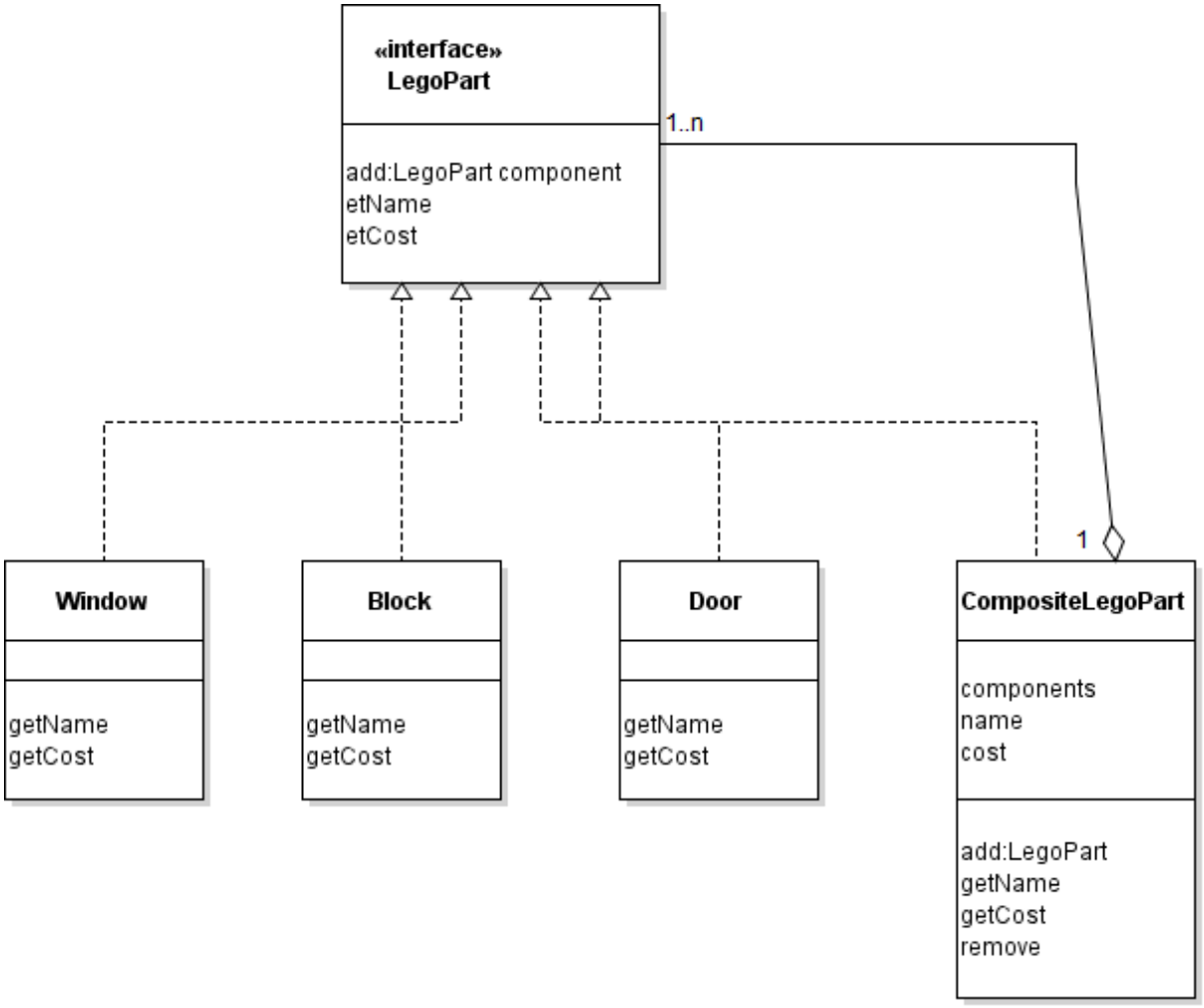
For the final exercise I decide to write it down in Java. I took the example with the Invoice Creator and the first thing was to apply the Composite Pattern. I created an Interface called LegoPart

```
import java.math.BigDecimal;

public interface LegoPart {

	public void add(LegoPart component);

	public String getName();

	public BigDecimal getCost();

}
```

This interface was implemented by all leafs (Window, Door and Block) the method add was left empty as they are leafs in the composition. The class CompositeLegoPart implemented the add method which is adding the leafs into an array.

```
public class CompositeLegoPart implements LegoPart{

	List<LegoPart> components = new ArrayList<LegoPart>();
	private String name;
	private BigDecimal cost;

	public CompositeLegoPart(String name, BigDecimal cost) {
		this.name = name;
		this.cost = cost;
	}

	public CompositeLegoPart() {
	}

	public void add(LegoPart component) {
		components.add(component);
	}

	public String getName() {
		return name;
	}

	public BigDecimal getCost() {
		return cost;
	}

	public void remove(LegoPart component) {
		components.remove(component);
	}

	public String toString() {
		BigDecimal total = BigDecimal.ZERO;
		String text = "";

		for (LegoPart component : components) {
			total = total.add(((LegoPart) component).getCost());
			text += ((LegoPart) component).getName() + ": " + ((LegoPart)
component).getCost() + "\n";
		}
```

```
            return text + "TOTAL: " + total.toString();
        }
}
```



2. A requirement come to add to the invoice not only the parts for building the house (like-door, window and blocks), but the furniture that are needed for the interior of the house, the requirement was that I need to be able to distinguish between the parts needed to build the house and the parts needed for the interior of the house. But all of them should be included in the invoice. The key was that for the furniture we do not need the name only, but most importantly we need the where is made(who is the manufacturer) So I decided to use the Adapter patter, this way I won't change the functionality already provide. For this purpose I create a Furniture Object which has one extra parameter – manufacturer passed in.

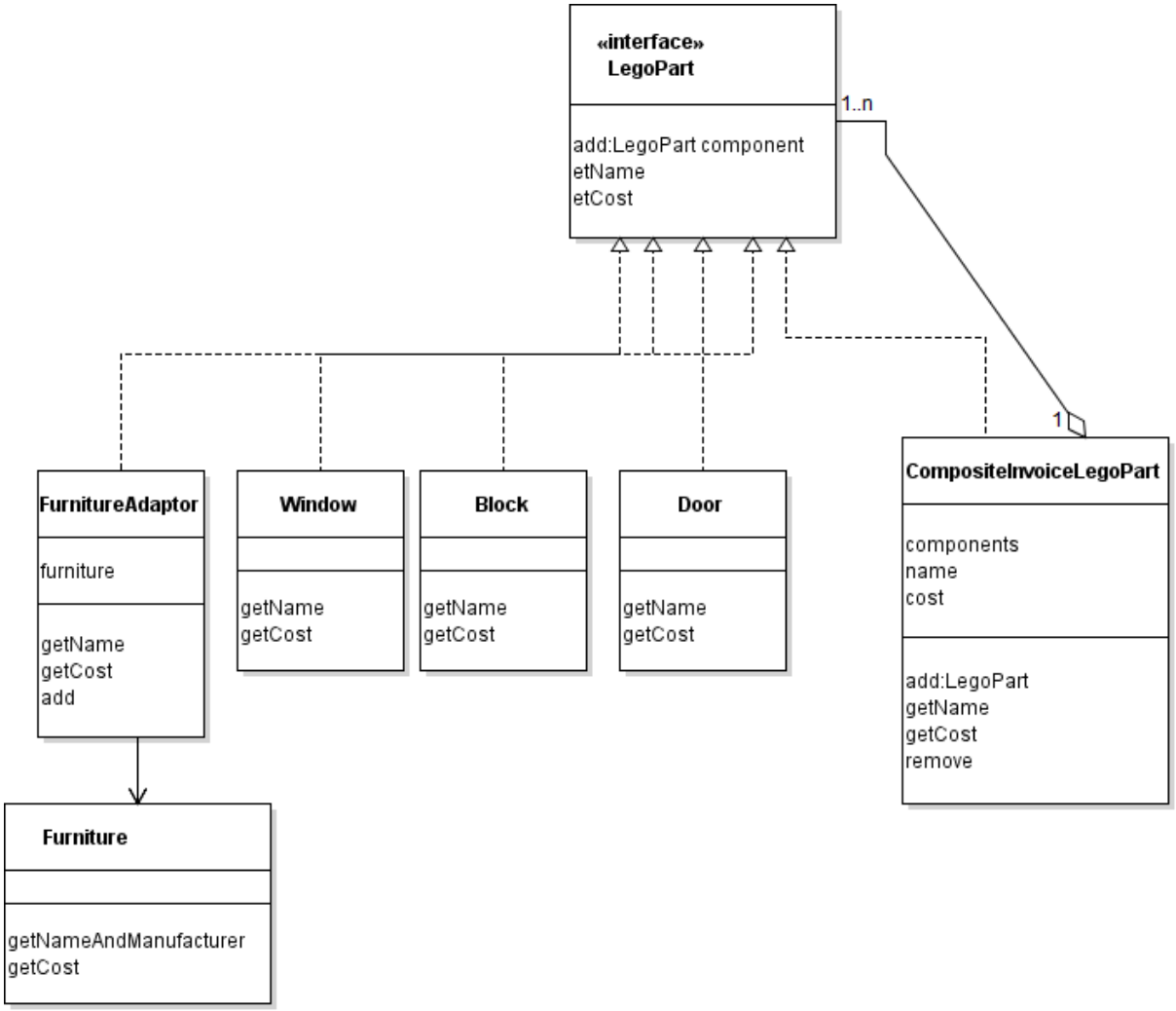
```
public class Furniture {
	private String name;
	private BigDecimal cost;
	private String manufacturer;

	public Furniture(String name, BigDecimal cost, String manufacturer) {
		this.name = name;
		this.cost = cost;
		this.manufacturer = manufacturer;
	}

	public String getNameAndManufacturer() {
		return name+ " made in " + manufacturer;
	}

	public BigDecimal getCost() {
		return cost;
	}
}
```

And the key here is to create an Adaptor Object which needs to implement the LegoPart interface. The constructor takes an instance of Furniture which we are going to adapt and when the getName is called we will delegate it to the Furniture getNameAndManufacturer() method which returns extra information.

```
import java.math.BigDecimal;


public class FurnitureAdaptor implements LegoPart{

	Furniture furniture;

	public FurnitureAdaptor(Furniture furniture) {
		this.furniture = furniture;
	}

	public String getName() {
		return furniture.getNameAndManufacturer();
	}

	public BigDecimal getCost() {
		return furniture.getCost();
	}

	public void add(LegoPart component) {
		// this is a leaf node so adding is not applicable here
	}
}
```

So our CompositeInvoiceLegoPart can take an Adaptor which implements the same interface but provides different result.

```
public InvoiceCreator() {
            invoice = new CompositeInvoiceLegoPart();
            invoice.add(new Block("Block", new BigDecimal("10.00")));
            invoice.add(new Door("Door", new BigDecimal("12.50")));
            invoice.add(new Window("Window", new BigDecimal("17.50")));
            invoice.add(new FurnitureAdaptor(new Furniture("Sofa", new
BigDecimal("52.50"), "Italy")));
      }
```



3. A new requirement come and I needed to adopt it to the current solution. I have been asked to provide the ability to count and include in the invoice how many Furniture Lego parts were used in creating the house. For this purpose I thought a good fit will be to use the Decorator pattern so thus I don't need to change my interface. The counter was incremented in the getName, as the name was included only once in the invoice.

The LegoPartFurnitureCounter was implemented as a Decorator implementing the LegoPart interface, all Furnitures parts only were wrapped with it.

```
import java.math.BigDecimal;
```

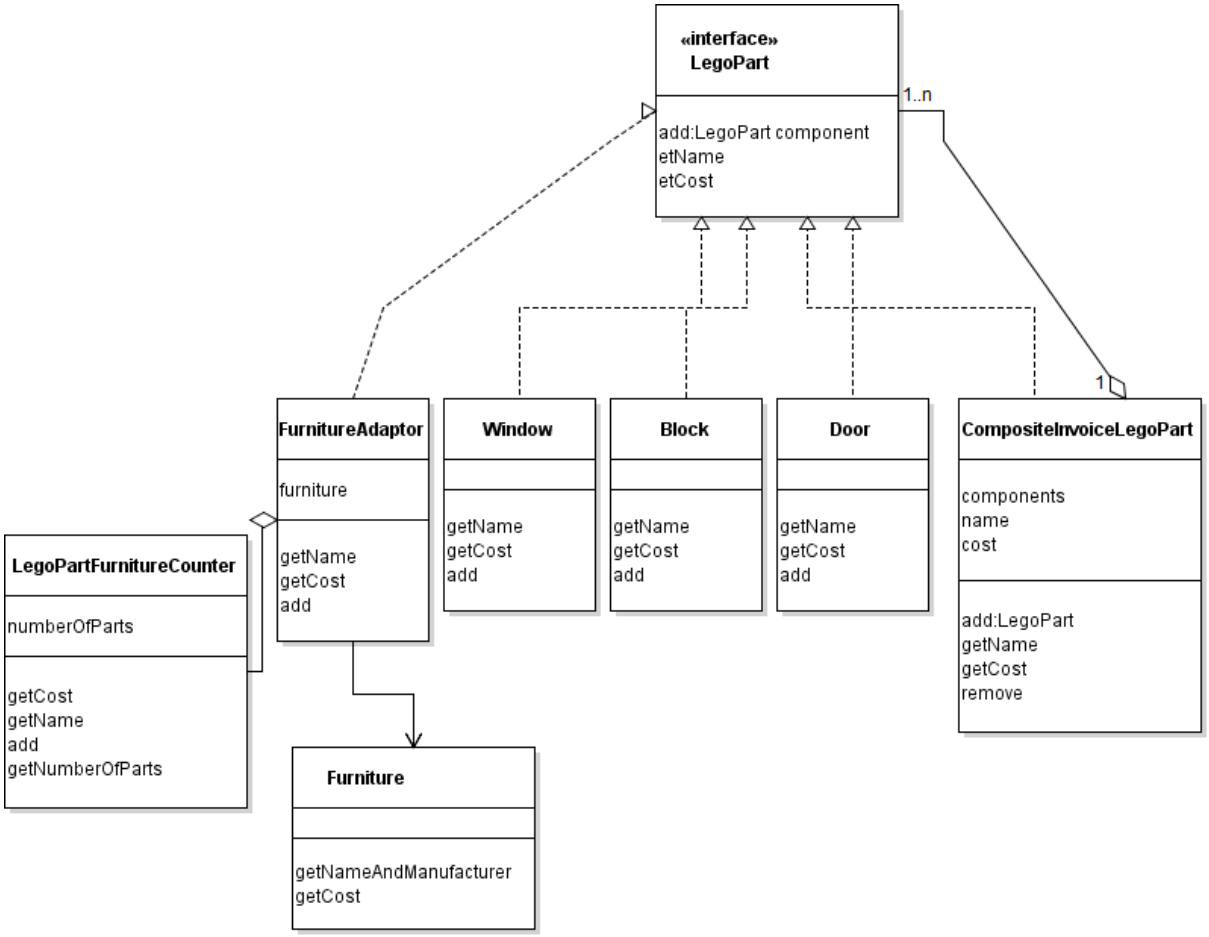
```
public class LegoPartFurnitureCounter implements LegoPart{

	LegoPart legoPart;
	static int numberOfParts = 0;

	public LegoPartCounter(LegoPart legoPart) {
		this.legoPart = legoPart;
	}

	@Override
	public void add(LegoPart component) {
	}

	@Override
	public String getName() {
		numberOfParts ++;
		return legoPart.getName();
	}

	@Override
	public BigDecimal getCost() {
		return legoPart.getCost();
	}

	public static int getNumberOfParts() {
		return numberOfParts;
	}
}
```

And adding the parts in the InvoiceCreator looks like that:

```
	public InvoiceCreator() {
		invoice = new CompositeInvoiceLegoPart();
		invoice.add(new Block("Block", new BigDecimal("10.00")));
		invoice.add(new Door("Door", new BigDecimal("12.50")));
		invoice.add(new Window("Window", new BigDecimal("17.50")));
		invoice.add(new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture("Sofa", new BigDecimal("52.50"), "Italy"))));
		invoice.add(new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture("Double bed", new BigDecimal("31.50"), "France"))));

	}
```

This is the output of the invoice:

```
Block: 10.00
Door: 12.50
Window: 17.50
Sofa is made in Italy: 52.50
Double bed is made in France: 31.50
TOTAL: 124.00 including 2 furniture parts
```



4. New requirement come where the customer asked to have a faster and convenient way for producing and adding the Furniture lego parts to the invoice. I decided to use the Abstract Factory Pattern for creating a furniture parts only.

I created an Abstract factory with only method getFurniture which accepts the name of the furniture. `FurnitureFactory` extends it and returns the desired furniture based on its name.

```
import java.math.BigDecimal;


public class FurnitureFactory extends AbstractFurnitureFactory{

	public static final String SOFA = "Sofa";
	public static final String DOUBLE_BED = "Double bed";
	public static final String BED = "Bed";
	public static final String TABLE = "Table";
	public static final String CHAIR = "Chair";

	@Override
	public LegoPart getFurniture(String name) {

		switch (name) {
		case SOFA:
			return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(SOFA, new BigDecimal("52.90"), "Italy")));
		case DOUBLE_BED:
```

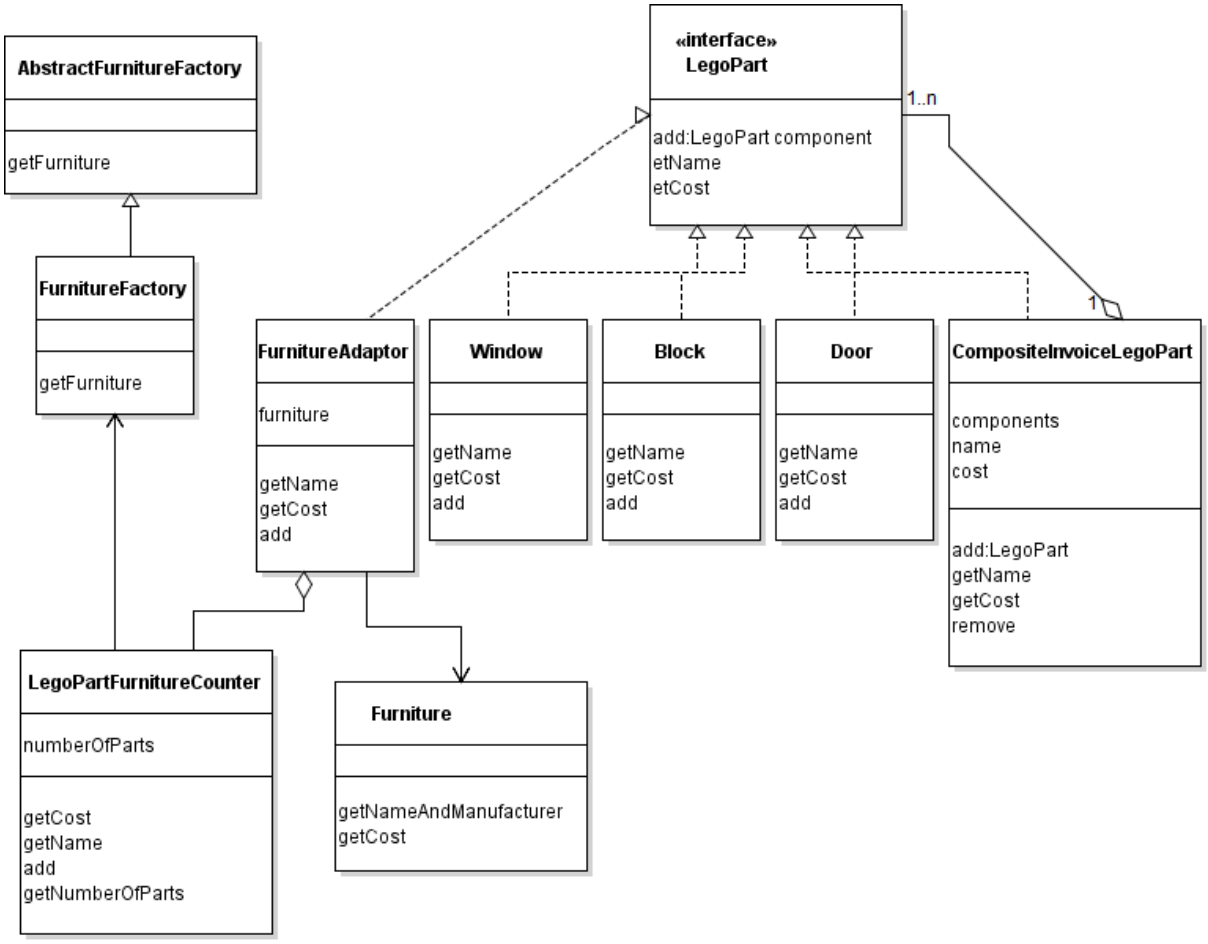
```
                    return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(DOUBLE_BED, new BigDecimal("31.70"), "France")));
             case BED:
                    return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(BED, new BigDecimal("27.50"), "Holand")));
             case TABLE:
                    return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(TABLE, new BigDecimal("15.30"), "Ireland")));
             case CHAIR:
                    return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(CHAIR, new BigDecimal("9.30"), "Italy")));
             default:
                    return new LegoPartFurnitureCounter(new FurnitureAdaptor(new
Furniture(CHAIR, new BigDecimal("9.30"), "Italy")));
             }
      }
}
```

So now in my Invoice creator I had the following code:

```
      public InvoiceCreator() {
             FurnitureFactory furnitureFactory = new FurnitureFactory();
             invoice = new CompositeInvoiceLegoPart();
             invoice.add(new Block("Block", new BigDecimal("10.00")));
             invoice.add(new Door("Door", new BigDecimal("12.50")));
             invoice.add(new Window("Window", new BigDecimal("17.50")));
             invoice.add(furnitureFactory.getFurniture(FurnitureFactory.BED));

      invoice.add(furnitureFactory.getFurniture(FurnitureFactory.DOUBLE_BED));
             invoice.add(furnitureFactory.getFurniture(FurnitureFactory.TABLE));
             invoice.add(furnitureFactory.getFurniture(FurnitureFactory.SOFA));
             invoice.add(furnitureFactory.getFurniture(FurnitureFactory.CHAIR));

      }
```

And the output produced is:

```
Block: 10.00
Door: 12.50
Window: 17.50
Bed is made in Holand: 27.50
Double bed is made in France: 31.70
Table is made in Ireland: 15.30
Sofa is made in Italy: 52.90
Chair is made in Italy: 9.30
TOTAL: 176.70 including 5 furniture parts
```



5. A new requirement comes along that we do need to track dynamically every time a new part is added to the invoice. This leads me to use the Observable Pattern. I created an interface LegoPartObservable where now our LegoPart interface needs to extends from it.

```
public interface LegoPartObservable {

	public void registerObserver(Observer observer);
	public void notifyObservers();
}
```

We do need a helper class to implement the LegoPartObservable which will hold a list of all observers.

```
public class Observable implements LegoPartObservable{

	ArrayList<Observer> observers = new ArrayList<Observer>();
	LegoPartObservable legoPart;

	public Observable(LegoPartObservable legoPart) {
		this.legoPart = legoPart;
	}
```

```
	@Override
	public void notifyObservers() {
		Iterator<Observer> it = observers.iterator();
		while(it.hasNext()) {
			Observer observer = (Observer)it.next();
			observer.update(legoPart);
		}
	}

	@Override
	public void registerObserver(Observer observer) {
		observers.add(observer);
	}
}
```

Now `registerObserver` and `notifyObservers` needs to be implemented in all Obkects implementing the LegoPart interface. Some examplein the constructor observables needs to be instantiated

```
	private Observable observable;

	public Block(String name, BigDecimal cost) {
		observable = new Observable(this);
		this.name = name;
		this.cost = cost;
	}
```

…….

```
	@Override
	public void registerObserver(Observer observer) {
		observable.registerObserver(observer);
	}

	@Override
	public void notifyObservers() {
		observable.notifyObservers();
	}
```

An interface `Observer` was created with a method update in it. The `LegoPartObserverImpl` implements it a where the update method is call on all Observable when notifyObservers is invoked.

In InvoiceCreator now we have to register the observers on the desired objects as follows:

```
	public InvoiceCreator() {
		LegoPartObserverImpl lpo = new LegoPartObserverImpl();
```

```
            invoice = new CompositeInvoiceLegoPart();

            LegoPart block = new Block("Block", new BigDecimal("10.00"));
            block.registerObserver(lpo);

            LegoPart door = new Door("Door", new BigDecimal("12.50"));
            door.registerObserver(lpo);

            LegoPart window = new Window("Window", new BigDecimal("17.50"));
            window.registerObserver(lpo);

            FurnitureFactory furnitureFactory = new FurnitureFactory();

            LegoPart bed = furnitureFactory.getFurniture(FurnitureFactory.BED);
            bed.registerObserver(lpo);

            LegoPart doubleBed =
furnitureFactory.getFurniture(FurnitureFactory.DOUBLE_BED);
            doubleBed.registerObserver(lpo);

            LegoPart table =
furnitureFactory.getFurniture(FurnitureFactory.TABLE);
            table.registerObserver(lpo);

            LegoPart sofa =
furnitureFactory.getFurniture(FurnitureFactory.SOFA);
            sofa.registerObserver(lpo);

            LegoPart chair =
furnitureFactory.getFurniture(FurnitureFactory.CHAIR);
            chair.registerObserver(lpo);

            invoice = new CompositeInvoiceLegoPart();

            invoice.add(block);
            invoice.add(door);
            invoice.add(window);
            invoice.add(bed);
            invoice.add(doubleBed);
            invoice.add(table);
            invoice.add(sofa);
            invoice.add(chair);
      }
```

To string was overridden to return the name of the Object:

This is sample output:

```
A lego part: Block was just added to the invoice!
A lego part: Door was just added to the invoice!
A lego part: Window was just added to the invoice!
A lego part: Bed is made in Holand was just added to the invoice!
A lego part: Double bed is made in France was just added to the invoice!
A lego part: Table is made in Ireland was just added to the invoice!
A lego part: Sofa is made in Italy was just added to the invoice!
A lego part: Chair is made in Italy was just added to the invoice!
Block: 10.00
```

```
Door: 12.50
Window: 17.50
Bed is made in Holand: 27.50
Double bed is made in France: 31.70
Table is made in Ireland: 15.30
Sofa is made in Italy: 52.90
Chair is made in Italy: 9.30
TOTAL: 176.70 including 5 furniture parts
```

See the final UML for reference: