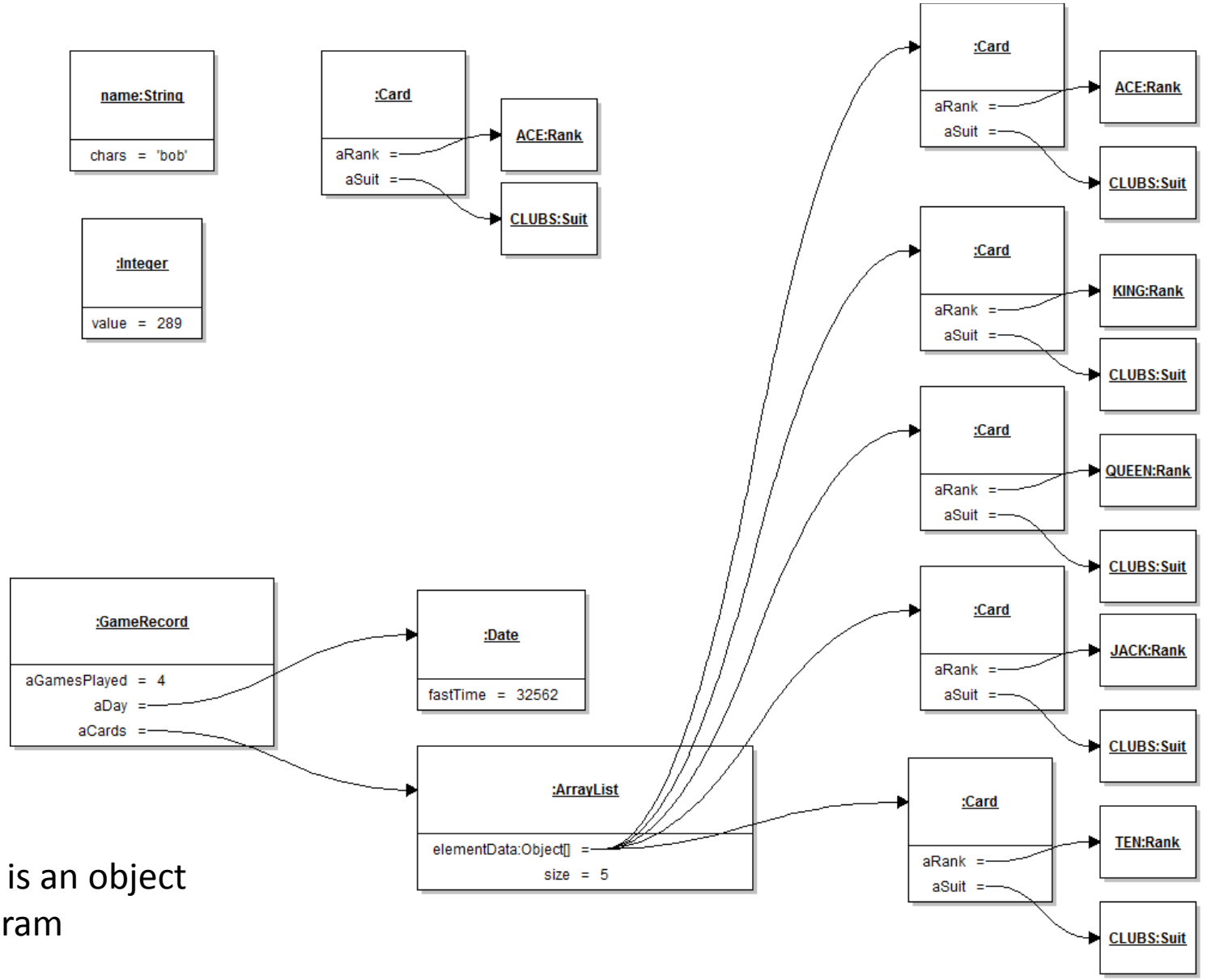
Today

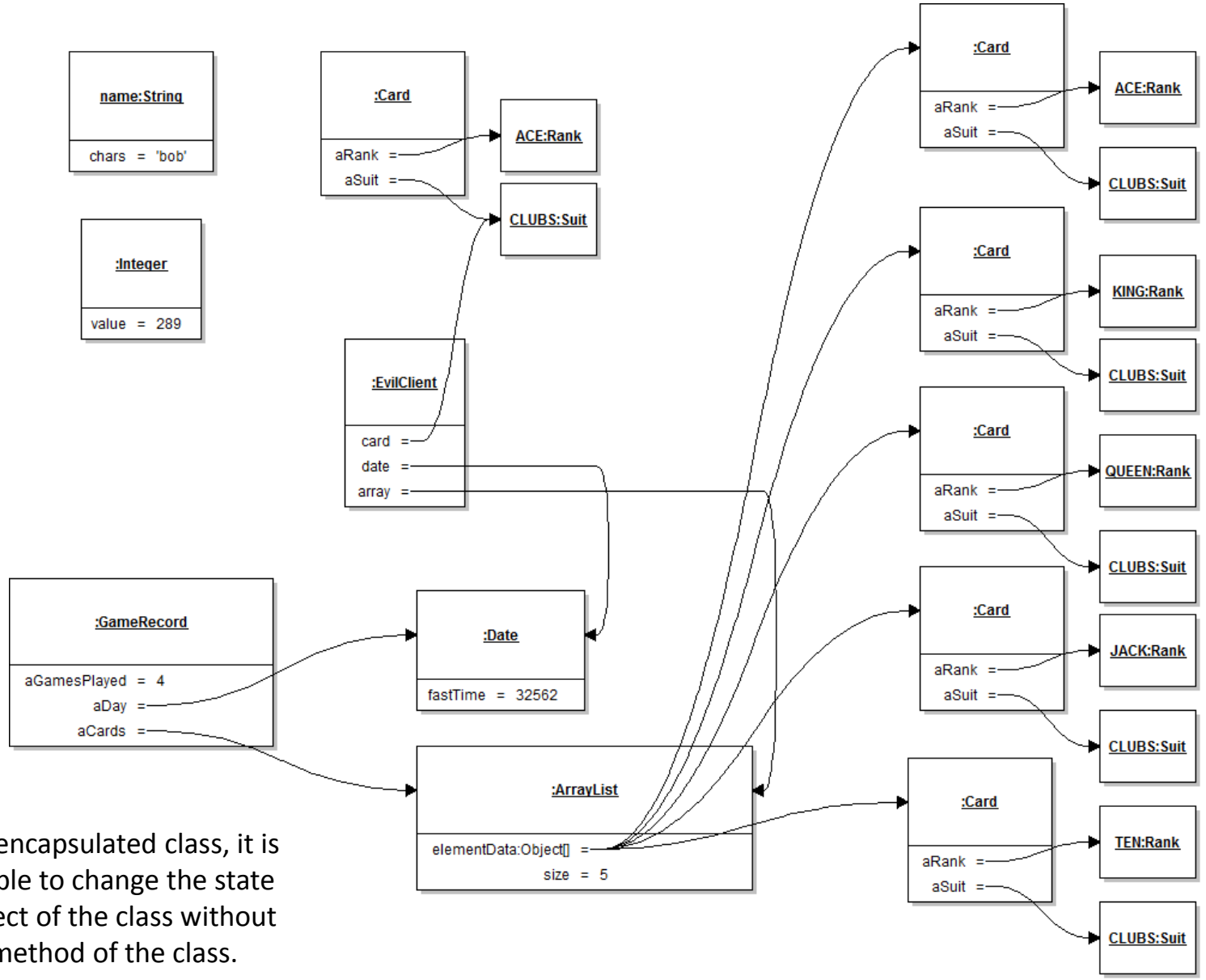
- More on encapsulation
- Introduction to UML – the Unified Modeling Language
- Polymorphism through interfaces
- A glance at the Iterator and Strategy design patterns

Object State



This is an object diagram

Shared Object State

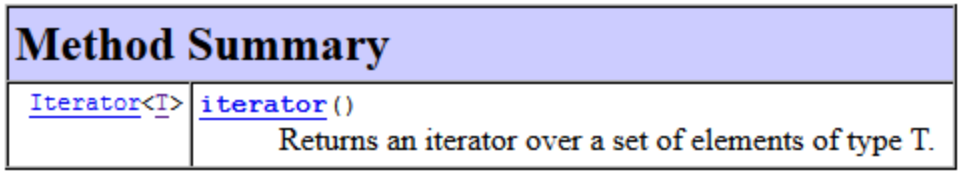


In a well-encapsulated class, it is not possible to change the state of an object of the class without calling a method of the class.

An interface declaration establishes a contract:
The class has to supply all the methods in the interface

```
public class GameRecord implements Iterable<Card>
```

The contract:

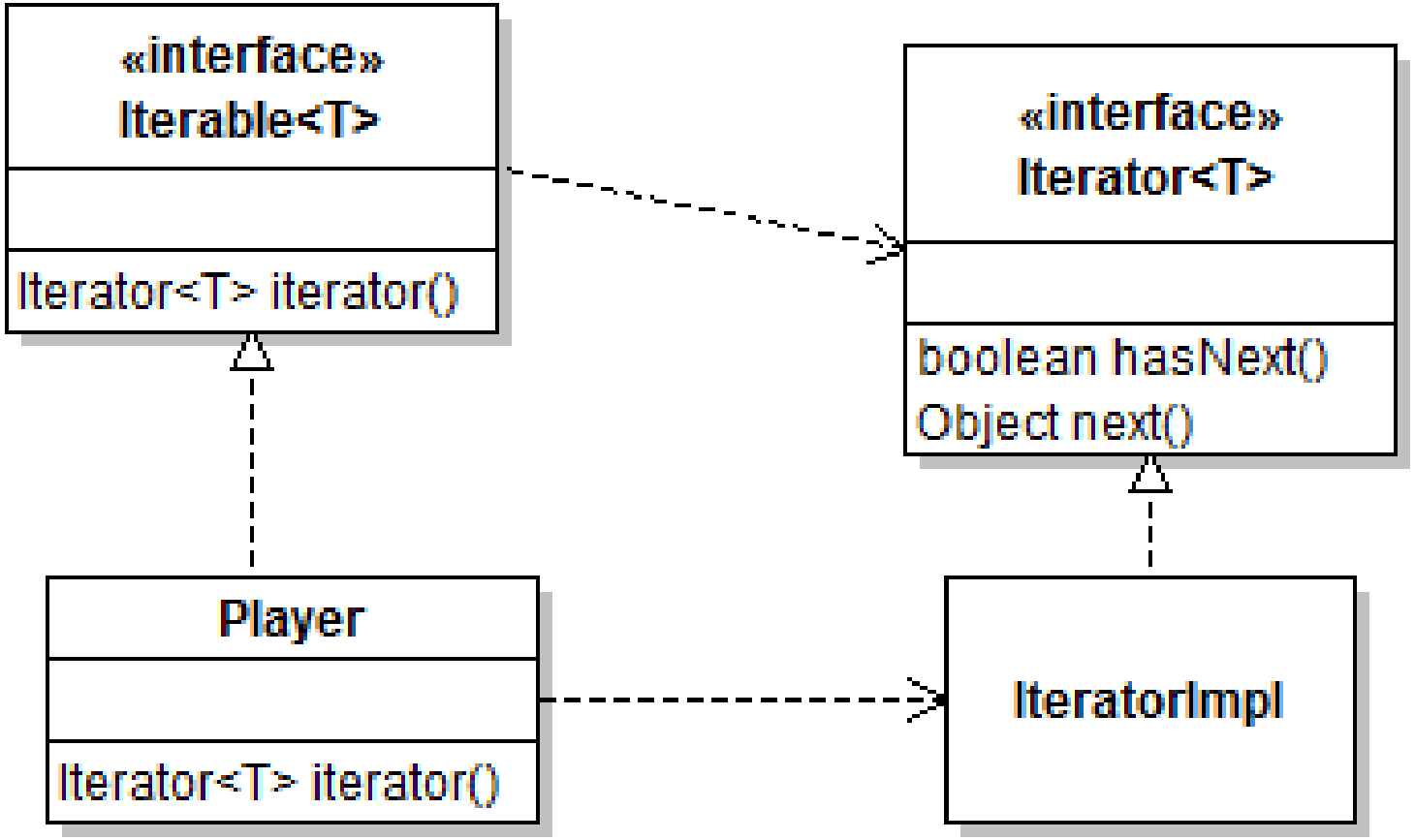


Method Summary	
Iterator<T>	iterator() Returns an iterator over a set of elements of type T.

```
Player player = new Player(…)
…
Iterable<Card> i = player;
```

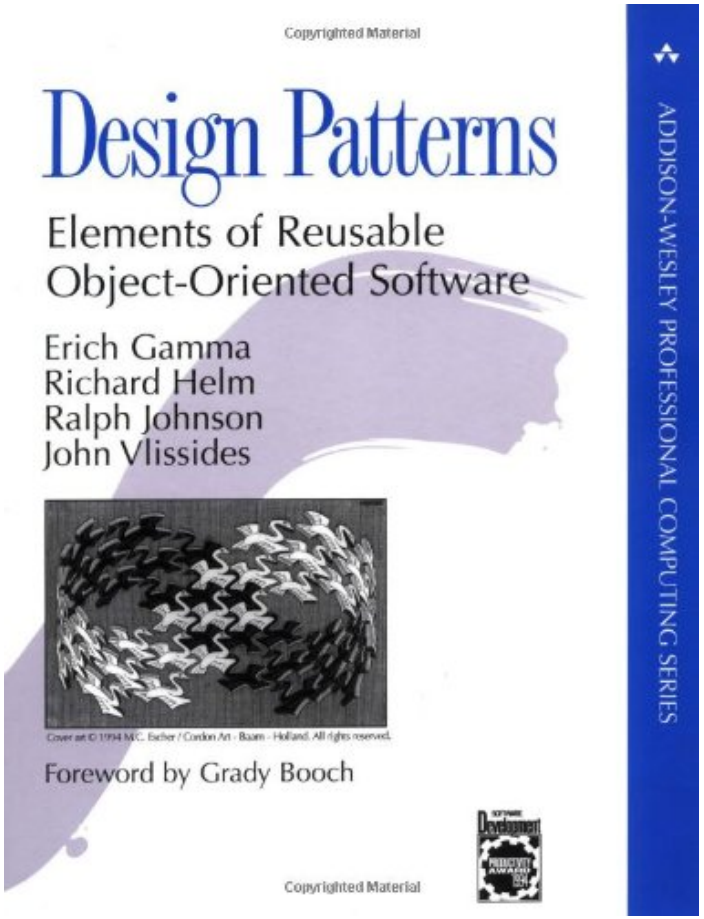
As a result of this contract, you can assign values of the class type to variables of the interface type. Polymorphism in action!

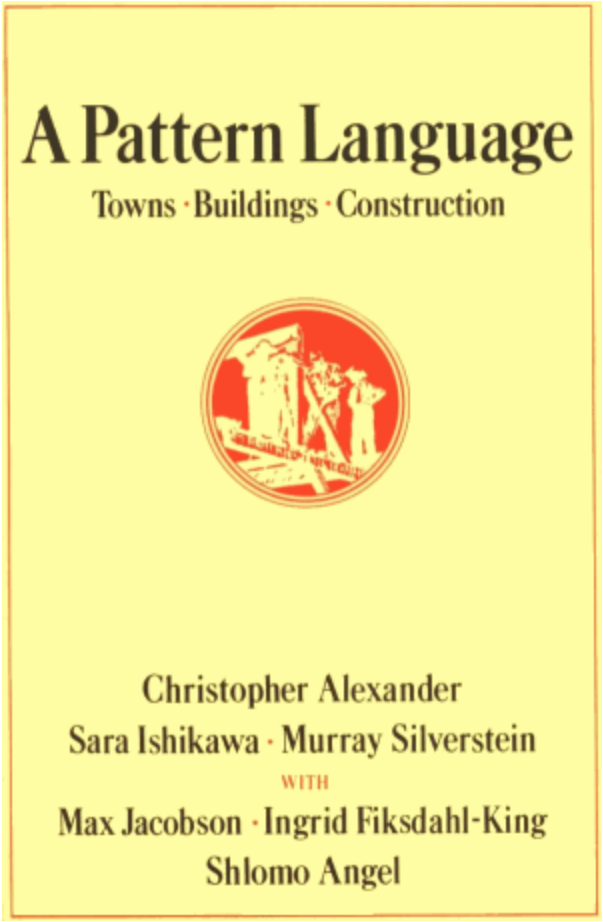
Representing Interface Contracts in UML



This is a class diagram

Seminal Work of on Design Patterns

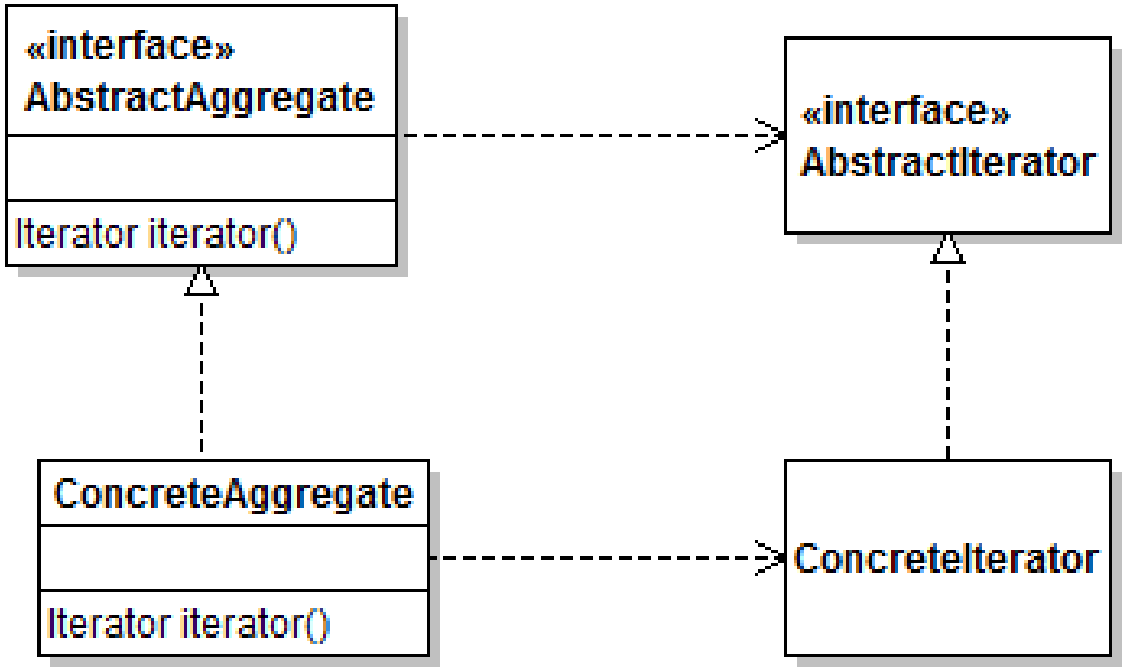




The Iterator Design Pattern

Context:

1. An object (the aggregate) contains other objects (the elements)
2. Clients need to access the elements
3. The aggregate should not expose its internal structure
4. There may be multiple clients that need simultaneous access



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