МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа № 4

По дисциплине «Технологии программирования»

Выполнил студент группы №M3209
Бабурин Тимур

Проверил
Собенников Виктор Леонидович

САНКТ-ПЕТЕРБУРГ
2021

Упражнение 4-1.Классы-коллекции.

Цель упражнения: Изучить преимущества использования классов-коллекций, предоставляемых в стандартной поставке JDK.

Описание упражнения: В этом упражнении вы реализуете класс ItemCatalog, хранящий список товаров, продаваемых интернет-магазином. Список в дальнейшем будет использоваться для демонстрации товаров покупателям на сайте магазина.

1) Добавьте в проект класс ItemCatalog, в котором будет реализована логика хранения списка товаров.

1) Включите в класс ItemCatalog поля, определенные следующим образом:

```
private HashMap<Integer,GenericItem> catalog =
                        new HashMap<Integer,GenericItem>();
private ArrayList<GenericItem> ALCatalog =
                        new ArrayList<GenericItem>();
```

Примечание: Обе эти коллекции будут хранить один и тот же список товаров. Коллекция HashMap более оптимальна для последующего поиска товаров в каталоге, а коллекция ArrayList нужна для сравнения с ней.

2) Реализуйте в классе ItemCatalog следующие методы:

1) public void addItem(GenericItem item) : добавляет товар в каталог

```
public void addItem(GenericItem item) {
      catalog.put(item.ID, item); // Добавляем товар в
HashMap
      ALCatalog.add(item); // Добавляем тот же товар в
ArrayList
}
```

2) public void printItems() : распечатывает товары из каталога на экране. Распечатку следует производить с использованием метода toString класса GenericItem.

```
 public void printItems(){
 for(GenericItem i : ALCatalog){
      System.out.println(i);
  }
}
```

3) public GenericItem findItemByID(int id) : производит поиск в каталоге по переданному id товара. Поиск следует производит в коллекции catalog типа HashMap public GenericItem findItemByID(int id){

```
//Если нет такого ID, возвращаем пустое значение
      if(!catalog.containsKey(id)) {
            return null;
      } else{
            return catalog.get(id);
      }
}
```

4) `public GenericItem findItemByIDAL(int id)` : производит поиск в каталоге по переданному `id` товара. Поиск следует производит в коллекции `ALCatalog` типа `ArrayList`

```
public GenericItem findItemByIDAL(int id){
      for(GenericItem i : ALCatalog){
            if(i.ID==id) return i;
      }
      return null;
}
```

3) В методе `main` класса `Main` создайте новый экземпляр класса `ItemCatalog`. С помощью метода `addItem` добавьте в него несколько (порядка `10`) товаров.
4) С помощью приема из упр. 3-1 сравните скорость поиска по двум типам коллекций:

```
long begin = new Date().getTime();

for(int i=0; i<100000;i++)
cat.findItemByID(10);
long end = new Date().getTime();
System.out.println("In HashMap: "+(end-begin)); begin = new
Date().getTime();
for(int i=0; i<100000;i++)
   cat.findItemByIDAL(10);
end = new Date().getTime();
System.out.println("In ArrayList: "+(end-begin));
```

Примечание: Поскольку у нас нет возможности сформировать действительно большие списки товаров, приходится производить большое количество циклов поиска, чтобы разница во времени накапливалась и становилась ощутимой.

Артефакты выполнения упражнения:

```java
package ru.billing.stocklist;

import java.util.ArrayList;
import java.util.HashMap;

public class ItemCatalog {
    private HashMap<Integer,GenericItem> catalog = new HashMap<~>();
    private ArrayList<GenericItem> ALCatalog = new ArrayList<~>();

    public void addItem(GenericItem item) {
        catalog.put(item.getID(), item); // Добавляем товар в HashMap
        ALCatalog.add(item); // Добавляем тот же товар в ArrayList
    }

    public void printItems(){
        for(GenericItem i : ALCatalog){
            System.out.println(i);
        }
    }

    public GenericItem findItemByID(int id){
        if(!catalog.containsKey(id)) {
            return null;
        } else{
            return catalog.get(id);
        }
    }

    public GenericItem findItemByIDAL(int id){
        for(GenericItem i : ALCatalog){
            if(i.getID() == id) return i;
        }
        return null;
    }
}
```

```
In HashMap: 7
In ArrayList: 15
```

Упражнение4-2. Абстрактные классы и интерфейсы.

Цель упражнения: Изучить полезные свойства абстрактных классов и интерфейсов.

Описание упражнения: В этом упражнении вы добавите в проект интерфейс `CatalogLoader` с методом `load`. Объект, реализующий метод `load` будет способен загружать список товаров в указанный каталог (`ItemCatalog`). В дальнейшем в нашем проекте появятся несколько классов реализующих метод `load` и заполняющих каталог из различных источников.

5) Добавьте в проект новый интерфейс `CatalogLoader`.
6) Опишите в интерфейсе `CatalogLoader` следующий метод:

```
public void load(ItemCatalog cat);
```

7) Добавьте в проект класс `CatalogStubLoader`, реализующий интерфейс `CatalogLoader`.
8) В методе `load` класса `CatalogStubLoader` реализуйте «ручной» способ загрузки каталога с помощью явно созданных программистом объектов:

```
GenericItem item1 = new GenericItem("Sony
TV",23000,Category.GENERAL);
FoodItem item2 = new FoodItem("Bread",12,null,new
Date(),(short)10);
cat.addItem(item1);
cat.addItem(item2);
```

9) В методе `main` класса `Main` создайте новый фрагмент кода, загружающий товары в каталог с помощью объекта-загрузчика:

```
CatalogLoader loader = new CatalogStubLoader();
loader.load(cat);
```

10) Запустите программу и проверьте корректность загрузки списка товаров.

Артефакты выполнения упражнения:

```java
package ru.billing.client;

import ru.billing.stocklist.*;

public interface CatalogLoader {
    void load(ItemCatalog cat);
}
```

```java
package ru.billing.client;

import ru.billing.stocklist.*;
import java.util.Date;

public class CatalogStubLoader implements CatalogLoader {
    @Override
    public void load(ItemCatalog cat) {
        GenericItem item1 = new GenericItem( name: "Sony TV", price: 23000, Category.GENERAL);
        FoodItem item2 = new FoodItem( name: "Bread", price: 12, analog: null, new Date(),(short)10);
        cat.addItem(item1);
        cat.addItem(item2);
    }
}
```

```java
CatalogLoader loader = new CatalogStubLoader();
loader.load(myCatalog);
myCatalog.printItems();
```

```
ru.billing.stocklist.FoodItem{ID=0, name='Pepsi', price=50.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=5}
ru.billing.stocklist.FoodItem{ID=1, name='Coca-cola', price=55.0, analog=ru.billing.stocklist.FoodItem{ID=0, name='Pepsi', price=50.0
ru.billing.stocklist.FoodItem{ID=2, name='ChipsLub', price=65.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=3}
ru.billing.stocklist.FoodItem{ID=3, name='AmericanShips', price=80.0, analog=ru.billing.stocklist.FoodItem{ID=2, name='ChipsLub', pri
ru.billing.stocklist.FoodItem{ID=4, name='Apple from Russia', price=25.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expir
ru.billing.stocklist.FoodItem{ID=5, name='Apple from France', price=55.0, analog=ru.billing.stocklist.FoodItem{ID=4, name='Apple from
ru.billing.stocklist.FoodItem{ID=6, name='Bananas', price=30.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=2}
ru.billing.stocklist.FoodItem{ID=7, name='Snickers', price=35.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=3}
ru.billing.stocklist.FoodItem{ID=8, name='Twix', price=35.0, analog=ru.billing.stocklist.FoodItem{ID=7, name='Snickers', price=35.0,
ru.billing.stocklist.FoodItem{ID=9, name='Pasta', price=40.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=9}
ru.billing.stocklist.GenericItem{ID=0, name='Sony TV', price=23000.0, analog=null, itemCategory=GENERAL}
ru.billing.stocklist.FoodItem{ID=10, name='Bread', price=12.0, analog=null, itemCategory=GENERAL, dateOfIncome=null, expires=10}

Process finished with exit code 0
```

Упражнение 4-3. Пакеты, модификаторы доступа и инкапсуляция.

Цель упражнения: Изучить на практике использование механизма пакетов и принципа инкапсуляции

Описание упражнения: В этом упражнении вы создадите пакеты в рамках проекта `StockListProject` и распределите классы и интерфейсы проекта по пакетам. Также вы создадите инкапсулированные версии классов проекта.

11) Инкапсулируйте классы `GenericItem`, `FoodItem`, `TechnicalItem` и `ItemCatalog` своего проекта. Для этого объявите все их поля как `private`, а для доступа к ним создайте соответствующие методы `set…` и `get…`

   `Это лучше сделать при помощи eclipse – Source/Generate Setters and Getters.`

12) Создайте в проекте `StockListProject` пакеты со следующим содержанием:
    1) `ru.billing.client`
        1) Класс `Main`
        2) Интерфейс `CatalogLoader`
        3) Класс `CatalogStubLoader`
    2) `ru.billing.exceptions`
    3) `ru.billing.stocklist`
        1) Перечисление `Category`
        2) Класс `GenericItem`
        3) Класс `FoodItem`
        4) Класс `TechnicalItem`
        5) Класс `ItemCatalog`
    4) `ru.lanit.warehouse`
13) Исправьте ошибки, появившиеся в проекте из-за необходимости импорта классов из других пакетов.
14) Запустите программу, проверьте работоспособность проекта.

Артефакты выполнения упражнения:

```java
package ru.billing.stocklist;

public class GenericItem {
    private static int currentID;
    private int ID;
    private String name;
    private float price;
    private GenericItem analog;
    private Category itemCategory = Category.GENERAL;

    public static int getCurrentID() { return currentID; }

    public static void setCurrentID(int currentID) { GenericItem.currentID = currentID; }

    public int getID() {
        return ID;
    }

    public void setID(int ID) { this.ID = ID; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public float getPrice() { return price; }

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

    public GenericItem getAnalog() { return analog; }

    public void setAnalog(GenericItem analog) { this.analog = analog; }

    public Category getItemCategory() { return itemCategory; }

    public void setItemCategory(Category itemCategory) { this.itemCategory = itemCategory; }
```