Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение вычислительной техники

и автоматизированных систем»

**Отчёт**

по лабораторной работе № 12

по дисциплине ***«Объектно-ориентированное программирование»***

тема: «Использование контейнеров в Java. Java Collections Framework, JCF»

Название бригады: «**Coding new generations**»

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студенты группы 10701216  Ассанович Кирилл,  Медведев Роман |
| Преподаватель: |  | Иванченко Виктор Викторович |

2017-2018 учебный год

**ЛАБОРАТОРНАЯ РАБОТА № 09(10)**

Абстрактные классы и интерфейсы в Java. Абстрактные структуры данных: Стек и Очередь. Обобщённое программирование в Java

Цель работы:

Ознакомиться с содержимым и архитектурой библиотеки контейнеров JCF (пакет java.util), а также получить практические навыки использования интер- фейсов и классов данного фреймворка для решения задач, связанных с хра- нением и обработкой наборов данных.

**Основное задание:**

Модифицировать программу, разработанную в предыдущей лабораторной работе, следующим образом:

− во всех классах заменить работу с контейнерами пользовательских типов и(или) массивами на работу с соответствующими контейнерами из библиотеки JCF;

− добавить возможность сортировки бизнес-объектов по нескольким критериям с использованием интерфейса Comparable для указания естественной сортировки и Comparator для реализации остальной логики сортировки бизнес-объектов (пример реализации шаблона Стратегия – The Strategy Pattern);

− выбор объекта, который содержит логику сортировки, должен выбираться из контейнера типа Map.

− добавить возможность поиска бизнес-объектов по разным параметрам;

− добавить возможность использования итераторов в пользовательских контейнерных классах (т.е. необходимо описать класс-контейнер таким образом, чтобы объект данного класса можно было использовать с моди- фицированным в JDK5.0 циклом for).

Требования к выполнению:

1) При разработке программ рекомендуется использовать шаблон MVC и SOLID (GRASP) принципы, особенно принцип единственной ответственности (Single Responsibility Principle), т.к. классы рекомендуется проектироваться и реализовываться таким образом, чтобы они были слабо зависимы от других классов.

2) При выполнении каждого задания необходимо по максимуму пытаться раз- рабатывать универсальный код.

3) Программа должна обязательно быть снабжена комментариями на англий- ском языке, в которых необходимо указать краткое предназначение про- граммы, номер лабораторной работы и её название, версию программы, ФИО разработчика, номер группы и дату разработки. Исходный текст классов и де- монстрационной программы рекомендуется также снабжать комментариями.

4) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом.

5) Обязательно покрыть тестами новое задание, а также новый функционал, который был добавлен при рефакторинге предыдущего проекта. Запустить тесты и проверить проекты на их работоспособность.

6) Модифицируйте свою UML-диаграмму классов с учётом внесённых изменения в архитектуру своего приложения.

7) Упаковать всю программу в jar-архив, который должен быть запускаемым.

8) При разработке программ придерживайтесь соглашений по написанию кода на JAVA (Java Code-Convention).

Результаты выполнения основного задания:

Результаты работы программы:

*ПРИЛОЖЕНИЕ А*

Листинг исходного кода на JAVA:

SmartTree.java

package by.bntu.fitr.povt.cng.model.entity.container;

import by.bntu.fitr.povt.cng.model.collections.deprecated.CustomCollections;

import by.bntu.fitr.povt.cng.model.entity.abstracts.Toy;

public class SmartTree {

private CustomCollections <Toy> toysMass = null;

private static SmartTree tree = null;

public SmartTree() {

}

public SmartTree(CustomCollections collections){

toysMass = collections;

}

public static SmartTree getInstance(CustomCollections collections){

return new SmartTree(collections);

}

public Toy[] getToysMass() {

return (Toy[]) toysMass.CollectionToArray();

}

public int getSize(){

return toysMass.size();

}

public Toy getToy(int index){

return toysMass.get(index);

}

//priceTree

public void changeToy(Toy newToy, int index){

toysMass.setElement(index,newToy);

}

public void setToy(Toy newToy, int index){

toysMass.setElement(index,newToy);

}

public double getPrice(int index){

return toysMass.get(index).getPrice();

}

public void add(Toy...newToys) {

for (Toy newToy : newToys){

toysMass.add(newToy);

}

}

@Override

public int hashCode() {

int HASHCONST = 31;

int result = 1;

result = HASHCONST \* result + getSize();

for (Object object : toysMass.CollectionToArray()){

result = HASHCONST \* result + object.hashCode();

}

return result;

}

@Override

public boolean equals(Object obj) {

return (obj != null && this.hashCode() == obj.hashCode() && obj.getClass() == this.getClass());

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < toysMass.size(); i++) {

sb.append(toysMass.get(i)).append("\n");

}

return sb.toString();

}

}

JSFTrees.java

package by.bntu.fitr.povt.cng.model.entity.container.JSFContainer;

import by.bntu.fitr.povt.cng.model.entity.abstracts.Toy;

public interface JSFTrees extends Iterable<Toy> {

int getSizeOfTree();

void remove(int index);

void changeToy(Toy toy, int index);

void add(Toy toy);

Toy[] getMasOfObjects();

Toy getToy(int index);

}

JSFLinkedTree.java

package by.bntu.fitr.povt.cng.model.entity.container.JSFContainer;

import by.bntu.fitr.povt.cng.model.entity.abstracts.Toy;

import java.util.Arrays;

import java.util.Iterator;

import java.util.LinkedList;

public class JSFLinkedTree implements JSFTrees{

private LinkedList<Toy> toys = new LinkedList<>();

public JSFLinkedTree(Toy...ts) {

toys.addAll(Arrays.asList(ts));

}

public JSFLinkedTree(){

}

public Toy[] getMasOfObjects(){

return (Toy[]) toys.toArray();

}

public Toy getToy(int index){

return toys.get(index);

}

public void changeToy(Toy toy, int index){

toys.remove(index);

toys.add(index, toy);

}

public void remove(int index){

toys.remove(index);

}

public void add(Toy toy){

toys.add(toy);

}

public int getSizeOfTree(){

return toys.size();

}

@Override

public int hashCode() {

int result = 0;

for (Toy toy : toys){

result += toy.hashCode();

}

return result;

}

@Override

public boolean equals(Object obj) {

if (obj == null && !(obj instanceof JSFArrayTree)){

return false;

}

return obj == this && this.hashCode() == obj.hashCode() && obj.getClass() == this.getClass();

}

@Override

public Iterator<Toy> iterator() {

return toys.iterator();

}

@Override

public String toString() {

return toys.toString();

}

}

JSFArrayTree.java

package by.bntu.fitr.povt.cng.model.entity.container.JSFContainer;

import by.bntu.fitr.povt.cng.model.entity.abstracts.Toy;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

public class JSFArrayTree implements JSFTrees{

private ArrayList<Toy> toys = new ArrayList<>();

public JSFArrayTree(Toy...ts) {

toys.addAll(Arrays.asList(ts));

}

public JSFArrayTree(){

}

public Toy[] getMasOfObjects(){

return (Toy[]) toys.toArray();

}

public Toy getToy(int index){

return toys.get(index);

}

public void changeToy(Toy toy, int index){

toys.remove(index);

toys.add(index, toy);

}

public void remove(int index){

toys.remove(index);

}

public void add(Toy toy){

toys.add(toy);

}

public int getSizeOfTree(){

return toys.size();

}

@Override

public int hashCode() {

int result = 0;

for (Toy toy : toys){

result += toy.hashCode();

}

return result;

}

@Override

public boolean equals(Object obj) {

if (obj == null && !(obj instanceof JSFArrayTree)){

return false;

}

return obj == this && this.hashCode() == obj.hashCode() && obj.getClass() == this.getClass();

}

@Override

public String toString() {

return toys.toString();

}

@Override

public Iterator<Toy> iterator() {

return toys.iterator();

}

}

Sort.java

package by.bntu.fitr.povt.cng.model.logic;

import by.bntu.fitr.povt.cng.model.entity.container.JSFContainer.JSFTrees;

import by.bntu.fitr.povt.cng.model.entity.container.Tree;

import by.bntu.fitr.povt.cng.model.entity.abstracts.Toy;

import java.util.Comparator;

public class Sort {

@Deprecated

public static void insert(Tree tree) {

Toy temp;

int j;

for (int i = 0; i < tree.getSize() - 1; i++) {

if (tree.getToy(i).getPrice() > tree.getToy(i + 1).getPrice()) {

temp = tree.getToy(i + 1);

tree.changeToy(tree.getToy(i), i + 1);

j = i;

while (j > 0 && temp.getPrice() < tree.getToy(j - 1).getPrice()) {

tree.changeToy(tree.getToy(j - 1), j);

j--;

}

tree.changeToy(temp, j);

}

}

}

@Deprecated

public static void bubble(Tree tree) {

boolean sorted = true;

for (int i = tree.getSize(); i > 0; i--) {

for (int j = 0; j < i - 1; j++) {

if (tree.getToy(j).getPrice() > tree.getToy(j + 1).getPrice()) {

Toy tmp = tree.getToy(j);

tree.changeToy(tree.getToy(j + 1), j);

tree.changeToy(tmp, j + 1);

sorted = false;

}

if (sorted) {

break;

}

}

}

}

}

ComparableSort.java

import java.util.\*;

public class ComparableSort {

public static void insert(JSFTrees tree, Comparator comparator) {

Toy temp;

int j;

for (int i = 0; i < tree.getSizeOfTree() - 1; i++) {

if (comparator.compare(tree.getToy(i), tree.getToy(i + 1)) > 0){//if (tree.getToy(i).getPrice() > tree.getToy(i + 1).getPrice()) {

temp = tree.getToy(i + 1);

tree.changeToy(tree.getToy(i), i + 1);

j = i;

while (j > 0 && comparator.compare(temp, tree.getToy(j - 1)) < 0) {

tree.changeToy(tree.getToy(j - 1), j);

j--;

}

tree.changeToy(temp, j);

}

}

}

public static void bubble(JSFTrees tree, Comparator comparator) {

boolean sorted = true;

for (int i = tree.getSizeOfTree(); i > 0; i--) {

for (int j = 0; j < i - 1; j++) {

if (comparator.compare(tree.getToy(j), tree.getToy(j + 1)) > 0){

Toy tmp = tree.getToy(j);

tree.changeToy(tree.getToy(j + 1), j);

tree.changeToy(tmp, j + 1);

sorted = false;

}

if (sorted){

break;

}

}

}

}

}