

CSSE 220 Day 25

Strategy Pattern, Search, Config Files

Checkout *StrategyPattern* project from SVN

Questions

Sorting Review

- ▶ Selection Sort
 - Find the smallest item in the unsorted part
 - Put it at the end of the sorted part, by swapping it with the first item in the unsorted part
- ▶ Insertion Sort
 - Take the first item in unsorted part
 - Slide it down to the correct place in the sorted part
- ▶ Merge Sort
 - If size is 0 or 1, we are done
 - Otherwise:
 - Divide list in half, recursively sort each half
 - Merge two halves

Polymorphism and Inheritance

```
interface Letters {  
    public void one();  
    public void two();  
    public void four();  
}  
  
class Lower implements Letters {  
    public void one() {  
        System.out.println("a");  
    }  
  
    public void two() {  
        System.out.println("b");  
        this.one();  
    }  
  
    public void four() {  
        System.out.println("d");  
    }  
}
```

```
class Upper extends Lower {  
    public void one() {  
        System.out.println("A");  
    }  
  
    public void four() {  
        System.out.println("D");  
        super.four();  
    }  
  
    public void five() {  
        System.out.println("E");  
    }  
}
```

```
Letters m = new Letters();  
m.one();
```

Polymorphism and Inheritance

```
interface Letters {  
    public void one();  
    public void two();  
    public void four();  
}  
  
class Lower implements Letters {  
    public void one() {  
        System.out.println("a");  
    }  
  
    public void two() {  
        System.out.println("b");  
        this.one();  
    }  
  
    public void four() {  
        System.out.println("d");  
    }  
}
```

```
class Upper extends Lower {  
    public void one() {  
        System.out.println("A");  
    }  
  
    public void four() {  
        System.out.println("D");  
        super.four();  
    }  
  
    public void five() {  
        System.out.println("E");  
    }  
}
```

Letters o = new Upper();
o.two();

Polymorphism and Inheritance

```
interface Letters {
    public void one();
    public void two();
    public void four();
}

class Lower implements Letters {
    public void one() {
        System.out.println("a");
    }

    public void two() {
        System.out.println("b");
        this.one();
    }

    public void four() {
        System.out.println("d");
    }
}
```

```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    public void four() {
        System.out.println("D");
        super.four()
    }

    public void five() {
        System.out.println("E");
    }
}
```

Letters p = new Upper();
p.four();

Polymorphism and Inheritance

```
interface Letters {  
    public void one();  
    public void two();  
    public void four();  
}  
  
class Lower implements Letters {  
    public void one() {  
        System.out.println("a");  
    }  
  
    public void two() {  
        System.out.println("b");  
        this.one();  
    }  
  
    public void four() {  
        System.out.println("d");  
    }  
}
```

```
class Upper extends Lower {  
    public void one() {  
        System.out.println("A");  
    }  
  
    public void four() {  
        System.out.println("D");  
        super.four();  
    }  
  
    public void five() {  
        System.out.println("E");  
    }  
}
```

Letters q = new Upper();
q.five();

Polymorphism and Inheritance

```
interface Letters {
    public void one();
    public void two();
    public void four();
}

class Lower implements Letters {
    public void one() {
        System.out.println("a");
    }

    public void two() {
        System.out.println("b");
        this.one();
    }

    public void four() {
        System.out.println("d");
    }
}
```

```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    public void four() {
        System.out.println("D");
        super.four()
    }

    public void five() {
        System.out.println("E");
    }
}
```

```
Lower r = new Upper();
((Upper) r).five();
```


Polymorphism and Inheritance

```
interface Letters {
    public void one();
    public void two();
    public void four();
}

class Lower implements Letters {
    public void one() {
        System.out.println("a");
    }

    public void two() {
        System.out.println("b");
        this.one();
    }

    public void four() {
        System.out.println("d");
    }
}
```

```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    public void four() {
        System.out.println("D");
        super.four()
    }

    public void five() {
        System.out.println("E");
    }
}
```

```
Upper s = new Lower();
s.one();
```

Polymorphism and Inheritance

```
interface Letters {  
    public void one();  
    public void two();  
    public void four();  
}  
  
class Lower implements Letters {  
    public void one() {  
        System.out.println("a");  
    }  
  
    public void two() {  
        System.out.println("b");  
        this.one();  
    }  
  
    public void four() {  
        System.out.println("d");  
    }  
}
```

```
class Upper extends Lower {  
    public void one() {  
        System.out.println("A");  
    }  
  
    public void four() {  
        System.out.println("D");  
        super.four();  
    }  
  
    public void five() {  
        System.out.println("E");  
    }  
}
```

```
Lower t = new Upper();  
t.one();
```

Strategy Design Pattern

»» An application of
function objects

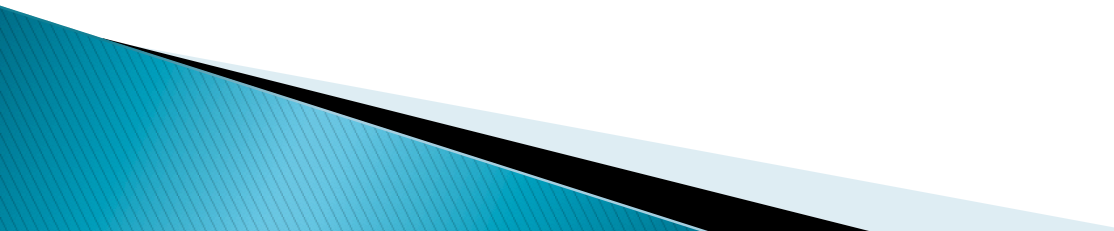
Design Pattern

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

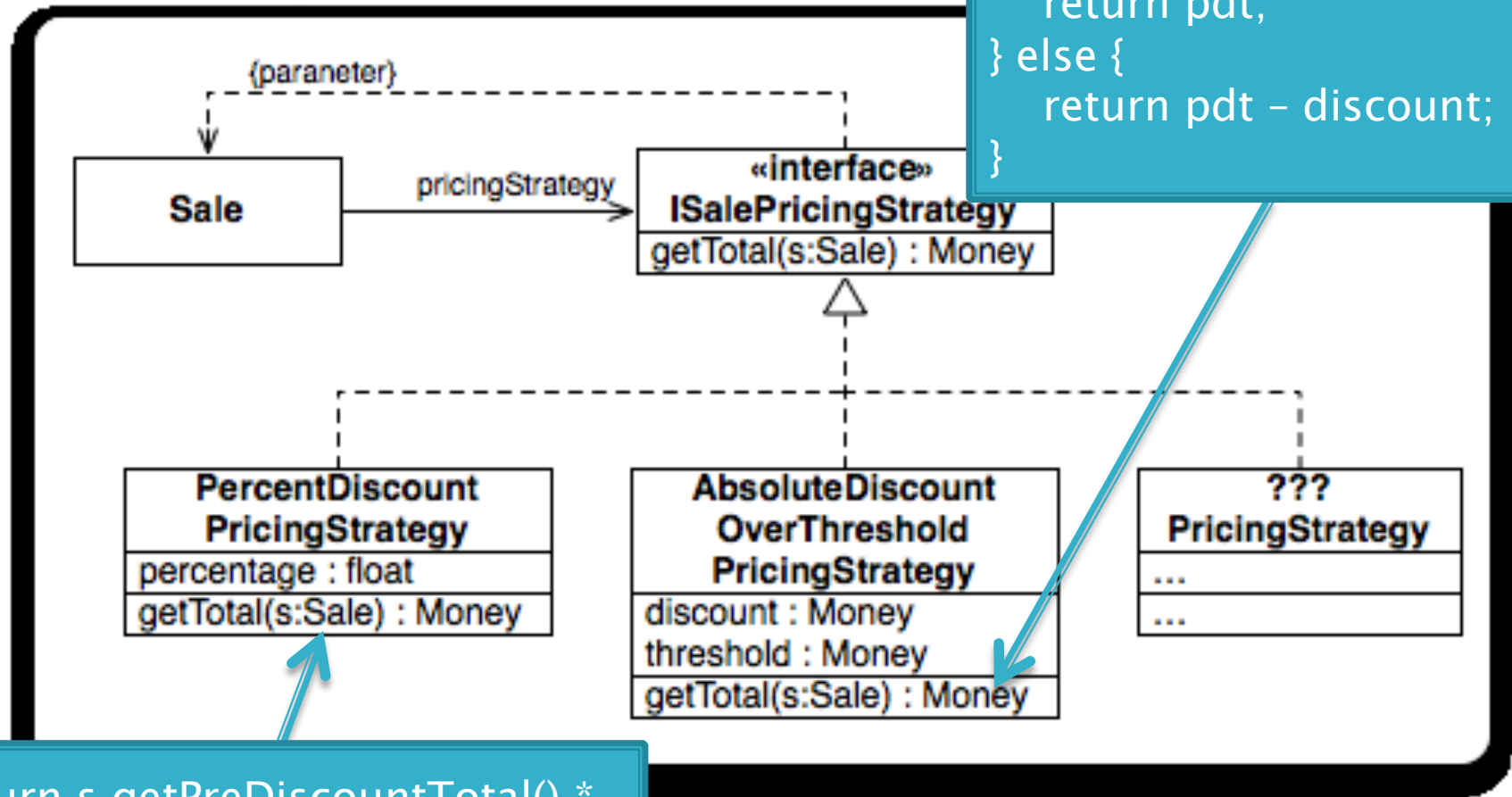
History

- ▶ *A Pattern Language: Towns, Building, Construction*
 - Alexander, Ishikawa, and Silverstein
- ▶ Kent Beck and Ward Cunningham at Tektronik
- ▶ *Design Patterns: Elements of Reusable Object-Oriented Software*
 - Gamma, Helm, Johnson, Vlissides
 - A.k.a., the Gang of Four (GoF)

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
- 

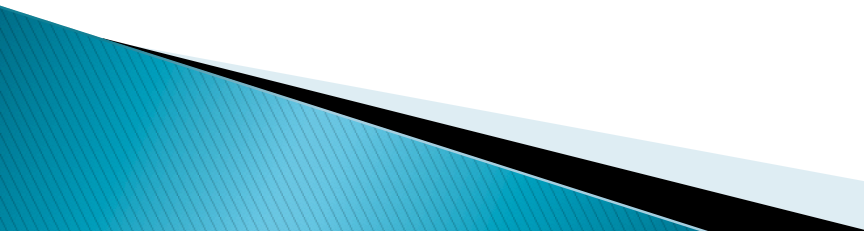
Strategy Example



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

Everyone should do this exercise, but you should discuss it with your team as you work on it

Help each other to understand

Team project work time

»» When you have finished the **StrategyPattern** exercise

Work with your team on the team project