


Lab 6 a

- 
- Given is the customers application which contains a CustomerCollection class with a list of customers.
 - Write an AgeIterator that iterates over the customer collection and returns the customer with the lowest age first, then the customer with the 2nd lowest age, etc.

Lab 6 a

```
public class AgeIterator implements Iterator<Customer> {
    private List<Customer> list;
    private Iterator<Customer> internalIterator;

    public AgeIterator(List<Customer> list) {
        this.list = new ArrayList<>(list);
        //sort the internal list
        Collections.sort(this.list, Comparator.comparingInt(Customer::getAge));
        internalIterator = this.list.iterator();
    }

    public Iterator<Customer> iterator() {
        return this;
    }
    @Override
    public boolean hasNext() {
        return internalIterator.hasNext();
    }
    @Override
    public Customer next() {
        return internalIterator.next();
    }
    @Override
    public void remove() {
        internalIterator.remove();
    }
}
```

Lab 6 b

- Write a filter iterator that can filter out customers based on their address. Using this filter iterator show the following customers:
 - All customers from Chicago
 - All customers whose zip code starts with “12”

```
System.out.println("-----All customers from Chicago");
Predicate<Customer> chicagoPredicate = c -> c.getAddress().getCity().equals("Chicago");
Iterator<Customer> chicagoIterator = customerCollection.filterIterator(chicagoPredicate);
while (chicagoIterator.hasNext()) {
    System.out.println(chicagoIterator.next());
}
```

```
System.out.println("-----All customers whose zip code starts with 12");
Predicate<Customer> zipcodePredicate = c -> c.getAddress().getZip().startsWith("12");
Iterator<Customer> zipcodeIterator = customerCollection.filterIterator(zipcodePredicate);
while (zipcodeIterator.hasNext()) {
    System.out.println(zipcodeIterator.next());
}
```

Lab 6 c



- Write an iterator that iterates only over the customers, but after every customer it skips the next customer. So the iterator returns first customer 1, then customer 3, then customer 5, etc.

Lab 4 c

```
public class SkipIterator implements Iterator<Customer> {
    private final List<Customer> list;
    private int position;

    public SkipIterator(List<Customer> list) {
        this.list = list;
        this.position = 0;
    }

    public Iterator<Customer> iterator() {
        return this;
    }

    @Override
    public boolean hasNext() {
        return position <= list.size();
    }

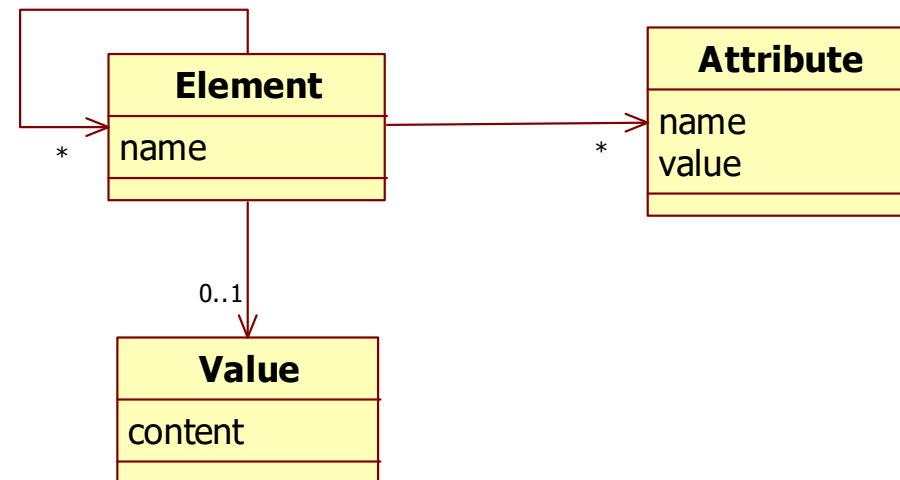
    @Override
    public Customer next() {
        Customer cust = list.get(position);
        position = position+2;
        return cust;
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Lab 6 d

- Suppose you have to write an XML parser. Draw the class diagram of the domain model for this XML parser without using the composite pattern.

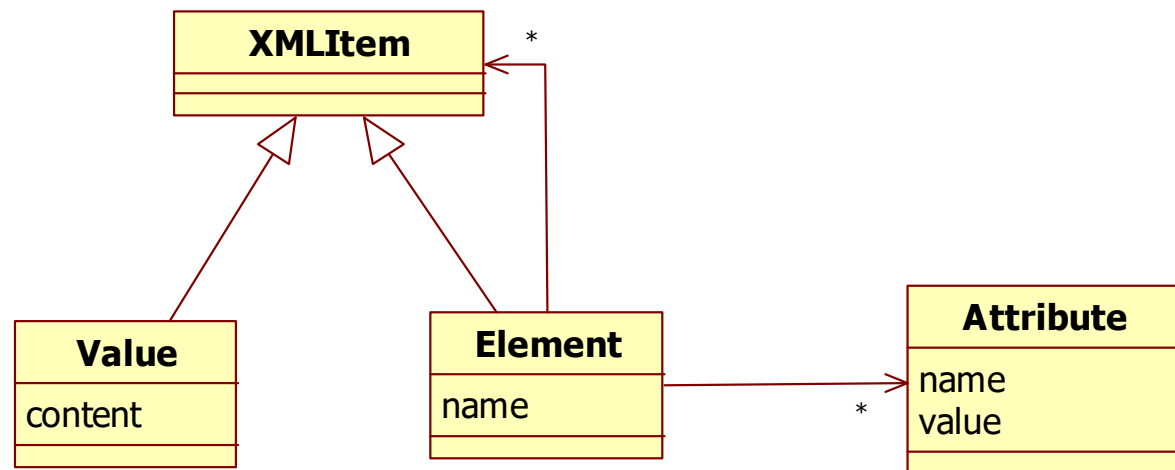
```
<person>  
  <name> Bart Simpson </name>  
  <tel type="mobile"> 02 – 444 7777 </tel>  
  <tel type="home"> 051 – 011 022 </tel>  
  <email> bart@tau.ac.il </email>  
</person>
```



Lab 6 e

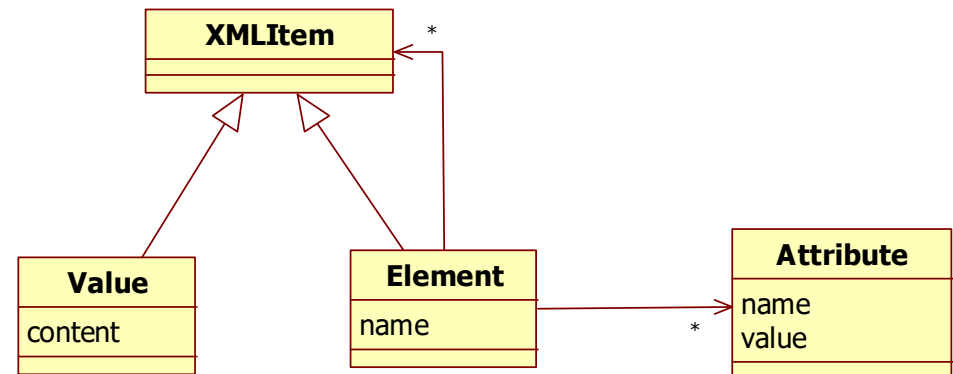
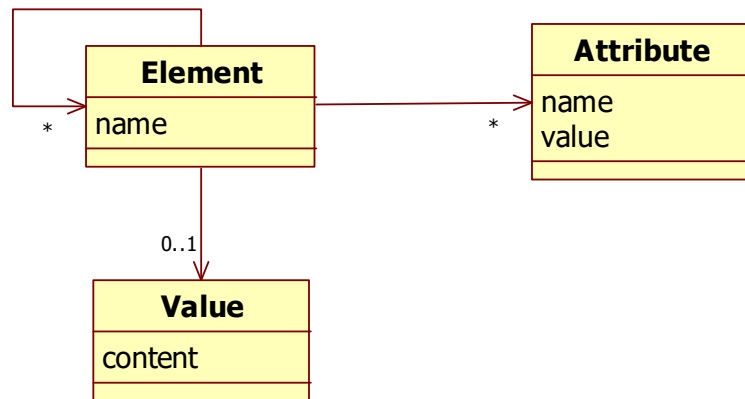
- Draw the class diagram of the domain model for this XML parser using the composite pattern

```
<person>  
  <name> Bart Simpson </name>  
  <tel type="mobile"> 02 – 444 7777 </tel>  
  <tel type="home"> 051 – 011 022 </tel>  
  <email> bart@tau.ac.il </email>  
</person>
```



Lab 6 f

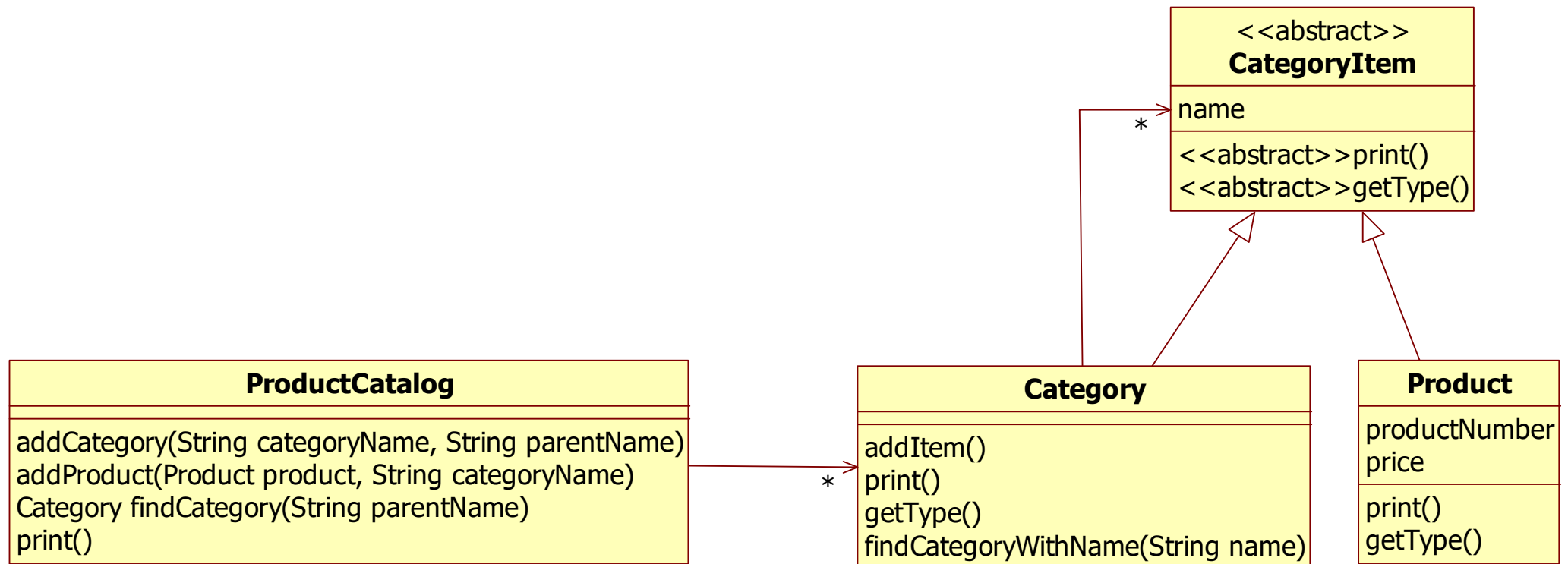
- Explain the advantage and/or disadvantage of the design of part c and part d.



Lab 6 g

- Suppose we have a webshop where the product catalog consists of categories and products. Design this with the composite pattern and implement it in Java. Write code to test your application, for example print out the whole product catalog with all categories and products

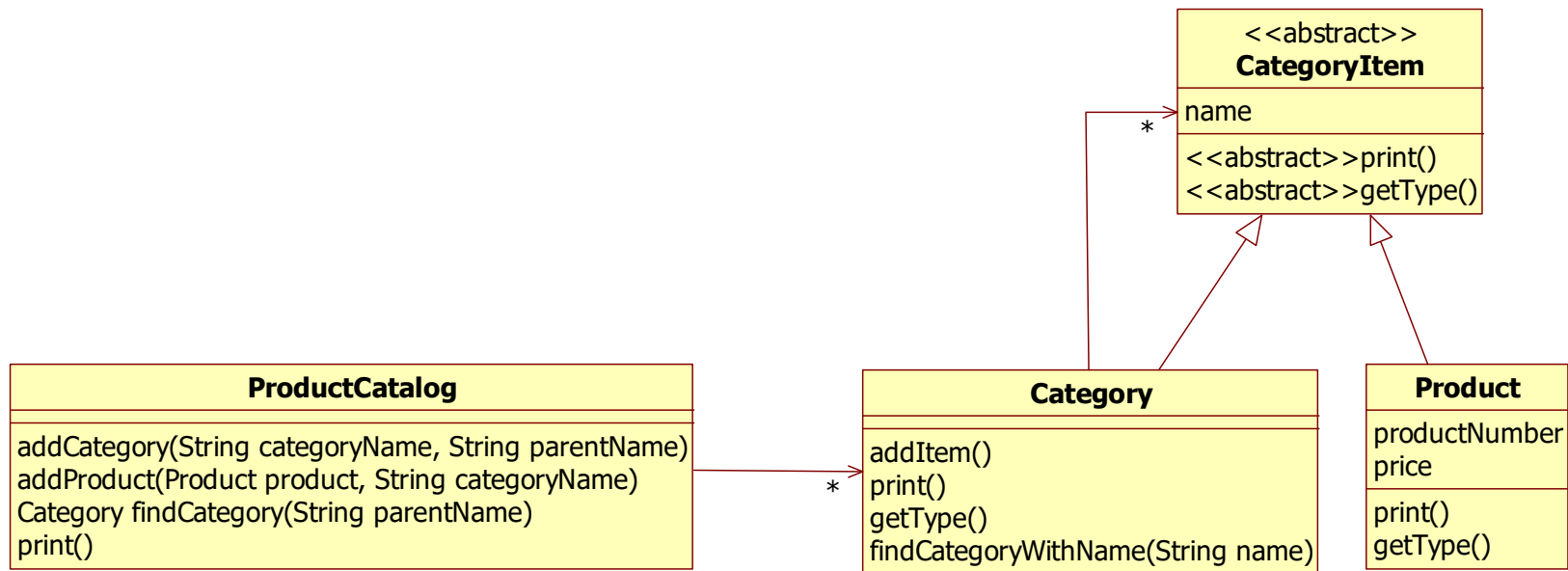
Lab 6 g



Lab 4 g

```
public abstract class CategoryItem {  
    private String name;  
  
    protected abstract void print();  
    protected abstract String getType();  
    ...  
}
```

```
public class Product extends CategoryItem {  
    private int productNumber;  
    private double price;  
  
    ...  
}
```



Lab 6 g: Category

```
public class Category extends CategoryItem{
    List<CategoryItem> items = new ArrayList<CategoryItem>();
```

```
@Override
```

```
protected void print() {
    System.out.println("Category [name=" + getName() + "]);
    for (CategoryItem item: items) {
        item.print();
    }
}
```

```
public Category(String name) {
    super(name);
}
```

```
public void addItem(CategoryItem item) {
    items.add(item);
}
```

```
public String getType() {
    return "Category";
}
```

```
public Category findCategoryWithName(String name) {
    Category result = null;
    if (name.equals(getName())) {
        return this;
    }
    else {
        for(CategoryItem subitem: items) {
            if (subitem.getType().equals("Category")) {
                result =
                    ((Category)subitem).findCategoryWithName(name);
                if (result != null)
                    return result;
            }
        }
    }
    return result;
}
```

Lab 6 g: ProductCatalog 1/2

```
public class ProductCatalog {  
  
    List<CategoryItem> catalog = new ArrayList<CategoryItem>();  
  
    public void addCategory(String categoryName, String parentName) {  
        Category category = new Category(categoryName);  
        if (parentName == null) {  
            catalog.add(category);  
        } else {  
            Category parentCategory = findCategory(parentName);  
            if (parentCategory != null) {  
                parentCategory.addItem(category);  
            }  
        }  
    }  
  
    public void addProduct(Product product, String categoryName) {  
        Category parentCategory = findCategory(categoryName);  
        if (parentCategory != null) {  
            parentCategory.addItem(product);  
        }  
    }  
}
```

Lab 6 g: ProductCatalog 2/2

```
private Category findCategory(String parentName) {
    Category result = null;
    for (CategoryItem item : catalog) {
        if (item.getName().equals(parentName)) {
            return (Category) item;
        }
        if (item.getType().equals("Category")) {
            result = ((Category) item).findCategoryWithName(parentName);
            if (result != null)
                return result;
        }
    }
    return result;
}

public void print() {
    for (CategoryItem item : catalog) {
        item.print();
    }
}
}
```