School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP5911M
Advanced Software Engineering

7: Fundamentals of Design Patterns

Nick Efford

https://comp5911m.info

Objectives

UNIVERSITY OF LEEDS

- To introduce the concept of **design patterns**
- To see, by means of an example, how a pattern provides a good solution to a design problem
- To discuss two examples of classic patterns

Good & Bad Designers

A Design Pattern Is…

UNIVERSITY OF LEEDS

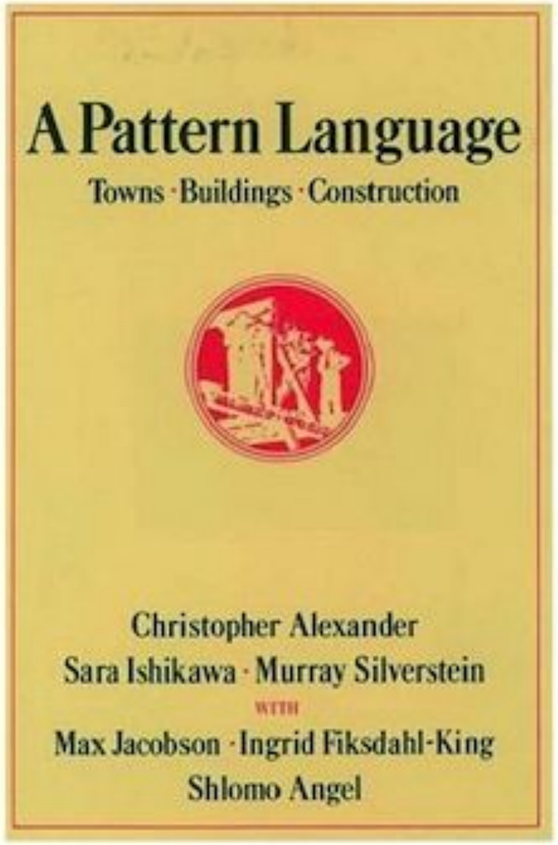
- A named, well-understood solution to a common object-oriented design problem
- A tried-and-tested approach
- A formalisation of experience-based knowledge
- A way for novices to learn to be experts, by example

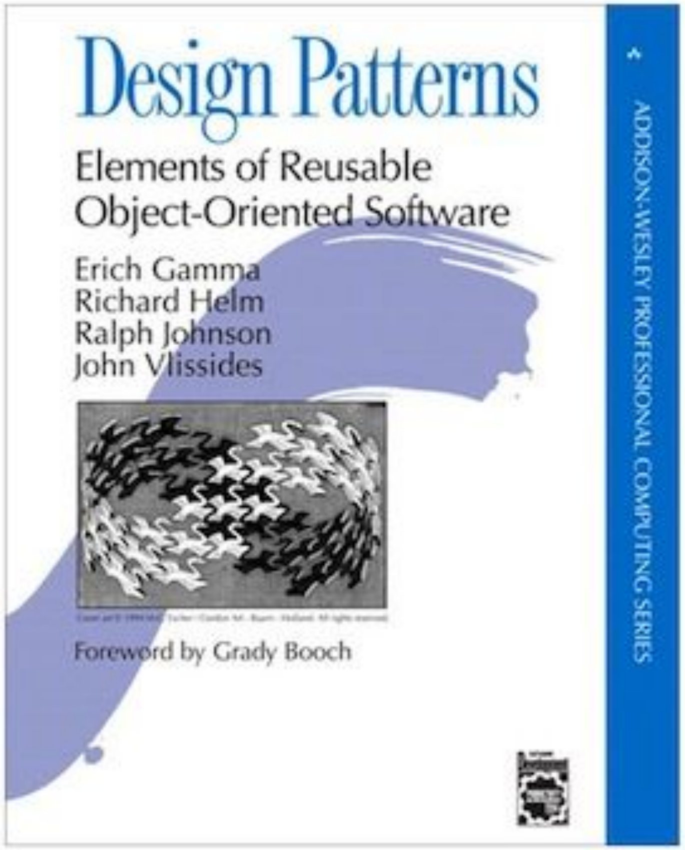
Pattern Catalogue

- A set of patterns, described consistently
- Typical elements are
  - Name (+ any aliases)
  - Brief overview
  - Detailed description of problem solved
  - Solution (diagrams + explanation)
  - Consequences of use
- Classic example: ‘Gang of Four’ book

Pattern Catalogues

UNIVERSITY OF LEEDS


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

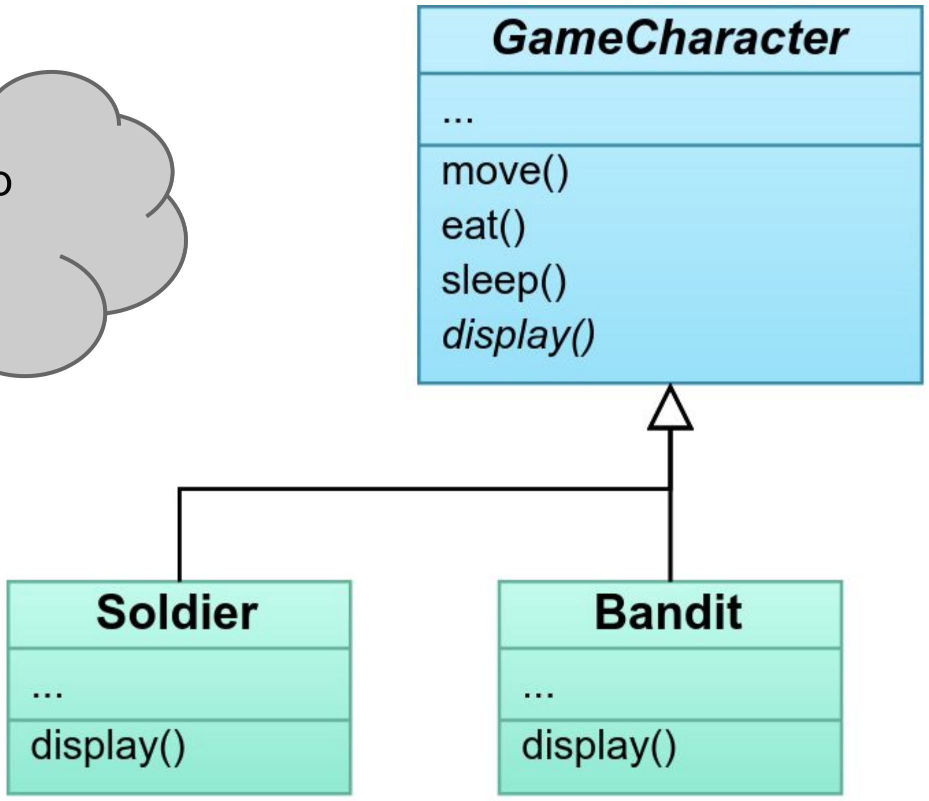
Game Development Scenario

UNIVERSITY OF LEEDS

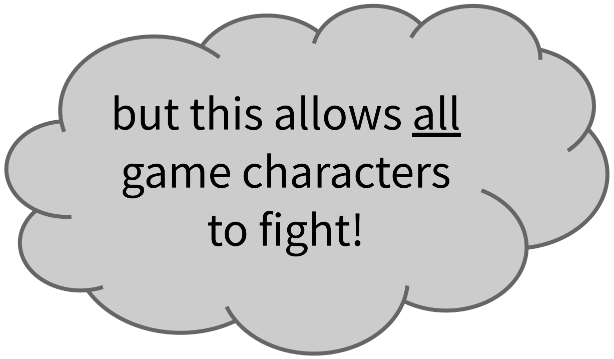
Starting Point

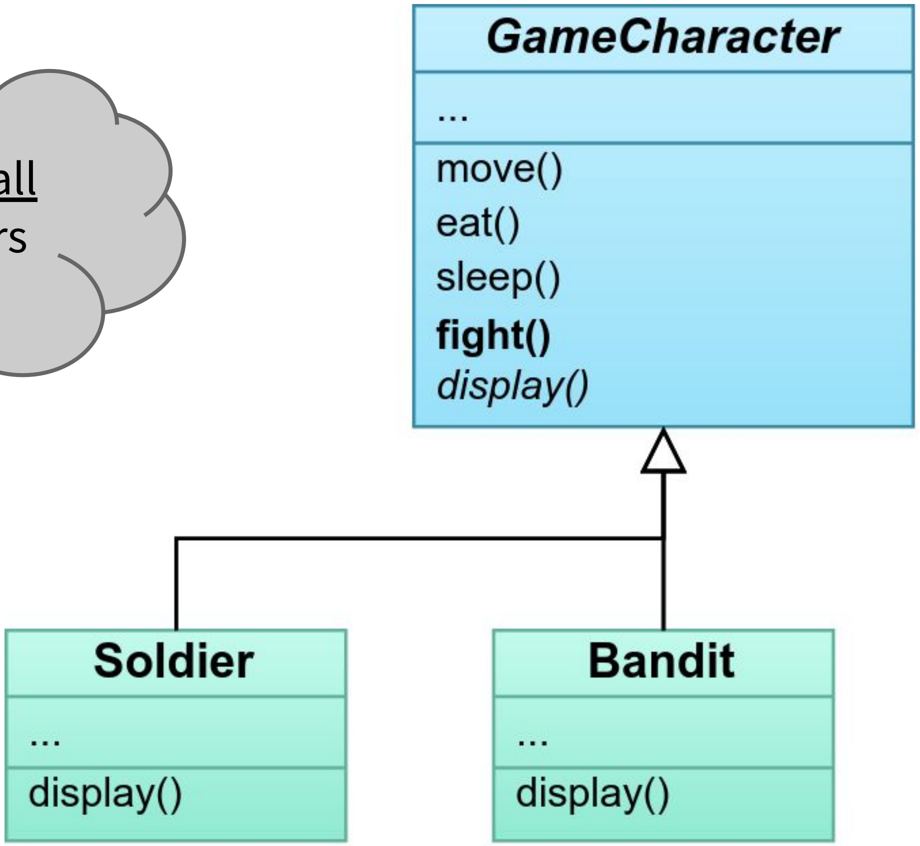
UNIVERSITY OF LEEDS

but we need to add fighting behaviour…

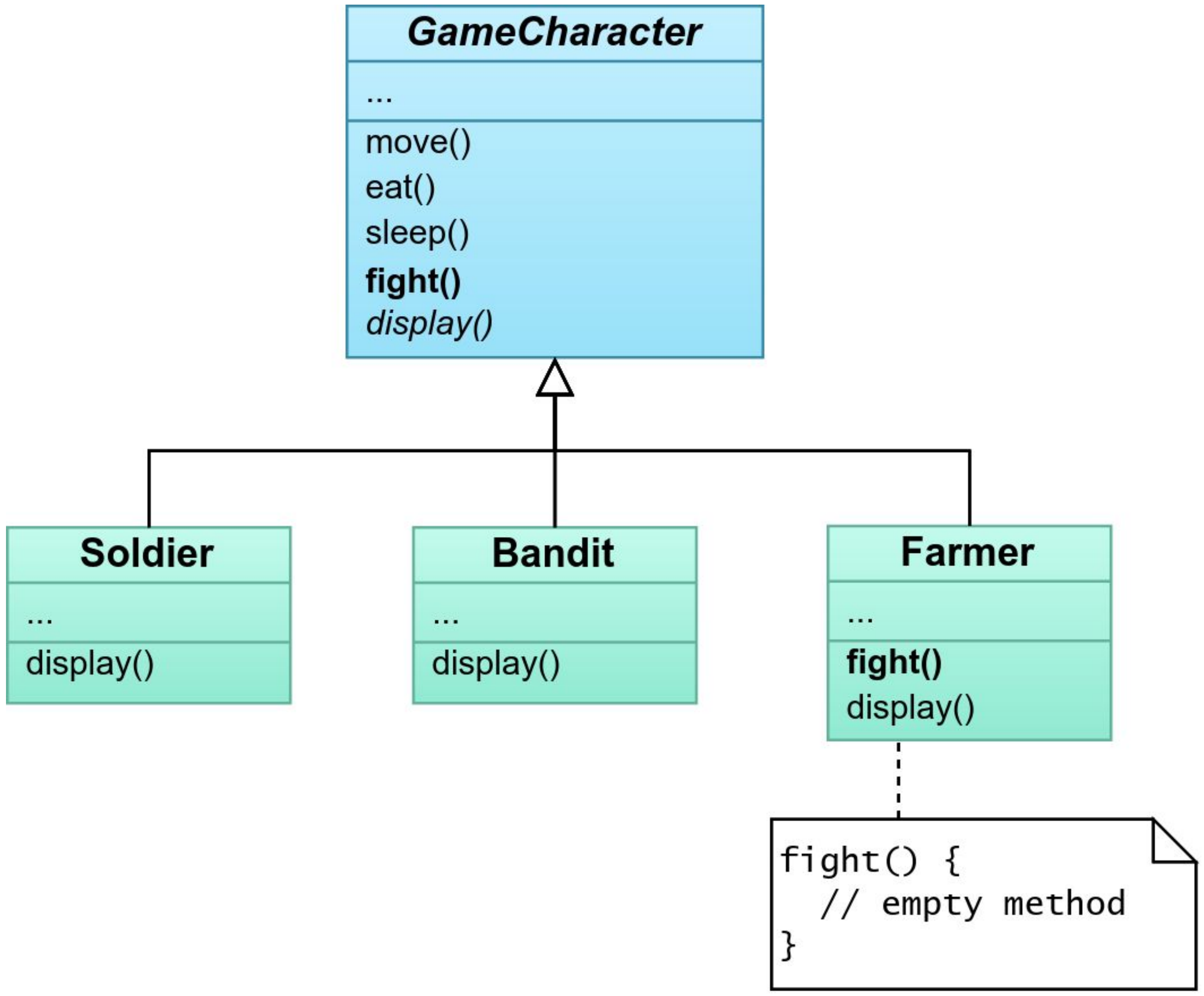


Adding a `fight()` Method





A Possible Solution



```
fight() {
  // empty method
}
```

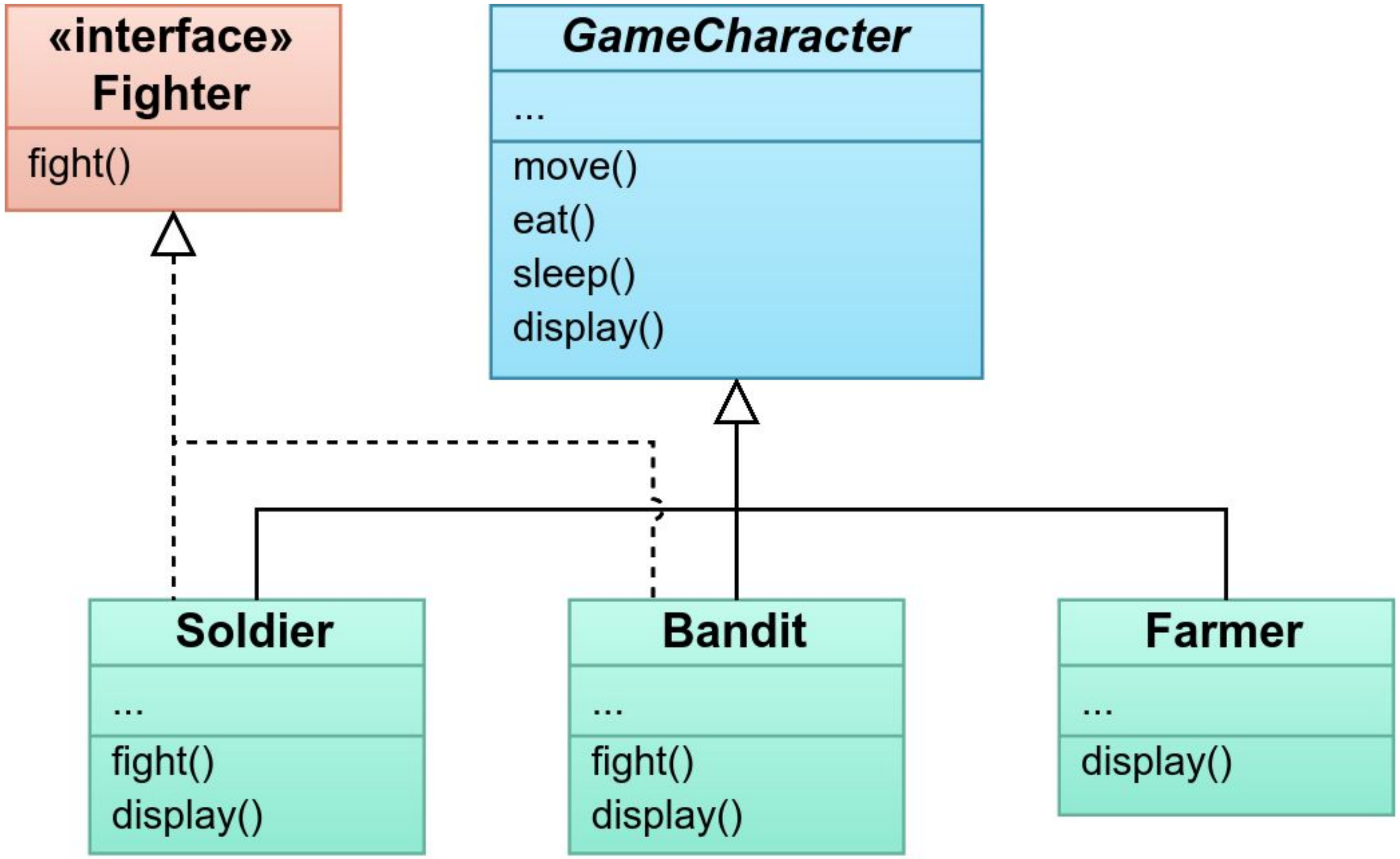
Questions

UNIVERSITY OF LEEDS

- How scalable is this approach?
- What are the limitations? How flexible is it?

Another Possible Solution

UNIVERSITY OF LEEDS



Questions

UNIVERSITY OF LEEDS

- Why is this better?
- What are the limitations? How flexible is it?

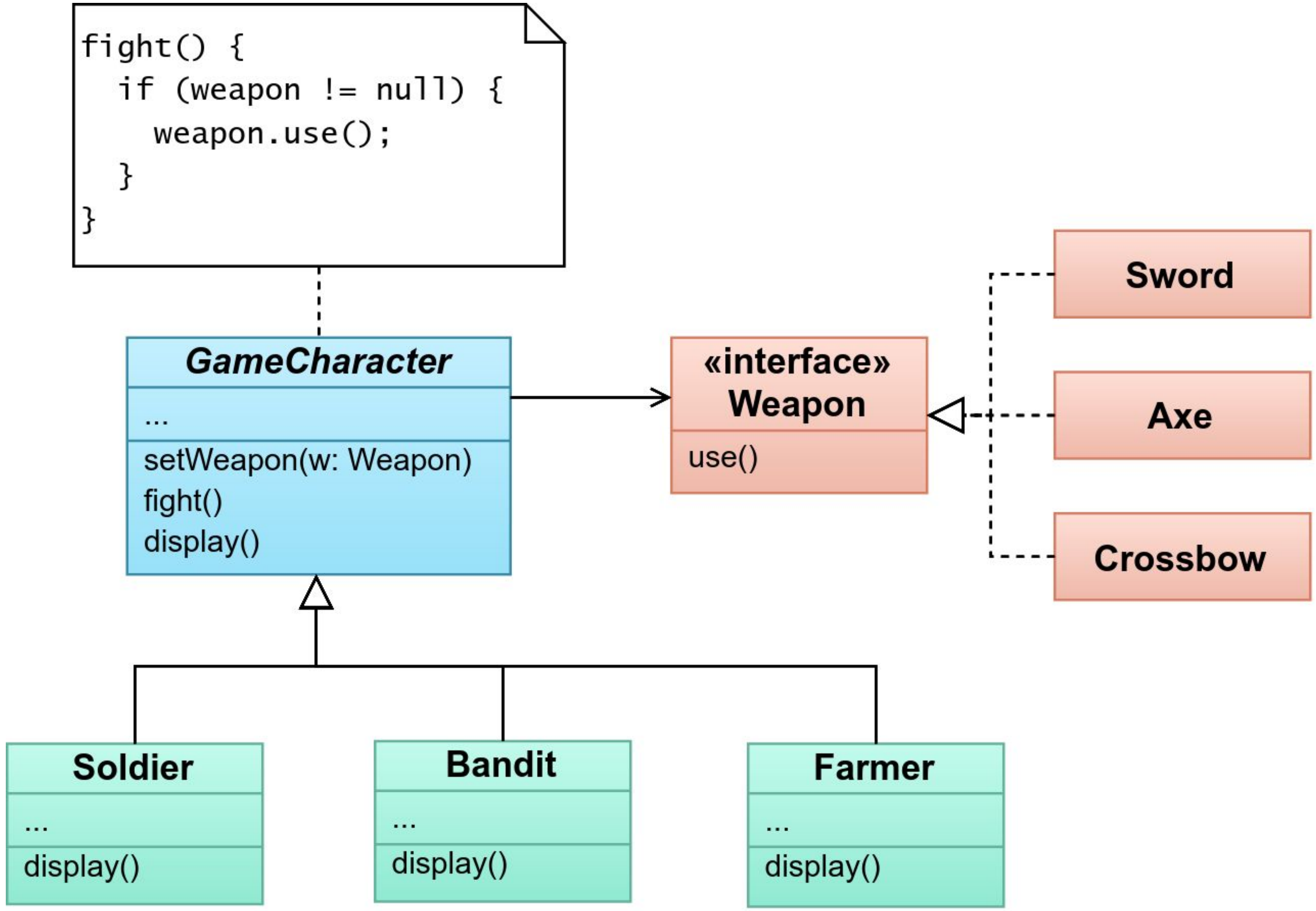
Important Design Principle

UNIVERSITY OF LEEDS

Identify the things in your application that vary;

Keep them separate from the things that stay the same

Our Final Solution


```
fight() {
  if (weapon != null) {
    weapon.use();
  }
}
```

As Code

UNIVERSITY OF LEEDS

```
public abstract class GameCharacter {
  private Weapon weapon;
  public void setWeapon(Weapon w) { weapon = w; }
}

public class Soldier extends GameCharacter {
  public Soldier() {
    setWeapon(new Sword());
  }
}

public class Farmer extends GameCharacter {
  public Farmer() {
    setWeapon(null);
  }
}
```

The Strategy Pattern

Problem

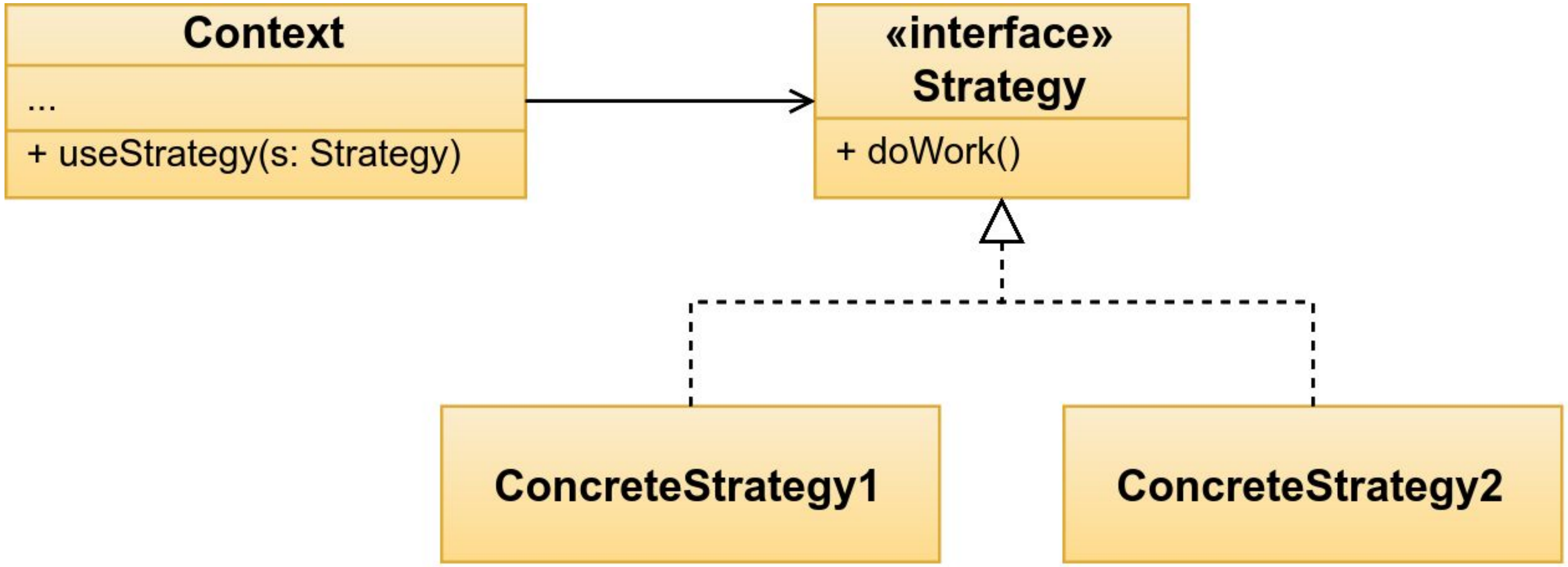
- A class can benefit from different business rules or algorithms, depending on the context in which it is used
- Particular algorithm or set of rules that are chosen depends upon the class user

Solution

- Abstract the algorithm into an interface, the **strategy**
- Provide algorithms as classes implementing this interface
- Have the context class maintain a strategy reference
- Have the user's code supply a concrete strategy object to the context class

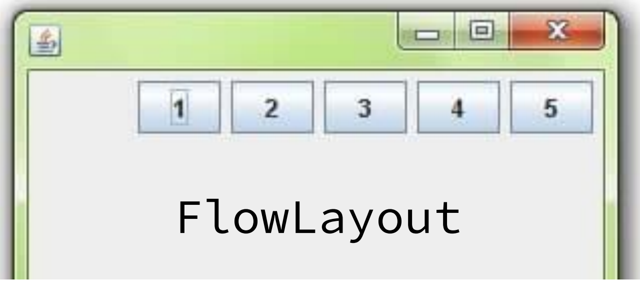
The Strategy Pattern

UNIVERSITY OF LEEDS

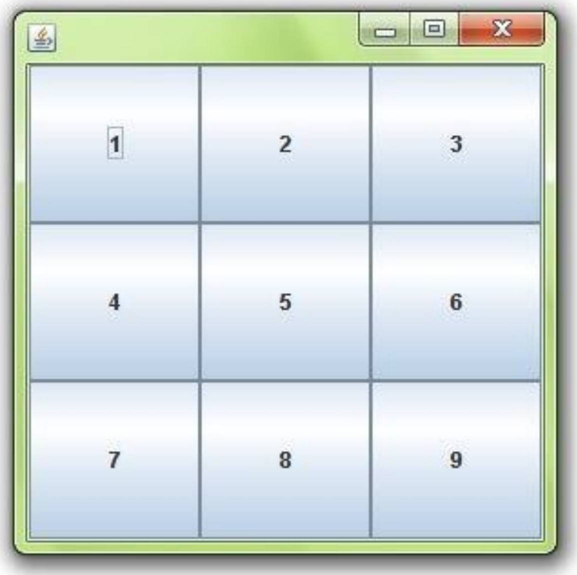


Example: Swing UI Layout

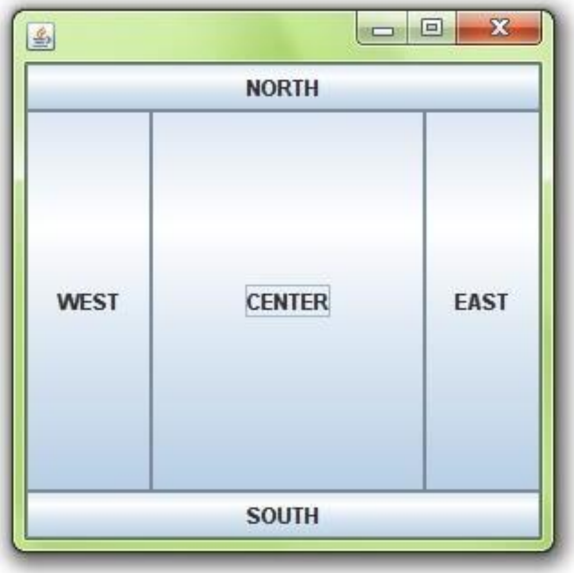
UNIVERSITY OF LEEDS



FlowLayout



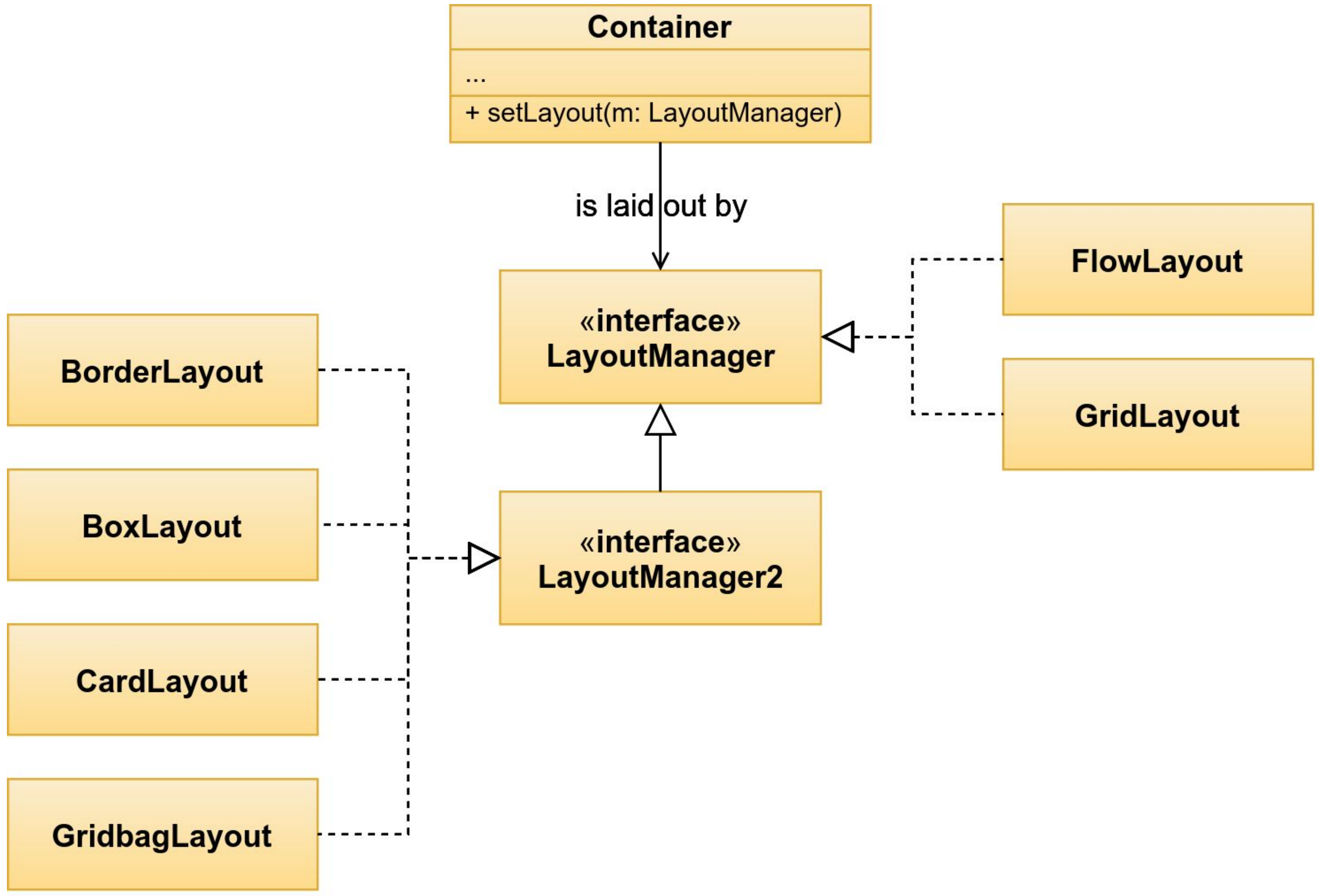
GridLayout



BorderLayout

Example: Swing UI Layout

UNIVERSITY OF LEEDS



The Observer Pattern

UNIVERSITY OF LEEDS

Problem

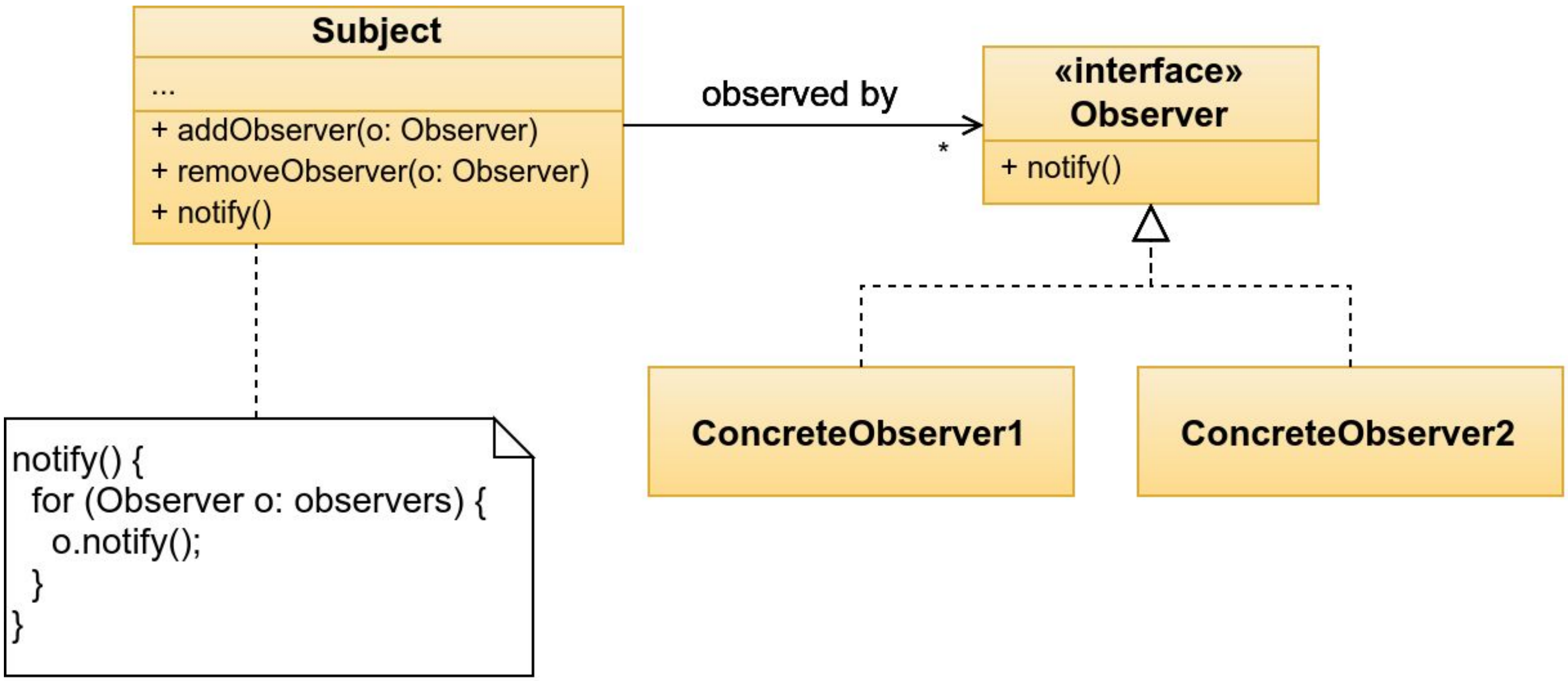
- A **subject** is a source of **events**
- One or more **observers** need to know when events occur

Solution

- Define an interface type for observers; concrete classes must implement this and provide a `notify` method specifying response to event
- Subject class maintains a collection of the observers that are interested in its events
- When an event occurs, subject must call `notify()` on all registered observers

The Observer Pattern

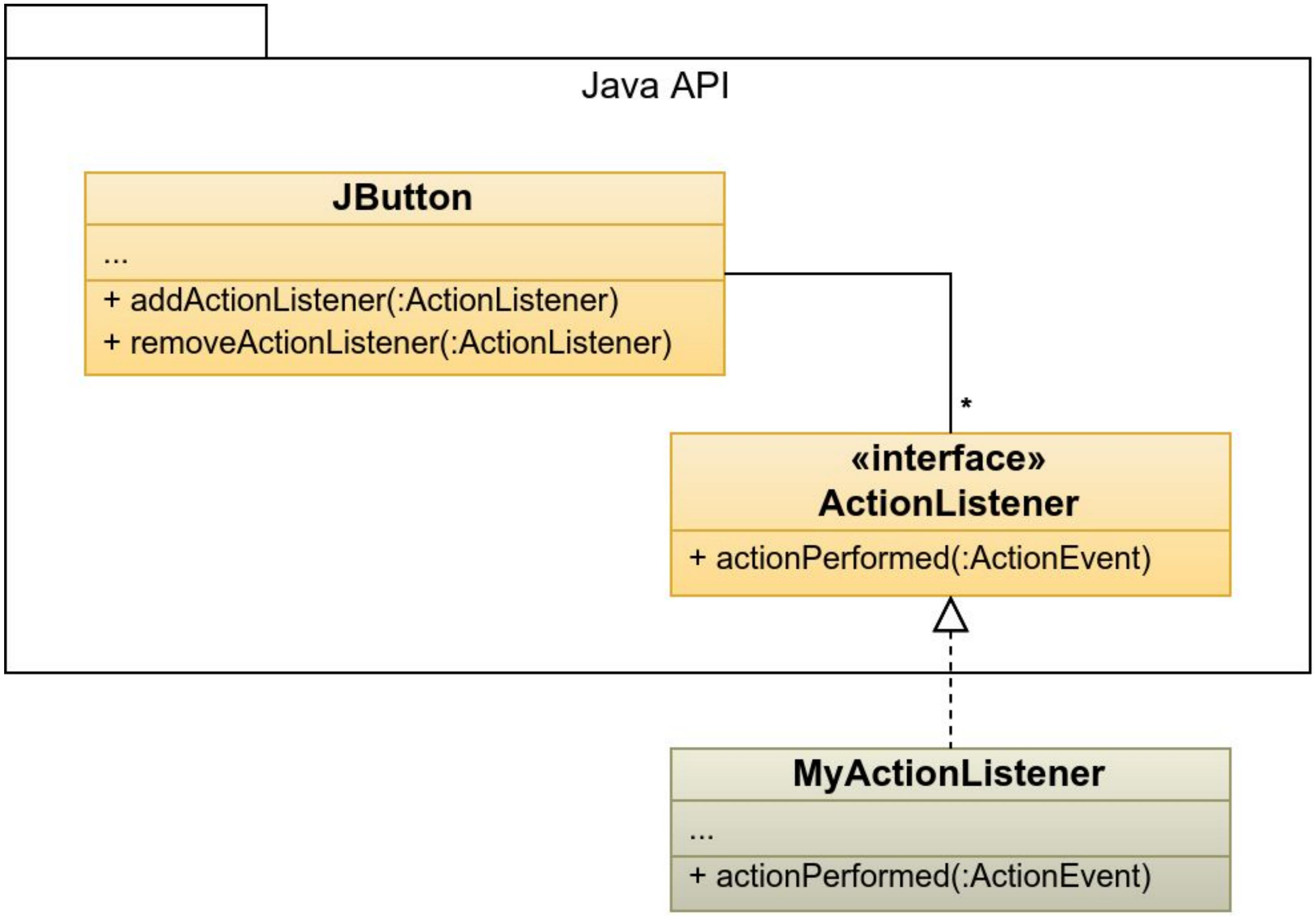
UNIVERSITY OF LEEDS



```
notify() {
  for (Observer o: observers) {
    o.notify();
  }
}
```

Example: Swing UI Event Handling

UNIVERSITY OF LEEDS


Java API
JButton
...
+ addActionListener(:ActionListener)
+ removeActionListener(:ActionListener)
*
«interface»
ActionListener
+ actionPerformed(:ActionEvent)
MyActionListener
...
+ actionPerformed(:ActionEvent)

Summary

UNIVERSITY OF LEEDS

We have

- Introduced the concept of design patterns as a way of capturing the experience & knowledge of expert designers
- Considered a scenario in which the **Strategy** pattern provides a good solution to a problem
- Seen how Java's Swing UI framework uses Strategy and another pattern: **Observer**

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Gamma et al, *Design Patterns: Elements of Reusable Software*
- Refactoring Guru's Catalog of Design Patterns
- Example code showing use of Strategy & Observer