Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 4**

по дисциплине

‘Программирование’

Вариант №0.3

*Выполнил:*

Студент группы P3109

Mallaev Sabur N.

*Преподаватель:*

Шешуков Дмитрий

Михайлович



Санкт-Петербург, 2021

Задание

* Доработанная модель должна соответствовать принципам SOLID.
* Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).
* В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().
* Программа должна содержать как минимум один перечисляемый тип (enum).
* В программе должны быть реализованы 2 собственных класса исключений (checked и unchecked), а также обработка исключений этих классов.
* В программу необходимо добавить использование локальных, анонимных и вложенных классов (static и non-static).

Текст

И вот 5 января в 7 часов 15 минут утра начался наш полет на север в самолете, который вел пилот Мактай. Кроме него в самолете находились еще десять человек, семь собак, сани, горючее, запас продовольствия, а также прочие необходимые вещи, в том числе и рация. Погода стояла безветренная, небо чистое, температура для этих мест не слишком низкая, так что особых трудностей не предвиделось. Мы были уверены, что с помощью указанных Лейком координат легко отыщем лагерь. Но дурные предчувствия нас не покидали: что обнаружим мы у цели? Ведь радио по-прежнему молчало, никто не отвечал на наши постоянные вызовы, Каждый момент этого четырехчасового полета навсегда врезался в мою память: он изменил всю мою жизнь. Именно тогда, в 54-летнем возрасте, я навсегда утратил мир и покой, присущий человеку с нормальным рассудком и живущему в согласии с природой и ее законами. С этого времени мы -- все десятеро, но особенно мы с Денфортом -- неотрывно следили за фантомами, таящимися в глубинах этого чудовищного искаженного мира, и ничто не заставит нас позабыть его. Мы не стали бы рассказывать, будь это возможно, о наших переживаниях всему человечеству. Газеты напечатали бюллетени, посланные нами с борта самолета, в которых сообщалось о нашем беспосадочном перелете; о встрече в верхних слоях атмосферы с предательскими порывами ветра; об увиденной с высоты шахте, которую Лейк пробурил три дня назад на полпути к горам, а также о загадочных снежных цилиндрах, замеченных ранее Амундсеном и Бэрдом,-- ветер гнал их по бескрайней ледяной равнине. Затем наступил момент, когда мы не могли адекватно передавать охватившие нас чувства, а потом пришел и такой, когда мы стали строго контролировать свои слова, введя своего рода цензуру. Первым завидел впереди зубчатую линию таинственных кратеров и вершин матрос Ларсен. Он так завопил, что все бросились к иллюминаторам. Несмотря на значительную скорость самолета, горы, казалось, совсем не приближались; это говорило о том, что они бесконечно далеки и видны только из-за своей невероятной, непостижимой высоты. И, все же постепенно они мрачно вырастали перед нами, застилая западную часть неба, и мы уже могли рассмотреть голые, лишенные растительности и незащищенные от ветра темные вершины. Нас пронизывало непередаваемое ощущение чуда, переживаемое при виде этих залитых розоватым антарктическим светом громад на фоне облаков ледяной пыли, переливающейся всеми цветами радуги. Эта картина рождала чувство близости к некоей глубочайшей тайне, которая могла вдруг раскрыться перед нами. За безжизненными жуткими хребтами, казалось, таились пугающие пучины подсознательного, некие бездны, где смешались время, пространство и другие, неведомые человечеству измерения. Эти горы представлялись мне вместилищем зла -- хребтами безумия, дальние склоны которых обрывались, уходя в пропасть, за которой ничего не было. Полупрозрачная дымка облаков, окутывающая вершины, как бы намекала на начинающиеся за ними бескрайние просторы, на затаенный и непостижимый мир вечной Смерти -- далекий, пустынный и скорбный.

Диаграмма классов реализованной объектной модели

Исходный код программы

Можно прочитать на GitHub: <https://github.com/suhankins/ITMO-Labs/tree/main/Programming/lab3/src>

Main.java

package planeflight;

import planeflight.entities.\*;

import planeflight.environment.\*;

import planeflight.inventory.\*;

/\*\*

\* Lab3 Programming. Based on the description

\* you can find in text.docx file in the parent folder.

\*

\* @author  Dimitri Sukhankin

\* @since   2021-12-18

\*/

public class Main {

    public static void main(String[] args) {

        Plane plane = new Plane();

        Human human = new Human("Main Character", 54);

        try {

            human.addToInventory(new Paper());

            plane.addToInventory(human);

            System.out.println("I'm " + human.toString());

            plane.addToInventory(new Human("McTighe", 43));

            plane.addToInventory(new Human("Danforth", 56));

            plane.addToInventory(new Human("Lake", 40));

            plane.addToInventory(new Human("Baird", 58));

            plane.addToInventory(new Human("Larsen", 39));

            for (int i = 0; i < 7; i++) plane.addToInventory(new Dog());

            plane.addToInventory(new Skies());

            plane.addToInventory(new Fuel());

            plane.addToInventory(new Radio());

        } catch(Exception e) {

            System.out.println(e.getMessage());

        }

        plane.describe();

        while (plane.update()) {}

    }

}

Intentory

Intentory.java

package planeflight.inventory;

import planeflight.entities.\*;

public interface Inventory {

    /\*\*

    \* Clears every item in the inventory

    \*/

    public void clearInventory();

    /\*\*

    \* Adds a specified entity to the inventory

    \* @param item Entity you want to add to the inventory

    \*/

    public void addToInventory(Entity item) throws inventoryInvalidItemException;

    /\*\*

    \* Returns entity stored at a specified position in the inventory

    \* @param itemPosition position of the required entity in the inventory

    \* @return entity that is stored at the requested position

    \* If provided position is higher, than the length of the inventory, last filled inventory slot will be returned

    \*/

    public Entity getFromInventory(int itemPosition);

    /\*\*

    \* Removes entity stored at a specified position in the inventory

    \* @param itemPosition position of the required entity in the inventory.

    \* If provided position is higher, than the length of the inventory, last filled inventory slot will be removed

    \*/

    public void removeFromInventory(int itemPosition);

    /\*\*

    \* Returns the whole inventory as an array

    \* @return the whole inventory as an array

    \*/

    public Entity[] getInventory();

    /\*\*

    \* Returns the position of an entity with a name that matches given one

    \* @param name name of the item you are looking for

    \* @return position in the inventory of the requested item

    \*/

    public int hasItem(String name);

}

invetoryFullException.java

package planeflight.inventory;

public class inventoryFullException extends RuntimeException {

    public inventoryFullException(String errorMessage) {

        super(errorMessage);

    }

}

intentoryInvalidItemException.java

package planeflight.inventory;

public class inventoryInvalidItemException extends Exception {

    public inventoryInvalidItemException(String errorMessage) {

        super(errorMessage);

    }

}

Environment

Location.java

package planeflight.environment;

public enum Location {

    NOTHING,

    MOUNTAIN,

    SNOW\_PILLAR,

    MINE;

}

Month.java

package planeflight.environment;

public enum Month {

    JANUARY(0),

    FEBRUARY(1),

    MARCH(2),

    APRIL(3),

    MAY(4),

    JUNE(5),

    JULY(6),

    AUGUST(7),

    SEPTEMBER(8),

    OCTOBER(9),

    NOVEMBER(10),

    DECEMBER(11);

    public final int number;

    private Month(int number) {

        this.number = number;

    }

    public int getNumber() {

        return this.number;

    }

    public static Month nameOfNumber(int number) {

        for (Month m : values()) {

            if (m.number == number) {

                return m;

            }

        }

        return null;

    }

}

Weather.java

package planeflight.environment;

public enum Weather {

    SUNNY,

    WINDY,

    RAINY;

}

World.java

package planeflight.environment;

public final class World {

    public static float temperature = 17.4f;

    public static int month = Month.JANUARY.getNumber();

    public static int day = 5;

    /\*\*

    \* Time, defined in seconds since midnight.

    \* Default value: 7 hours, 15 minutes

    \*/

    public static int time = (7 \* 60 + 15) \* 60;

    public static Weather weather = Weather.SUNNY;

    private static final int SECONDS\_IN\_DAY = 24 \* 60 \* 60;

    private static final int DAYS\_IN\_MONTH = 30;

    /\*\*

     \* Updates time

     \* @param seconds number of seconds to append to the clock

     \*/

    public static void updateTime(int seconds) {

        time += seconds;

        if (time >= SECONDS\_IN\_DAY) {

            day += time / SECONDS\_IN\_DAY;

            time = time % SECONDS\_IN\_DAY;

        }

        if (day >= DAYS\_IN\_MONTH) {

            month += day / DAYS\_IN\_MONTH;

            day = day % DAYS\_IN\_MONTH;

        }

        if (month >= 12) {

            month = month % 12;

        }

    }

    /\*\*

     \* Describes the environment

     \*/

    public static void describe() {

        String result = "It was ";

        //Month and day

        result += Month.nameOfNumber(month).name() + " " + day;

        //Time

        result += ", " + zeropad(time / (60 \* 60), 2) + ":" + zeropad((time % (60 \* 60)) / 60, 2) + ":" + zeropad(time % 60, 2) + ". ";

        //Weater and temperature

        result += "Weather was " + weather + ", temperature was " + temperature + " degrees.";

        System.out.println(result);

    }

    /\*\*

     \*

     \* @param number

     \* @param length

     \* @return a zeropadded number

     \*/

    public static String zeropad(int number, int length) {

        String toReturn = Integer.toString(number);

        for (int i = toReturn.length(); i < length; i++) {

            toReturn = "0" + toReturn;

        }

        return toReturn;

    }

}

Entities

React.java

package planeflight.entities;

import planeflight.environment.\*;

public interface React {

    /\*\*

     \* Used by plane to make entities inside react to the environment

     \* @param location Location entity is reacting to

     \*/

    public void react(Location location);

}

Entity.java

package planeflight.entities;

import planeflight.environment.\*;

/\*\*

\* Base class upon which every thing that

\* PHYSICALLY EXISTS is based on.

\*/

public abstract class Entity implements React {

    private String name;

    public void setName(String newName) {

        this.name = newName;

    }

    public String getName() {

        return this.name;

    }

    public void react(Location location) {

    }

    @Override

    public String toString() {

        return getName();

    }

    @Override

    public int hashCode() {

        return 1 + name.hashCode();

    }

    @Override

    public boolean equals(Object o){

        if (o == this) {

            return true;

        }

        if (!(o instanceof Entity)) {

            return false;

        }

        Entity s = (Entity)o;

        return s.hashCode() == this.hashCode();

    }

}

EntityWithInventory.java

package planeflight.entities;

import planeflight.inventory.\*;

/\*\*

\* Everything that has an inventory

\*/

public abstract class EntityWithInventory extends Entity implements Inventory {

    private Entity[] inventory;

    public EntityWithInventory(int inventorySize) {

        inventory = new Entity[inventorySize];

    }

    public void clearInventory() {

        for (int i = 0; i < inventory.length; i++) {

            inventory[i] = null;

        }

    }

    public void addToInventory(Entity item) throws inventoryInvalidItemException {

        if (item.getName().equals("News") | item.getName().equals("Plane")) {

            throw new inventoryInvalidItemException("Entity types 'News' and 'Plane' cannot be stored in an inventory!");

        }

        for (int i = 0; i < inventory.length; i++) {

            //The position is empty, we can put our new item there

            if (inventory[i] == null) {

                inventory[i] = item;

                return;

            }

        }

        throw new inventoryFullException("Inventory is full, can't add more items!");

    }

    public Entity getFromInventory(int itemPosition) {

        if (itemPosition >= inventory.length) {

            for (int i = inventory.length - 1; i >= 0; i--) {

                if (inventory[i] != null) {

                    return inventory[i];

                }

            }

            return null;

        } else {

            return inventory[itemPosition];

        }

    }

    public void removeFromInventory(int itemPosition) {

        int removedItemLocation = itemPosition;

        if (itemPosition >= inventory.length) {

            for (int i = inventory.length - 1; i >= 0; i--) {

                if (inventory[i] != null) {

                    removedItemLocation = i;

                    break;

                }

            }

            //Nothing to remove

            return;

        }

        inventory[removedItemLocation] = null;

        //Moving everything higher than the removed item position lower.

        //i.e.

        // Before                         | After

        //                                |

        // Item0 <--- this one is removed | Item1

        // Item1                          | Item2

        // Item2                          | null

        for (int i = removedItemLocation + 1; i < inventory.length; i++) {

            inventory[i - 1] = inventory[i];

            inventory[i] = null;

        }

    }

    public Entity[] getInventory() {

        return inventory;

    }

    public int hasItem(String name) {

        for (int i = 0; i < inventory.length; i++) {

            if (inventory[i] != null) {

                if (inventory[i].getName().equals(name)) {

                    return i;

                }

            }

        }

        return -1;

    }

    @Override

    public String toString() {

        return getName() + ", carries: " + inventory.toString();

    }

    @Override

    public int hashCode() {

        return 1 + getName().hashCode() + inventory.hashCode();

     }

}

Human.java

package planeflight.entities;

import planeflight.environment.\*;

/\*\*

\* Class for humans, duh

\*/

public class Human extends EntityWithInventory {

    public int age;

    public Location[] memories = new Location[3];

    @Override

    public void react(Location location) {

        switch(location) {

            case NOTHING:

                if (Math.random() < 0.25) System.out.println("Passenger " + getName() + " has a bad feeling about this flight.");

                break;

            default:

                if (Math.random() < 0.75) System.out.println("Passenger " + getName() + " is amazed at sight of " + location);

                addMemory(location);

                break;

        }

    }

    /\*\*

     \* Adds a location to person's memory. Only used when sending letters.

     \* @param location Location to add to human's memory.

     \*/

    public void addMemory(Location location) {

        for (int i = 0; i < memories.length; i++) {

            if (memories[i] == null) {

                memories[i] = location;

                //This is the last memory

                if (i == memories.length - 1 && hasItem("Paper") != -1) {

                    removeFromInventory(hasItem("Paper"));

                    class Mail {

                        Location[] locations;

                        public Mail(Location[] locations) {

                            this.locations = locations;

                        }

                        @Override

                        public String toString() {

                            String result = "";

                            for (int i = 0; i < locations.length; i++) {

                                result += locations[i].name().replace("\_", " ");

                                if (i != locations.length - 1) {

                                    result += ", ";

                                }

                            }

                            return result;

                        }

                    }

                    Mail letter = new Mail(memories);

                    News.receive(letter.toString());

                }

                return;

            }

        }

        return;

    }

    /\*\*

     \* @param name Human's name

     \* @param age Human's age

     \*/

    public Human(String name, int age) {

        super(3);

        this.setName(name);

        this.age = age;

    }

    /\*\*

     \* Creates a human named "Unnamed", age 18

     \*/

    public Human() {

        this("Unnamed", 18);

    }

    @Override

    public String toString() {

        return getName() + ", " + age + ", memories right now: " + memories[0] + ", " + memories[1] + ", " + memories[2];

    }

    @Override

    public int hashCode() {

        return 1 + getName().hashCode() + age \* 2 + memories.hashCode() \* 4;

    }

}

Plane.java

package planeflight.entities;

import java.util.spi.ToolProvider;

import planeflight.environment.\*;

/\*\*

\* Plane

\*/

public class Plane extends EntityWithInventory {

    public Plane() {

        super(70);

        this.setName("Plane");

        World.describe();

    }

    class Journey {

        private static final Location[] LOCATIONS={Location.NOTHING,Location.SNOW\_PILLAR,Location.NOTHING,

                                                   Location.NOTHING, Location.NOTHING, Location.MINE,

                                                   Location.MOUNTAIN, Location.NOTHING};

        private static int position = -1;

        public static int getPosition() {

            return position;

        }

        public static Location getNextLocation() {

            position++;

            if (position >= LOCATIONS.length) {

                return null;

            }

            return LOCATIONS[position];

        }

    }

    /\*\*

     \* Update plane's position and cargo's reaction to the environment.

     \* @return Is flight over or not

     \*/

    public boolean update() {

        Entity[] cargo = getInventory();

        Location currentLocation = Journey.getNextLocation();

        if (currentLocation == null) {

            System.out.println("Flight is finished!");

            World.describe();

            return false;

        }

        for (int i = 0; i < cargo.length; i++) {

            if (cargo[i] == null) {

                break;

            }

            cargo[i].react(currentLocation);

        }

        World.updateTime(30 \* 60 + (int)(Math.random() \* 60.0));

        return true;

    }

    /\*\*

     \* Mainly prints all the cargo

     \*/

    public void describe() {

        String toReturn = "On the plane there was ";

        String[] checked = new String[70];

        int[] checkedCount = new int[70];

        Entity[] cargo = getInventory();

        for (int i = 0; i < cargo.length; i++) {

            if (cargo[i] == null) break;

            String currentName = cargo[i].getName();

            for (int j = 0; j < checked.length; j++) {

                if (checked[j] == null) {

                    checked[j] = currentName;

                    checkedCount[j] = 1;

                    break;

                } else if (checked[j].equals(currentName)) {

                    checkedCount[j]++;

                    break;

                }

            }

        }

        for (int i = 0; i < checked.length; i++) {

            if (checked[i] == null) {

                toReturn += "and that's about it.";

                break;

            } else {

                if (checkedCount[i] > 1) {

                    toReturn += checkedCount[i] + " " + checked[i] + "s, ";

                } else {

                    toReturn += checked[i] + ", ";

                }

            }

        }

        System.out.println(toReturn);

    }

    @Override

    public String toString() {

        return getName() + " at position " + Journey.getPosition() + ", cargo: " + getInventory().toString();

    }

    @Override

    public int hashCode() {

        return 1 + getName().hashCode() + 2 \* Journey.getPosition() + 4 \* getInventory().hashCode();

    }

}

Dog.java

package planeflight.entities;

import planeflight.environment.\*;

/\*\*

\* Dog. Absolutely useless.

\*/

public class Dog extends Entity {

    public Dog() {

        setName("Dog");

    }

    @Override

    public void react(Location location) {

        if (Math.random() < 0.1) {

            System.out.println("Bark!");

        }

    }

}

Fuel.java

package planeflight.entities;

/\*\*

\* Absolutely useless.

\*/

public class Fuel extends Entity {

    public Fuel() {

        setName("Fuel");

    }

}

News.java

package planeflight.entities;

/\*\*

\* News office that prints bulletins

\*/

public final class News extends Entity {

    {

        setName("News");

    }

    /\*\*

     \* Prints contents of the string

     \* @param letter string you want to be printed

     \*/

    public static void receive(String letter) {

        System.out.println("News released bulletins about our crew flying by " + letter);

    }

}

Paper.java

package planeflight.entities;

/\*\*

\* Paper. Pretty much useless, but using it,

\* humans can write mail

\*/

public class Paper extends Entity {

    public Paper() {

        setName("Paper");

    }

}

Radio.java

package planeflight.entities;

import planeflight.environment.\*;

/\*\*

\* Radio. Doesn't respond.

\*/

public class Radio extends Entity {

    public Radio() {

        setName("Radio");

    }

    @Override

    public void react(Location location) {

        if (Math.random() < 0.25) {

            System.out.println("Radio doesn't respond");

        }

    }

}

Skies.java

package planeflight.entities;

/\*\*

\* Absolutely useless.

\*/

public class Skies extends Entity {

    public Skies() {

        setName("Skies");

    }

}

Результат работы программы

Многие события в программе случайны, результат выполнения может отличаться.

It was JANUARY 5, 07:15:00. Weather was SUNNY, temperature was 17.4 degrees.

I'm Main Character, 54, memories right now: null, null, null

On the plane there was Main Character, McTighe, Danforth, Lake, Baird, Larsen, 7 Dogs, Skies, Fuel, Radio, and that's about it.

Passenger Lake has a bad feeling about this flight.

Passenger Baird has a bad feeling about this flight.

Bark!

Passenger Main Character is amazed at sight of SNOW\_PILLAR

Passenger McTighe is amazed at sight of SNOW\_PILLAR

Passenger Danforth is amazed at sight of SNOW\_PILLAR

Passenger Lake is amazed at sight of SNOW\_PILLAR

Passenger Baird is amazed at sight of SNOW\_PILLAR

Passenger McTighe has a bad feeling about this flight.

Passenger Lake has a bad feeling about this flight.

Passenger Larsen has a bad feeling about this flight.

Passenger Danforth has a bad feeling about this flight.

Bark!

Bark!

Passenger Larsen has a bad feeling about this flight.

Passenger Main Character is amazed at sight of MINE

Passenger McTighe is amazed at sight of MINE

Passenger Danforth is amazed at sight of MINE

Passenger Lake is amazed at sight of MINE

Passenger Baird is amazed at sight of MINE

Bark!

Passenger Main Character is amazed at sight of MOUNTAIN

News released bulletins about our crew flying by SNOW PILLAR, MINE, MOUNTAIN

Passenger McTighe is amazed at sight of MOUNTAIN

Passenger Danforth is amazed at sight of MOUNTAIN

Passenger Lake is amazed at sight of MOUNTAIN

Passenger Baird is amazed at sight of MOUNTAIN

Bark!

Bark!

Flight is finished!

It was JANUARY 5, 11:19:04. Weather was SUNNY, temperature was 17.4 degrees.

Вывод

Вложенные и локальные классы могут быть очень полезны в сложных программах, но я думаю моя программа еще не настолько сложная, чтобы от них была реальная польза. Жаль, что выдача вариантов на сайте сломалась и я не мог просто взять свою прошлую лабу и улучшить.