CSSE 220 Day 16

Strategy Pattern, Search, Config Files
Linked List Implementation

Checkout *StrategyPattern* project from SVN
Checkout *LinkedLists* project from SVN

Questions

Strategy Design Pattern

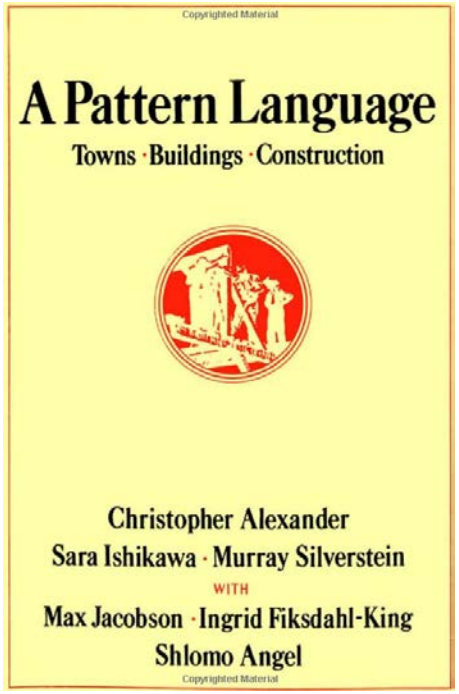
- An application of function objects

Design Pattern

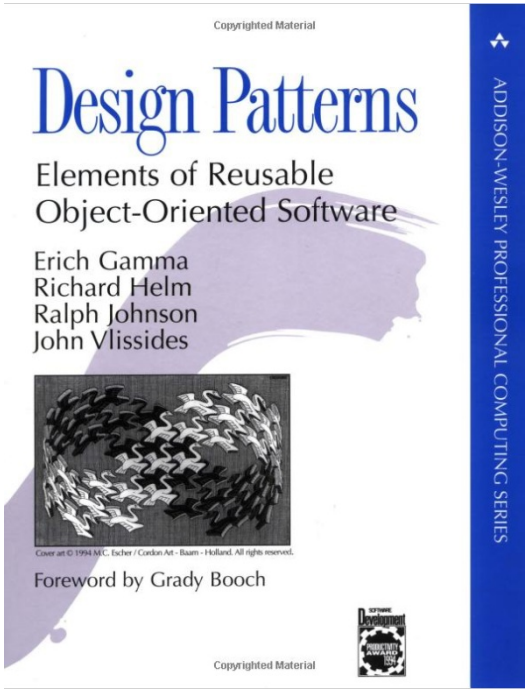
- A *named* and *well-known* problem-solution pair that can be applied in a new context.

History

1977



2004

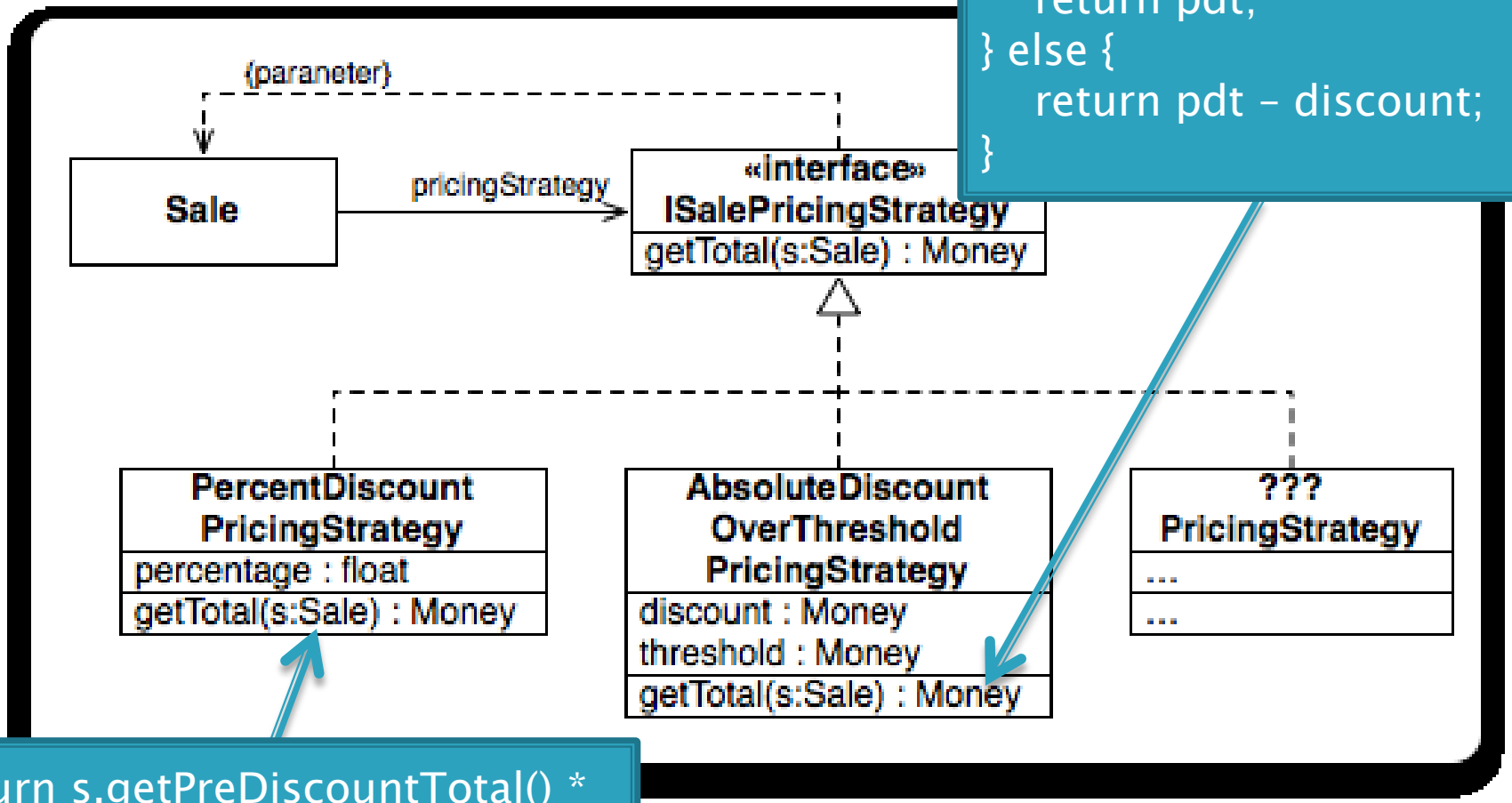


Strategy Pattern

- **Problem**: How do we design for varying, but related, algorithms or policies?
- **Solution**: Define each algorithm or policy in a separate class with a common interface

Strategy Example

```
double pdt =
s.getPreDiscountTotal();
if (pdt < this.threshold) {
    return pdt;
} else {
    return pdt - discount;
}
```




```
return s.getPreDiscountTotal() *
        this.percentage;
```

Search Review

- Linear vs. Binary Search

Searching

- Consider:
  - Find Cary Laxer's number in the phone book
  - Find who has the number 232-2527
- Is one task harder than the other? Why?
- For searching unsorted data, what's the worst case number of comparisons we would have to make?

Binary Search of Sorted Data

- A divide and conquer strategy

- Basic idea:
  - Divide the list in half
  - Decide whether result should be in upper or lower half
  - Recursively search that half

Analyzing Binary Search

- What's the best case?
- What's the worst case?

Putting It All Together

- Represent search algorithms using a strategy pattern
- Use a configuration file to specify the strategy
- Check out from repo and work as a team
- Help each other to understand

Data Structures

- Understanding the engineering trade-offs when storing data

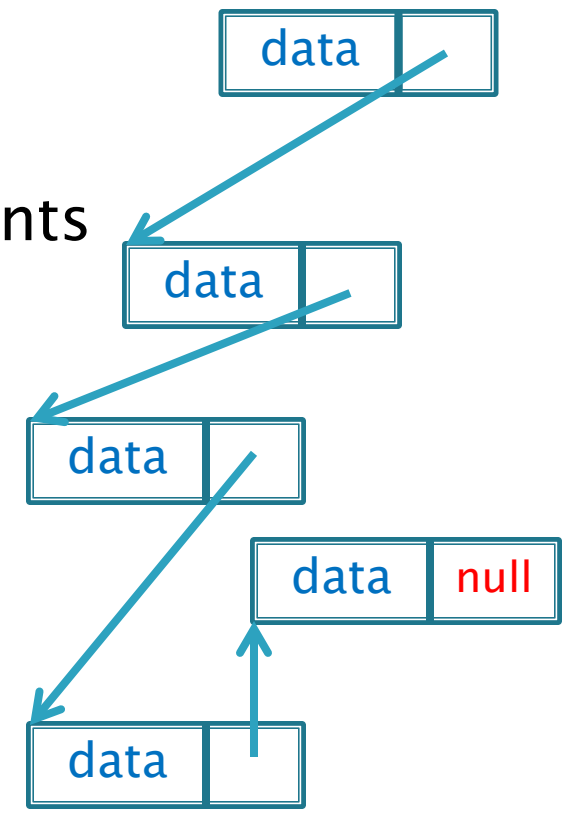
Data Structures

- Efficient ways to store data based on how we'll use it
- The main theme for the rest of the course
- So far we've seen `ArrayLists`
  - Fast addition **to end of list**
  - Fast access to any existing position
  - Slow inserts to and deletes from middle of list

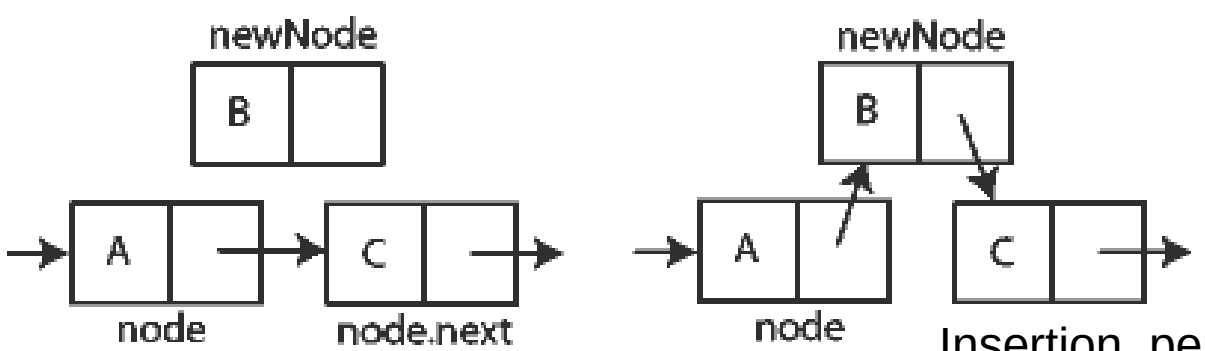
Q1

Another List Data Structure

- What if we have to add/remove data from a list frequently?
- `LinkedLists` support this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements



"random access"



Insertion, per Wikipedia

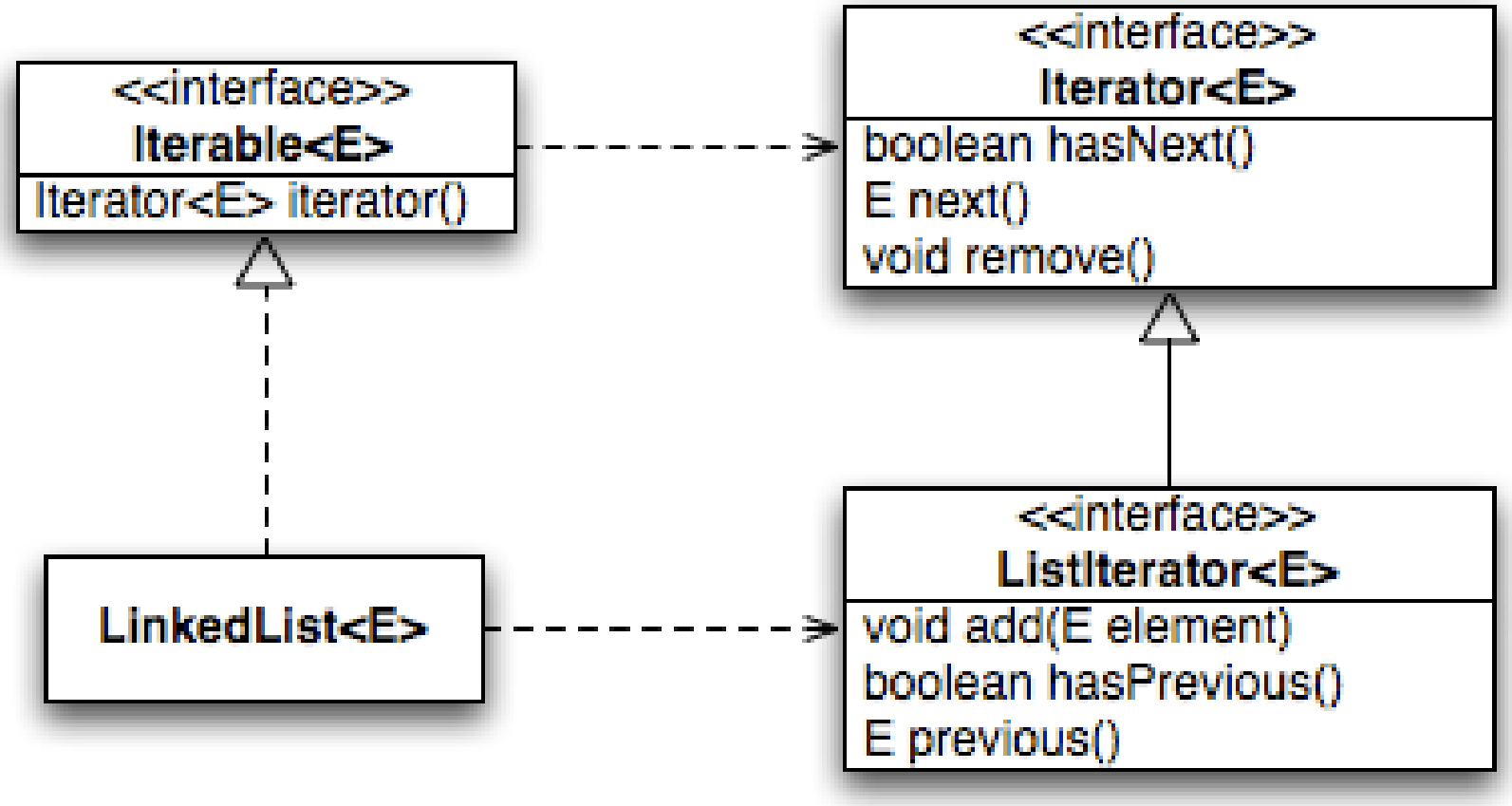
Q2, Q3

LinkedList<E> Methods

- `void addFirst(E element)`
- `void addLast(E element)`
- `E getFirst()`
- `E getLast()`
- `E removeFirst()`
- `E removeLast()`

- What about accessing the middle of the list?
  - `LinkedList<E> implements Iterable<E>`

Accessing the Middle of a LinkedList



An Insider's View

```
for (String s : list) {
  // do something
}
```

Enhanced For Loop

```
Iterator<String> iter =
    list.iterator();

while (iter.hasNext()) {
  String s = iter.next();
  // do something
}
```

What Compiler Generates

Implementing LinkedList

- A simplified version, with just the essentials
- Won't implement the java.util.List interface
- Will have the usual linked list behavior
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow random access

Team project work time

- When you have finished the StrategyPattern exercise

  Work with your team on the team project