|  |  |
| --- | --- |
| Course | Advanced Software Design – CS525 |
| Assignment | Lab 4 |
| Week | 04 |
| Due | Mar 8, 2020 |
| Student name | Quan Hong Doan |
| Student ID | 986956 |

Online version:

<https://github.com/zithiat/asd/blob/master/labs%20%26%20assignments/Answers/Assignment_CS525_Week04_986956.docx>

**Problem 1**:

Problem 1 is the question a, b, c from the PDF file.

**Answer**:

The online code repository for customer changes is on my GitHub <https://github.com/zithiat/asd/tree/master/codes/code%20for%20labs/lab4/customers>

Question a – Sort ascending age

**AgeIterator.java**

**public** **class** AgeIterator **implements** Iterator<Customer> {

**private** List<Customer> list;

**private** Iterator<Customer> it;

**public** AgeIterator(List<Customer> list) {

**this**.list = list;

Collections.*sort*(list, Comparator.*comparingInt*(Customer::getAge));

**this**.it = **this**.list.iterator();

}

@Override

**public** **boolean** hasNext() {

**if** (it.hasNext())

**return** **true**;

**return** **false**;

}

@Override

**public** Customer next() {

**return** it.next();

}

}

Question b – City + Zip iterator

**FilterIterator.java**

**public** **class** FilterIterator **implements** Iterator<Customer> {

**private** List<Customer> list;

**private** Iterator<Customer> it;

**public** FilterIterator(List<Customer> list) {

**this**.list = list;

**this**.it = **this**.list.iterator();

}

@Override

**public** **boolean** hasNext() {

**if** (it.hasNext())

**return** **true**;

**return** **false**;

}

@Override

**public** Customer next() {

**return** it.next();

}

**public** **void** filterByCity(String city) {

**this**.list.stream().filter(e -> city.equals(e.getAddress().getCity())).collect(Collectors.*toList*()).forEach(System.***out***::println);

}

**public** **void** filterByZipcode(String zip) {

**this**.list.stream().filter(e -> e.getAddress().getZip().startsWith(zip)).collect(Collectors.*toList*()).forEach(System.***out***::println);

}

}

Question c – Filter the even customers, only show the odd customers.

**SkipIterator.java**

**public** **class** SkipIterator **implements** Iterator<Customer> {

**private** List<Customer> list;

**private** Iterator<Customer> it;

**public** SkipIterator(List<Customer> list) {

**this**.list = list;

**this**.it = **this**.list.iterator();

}

@Override

**public** **boolean** hasNext() {

**if** (it.hasNext())

**return** **true**;

**return** **false**;

}

@Override

**public** Customer next() {

**return** it.next();

}

**public** **void** skipIterate() {

**while**(it.hasNext()) {

System.***out***.println(it.next());

// manual move to the next item

// We can use Spliterator interface for this case

**if** (it.hasNext())

it.next();

}

}

}

**CustomerCollection.java**

We can add other functions for the Iterators, but I simplify adding to the existing print() method.

**public** **class** CustomerCollection {

List<Customer> customers = **new** ArrayList<Customer>();

**public** **void** add(Customer customer) {

customers.add(customer);

}

**public** **void** print() {

System.***out***.println("=================== Original print() ===================");

**for** (Customer customer : customers) {

System.***out***.println(customer);

}

System.***out***.println("=================== With AgeIterator ===================");

AgeIterator ai = **new** AgeIterator(customers);

**while**(ai.hasNext()) {

System.***out***.println(ai.next());

}

System.***out***.println("=================== With FilterIterator ===================");

FilterIterator fi = **new** FilterIterator(customers);

System.***out***.println("Filter ---- from Chicago");

fi.filterByCity("Chicago");

System.***out***.println("Filter ---- zipcode starts with 12");

fi.filterByZipcode("12");

System.***out***.println("=================== With SkipIterator ===================");

SkipIterator si = **new** SkipIterator(customers);

si.skipIterate();

}

}

Console logs:

=================== Original print() ===================

Customer [firstName=Frank, lastName=Brown, email=fbrown@gmail.com, phone=0643267843, age=34, address=Address [street=Mainstreet 10, city=Chicago, zip=59223]]

Customer [firstName=John, lastName=Johnson, email=jjohnson@hotmail.com, phone=0609834544, age=44, address=Address [street=Mainstreet 2, city=San Fransisco, zip=12225]]

Customer [firstName=Bill, lastName=Waters, email=bill34@gmail.com, phone=0632416786, age=82, address=Address [street=456 S 2th street, city=Chicago, zip=59223]]

Customer [firstName=Mary, lastName=Clinton, email=maclinton@gmail.com, phone=0655676689, age=64, address=Address [street=421 Jeffersen Stret, city=New York, zip=45231]]

Customer [firstName=Sue, lastName=Jackson, email=sue@yahoo.com, phone=0644332256, age=78, address=Address [street=101 South D street, city=New York, zip=45786]]

Customer [firstName=Ed, lastName=Trump, email=edwardtrump@yahoo.com, phone=063322667893, age=66, address=Address [street=34 North E street, city=Chicago, zip=59223]]

Customer [firstName=Bob, lastName=Brown, email=bobbrown@gmail.com, phone=063245633221, age=26, address=Address [street=Jeffersen Av 324, city=San Fransisco, zip=12897]]

Customer [firstName=Eric, lastName=Vanderberg, email=eberg@gmail.com, phone=0643215673, age=26, address=Address [street=305 South B street, city=Chicago, zip=59223]]

Customer [firstName=Joe, lastName=Johnson, email=jj@gmail.com, phone=0643256743, age=48, address=Address [street=453 N C Street, city=Chicago, zip=59223]]

Customer [firstName=Frank, lastName=Cohen, email=fcohen@gmail.com, phone=0643232178, age=53, address=Address [street=Jeffersen Av 3, city=New York, zip=54221]]

Customer [firstName=Eric, lastName=Johnson, email=ejohnson@hotmail.com, phone=0612342345, age=36, address=Address [street=1000 S 4th street, city=San Fransisco, zip=12544]]

=================== With AgeIterator ===================

Customer [firstName=Bob, lastName=Brown, email=bobbrown@gmail.com, phone=063245633221, age=26, address=Address [street=Jeffersen Av 324, city=San Fransisco, zip=12897]]

Customer [firstName=Eric, lastName=Vanderberg, email=eberg@gmail.com, phone=0643215673, age=26, address=Address [street=305 South B street, city=Chicago, zip=59223]]

Customer [firstName=Frank, lastName=Brown, email=fbrown@gmail.com, phone=0643267843, age=34, address=Address [street=Mainstreet 10, city=Chicago, zip=59223]]

Customer [firstName=Eric, lastName=Johnson, email=ejohnson@hotmail.com, phone=0612342345, age=36, address=Address [street=1000 S 4th street, city=San Fransisco, zip=12544]]

Customer [firstName=John, lastName=Johnson, email=jjohnson@hotmail.com, phone=0609834544, age=44, address=Address [street=Mainstreet 2, city=San Fransisco, zip=12225]]

Customer [firstName=Joe, lastName=Johnson, email=jj@gmail.com, phone=0643256743, age=48, address=Address [street=453 N C Street, city=Chicago, zip=59223]]

Customer [firstName=Frank, lastName=Cohen, email=fcohen@gmail.com, phone=0643232178, age=53, address=Address [street=Jeffersen Av 3, city=New York, zip=54221]]

Customer [firstName=Mary, lastName=Clinton, email=maclinton@gmail.com, phone=0655676689, age=64, address=Address [street=421 Jeffersen Stret, city=New York, zip=45231]]

Customer [firstName=Ed, lastName=Trump, email=edwardtrump@yahoo.com, phone=063322667893, age=66, address=Address [street=34 North E street, city=Chicago, zip=59223]]

Customer [firstName=Sue, lastName=Jackson, email=sue@yahoo.com, phone=0644332256, age=78, address=Address [street=101 South D street, city=New York, zip=45786]]

Customer [firstName=Bill, lastName=Waters, email=bill34@gmail.com, phone=0632416786, age=82, address=Address [street=456 S 2th street, city=Chicago, zip=59223]]

=================== With FilterIterator ===================

Filter ---- from Chicago

Customer [firstName=Eric, lastName=Vanderberg, email=eberg@gmail.com, phone=0643215673, age=26, address=Address [street=305 South B street, city=Chicago, zip=59223]]

Customer [firstName=Frank, lastName=Brown, email=fbrown@gmail.com, phone=0643267843, age=34, address=Address [street=Mainstreet 10, city=Chicago, zip=59223]]

Customer [firstName=Joe, lastName=Johnson, email=jj@gmail.com, phone=0643256743, age=48, address=Address [street=453 N C Street, city=Chicago, zip=59223]]

Customer [firstName=Ed, lastName=Trump, email=edwardtrump@yahoo.com, phone=063322667893, age=66, address=Address [street=34 North E street, city=Chicago, zip=59223]]

Customer [firstName=Bill, lastName=Waters, email=bill34@gmail.com, phone=0632416786, age=82, address=Address [street=456 S 2th street, city=Chicago, zip=59223]]

Filter ---- zipcode starts with 12

Customer [firstName=Bob, lastName=Brown, email=bobbrown@gmail.com, phone=063245633221, age=26, address=Address [street=Jeffersen Av 324, city=San Fransisco, zip=12897]]

Customer [firstName=Eric, lastName=Johnson, email=ejohnson@hotmail.com, phone=0612342345, age=36, address=Address [street=1000 S 4th street, city=San Fransisco, zip=12544]]

Customer [firstName=John, lastName=Johnson, email=jjohnson@hotmail.com, phone=0609834544, age=44, address=Address [street=Mainstreet 2, city=San Fransisco, zip=12225]]

=================== With SkipIterator ===================

Customer [firstName=Bob, lastName=Brown, email=bobbrown@gmail.com, phone=063245633221, age=26, address=Address [street=Jeffersen Av 324, city=San Fransisco, zip=12897]]

Customer [firstName=Frank, lastName=Brown, email=fbrown@gmail.com, phone=0643267843, age=34, address=Address [street=Mainstreet 10, city=Chicago, zip=59223]]

Customer [firstName=John, lastName=Johnson, email=jjohnson@hotmail.com, phone=0609834544, age=44, address=Address [street=Mainstreet 2, city=San Fransisco, zip=12225]]

Customer [firstName=Frank, lastName=Cohen, email=fcohen@gmail.com, phone=0643232178, age=53, address=Address [street=Jeffersen Av 3, city=New York, zip=54221]]

Customer [firstName=Ed, lastName=Trump, email=edwardtrump@yahoo.com, phone=063322667893, age=66, address=Address [street=34 North E street, city=Chicago, zip=59223]]

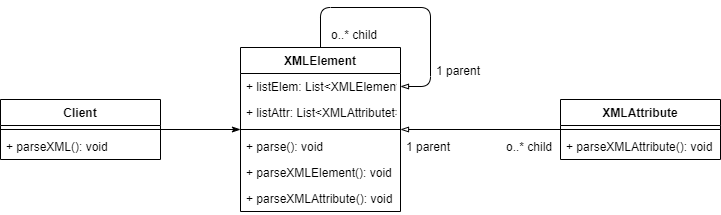
Customer [firstName=Bill, lastName=Waters, email=bill34@gmail.com, phone=0632416786, age=82, address=Address [street=456 S 2th street, city=Chicago, zip=59223]]

**Problem 2**:

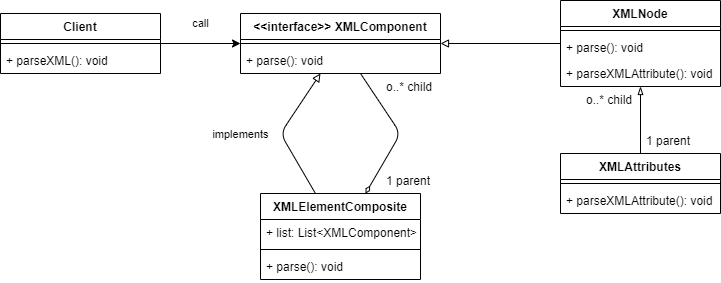
Problem 2 is the question d, e, f from the PDF file.

**Answer**:

Class diagram with non-composite pattern.



Class diagram with composite pattern.



As we know, this pattern is already used in many places within java itself like XML parsing and in some of the collection methods like: List#addAll(Collection), Set#addAll(Collection) etc.

Advantages:

* Clients use the component class interface to interact with objects in the composite structure
* Calling a composite forwards, the request to its child components

Disadvantages

* Leaf classes have to create some methods which has to empty in some cases
* Once tree structure is defined, the composite pattern makes the tree overly general

So when we use composite pattern:

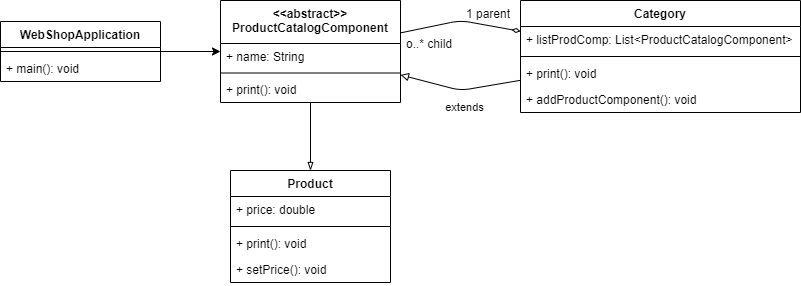
* When we need to treat all objects in the composite structure uniformly, like XLComponent.
* When the group of objects should behave as the single object, like our XMLElement, XMLNode

**Problem 3**:

Problem 3 is the question g from the PDF file.

**Answer**:

Class diagram of WebShop application



Implementation is on my GitHub <https://github.com/zithiat/asd/tree/master/codes/code%20for%20labs/lab4/webshop>

**WebShopApplication.java**

**package** webshop;

**public** **class** WebShopApplication {

**public** **static** **void** main(String[] args) {

Category allCats = **new** Category("Shop");

Category bookCat = **new** Category("Book");

Category groceryCat = **new** Category("Grocery");

Category scifiCat = **new** Category("Science Fiction");

Category advCat = **new** Category("Adventure");

Category milkCat = **new** Category("Milk products");

Category meatCat = **new** Category("Meat products");

Product lotr = **new** Product("Lord of the Rings");

lotr.setPrice(49.99);

Product journey = **new** Product("Journey to the center of the Earth");

journey.setPrice(21.82);

Product whipping = **new** Product("Whipping cream");

whipping.setPrice(4.59);

Product chicken = **new** Product("Chicken wings");

chicken.setPrice(5.79);

allCats.addProductComponent(bookCat);

allCats.addProductComponent(groceryCat);

bookCat.addProductComponent(scifiCat);

bookCat.addProductComponent(advCat);

scifiCat.addProductComponent(lotr);

advCat.addProductComponent(journey);

groceryCat.addProductComponent(milkCat);

groceryCat.addProductComponent(meatCat);

milkCat.addProductComponent(whipping);

meatCat.addProductComponent(chicken);

allCats.print();

}

}

**ProductCatalogComponent.java**

**package** webshop;

**public** **abstract** **class** ProductCatalogComponent {

**protected** String name;

**public** ProductCatalogComponent(String name) {

**this**.name = name;

}

**public** **abstract** **void** print();

}

**Category.java**

**package** webshop;

**import** java.util.ArrayList;

**import** java.util.Collection;

**public** **class** Category **extends** ProductCatalogComponent {

**protected** Collection<ProductCatalogComponent> listProducts = **new** ArrayList<ProductCatalogComponent>();

**public** Category(String name) {

**super**(name);

}

**public** **void** addProductComponent(ProductCatalogComponent product) {

listProducts.add(product);

}

@Override

**public** **void** print() {

System.***out***.println("Category: " + **this**.name);

**this**.listProducts.stream().forEach(p -> p.print());

}

}

**Product.java**

**package** webshop;

**public** **class** Product **extends** ProductCatalogComponent {

**protected** **double** price;

**public** Product(String name) {

**super**(name);

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

@Override

**public** **void** print() {

System.***out***.println("Product: " + **this**.name + ", price: " + getPrice());

}

}

**Console logs:**

Category: Shop

Category: Book

Category: Science Fiction

Product: Lord of the Rings, price: 49.99

Category: Adventure

Product: Journey to the center of the Earth, price: 21.82

Category: Grocery

Category: Milk products

Product: Whipping cream, price: 4.59

Category: Meat products

Product: Chicken wings, price: 5.79