CSSE 220 Day 25

Strategy Pattern, Search, Config Files

Checkout *StrategyPattern* project from SVN

Questions

Makeup Exam

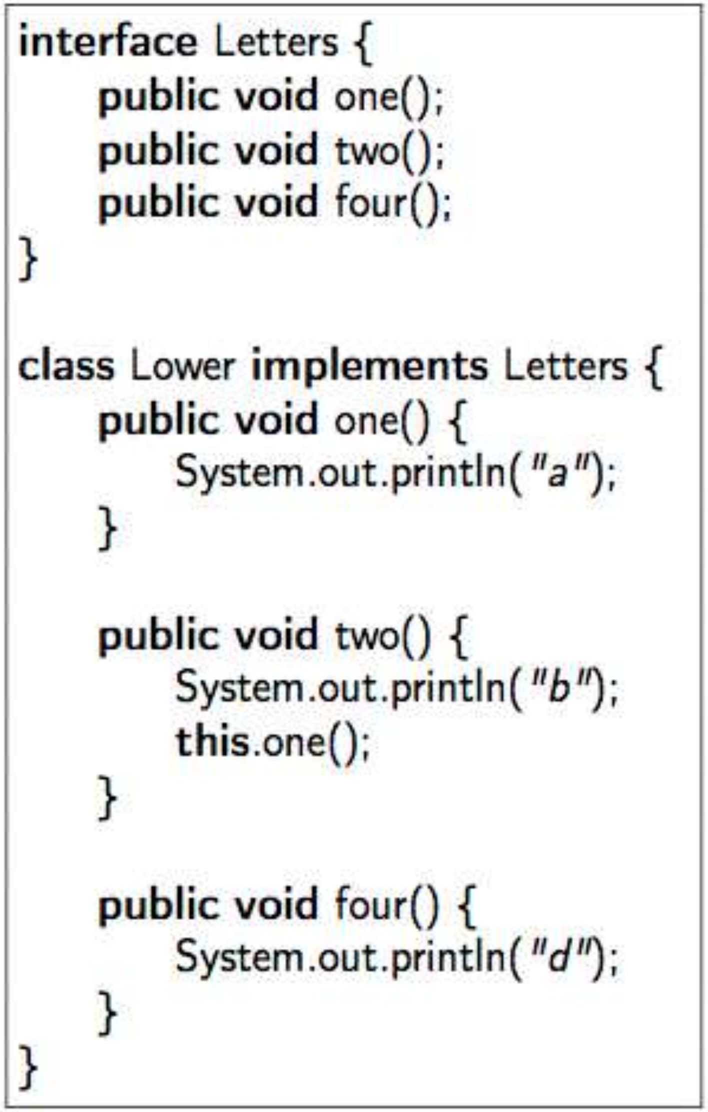
- Makeup for questions 3 and 4 of paper part and all of computer part of Exam 2
  - Can re-do one or two or all questions
  - Will only increase your grade

- Thursday, 7:00-9:00 p.m.
- Room O267

Sorting Review

- Selection Sort
  - Find the smallest item in the unsorted part
  - Swap it to 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
  - Size 0 or 1, then done
  - Otherwise:
    - Divide list in half, recursively sort each half
    - Merge two halves

Polymorphism and Inheritance

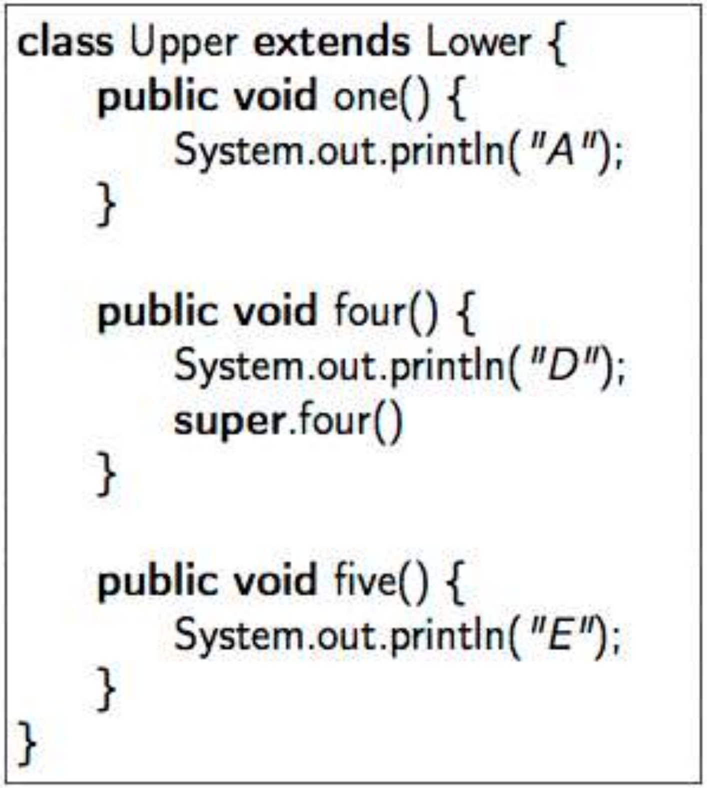


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



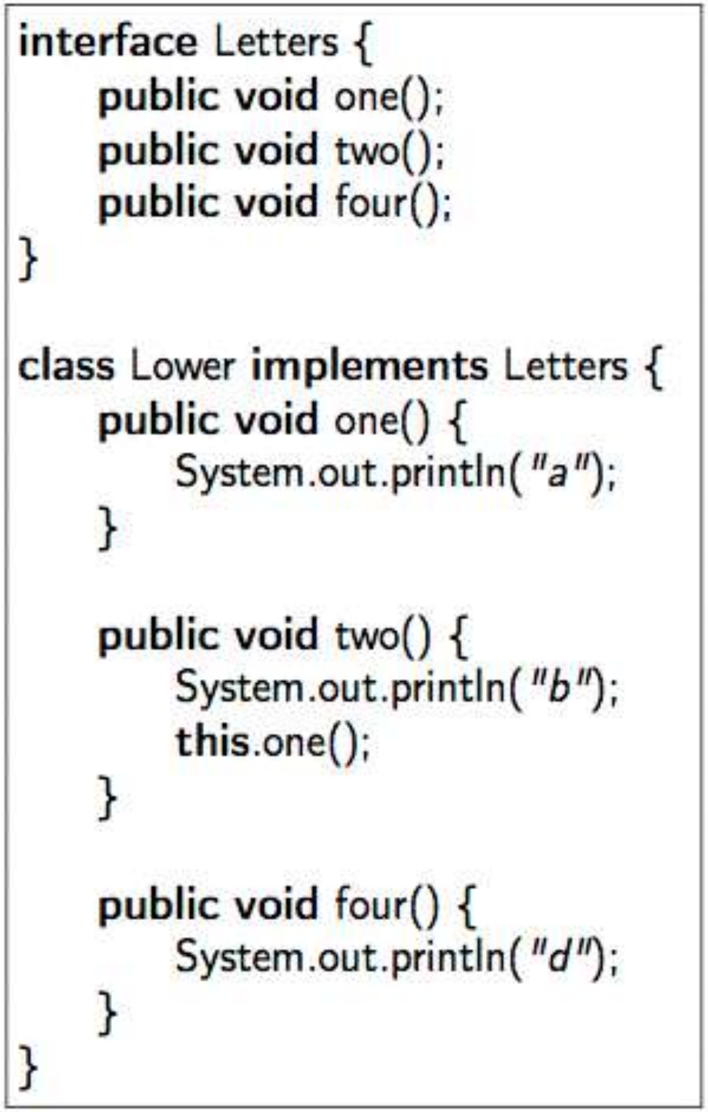
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

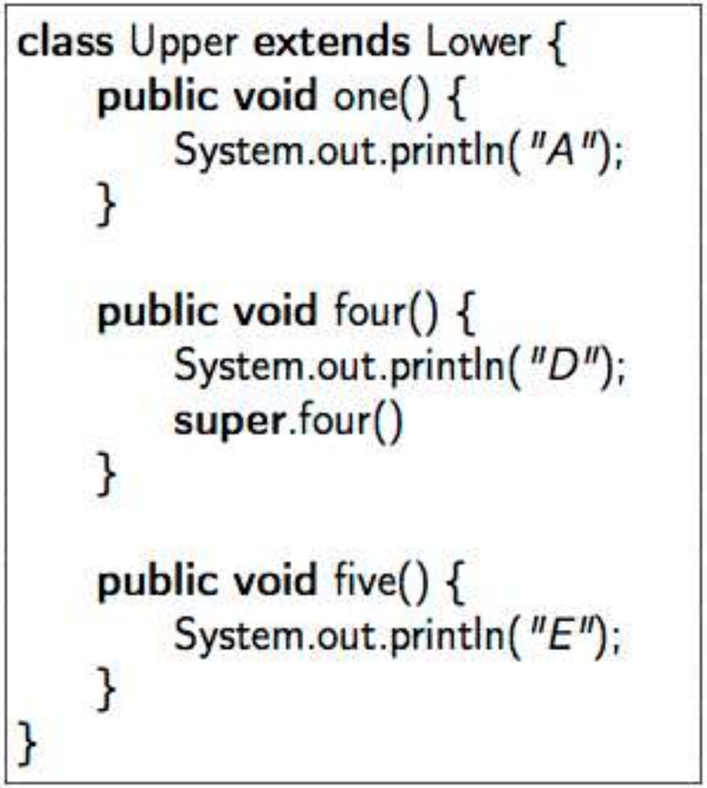


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



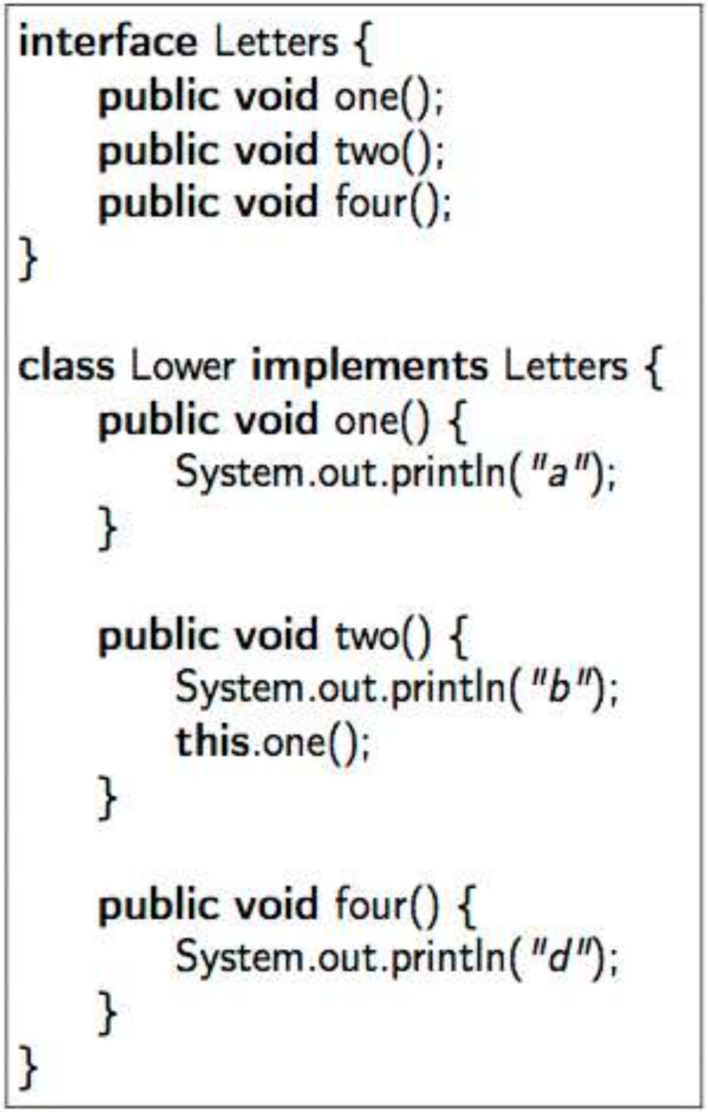
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

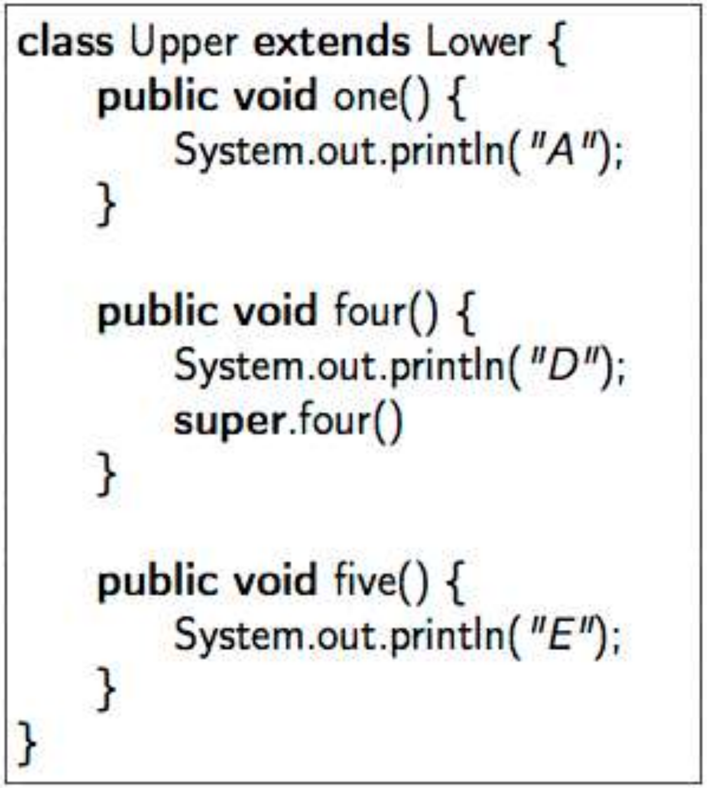


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



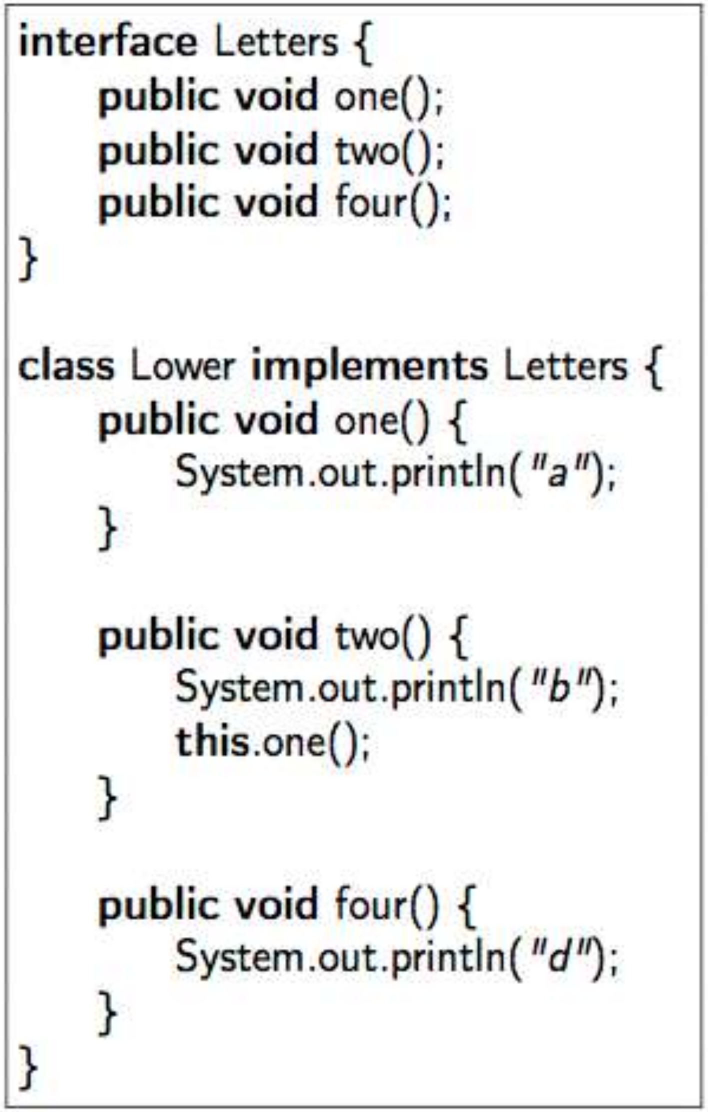
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

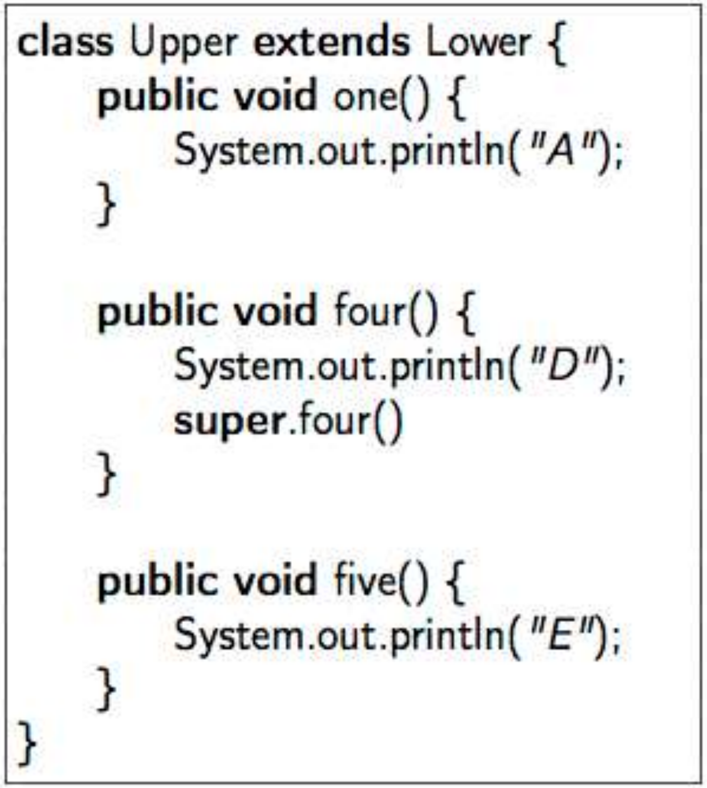


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



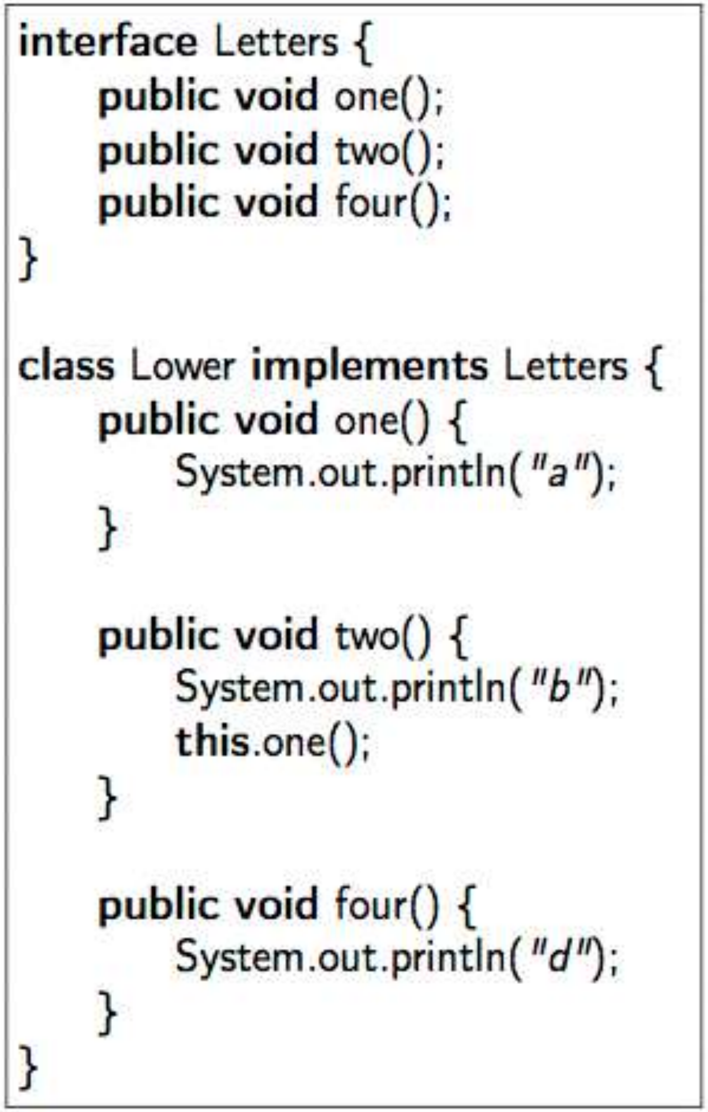
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

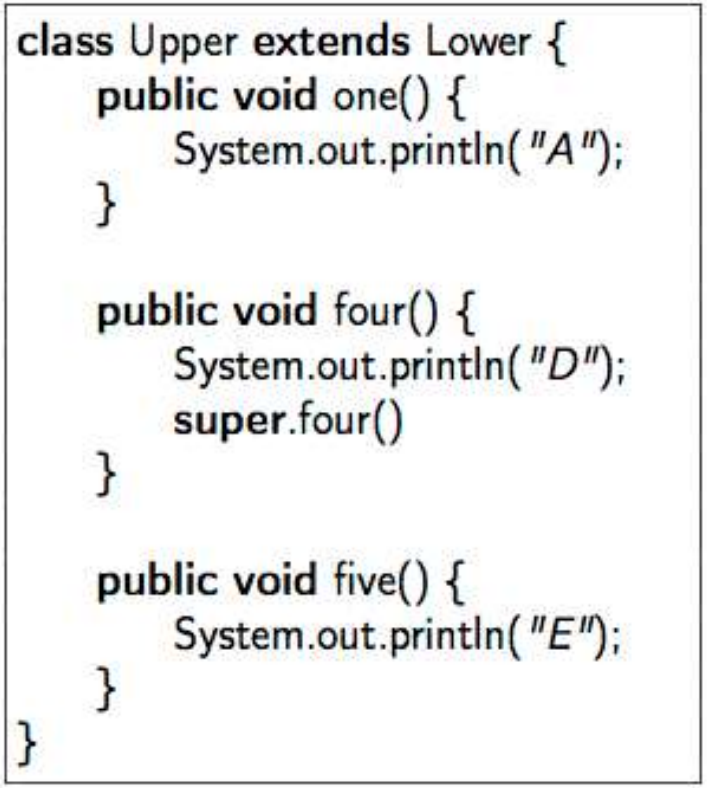


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



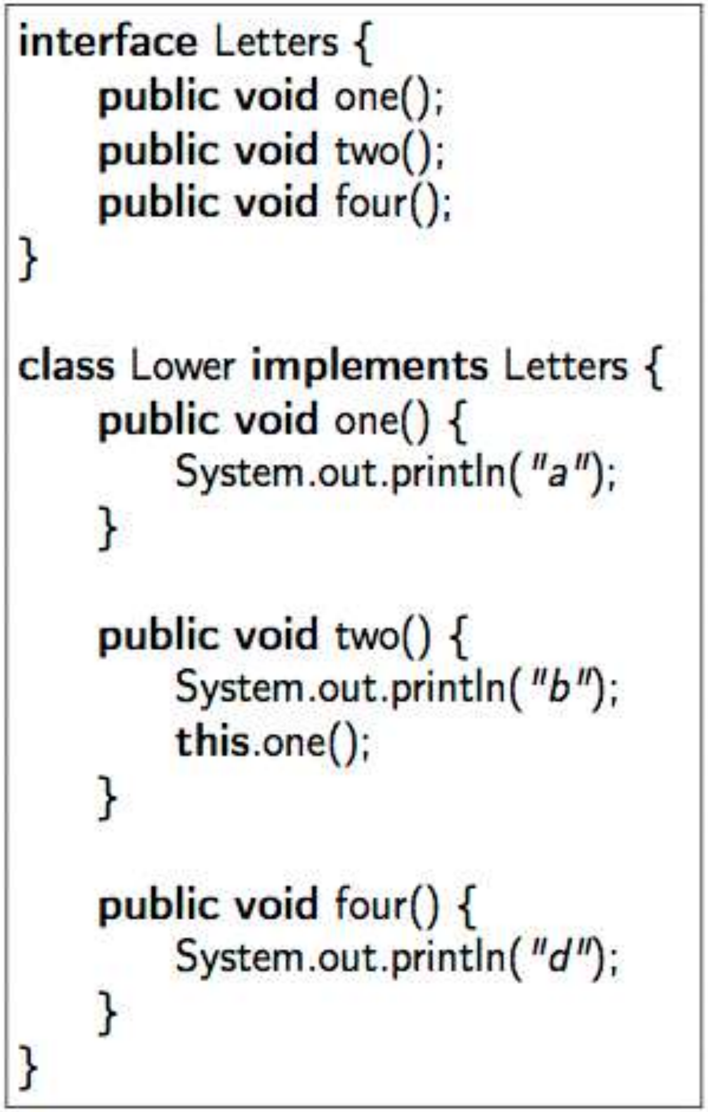
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

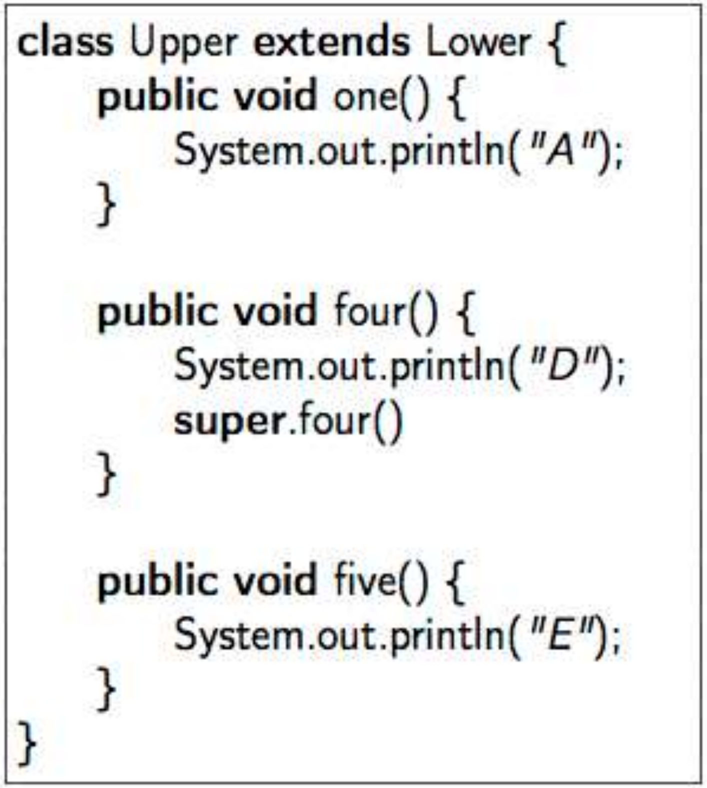


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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");
    }
}
```



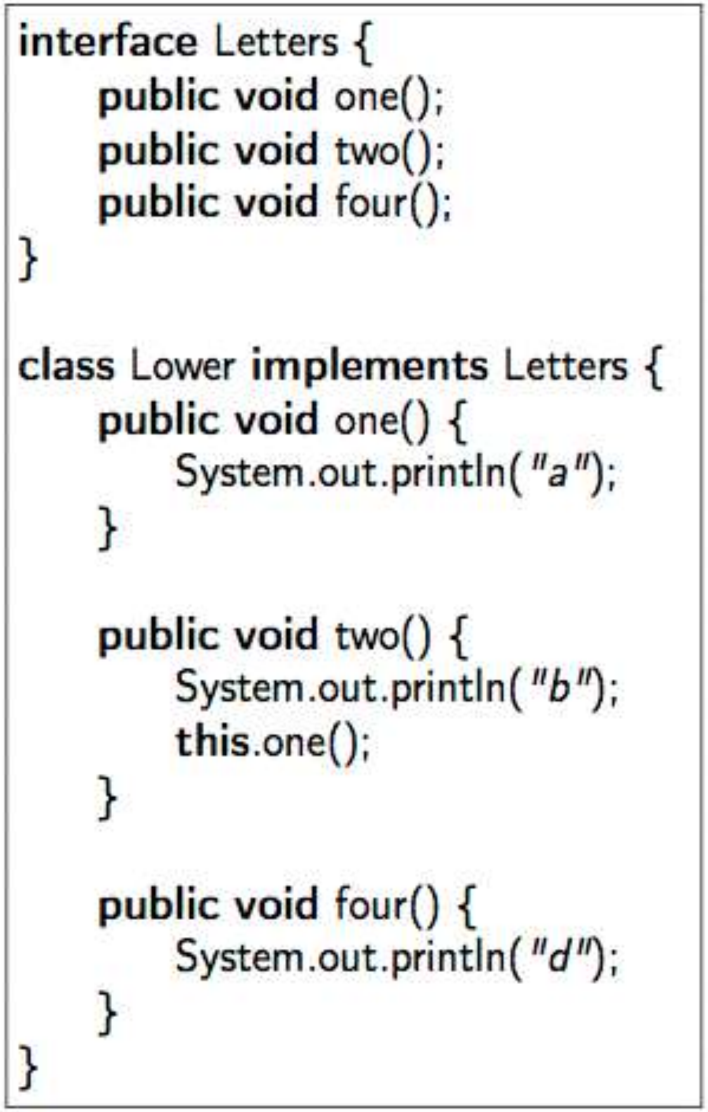
```
class Upper extends Lower {
    public void one() {
        System.out.println("A");
    }

    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

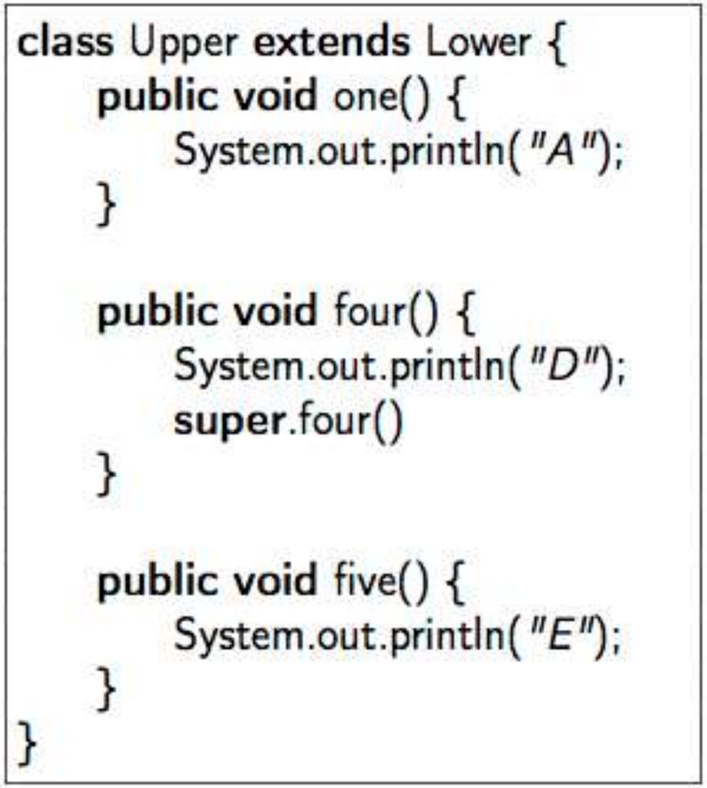


```
interface Letters {
    public void one();
    public void two();
    public void four();
}

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.

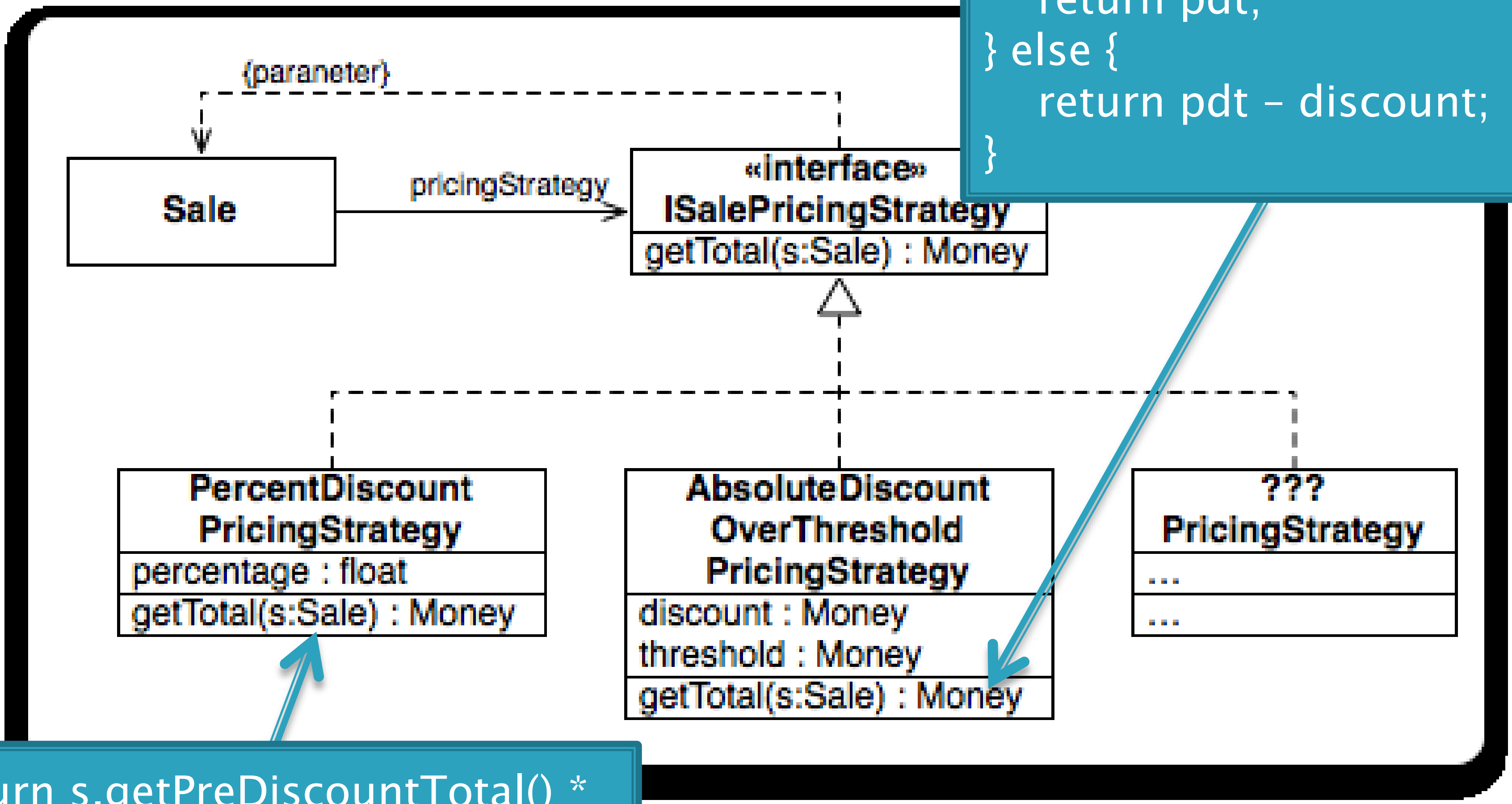
History

- *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

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

- Representing search algorithms using strategy pattern
- Using configuration files to specify the strategy