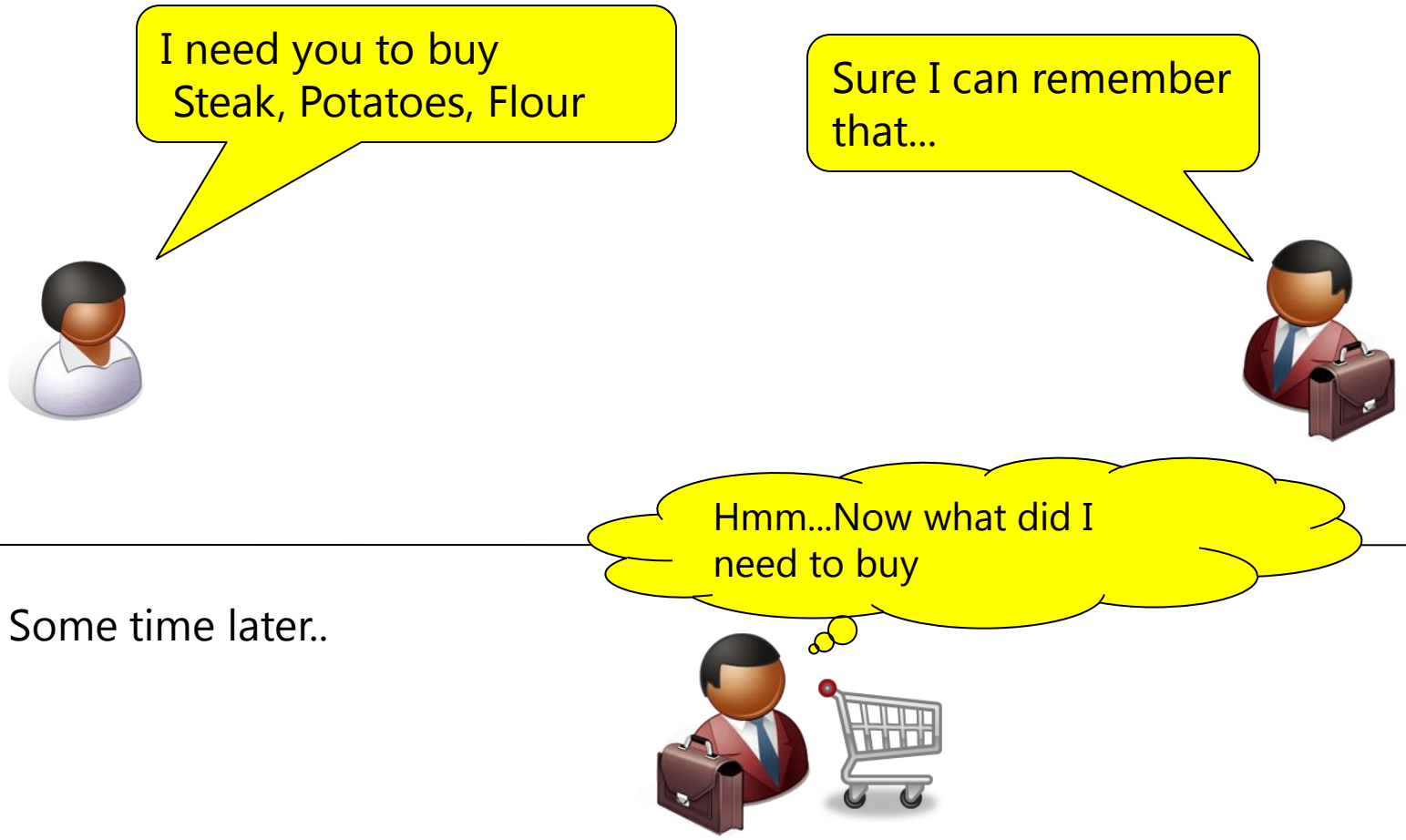
- Reuse Solutions
  - NOT CODE
- Establish Common Terminology
  - Efficient and accurate communication
- Higher Level perspective
  - Spend more time in the problem domain

What is a Pattern

- A pattern is a solution to a reoccurring problem in a context
  - Context
    - Situation in which the pattern applies
  - Problem
    - Goal in the given context
    - referencing the constraints imposed by the context
  - Solution
    - Describes a general way to achieve the goal and the set of constraints

Real Life Patterns

- "On your way home, go to the shops and buy"



Shopping List

- Problem
  - Wife has asked to buy items from the shops must remember everything.
- Context
  - Can’t contact the wife.
- Solution
  - Take a piece of paper and pen.
  - Write down each item the wife states to buy.
  - Cross of items as you buy them.

Is this a pattern ?

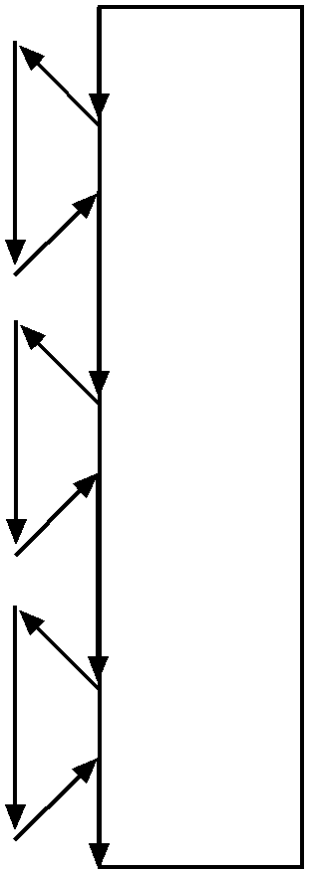
List Pattern

- A Pattern is a general solution
  - Problem
    - Need to remember a set of requests
  - Context
    - Can’t contact the requester
  - Solution
    - Take a recording device
    - Record each request as given
    - Remove requests as you complete them

Two Carpenters deciding on a joint

- Carpenter 1:
  - How do you think we should build these drawers?

- Carpenter 2:
  - Well, I think we should make the joint by cutting straight down into the wood, and then cut back up 45 degrees, and then going straight back down, and then back up the other way 45 degrees, and then going straight back down, and then …



What are they talking about ?

A More likely scenario

- Carpenter 1:

  Should we use a dovetail joint or a mitre joint?
- Discussion is at a higher level
  - More Abstract
  - Avoid details
- The word mitre joint to a carpenter automatically implies
  - Simpler solution; Light weight ; Inconspicuous
- Dove tail implies
  - More Complex; Looks Better; Independent of the fastening system
- The essence of the conversation between the two carpenters is:
  - “Should we use a complex, strong expensive joint or just make a quick and dirty joint”

The two conversations

- Conversation One
  - Carpenter 2 obscures the real issues by discussing the details of the implementations of the joints
- Conversation Two
  - Carpenter 1 wants to decide which joint to use based on costs and quality of the joint
- Carpenter 1 is talking in terms of carpentry patterns, which reflects the real design issues and provides an efficient means of communication

Software Patterns

- Software Engineering is now reaching a maturity that it has patterns
- The “Gang of Four” sat down and produced a initial set of patterns for Software, and released them in a book
- Called Design patterns book, which became known as the Gang of Four, references to the patterns are often written
  - GOF:XXX
- There are plenty of other pattern catalogues
  - Compound Patterns
  - Enterprise Patterns
  - Unit Test Patterns

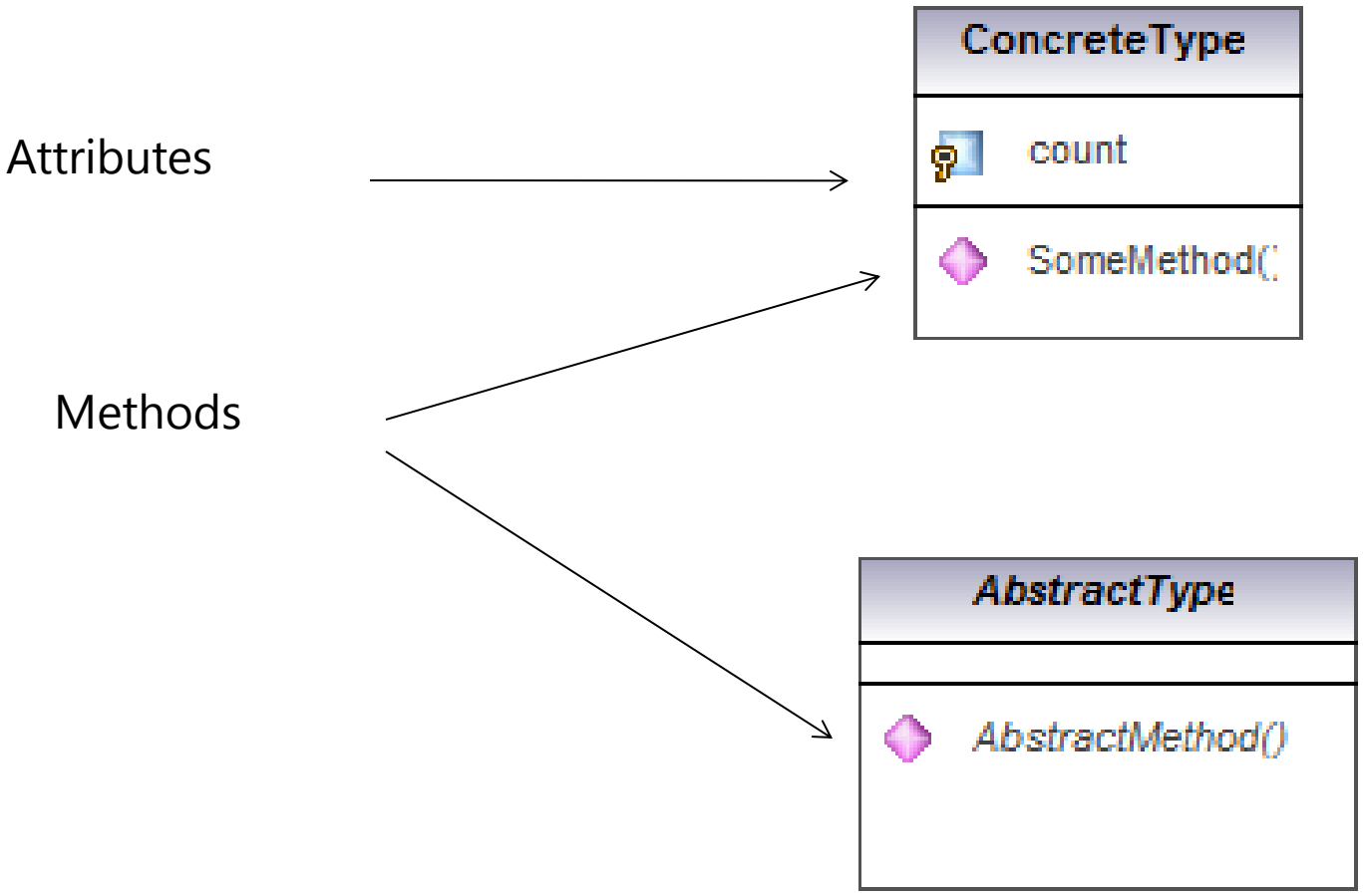
OO Design Pattern Principles

- Design to interfaces
  - Write interface classes and then produce implementation classes
- Favour composition over inheritance
  - Large inheritance hierarchies have failed in the past
- Encapsulate what varies
  - Separate code that stays the same to code that varies
- Types should be closed for modification but open for extension

Design Patterns by pictures

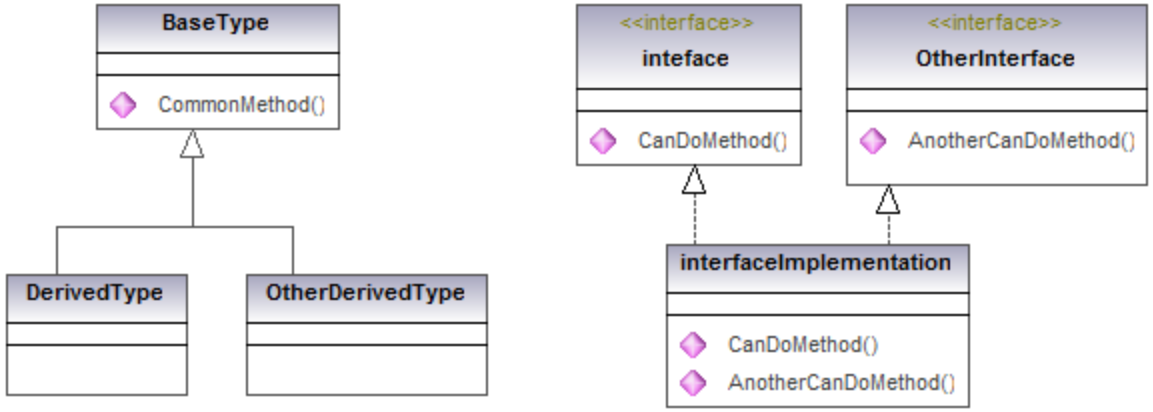
- UML is a picture language
  - Efficient way to capture the essence of a pattern
- UML is not a process
- Range of diagram types
  - Static
    - Class, Deployment, Use Case, Package
  - Interaction
    - Activity, Sequence, Collaboration, Finite State

Class Diagrams



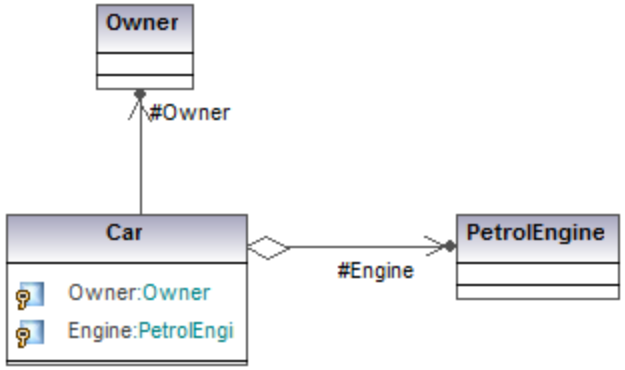
Class Diagrams, Relationships

Inheritance
Is-kind-of
Can Do



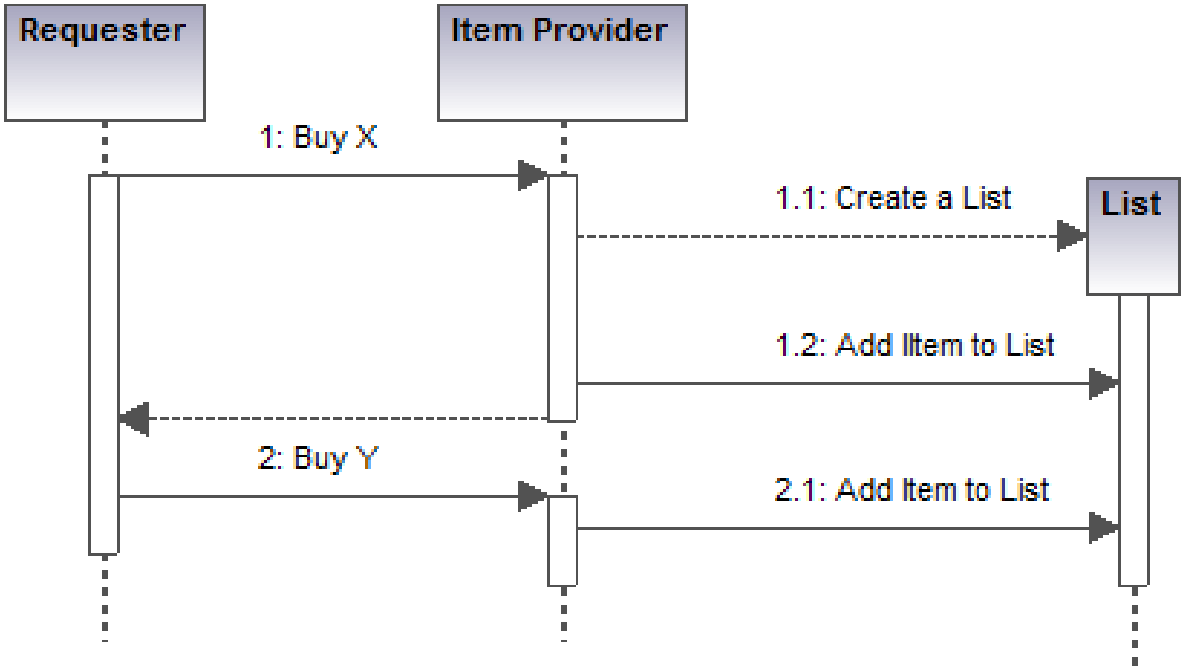
Association

Aggregation, Composition



Sequence Diagrams

- Visualise the interaction between objects typically for a given use case/part of use case



WARNING

- Knowledge of Design Patterns is akin to
  - Carrying a concealed weapon, just because you have one doesn't mean you need to use it
- Follow the KISS principle…
  - **K**eep **I**t **S**hort And **S**imple
- Don't engineer hypothetical extensibility
  - Overuse of patterns increases complexity
  - Use refactoring to create extensibility when required

Summary

- Patterns are solution re-use not code re-use
- Learning and understanding patterns
  - Will enhance your understanding of OO
  - Will allow simpler and more efficient communication
  - Will allow you to reuse well thought out designs and robust solutions