# Introduction to Information Systems and Programming

Visitor Design Pattern

# Visitor

- Useful if you need to perform operations across a diverse set of objects
- GoF: “Allows for one or more operation to be applied to a set of objects at runtime, decoupling the operations from the object structure”
- Provide additional functionality to a class without changing it

# Example: Postage

- Postage calculation depends on the item type:
  - Book
  - CD
  - Clothing
- Also depends on where the item is sent to
- Given a list of items of variety types, determine the total postage cost

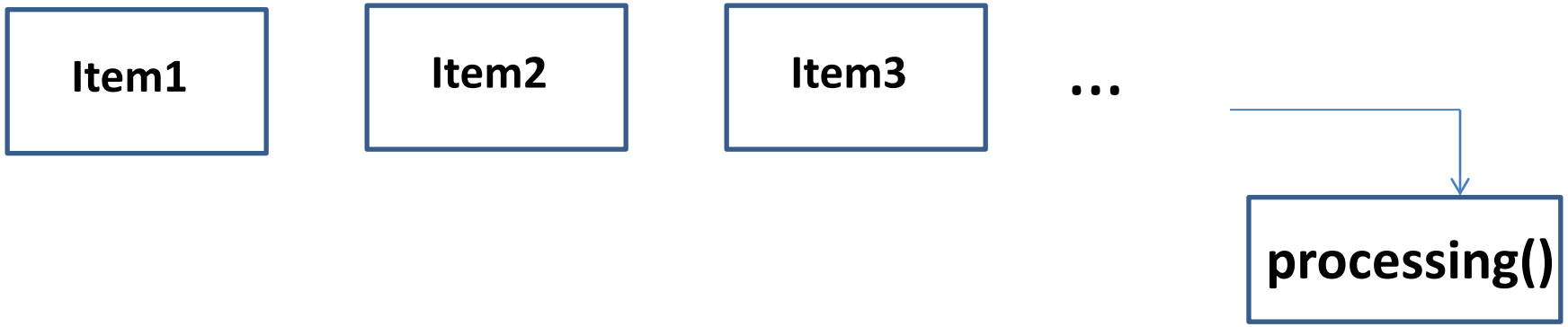
# Non-visitor design

```
public static double calPostage(ArrayList<Object> items) {
  double total=0;
  for (Object o: items) {
    if (o instanceof Book) {
      ...
    }
    else if (o instanceof CD) {
      ...
    }
    else if (o instanceof Clothing) {
      ...
    }
    else
      throw new AssertionError("not supported");
  }
  return total;
}
```

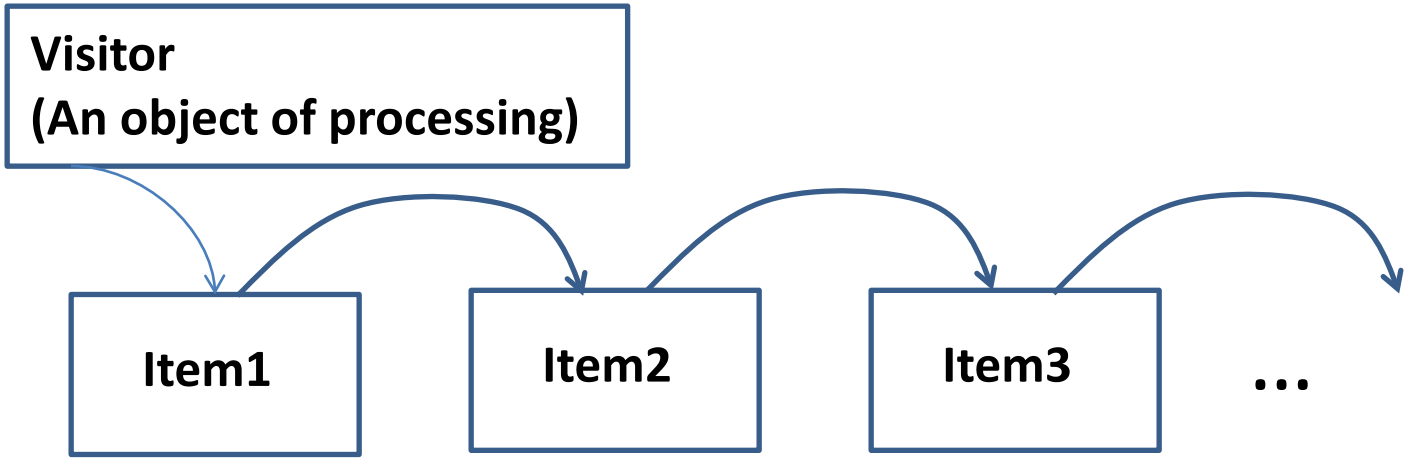
-Another procedure for another region

# Visitor Design Pattern is a more OO way to perform operation on a list of items

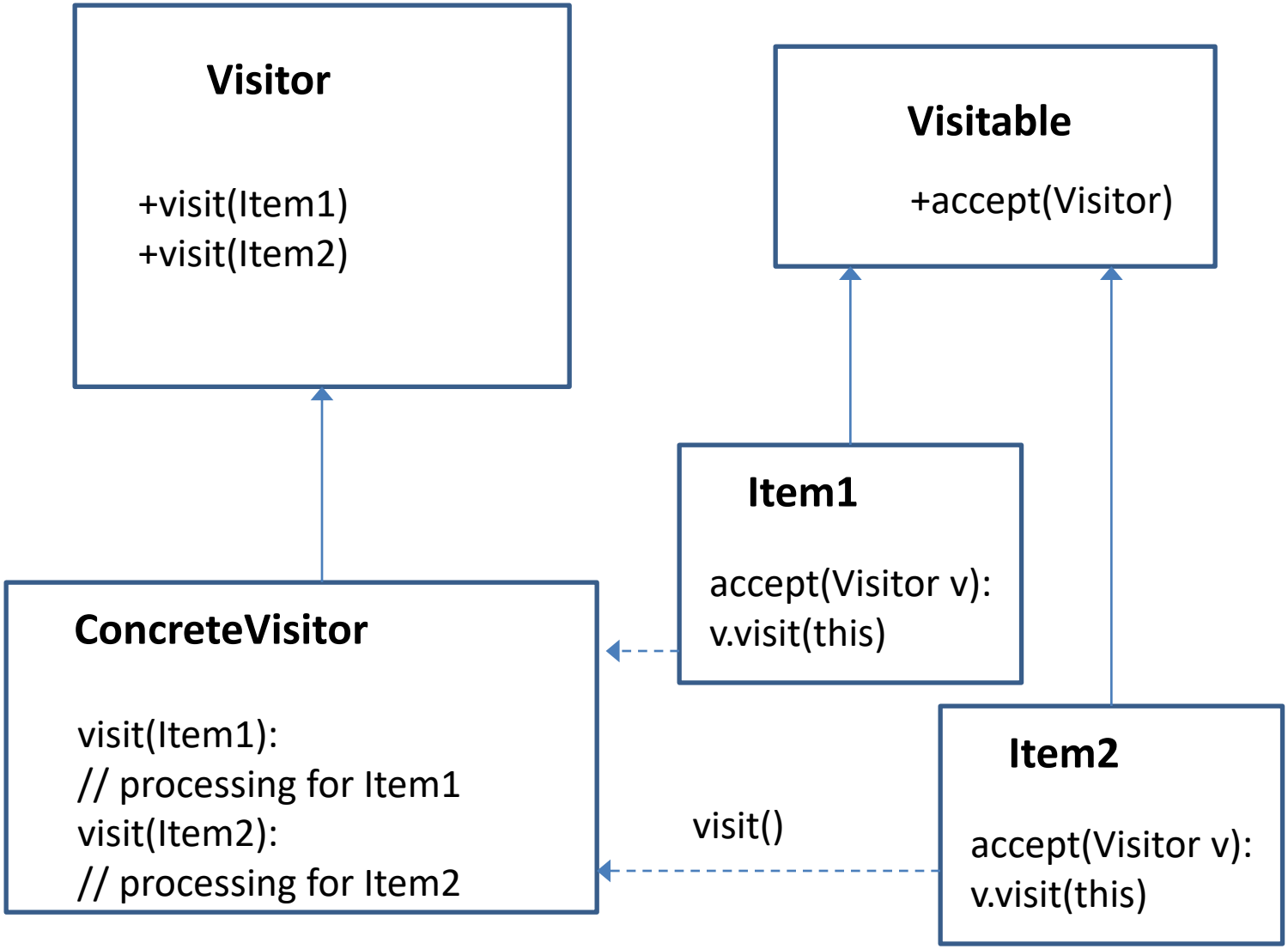
Non-Visitor Design:




Visitor Design:



# Visitor Design Pattern



# Visitor Design Pattern: Visitor Interface

```
// Visitor.java
public interface Visitor {
        void visit(Book b);
        void visit(CD c);
        void visit(Clothing c);
}
```

-Different visit methods for different object types to be processed

-Later, use method overloading to pick the method for processing (instead of if-then-else)

# PostageVisitor.java : concrete class implements Visitor

```
public class PostageVisitor implements Visitor {

  private double total =0;

  @Override
  public void visit(Book b) {
    total += b.getWeight() * 5;
  }

  @Override
  public void visit(CD c) {
    ...
  }

}
```

-Concrete implementation of Visitor provides the specifics of what to do with different object types

# Visitable Interface

```
public interface Visitable {

  void accept(Visitor v);

}
```

- Allows the Visitor to be passed in

# Item implements Visitable

```
class Book implements Visitable {
  private double weight;

  public void accept(Visitor v) {
    v.visit(this);
  }

  public Book (double weight) {
    this.weight=weight;
  }

  public double getWeight() {
    return weight;
  }
}
```

Simple but important change to make a class visitable: dispatch of visit depends on data type “this”

# Test.java

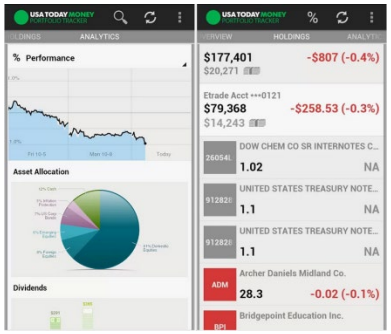
```
ArrayList<Visitable> items = new ArrayList<Visitable>();
PostageVisitor postage = new PostageVisitor();

items.add(new Book(1));
items.add(new CD("psy"));
items.add(new Book(2));
items.add(new Clothing(10));

for (Visitable o: items){
  o.accept(postage);

}

System.out.println(postage.getTotal());
```

Visitor postage visits each item one by one, processes them, information kept inside visitor

# Advantage of Visitor

- Separate out certain logic (e.g., postage calculation) from the items themselves, keeping the item classes simple
  - Item classes only need to implement Visitable
- Add methods to classes: another concrete class implements Visitor
  - No need to change item classes
- Enable static checking: code cannot be compiled without the visit method of the corresponding type

# Activity




- Using the Observer design pattern, develop a stock alert system that sends alerts to subscribers for any update of the stock prices
  - Develop StockGrabber class that keeps the list of subscribers for several stock prices, and notifies them for update
  - Develop StockObserver class that monitors changes in the stock prices
  - Develop a Test class to simulate the system

# Activity

- Use the Visitor design pattern, compute the total tax income of a list of items of the following types: Car, Alcohol, Chocolate. Two taxing systems are in place:
  - TaxNormal:
    - Car: 30%, Alcohol: 50%, Chocolate: 10%
  - TaxHoliday:
    - Car: 40%, Alcohol: 80%, Chocolate: 20%

Starting code in the course directory.