M5 (b) - Composition

Jin L.C. Guo

Image source: https://cdn.pixabay.com/photo/2017/11/05/21/21/container-2921882_960_720.jpg

Recap

- Divide and Conquer Principle
- Aggregation and Delegation
- Sequence Diagram
- Composite Pattern
- Decorator Pattern

Objective

- Polymorphic Object Cloning
- Prototype Pattern
- Command Pattern
- Principle: Law of Demeter

Object Copying

- Copy Constructor

```
public Deck(Deck pDeck)
{
    aCards = new  ArrayList<>(pDeck.aCards);
}
```

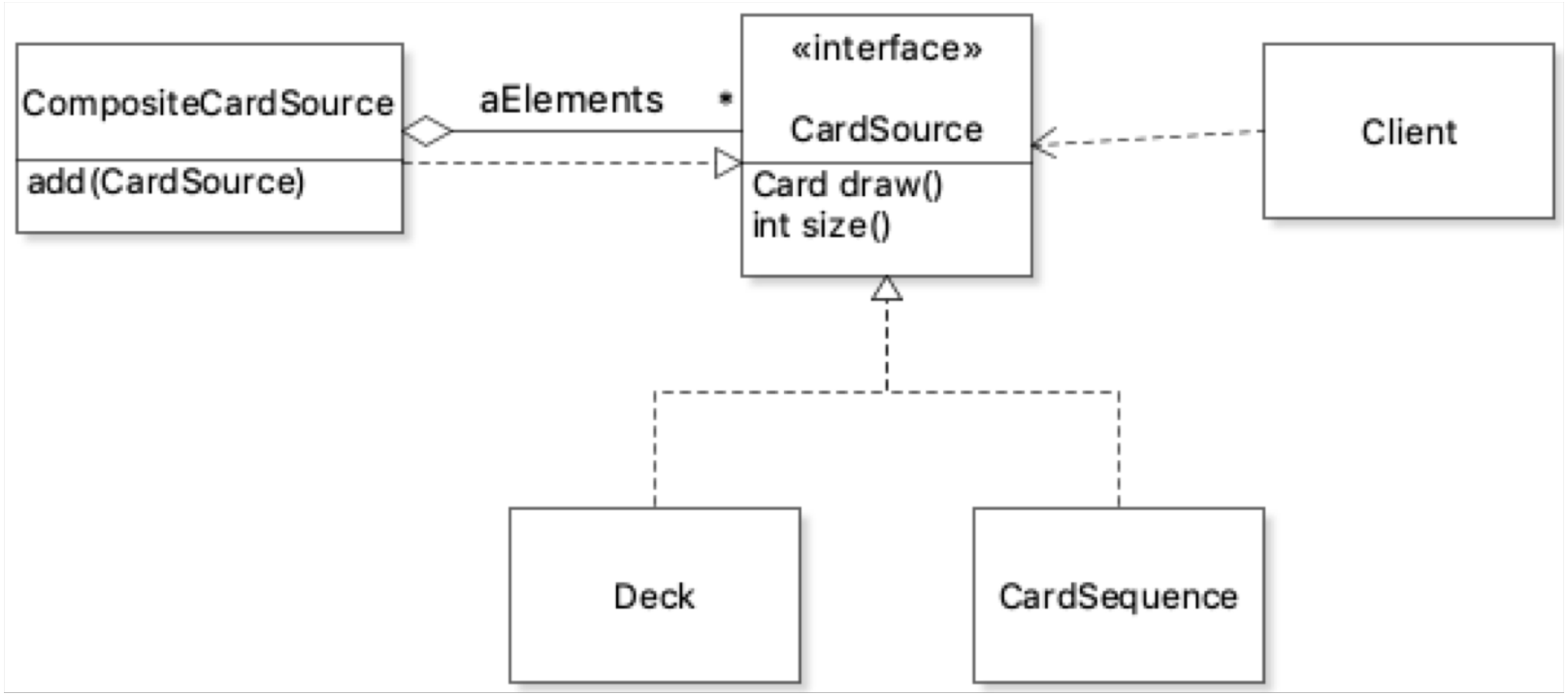
- Static factory method

```
public static Deck newInstance(Deck pDeck)
{
    return new Deck(pDeck);
}
```

```java
/**
 * Aggregate a collection of card sources.
 * The client can get existing card sources
 * and add new card sources on demand
 *
 */
public class CardSourceManager
{
    private final List<CardSource> aCardSources = new ArrayList<>();

    public List<CardSource> getCardSources()
    {
        // return a copy of aCardSources;
    }

    public void addCardSource(CardSource pCardSource)
    {
        // add a copy of pCardSources;
    }
}
```


CompositeCardSource
add(CardSource)
aElements
*
«interface»
CardSource
Card draw()
int size()
Client
Deck
CardSequence

```
public List<CardSource> getCardSources()
{
    // return a copy of aCardSources;
    List<CardSource> cardSourcesCopy = new ArrayList<>();
    for(CardSource cs:aCardSources)
    {
        if (cs instanceof Deck)
        {
            cardSourcesCopy.add(new Deck(cs));
        }
        else if (cs instanceof CardSequence)
        {
            cardSourcesCopy.add(new CardSequence(cs));
        }
        else if (cs instanceof CompositeCardSource)
        {
            ……
        }
        ……
    }
    return cardSourcesCopy;
}
```

How to achieve polymorphic object copying?

Implements Cloneable

- java.lang.Cloneable

this interface does *not* contain the clone method.

implement this interface should override Object.clone with a public method.

A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class.

Invoking Object's clone method on an instance that does not implement the Cloneable interface results in the exception CloneNotSupportedException being thrown.

Override Object.clone()

protected Object clone()
throws CloneNotSupportedException

Creates and returns a copy of this object.

x.clone() != x

x.clone().getClass() == x.getClass()

x.clone().equals(x)

object should be obtained by calling super.clone

the object returned by this method should be independent of this object

Polymorphic Object Copying Demo

```java
public class CompositeCardSource implements CardSource
{
    private List<CardSource> aElements = new ArrayList<>();

    @Override
    public CardSource clone()
    {
        try
        {
            CompositeCardSource clone = (CompositeCardSource) super.clone();
            clone.aElements = new ArrayList<CardSource>();
            for (CardSource cs:aElements)
            {
                clone.aElements.add(cs.clone());
            }
            return clone;
        }
        catch (CloneNotSupportedException e)
        {
            assert false;
            return null;
        }
    }
}
```

```java
/**
 * Aggregate a collection of card sources.
 * The client can get existing card sources
 * and add new card sources on demand
 *
 */
public class CardSourceManager
{
    private final List<CardSource> aCardSources = new ArrayList<>();


    public List<CardSource> getCardSources()
    {
        // return a copy of aCardSources;
    }

    public void addCardSource(CardSource pCardSource)
    {
        // Add a copy of pCardSource
       aCardSources.add(pCardSource.clone());
    }
}
```

Create default card source to add?

```java
/**
 * Aggregate a collection of card sources.
 * The client can get existing card sources
 * and add new card sources on demand
 *
 */
public class CardSourceManager
{
    private final List<CardSource> aCardSources = new ArrayList<>();
    private CardSource aPrototype;

    public List<CardSource> getCardSources()
    {
        // return a copy of aCardSources;
    }

    public void addCardSource(CardSource pCardSource)
    {
        // add a copy of pCardSource
        aCardSources.add(pCardSource.clone());
    }
}
```

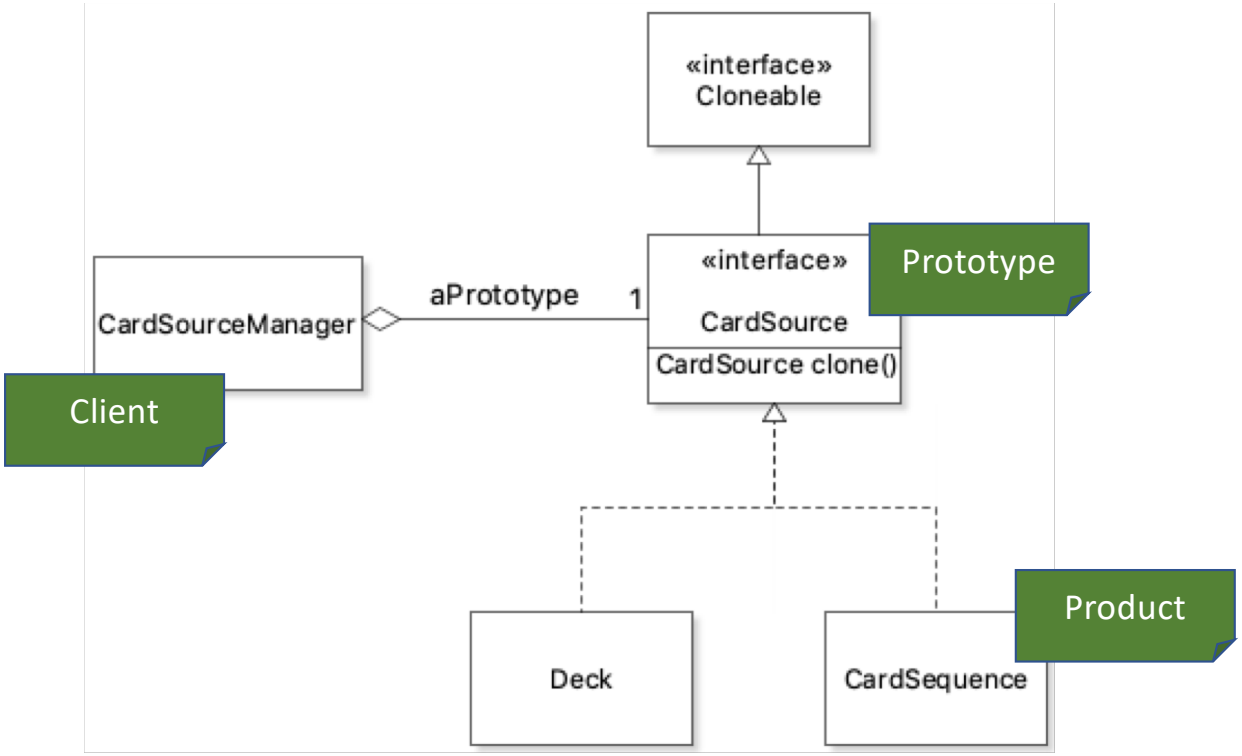
Prototype

- Intent
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Participants
  - Prototype

    declares an interface for cloning itself.
  - Product (Concrete Prototype)

    implements an operation for cloning itself.
  - Client

    creates a new object by asking a prototype to clone itself.


«interface»
Cloneable
«interface»
CardSource
CardSource clone()
Prototype
CardSourceManager
aPrototype
1
Client
Deck
CardSequence
Product

Ways for provide services

- Define methods (functions)

```
deck.draw();
```

- Function objects

```
Collections.sort(aCards, new Comparator<Card>() {
    public int compare(Card pCard1, Card pCard2) {
        return pCard1.getRank().compareTo(pCard2.getRank());
    }
});
```

Manage Function

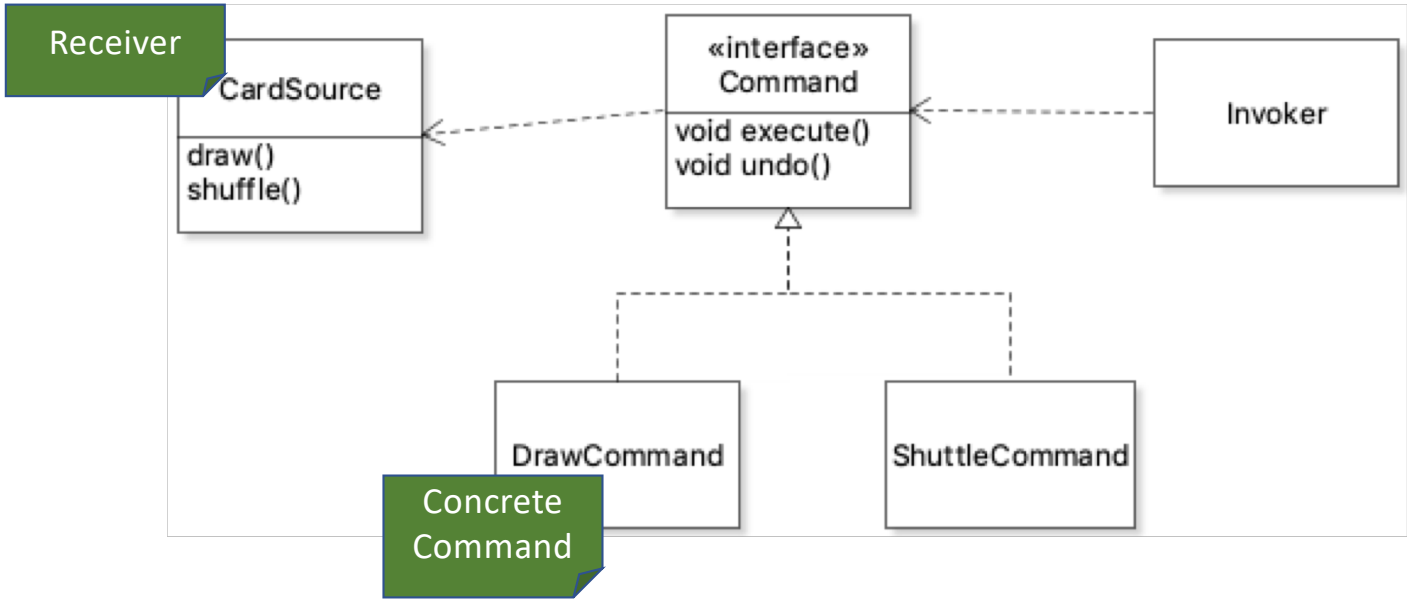
- Memorize past operation
- Redo/undo operation
- Schedule operation
- ……

Command

- Intent:
  - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
- Participants:
  - Command
    declares an interface for executing an operation.
  - ConcreteCommand
    implements execute by invoking the corresponding operation(s) on Receiver.
  - Receiver
    knows how to perform the actual operation
  - Invoker
    execute the operation through function calls declared in Command Interface.

Image source: https://c1.staticflickr.com/4/3111/3145776919_74f05d63fd_b.jpg

Class Diagram



Consideration

- Access of command target and its state

  Pass target as argument or use inner class

- Data flow

  Return value of execution

- Command execution correctness

  Respect precondition

- Storing state

```java
public interface CardSourceCommand
{
   /**
   *
   * @return the production of the execution if it's a card,
   * empty if the execute doesn't produce output.
   */
   Optional<Card> execute();
   /**
   * Undo the immediate previous execution.
   */
   void undo();


}
```

```java
public class DrawCommand implements CardSourceCommand
{
    private CardSource aCardSource;
    private Optional<Card> aCard;
    DrawCommand(CardSource pCardSource)
    {
        aCardSource = pCardSource;
    }

    @Override
    public Optional<Card> execute()
    {
        if(aCardSource.size()>0)
        {
            Card card = aCardSource.draw();
            aCard = Optional.of(card);
            return Optional.of(card);
        }
        else
        {
            return Optional.empty();
        }
    }
```

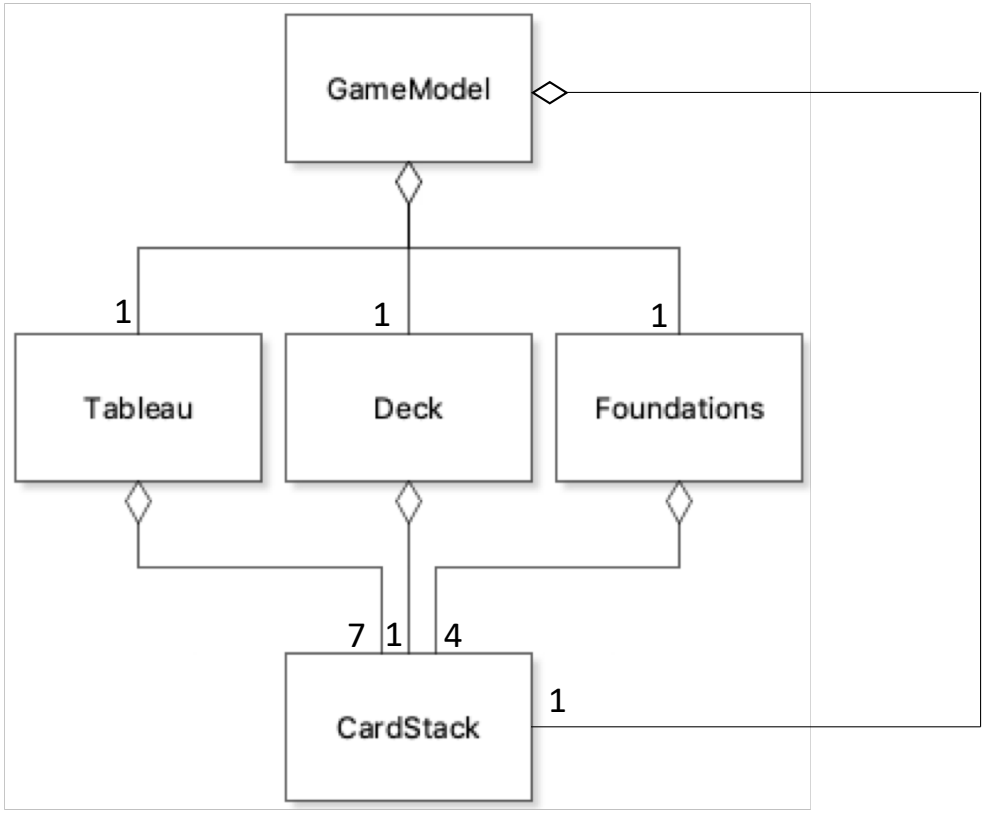
```
@Override
public void undo()
{
    if(aCard.isPresent())
    {
        aCardSource.push(aCard.get());
        aCard = Optional.empty();
    }

}
```

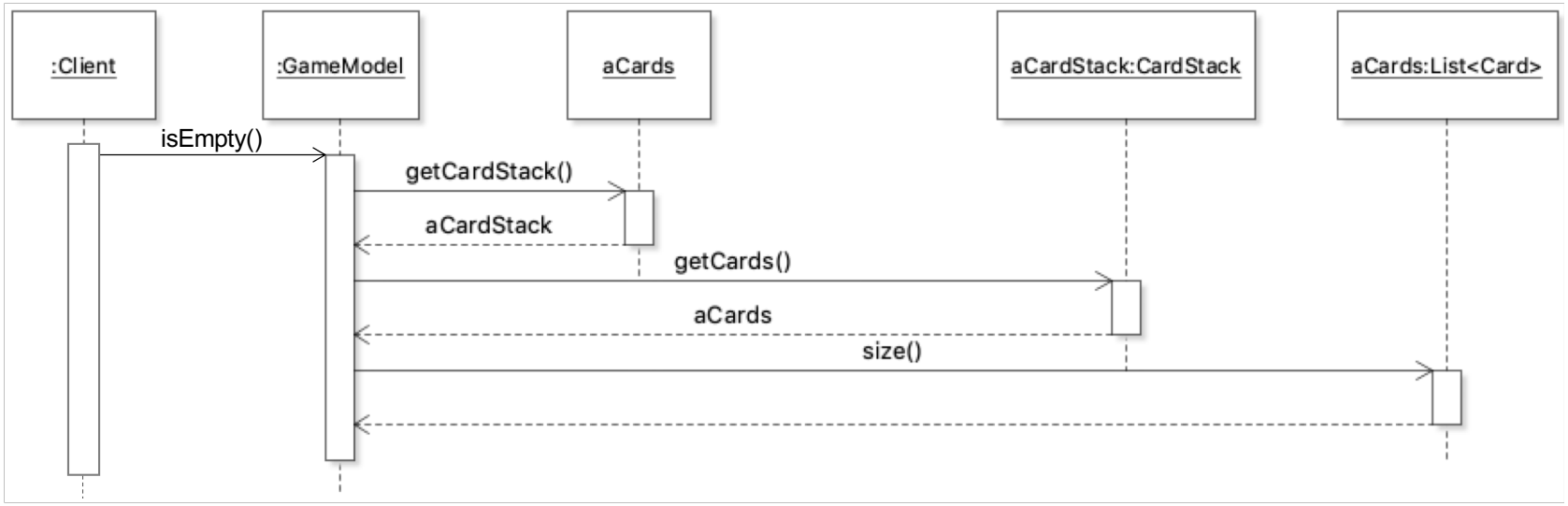
Modify CardSource interface to support undo function.

Anonymous Class for Shuffle Command Demo

Delegation Chains



Scenario of checking if the deck is empty



Who is in charge?

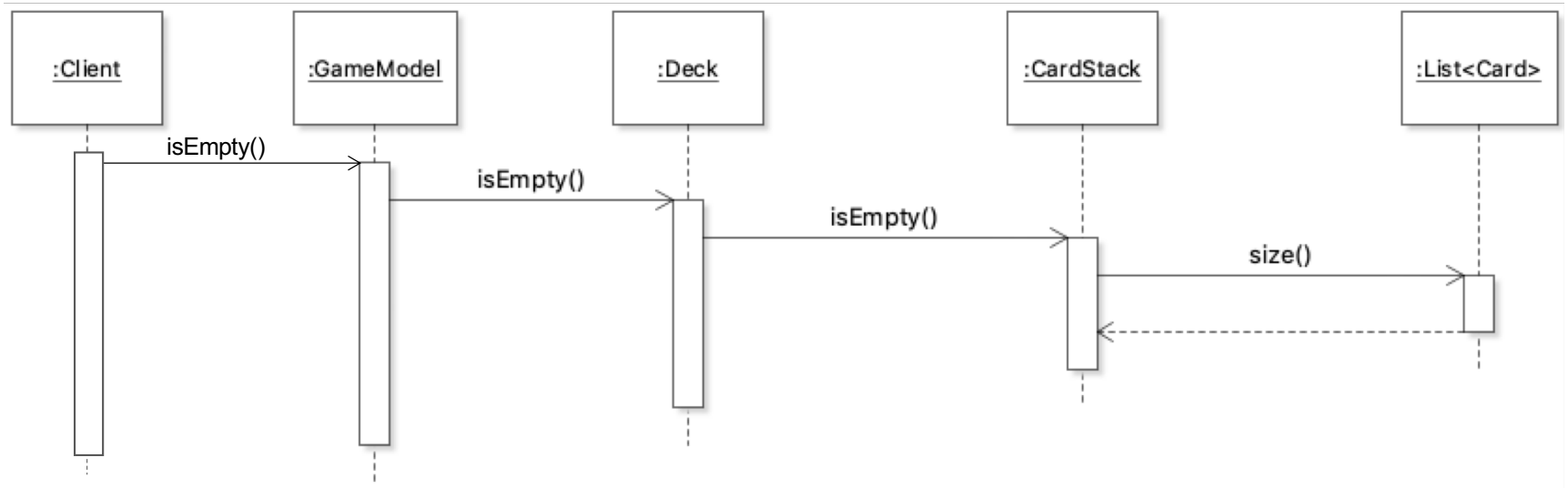
Law of Demeter

“Only talk to your friends”

The code of a method should only access:

- The instance variables of its implicit parameter;
- The arguments passed to the method;
- Any new object created within the method;
- (If need be) globally available objects.

Following the Law of Demeter



Activity

- Determine if the method calls are allowed according to the Law of Demeter:

```java
public class Colada {
   private Blender aBlender;
   private Vector aIngredients;
   public Colada()
   {
      aBlender = new Blender();
      aIngredients  = new Vector();
   }
   public void addIngredientsToBlender()
   {
      aBlender.addIngredients(aIngredients.elements());
   }
   public void printReceipt(Inventory pInventory)
   {
      PriceCalculator priceCalculator = pInventory.getPriceCalculator();
      Price price = priceCalculator.compute(aIngredients.elements());
      System.out.print(price);
   }
}
```

Acknowledgement

- Some examples are from the following resources:
  - *COMP 303 Lecture note* by Martin Robillard.
  - *The Pragmatic Programmer* by Andrew Hunt and David Thomas, 2000.
  - *Effective Java* by Joshua Bloch, 3rd ed., 2018.