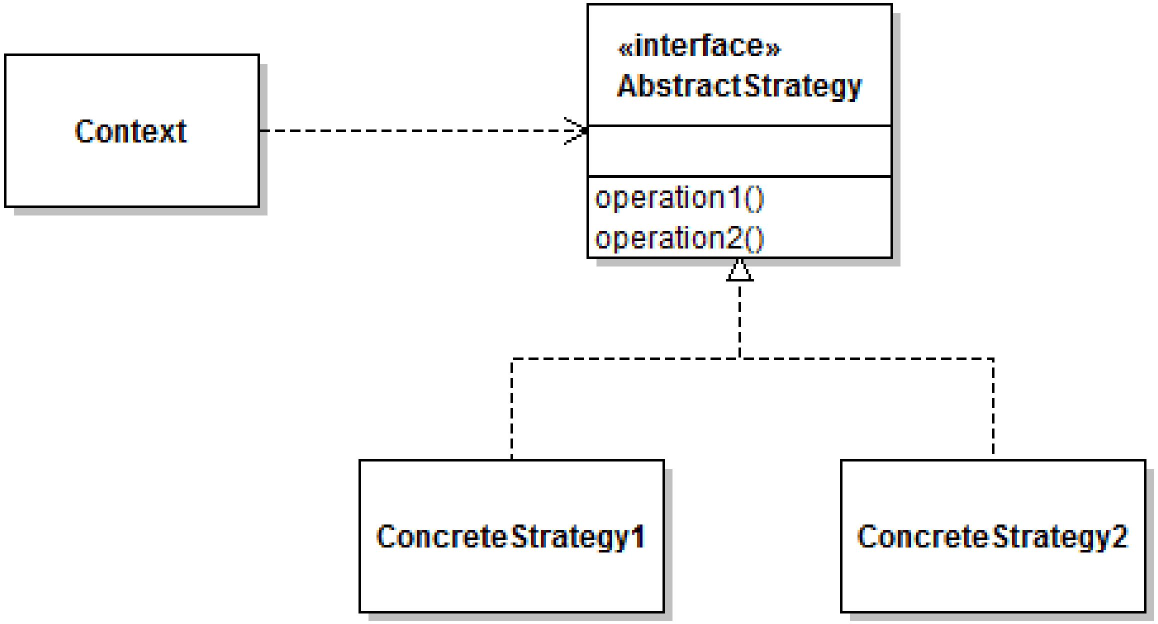
Today

- Review of Polymorphism
- The Strategy Design Pattern
- Class Quality:
  - Static Program Checking
  - Design by Contract
  - Introduction to Unit Testing

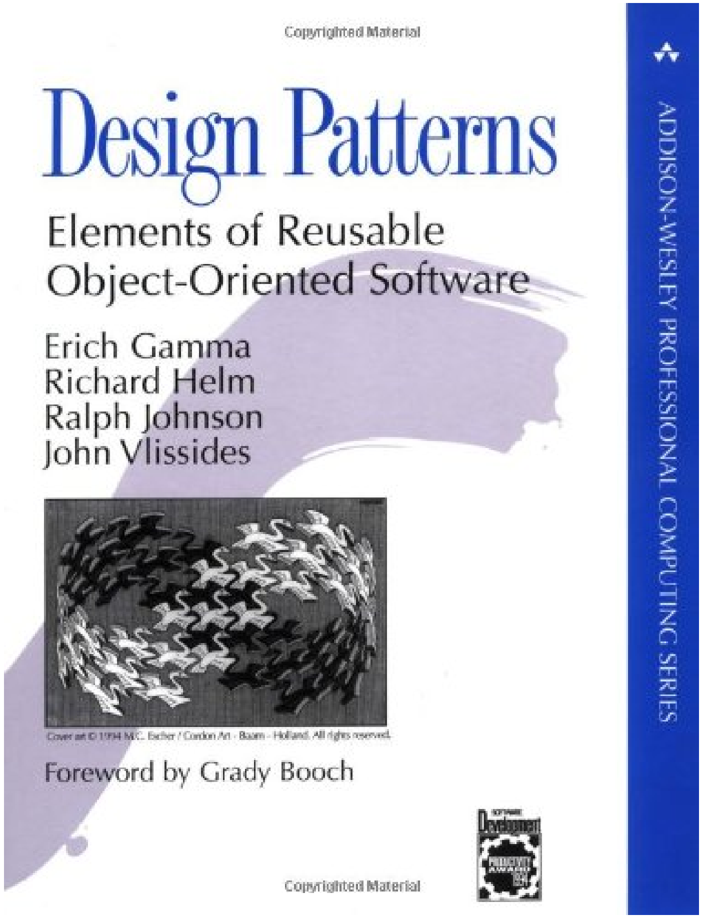
The Strategy Design Pattern

Context:

1. A class (the context) can benefit from different variants of an algorithm
2. Clients of the context may want to supply custom versions of the algorithm



Seminal Work of on Design Patterns



Design Patterns

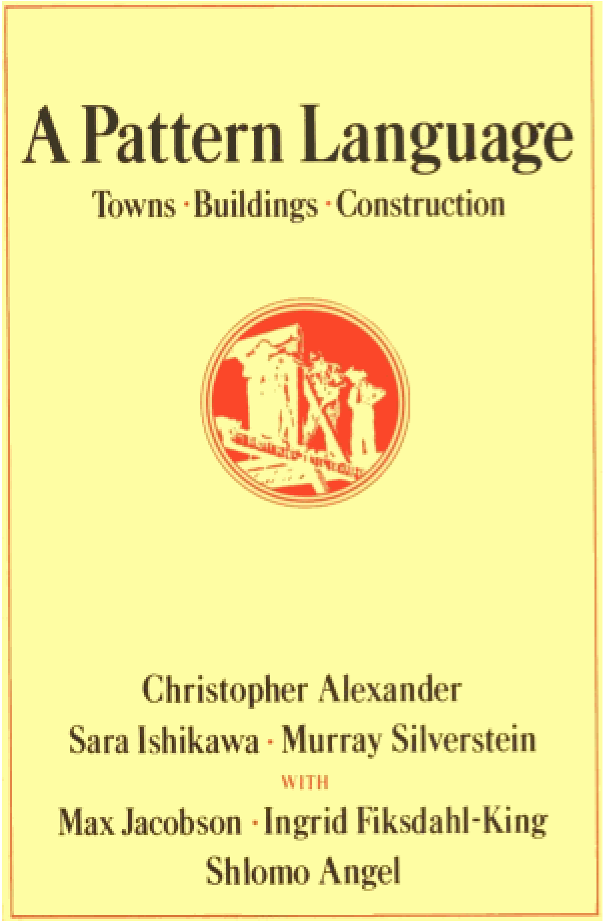
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



A Pattern Language

Towns · Buildings · Construction

Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

The Purpose of Formatting

“The coding style and readability set precedents that continue to affect maintainability and extensibility long after the original code has been changed beyond recognition. Your style and discipline survives, even though your code does not.” --- Robert Martin, *Clean Code*.

Bertrand Meyer on Defensive Programming

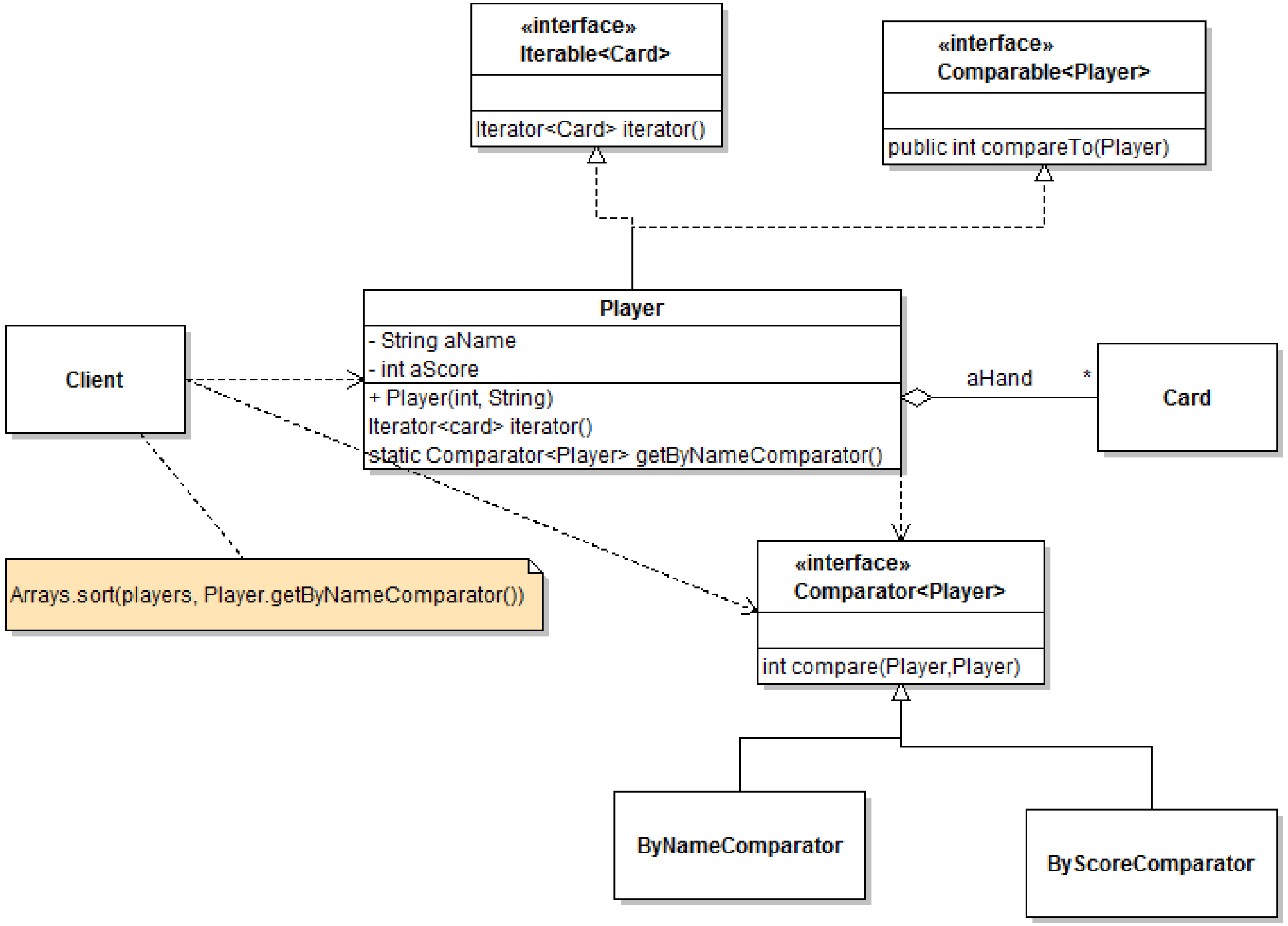
[Defensive programming] often defeats its own purposes. Adding possibly redundant code “just in case” only contributes to the software’s complexity – the single worst obstacle to software quality in general, and to reliability in particular. The result of such blind checking is simply to introduce more software, hence more sources of things that could go wrong at execution time, hence the need for more checks, and so on ad infinitum…

- B. Meyer, Applying “Design by Contract”, *IEEE Computer*, October 1992.

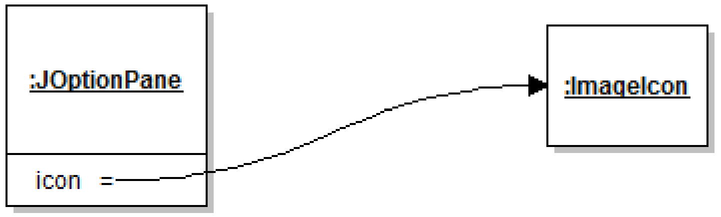
Specifying Method Interfaces

- Acceptable and unacceptable inputs
- Return values and their meaning
- Errors and exceptions
- Side effects
- **Preconditions**
- **Postconditions**
- **Invariants**
- Performance guarantees

Class Diagram Representing Player


«interface»
Iterable<Card>
Iterator<Card> iterator()
«interface»
Comparable<Player>
public int compareTo(Player)
Player
- String aName
- int aScore
+ Player(int, String)
Iterator<card> iterator()
static Comparator<Player> getByNameComparator()
Client
aHand
*
Card
Arrays.sort(players, Player.getByNameComparator())
«interface»
Comparator<Player>
int compare(Player,Player)
ByNameComparator
ByScoreComparator

Object Diagram of the Icon Scenario


:JOptionPane
icon =
:ImageIcon