Design Patterns

CSC207 Fall 2016

Design Patterns

A **design pattern** is a general description of the solution to a well-established problem using an arrangement of classes and objects.

Patterns describe the shape of code rather than the details.

They're a means of communicating design ideas.

They are not specific to any one programming language.

You'll learn about lots of patterns in CSC301 (Introduction to Software Engineering) and CSC302 (Engineering Large Software Systems).

Gang of Four

First codified by the Gang of Four in 1995

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides

Original Gang of Four book described 23 patterns

- More have been added

- Other authors have written books

Book provides an overview of:

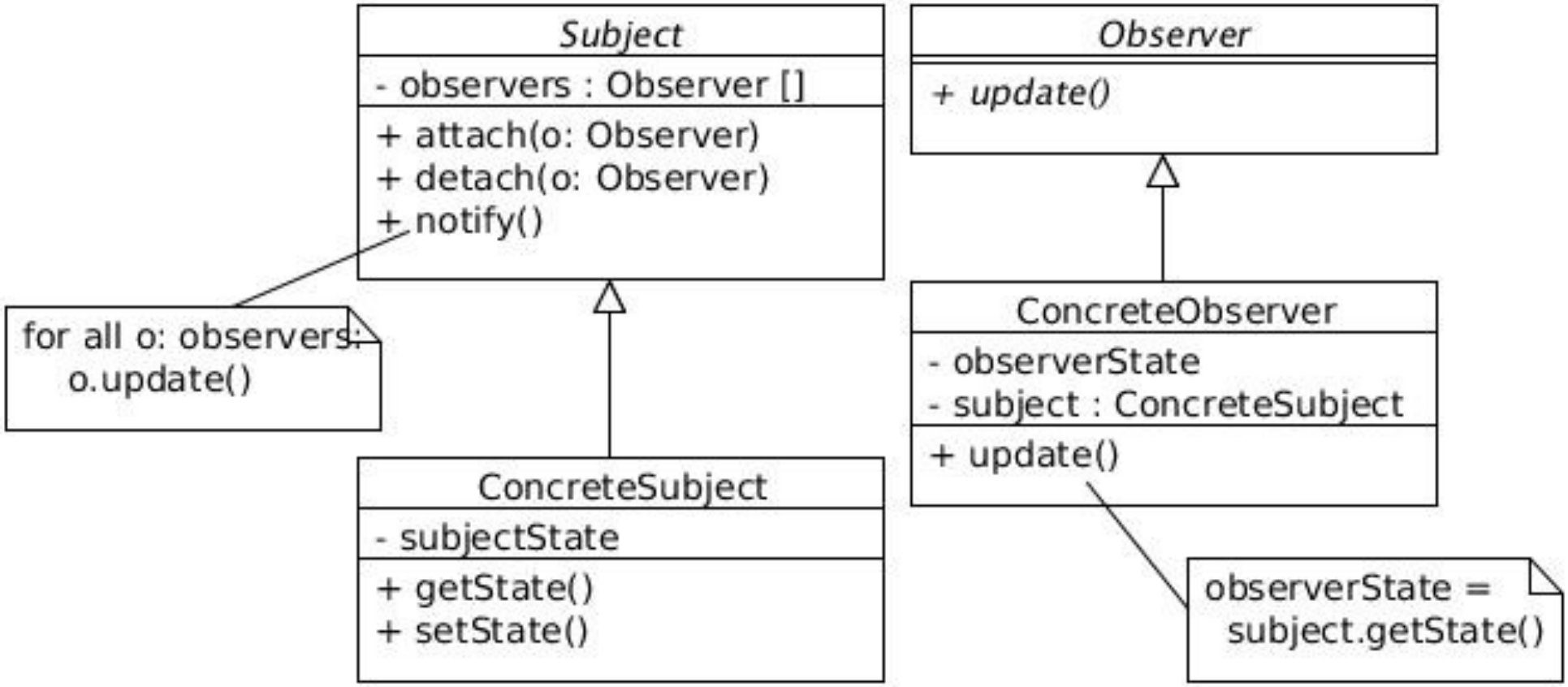
- **Name**
- **Problem**: when to use the pattern
  - motivation: sample application scenario
  - applicability: guidelines for when your code needs this pattern
- **Solution**:
  - structure: UML Class Diagram of generic solution
  - participants: description of the basic classes involved in generic solution
  - collaborations: describes the relationships and collaborations among the generic solution participants
  - sample code
- Consequences, Known Uses, Related Patterns, Anti-patterns

Observer Design Pattern

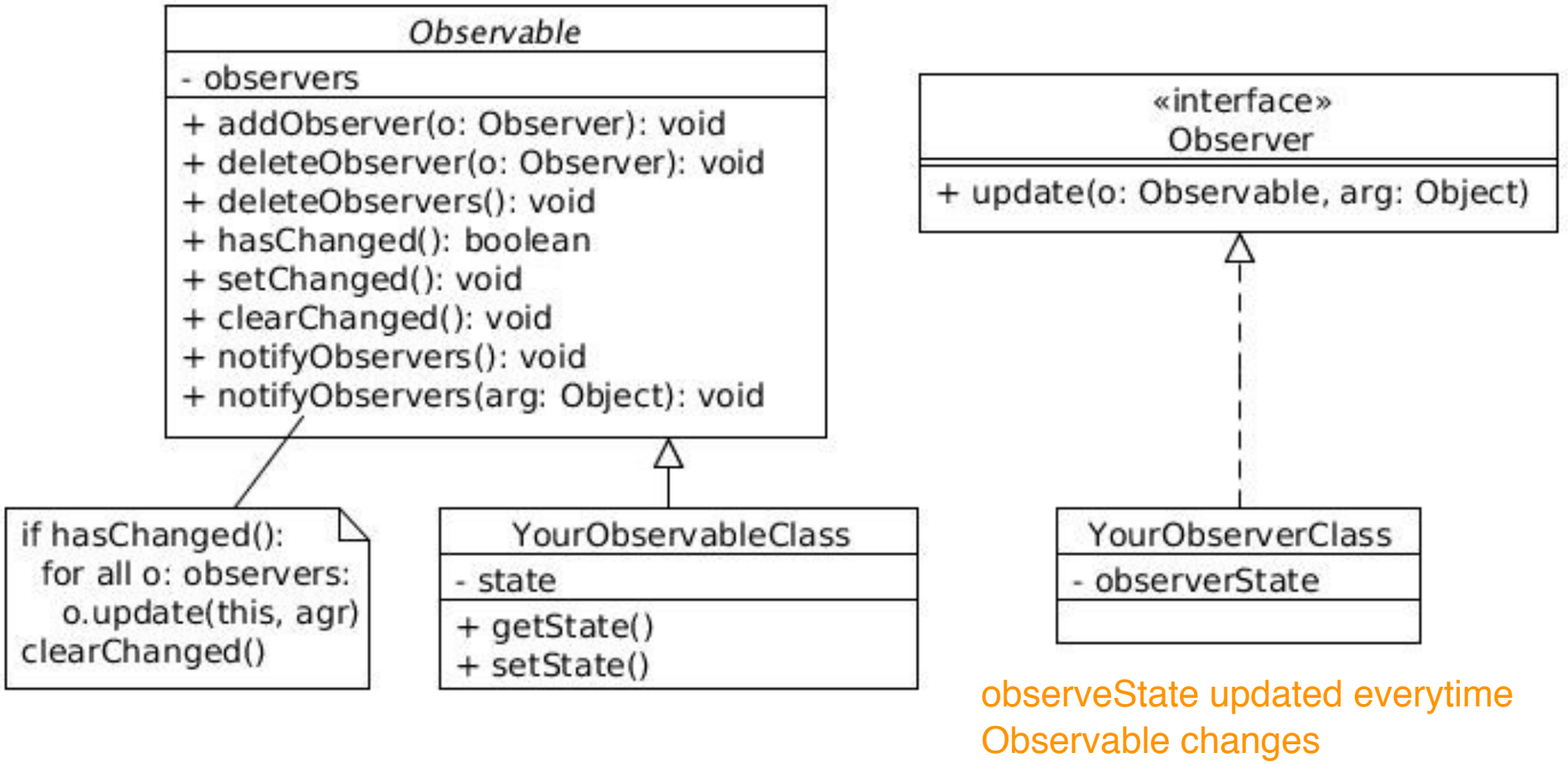
Problem:

- Need to maintain consistency between related objects.
- Two aspects, one dependent on the other.
- An object should be able to notify other objects without making assumptions about who these objects are.

Observer: Standard Solution

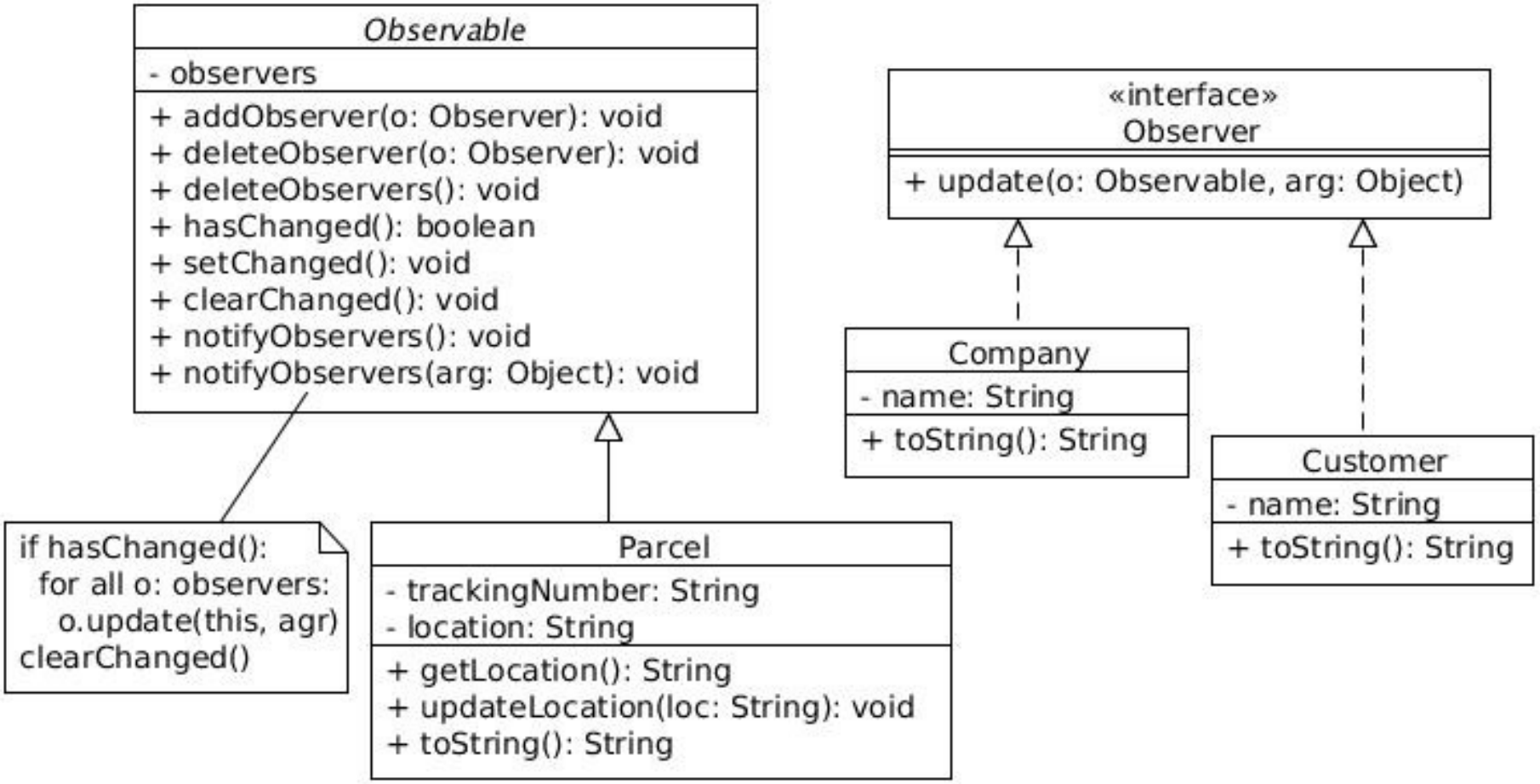


Observer: Java Implementation



observeState updated everytime Observable changes

Observer: Example in Java



notifyObservers called upon state change, i.e. updateLocation()

Uses of Observer in Java

In reality, people usually implement their own

- Usually can't or don't want to subclass from `Observable`
- Can't have your own class hierarchy and multiple inheritance is not available
- Has been replaced by the Java Delegation Event Model (DEM)

use this!

  - Passes event objects instead of update/notify

Listener is specific to GUI classes

Singleton Design Pattern

Context

- Classes for which only one instance should exist (singleton).
- Provide a global point of access.

Problem

- How do you ensure that it is never possible to create more than one instance of a singleton class?

Forces

- The use of a public constructor cannot guarantee that no more than one instance will be created.
- The singleton instance must be accessible to all classes that require it.

declares a public getter method

Singleton: Solution

Clients access a `Singleton` instance solely though `Singleton`'s `getInstance()` operation.

a problem is that instance is created when class Singleton is instantiated not when getInstance() is called: solve this problem by creating a holder method inside Singleton method

Iterator Design Pattern

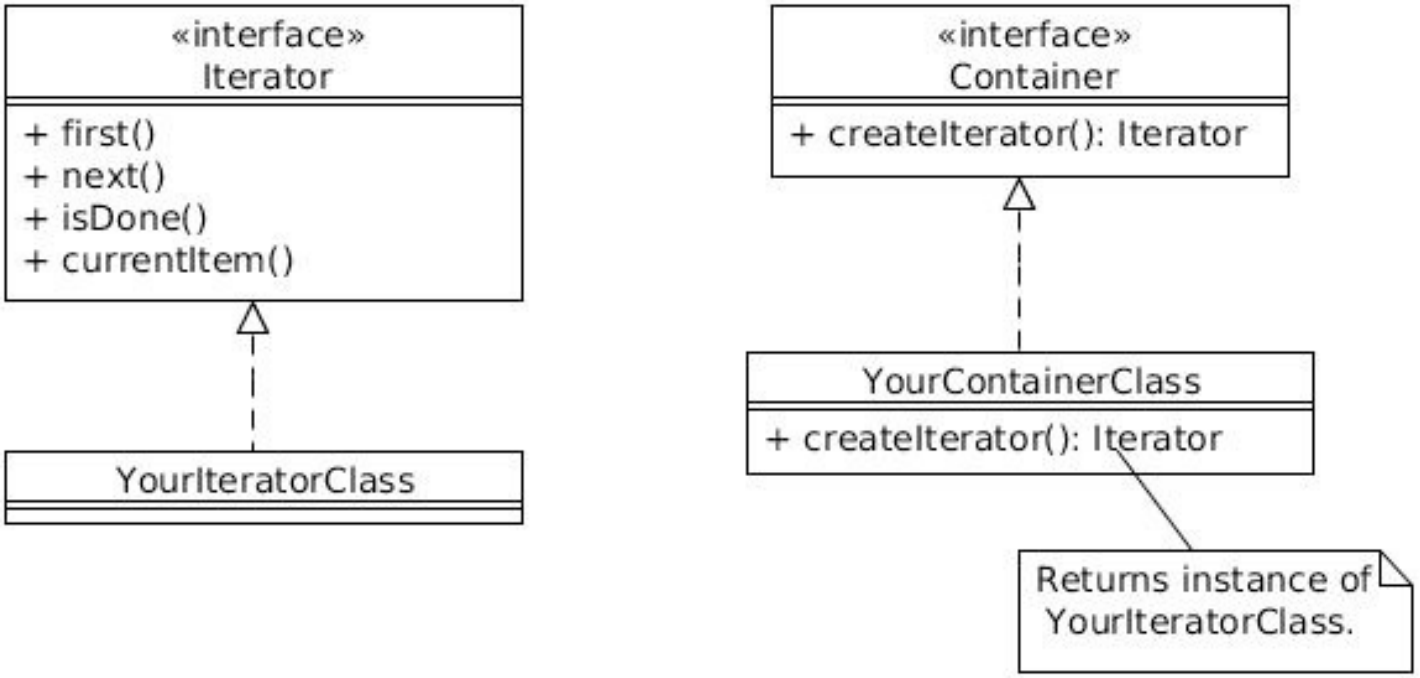
Context

- A container/collection object.

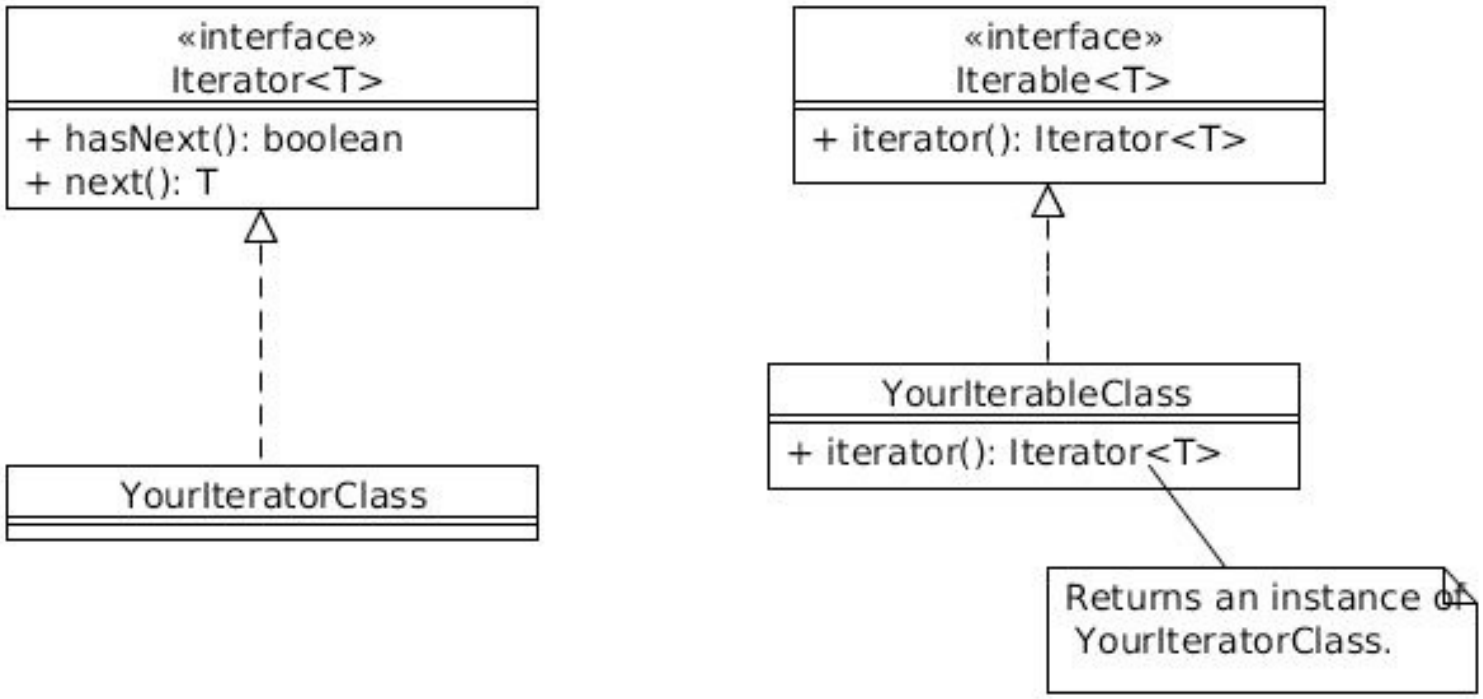
Problem

- Want a way to iterate over the elements of the container.
- Want to have multiple, independent iterators over the elements of the container.
- Do not want to expose the underlying representation: should not reveal how the elements are stored.

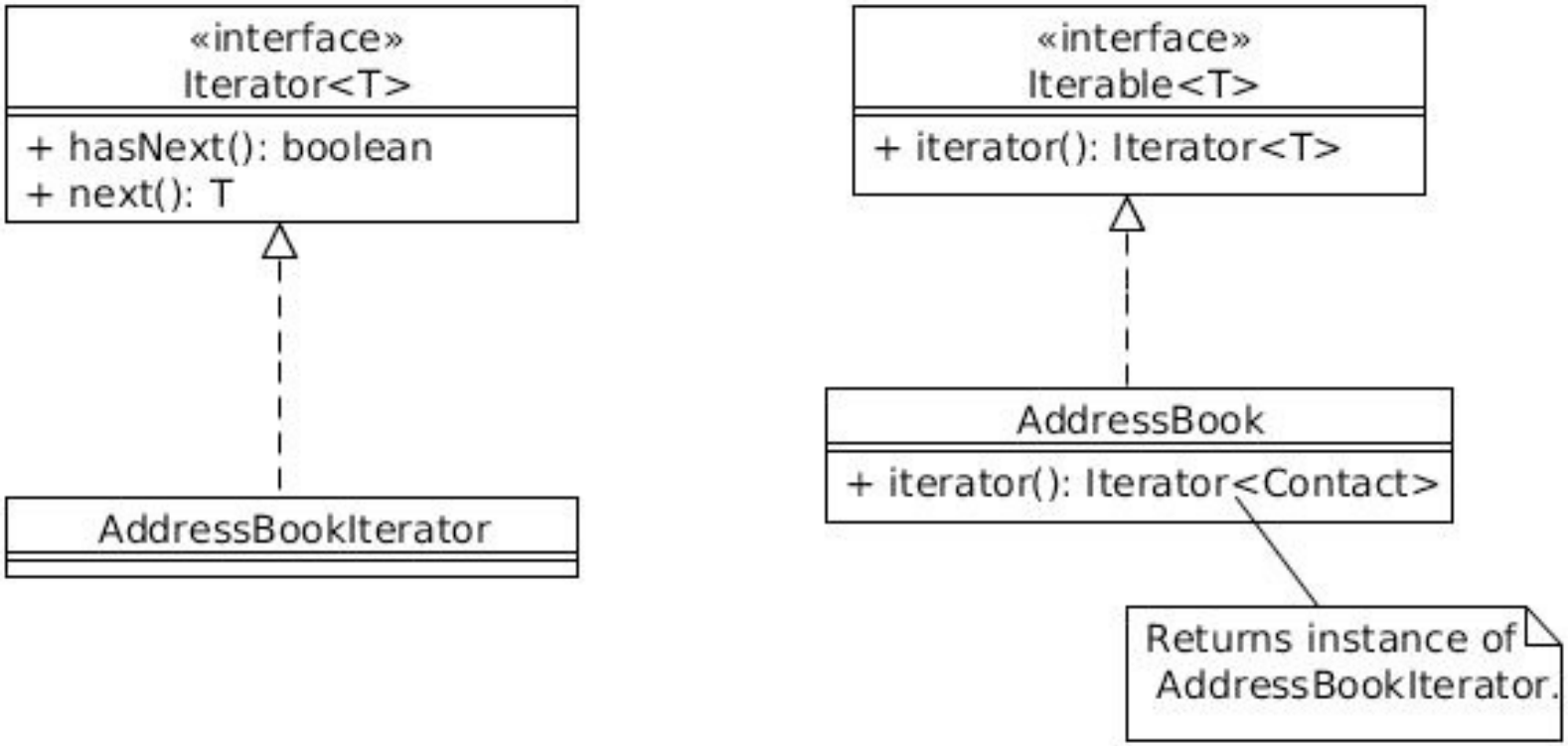
Iterator Design Pattern: Solution



Iterator Design Pattern: Java



Iterator: Example in Java



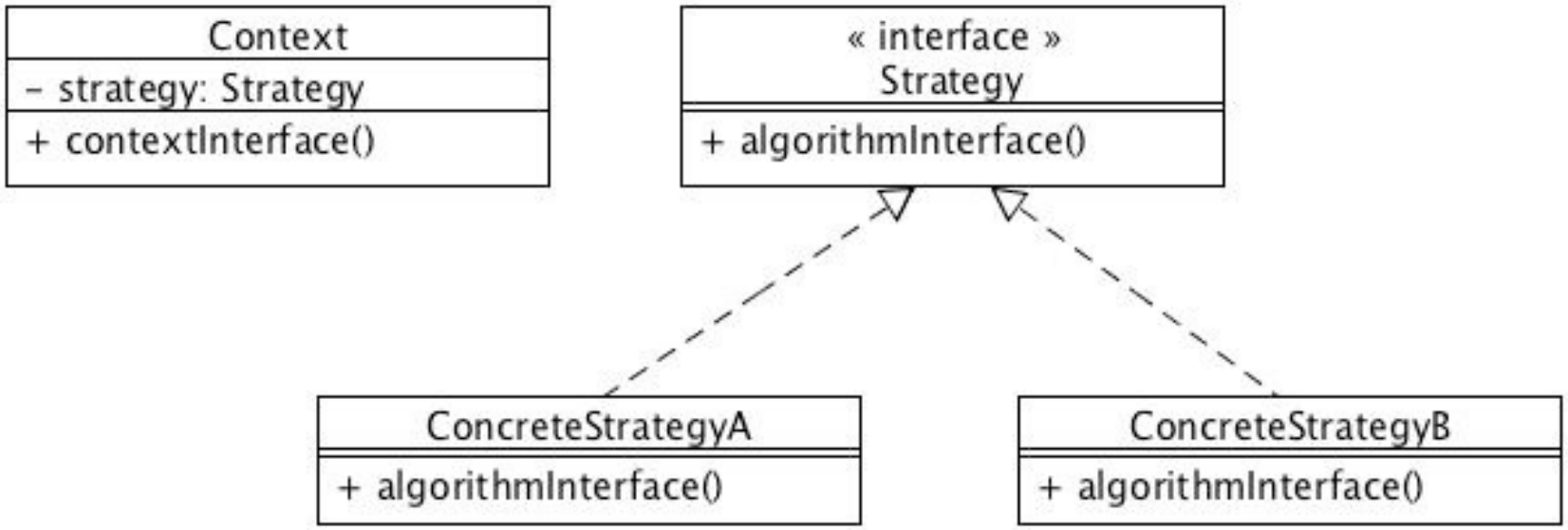
Strategy Design Pattern

Problem:

- multiple classes that differ only in their behaviour (for example, use different versions of an algorithm)
- but the various algorithms should not be implemented within the class
- want the implementation of the class to be independent of a particular implementation of an algorithm
- the algorithms could be used by other classes, in a different context
- want to **decouple** — separate — the implementation of the class from the implementations of the algorithms

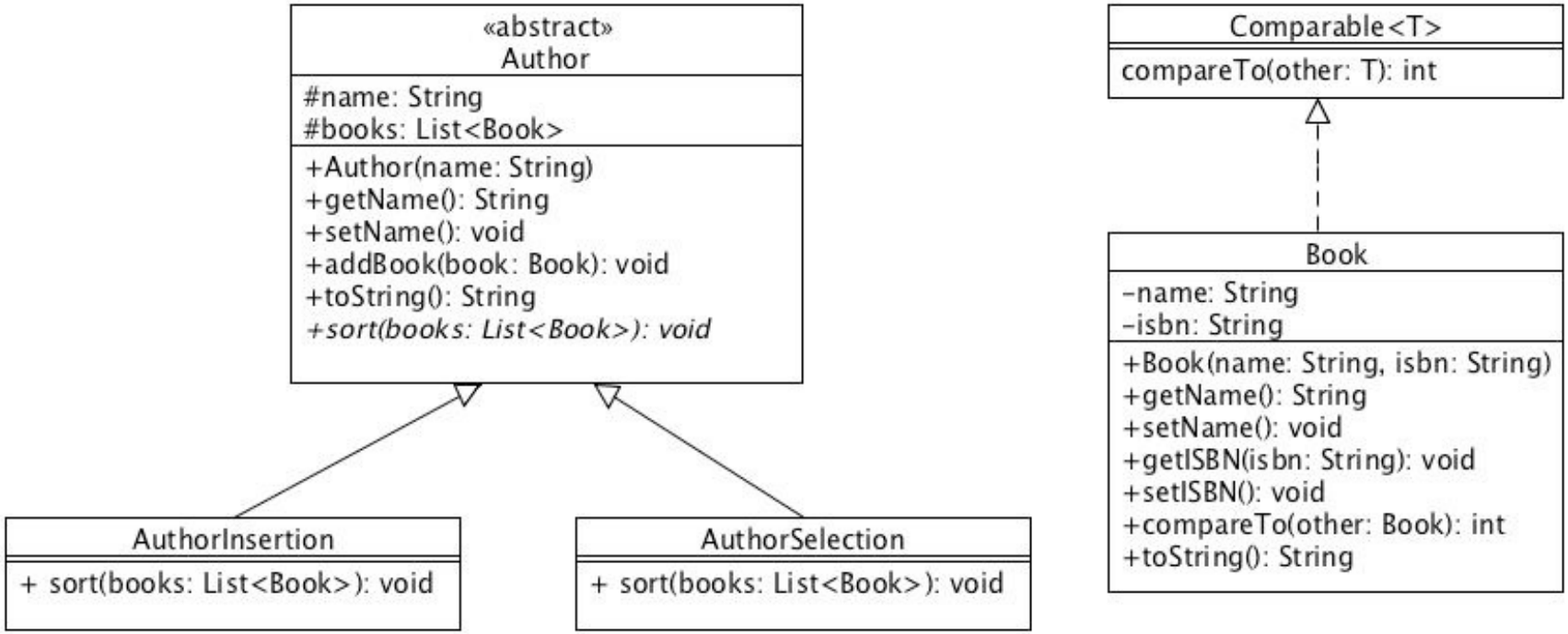
Strategy: Standard Solution

Context uses this interface to call the algorithm defined by a ConcreteStrategy.



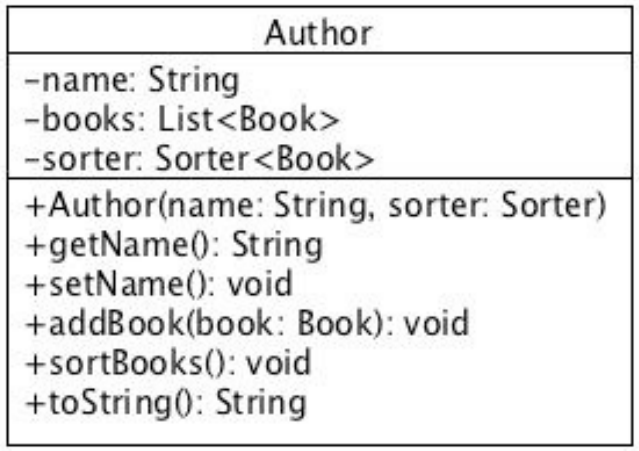
each concrete strategy implements an algorithm.

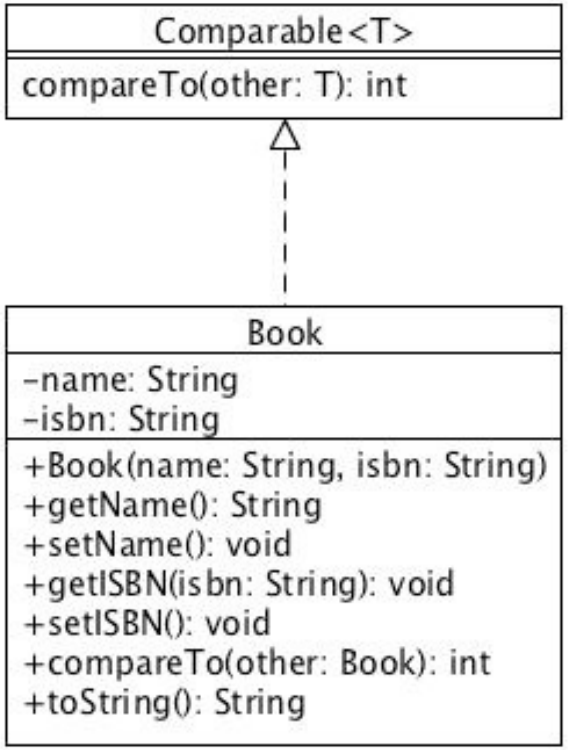
Example: without the Strategy pattern



Example: using the Strategy pattern

sorter determined upon creation of Author





algorithm is independent of context… can be reused