LESSON 10
DECORATOR, VISITOR PATTERN

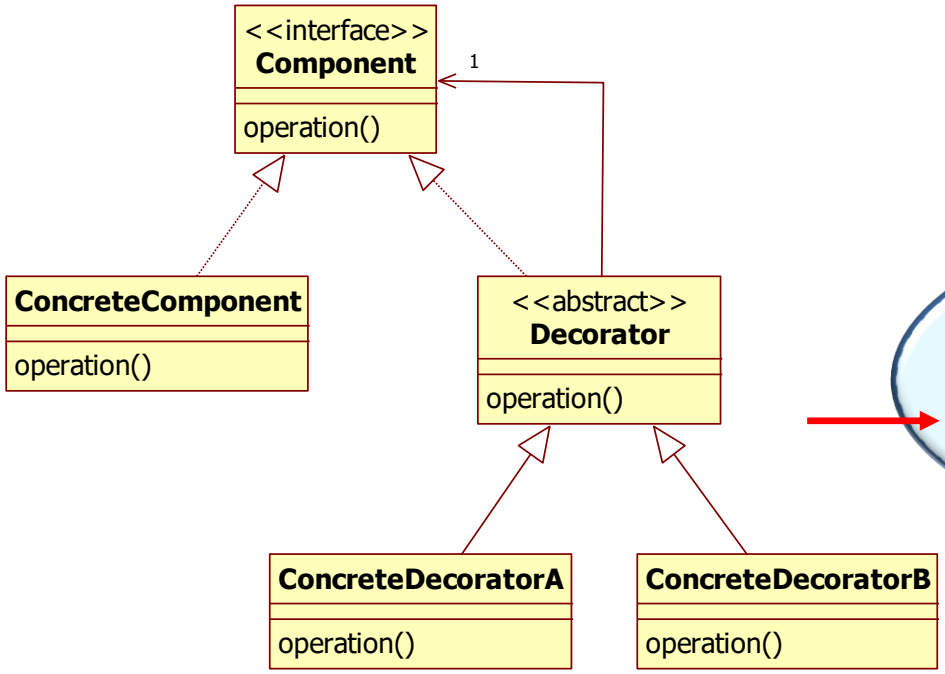
Decorator pattern

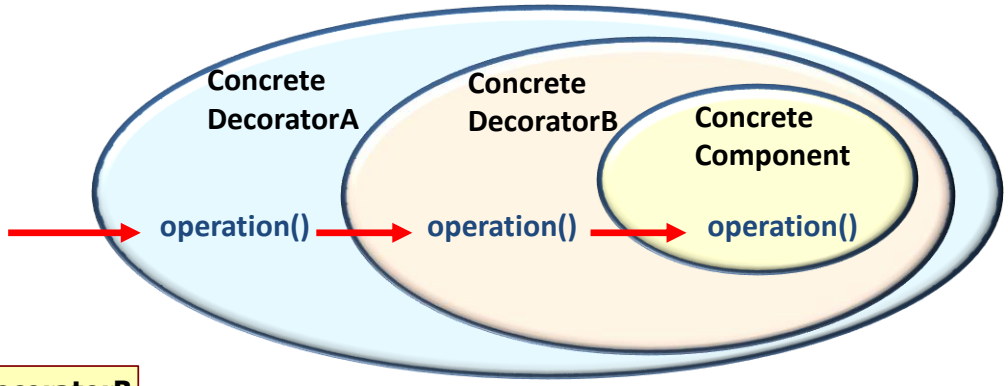
- Allows to dynamically add new behavior to an existing object.

Plain pizza crust

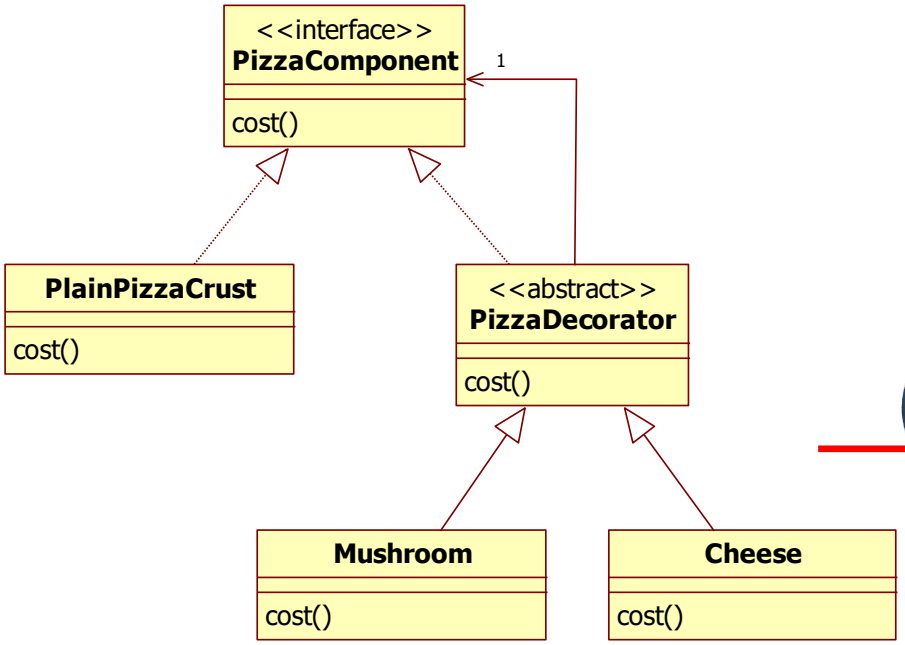
Pizza toppings

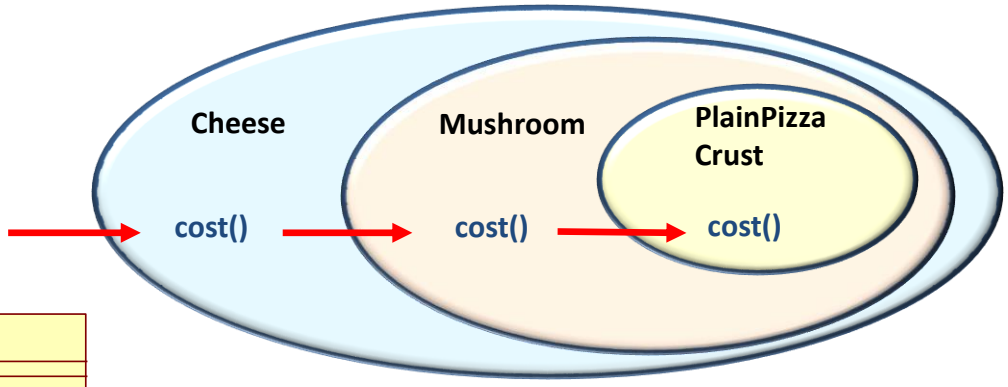
Decorator pattern



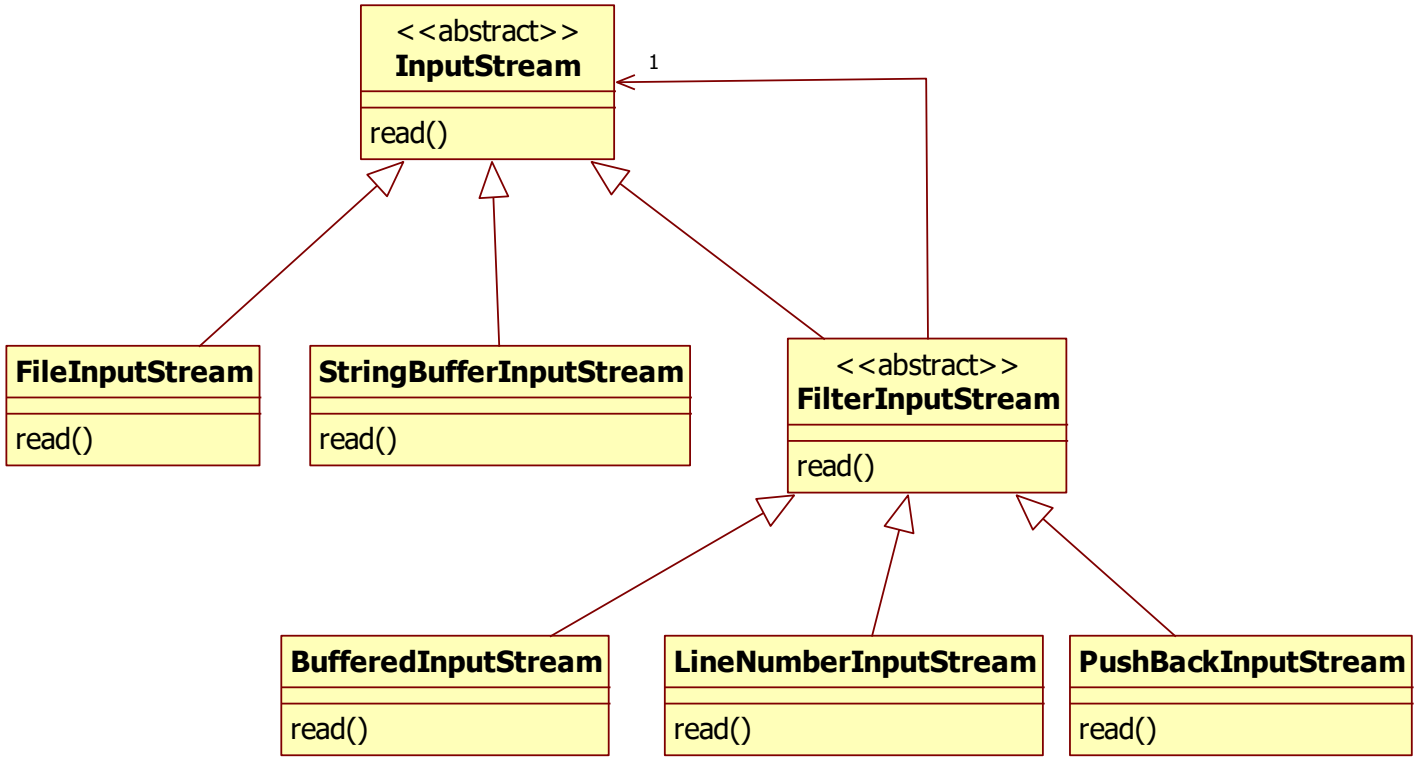


Decorating a pizza





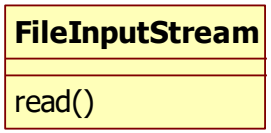
Java.io

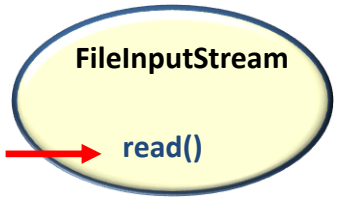

<<abstract>>
InputStream
read()
1
FileInputStream
read()
StringBufferInputStream
read()
<<abstract>>
FilterInputStream
read()
BufferedInputStream
read()
LineNumberInputStream
read()
PushBackInputStream
read()

FileInputStream

```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream = new FileInputStream(rootPath + "/input.txt");

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Reads a byte of data

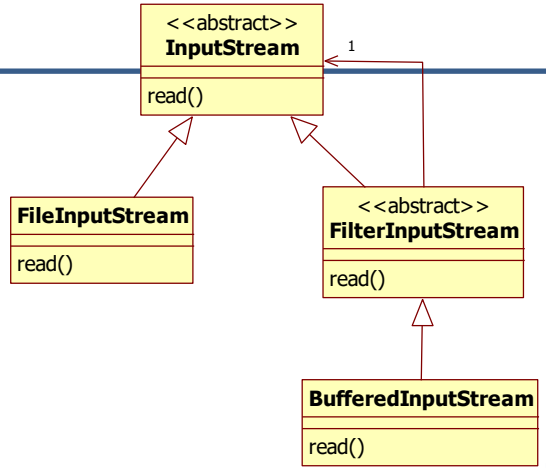


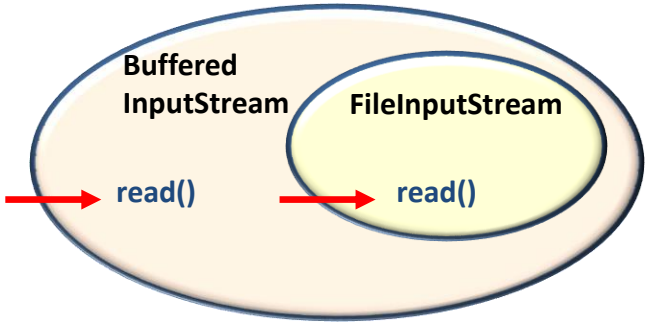


BufferedInputStream

```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream =
            new BufferedInputStream(new FileInputStream(rootPath + "/input.txt"));

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

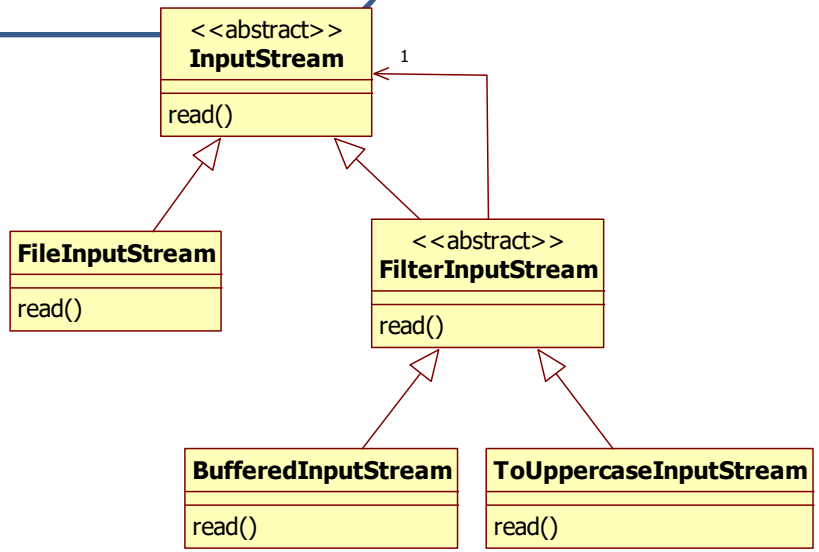
Reads 8 kilobytes of data and buffers them





Write your own decorator

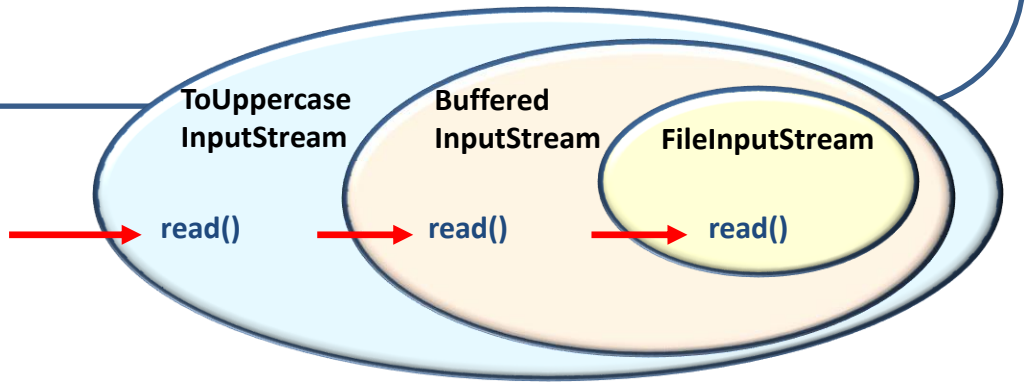
```
public class ToUppercaseInputStream extends FilterInputStream {

  protected ToUppercaseInputStream(InputStream in) {
    super(in);
  }

  @Override
  public int read() throws IOException {
    int c = super.read();
    if (c != -1)
      c = Character.toUpperCase((char)c);
    return c;
  }
}
```



ToUppercaseInputStream

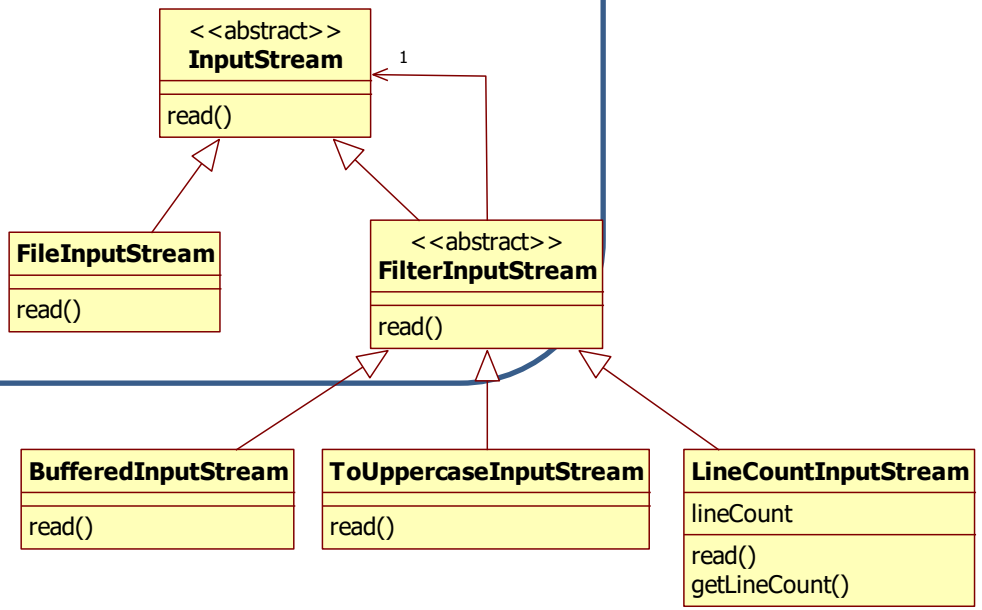
```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      InputStream inputStream =
            new ToUppercaseInputStream(new BufferedInputStream(
                new FileInputStream(rootPath + "/input.txt")));
      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }

      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Add decorators to the FileInputStream



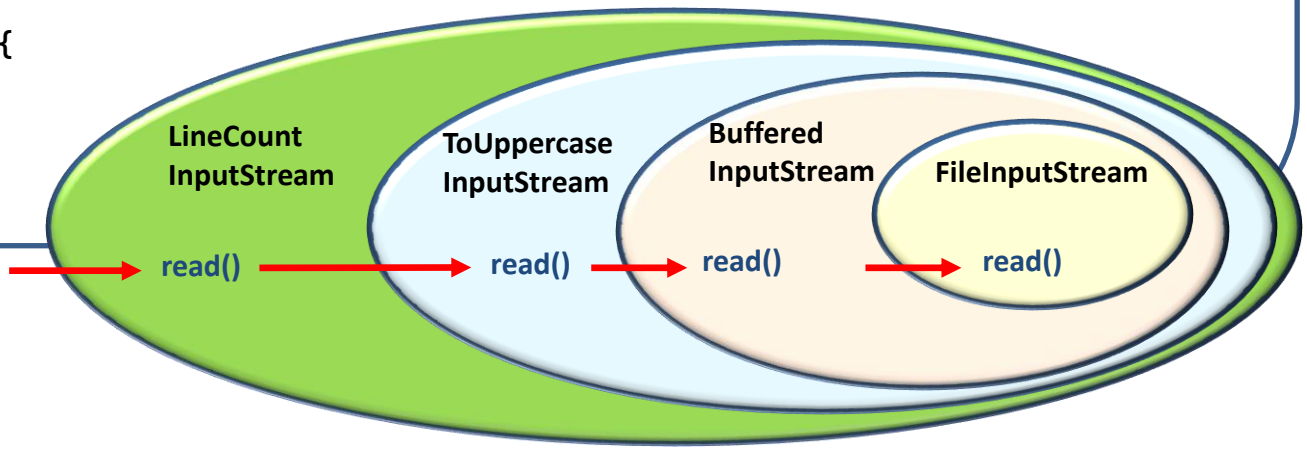
Write your own decorator

```
public class LineCountInputStream extends FilterInputStream {
  int lineCount = 0;

  protected LineCountInputStream(InputStream in) {
    super(in);
  }

  @Override
  public int read() throws IOException {
    int c = super.read();
    if (c != -1 && c==10 ) //carriage return = 10
      lineCount++;
    return c;
  }

  public int getLineCount() {
    return lineCount;
  }
}
```

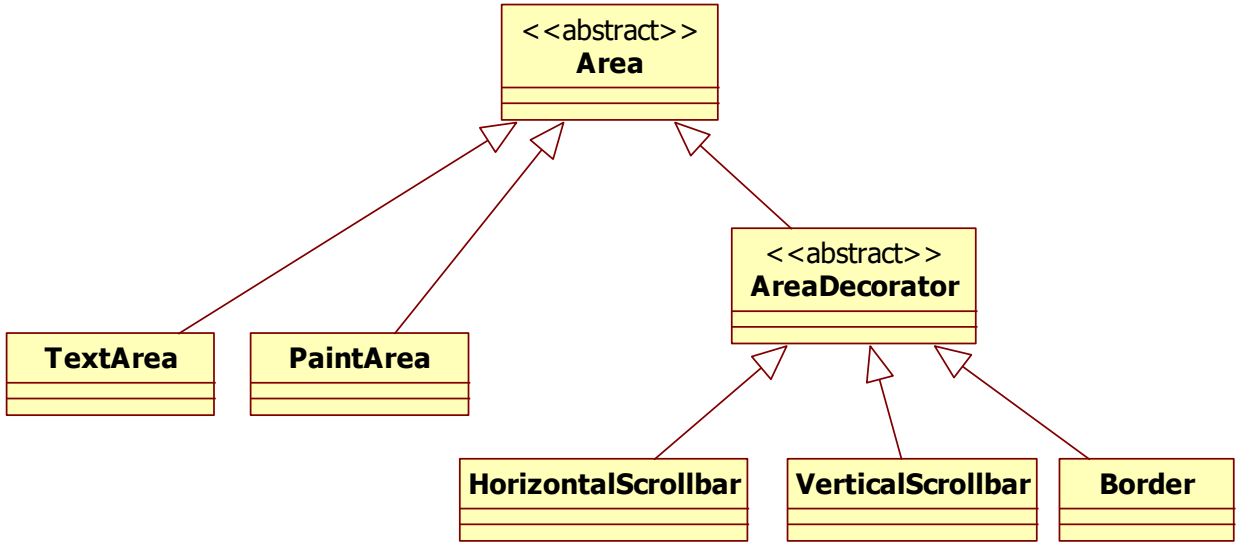


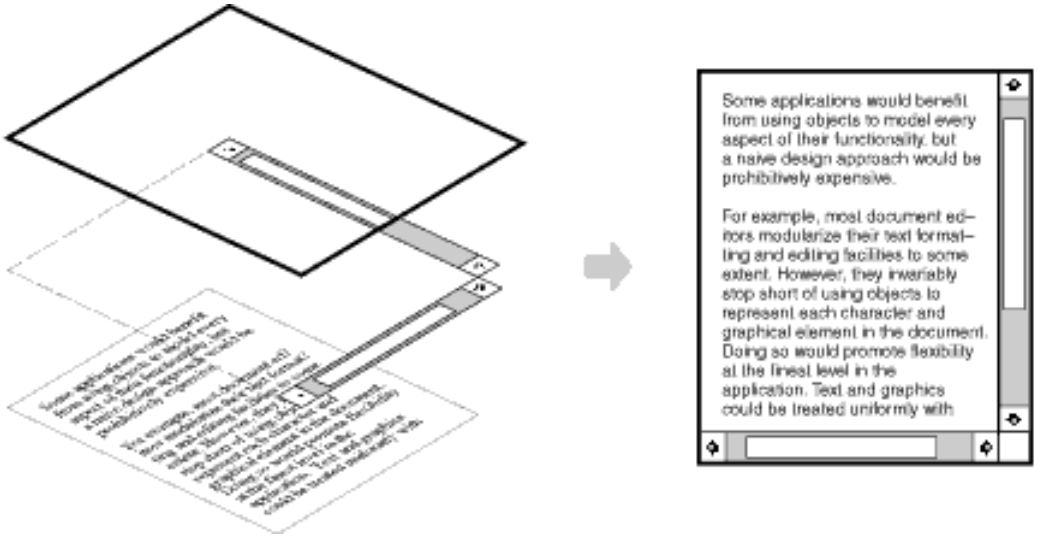
LineCountInputStream

```
public class Application {

  public static void main(String[] args) {
    int c;
    String rootPath = Thread.currentThread().getContextClassLoader().getResource("").getPath();
    try {
      LineCountInputStream inputStream =
        new LineCountInputStream(new ToUppercaseInputStream(new BufferedInputStream(
          new FileInputStream(rootPath + "/input.txt"))));

      while ((c = inputStream.read()) >= 0) {
        System.out.print((char) c);
      }
      System.out.println("");
      System.out.println("This file contains "+inputStream.getLineCount()+" lines");
      inputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```



Decorator example

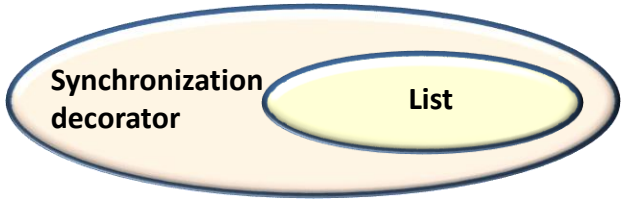




Decorator in Java collections

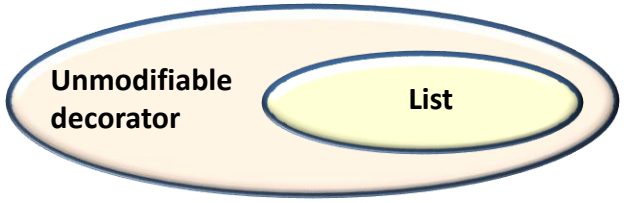
```
public static <T> Collection<T> synchronizedCollection(Collection<T> c);
public static <T> Set<T> synchronizedSet(Set<T> s);
public static <T> List<T> synchronizedList(List<T> list);
```

Factory methods that return a decorated collection



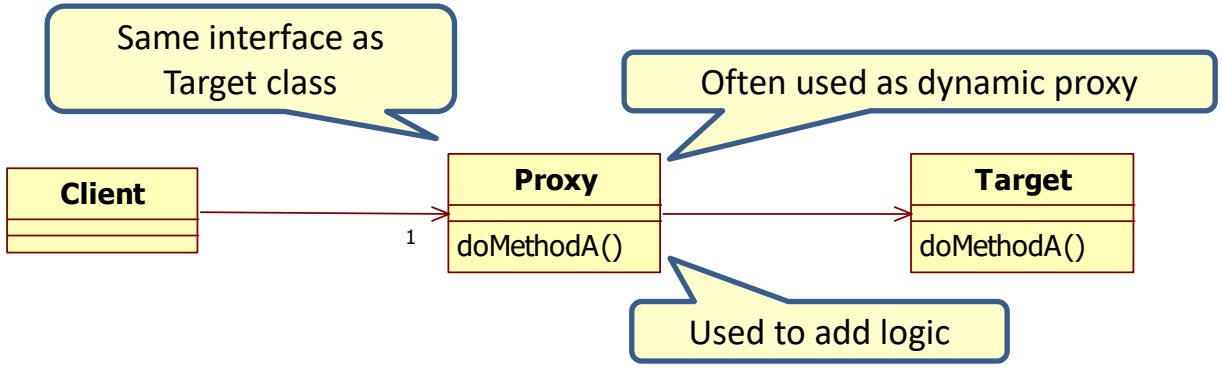

```
public static <T> Collection<T> unmodifiableCollection(Collection<? extends T> c);
public static <T> Set<T> unmodifiableSet(Set<? extends T> s);
public static <T> List<T> unmodifiableList(List<? extends T> list);
```

Factory methods that return an unmodifiable (immutable) collection

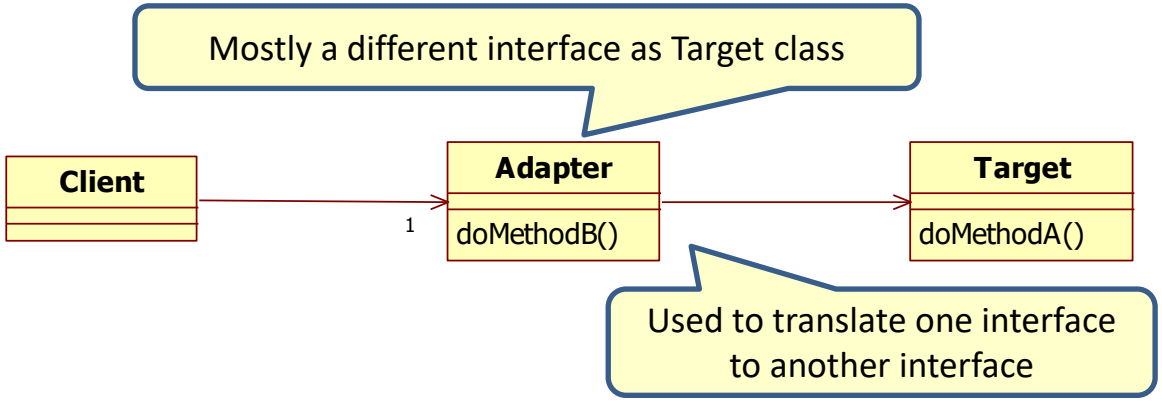


Wrappers

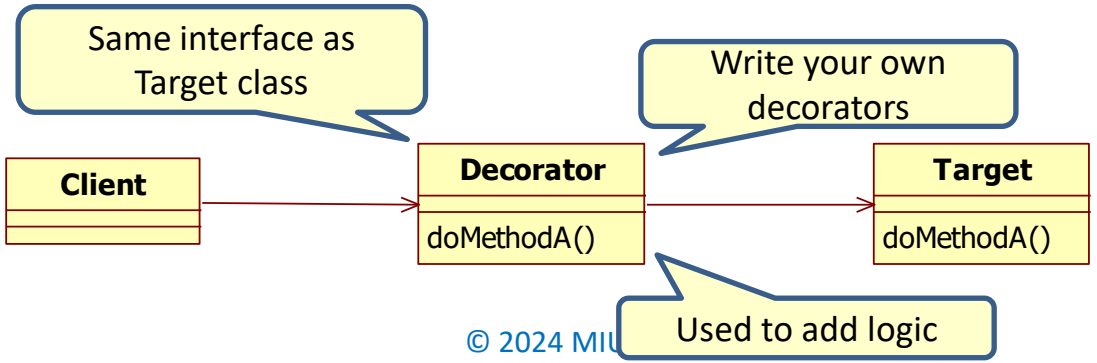
- Proxy



- Adapter



- Decorator

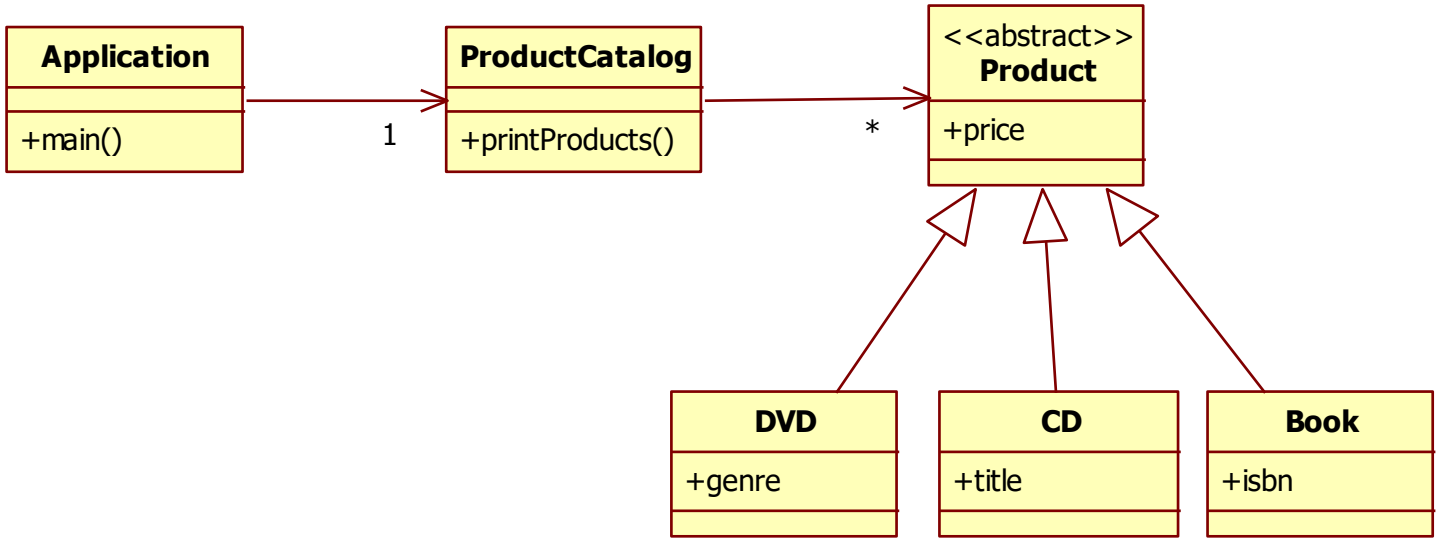


VISITOR PATTERN

Visitor pattern

- separating an algorithm from an object structure it operates on. A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures

Example application



ProductCatalog

```
public class ProductCatalog {
  private Collection<Product> productlist = new ArrayList<Product>();

  public ProductCatalog() {
    productlist.add(new DVD(12.95, "drama"));
    productlist.add(new CD(6.00, "Dancing Queen"));
    productlist.add(new Book(22.85, "Harry Potter"));
    productlist.add(new DVD(11.95, "action"));
  }

  public void printProducts(){
    for (Product product : productlist){
      if (product instanceof Book){
        System.out.println("Book: title="+((Book)product).getIsbn()+"
                           price="+product.getPrice());
      }
      if (product instanceof DVD){
        System.out.println("DVD: genre="+((DVD)product).getGenre()+"
                           price="+product.getPrice());
      }
      if (product instanceof CD){
        System.out.println("CD: artist="+((CD)product).getArtist()+"
                           price="+product.getPrice());
      }
    }
  }
}
```

The products

```
public abstract class Product {
  private double price;

  public Product(double price) {
    super();
    this.price = price;
  }

  public double getPrice() {
    return price;
  }

  public void setPrice(double price) {
    this.price = price;
  }
}
```

```
public class Book extends Product{
  private String isbn;

  public Book(double price, String isbn) {
    super(price);
    this.isbn=isbn;
  }

  public String getIsbn() {
    return isbn;
  }
}
```

```
public class CD extends Product{
  private String artist;

  public CD(double price, String artist) {
    super(price);
    this.artist=artist;
  }

  public String getArtist() {
    return artist;
  }
}
```

```
ublic class DVD extends Product{
private String genre;

public DVD(double price, String genre) {
  super(price);
  this.genre=genre;
}

public String getGenre() {
  return genre;
}
```

Example application

```
public class Application {

  public static void main(String[] args) {
    ProductCatalog catalog = new ProductCatalog();
    catalog.printProducts();
  }
}
```

```
DVD: genre=drama price=12.95
CD: artist=Dancing Queen price=6.0
Book: title=Harry Potter price=22.85
DVD: genre=action price=11.95
```

Suppose we give a discount

```
if (product == DVD){
   if (price < 8) price=price-(0.1*price)
   if (price > 8) price=price-(0.15*price)
}
if (product == CD){
   if (price < 8) price=price-(0.1*price)
   if (8 <price < 12) price=price-(0.175*price)
   if (price > 12) price=price-(0.2*price)
}
if (product == Book){
   price=price-(0.125*price)
}
```

Option 1

```
public class ProductCatalog {
  private Collection<Product> productlist = new ArrayList<Product>();

  public ProductCatalog() {
    productlist.add(new DVD(12.95, "drama"));
    productlist.add(new CD(6.00, "Dancing Queen"));
    productlist.add(new Book(22.85, "Harry Potter"));
    productlist.add(new DVD(11.95, "action"));
  }

  public void printProducts(){
    for (Product product : productlist){
      if (product instanceof Book){
        System.out.println("Book: title="+((Book)product).getIsbn()+"
                           price="+calculatePrice(product));
      }
      if (product instanceof DVD){
        System.out.println("DVD: genre="+((DVD)product).getGenre()+"
                           price="+calculatePrice(product));
      }
      if (product instanceof CD){
        System.out.println("CD: artist="+((CD)product).getArtist()+"
                           price="+calculatePrice(product));
      }
    }
  }
```

Option 1

```
private double calculatePrice(Product product){
  double price = product.getPrice();
  if (product instanceof DVD){
    if (price < 8) return price-(0.1*price);
    if (price > 8) return price-(0.15*price);
  }
  if (product instanceof CD){
    if (price < 8) return price-(0.1*price);
    if ((8 < price) && (price < 12)) return price-(0.175*price);
    if (price > 12) return price-(0.2*price);
  }
  if (product instanceof Book){
    return price-(0.125*price);
  }
  return 0;
}
```

Option 2

```
public class DVD extends Product{
  private String genre;

...

  public double getPrice(){
    if (price < 8) return price-(0.1*price);
    if (price >= 8) return price-(0.15*price);
    return 0;
  }

...
}
```

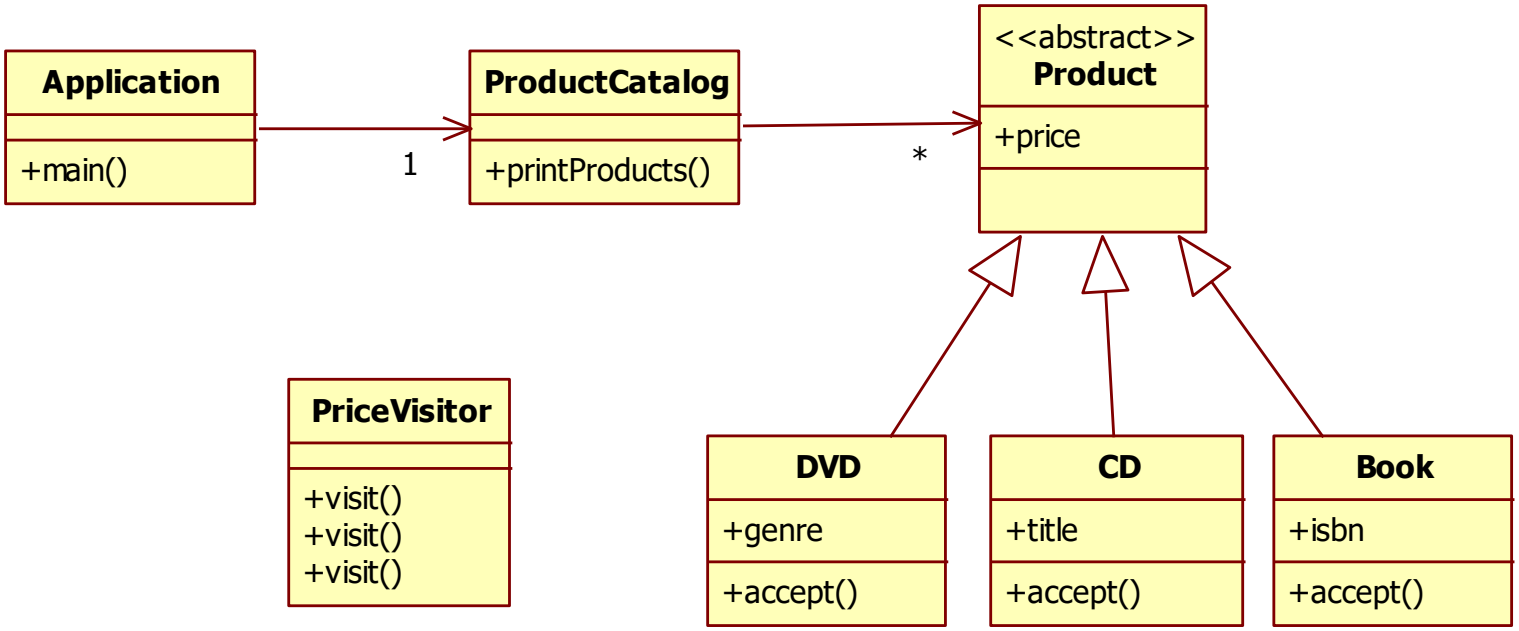
```
public class Book extends Product{
  private String isbn;

...

  public double getPrice(){
    return price-(0.125*price);
  }

...
}
```

```
public class CD extends Product{
  private String artist;

...

  public double getPrice(){
    if (price <= 8) return price-(0.1*price);
    if ((8 < price) && (price < 12)) return price-(0.175*price);
    if (price >= 12) return price-(0.2*price);
    return 0;
  }

...
}
```

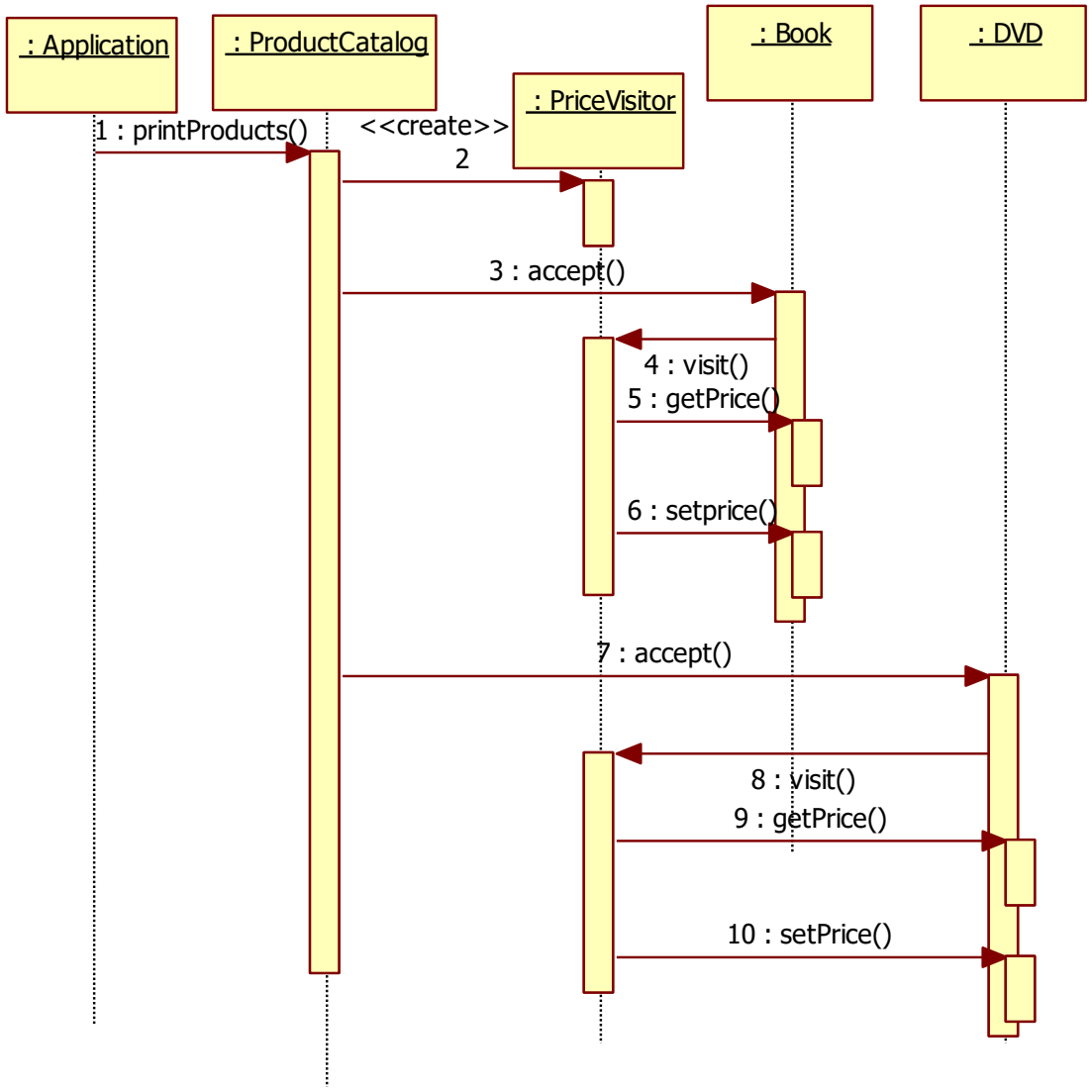
Option 2

```
public void printProducts(){
  for (Product product : productlist){
    if (product instanceof Book){
      Book book = (Book)product;
      System.out.println("Book: title="+book.getIsbn()+" price="+book.getPrice());
    }
    if (product instanceof DVD){
      DVD dvd=(DVD)product;
      System.out.println("DVD: genre="+dvd.getGenre()+" price="+dvd.getPrice());
    }
    if (product instanceof CD){
      CD cd = (CD)product;
      System.out.println("CD: artist="+cd.getArtist()+" price="+cd.getPrice());
    }
  }
}
```

With visitor pattern



With visitor



With Visitor

```
public class PriceVisitor implements Visitor {

  public void visit(Book product) {
    double price = product.getPrice();
    price = price - (0.125 * price);
    product.setPrice(price);
  }

  public void visit(CD product) {
    double price, newprice = 0;
    price = product.getPrice();
    if (price < 8) newprice = price - (0.1 * price);
    if ((8 < price) && (price < 12)) newprice = price - (0.175 * price);
    if (price > 12) newprice = price - (0.2 * price);
    product.setPrice(newprice);
  }

  public void visit(DVD product) {
    double price, newprice = 0;
    price = product.getPrice();
    if (price < 8) newprice = price - (0.1 * price);
    if (price > 8) newprice = price - (0.15 * price);
    product.setPrice(newprice);
  }
}
```

```
public interface Visitor {
  public void visit(Book product);
  public void visit(CD product);
  public void visit(DVD product);
}
```

The products

```
public class Book extends Product{
  private String isbn;

  public Book(double price, String isbn)
{
    super(price);
    this.isbn=isbn;
  }

  public String getIsbn() {
    return isbn;
  }

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

```
public class DVD extends Product{
  private String genre;

  public DVD(double price, String genre) {
    super(price);
    this.genre=genre;
  }

  public String getGenre() {
    return genre;
  }

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

```
public class CD extends Product{
  private String artist;

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

ProductCatalog

```
public class ProductCatalog {
  private Collection<Product> productlist = new ArrayList<Product>();

  public ProductCatalog() {
    productlist.add(new DVD(12.95, "drama"));
    productlist.add(new CD(6.00, "Dancing Queen"));
    productlist.add(new Book(22.85, "Harry Potter"));
    productlist.add(new DVD(11.95, "action"));
  }

  public void printProducts(){
    PriceVisitor priceVisitor = new PriceVisitor();
    for (Product product : productlist){
      if (product instanceof Book){
        Book book = (Book)product;
        book.accept(priceVisitor);
        System.out.println("Book: title="+book.getIsbn()+" price="+book.getPrice());
      }
      if (product instanceof DVD){
        DVD dvd=(DVD)product;
        dvd.accept(priceVisitor);
        System.out.println("DVD: genre="+dvd.getGenre()+" price="+dvd.getPrice());
      }
      if (product instanceof CD){
        CD cd = (CD)product;
        cd.accept(priceVisitor);
        System.out.println("CD: artist="+cd.getArtist()+" price="+cd.getPrice());
      }
    }
  }
}
```

The application

```
public class Application {

  public static void main(String[] args) {
    ProductCatalog catalog = new ProductCatalog();
    catalog.printProducts();
  }
}
```

```
DVD: genre=drama price=12.95
CD: artist=Dancing Queen price=6.0
Book: title=Harry Potter price=22.85
DVD: genre=action price=11.95
```

Visitor structure