Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

по дисциплине

‘ПРОГРАММИРОВАНИЕ’

Вариант №5219

*Выполнил:*

Студент группы P3109

Бугаев Сергей

Юрьевич

*Преподаватель:*

Письмак Алексей Евгеньевич



Санкт-Петербург, 2021

Оглавление

[Диаграмма классов 3](#__RefHeading___Toc897_570131292)

[Описание предметной области 4](#__RefHeading___Toc899_570131292)

[Исходный код 4](#__RefHeading___Toc901_570131292)

[Package run 4](#__RefHeading___Toc903_570131292)

[Story.java 4](#__RefHeading___Toc905_570131292)

[Package util 5](#__RefHeading___Toc907_570131292)

[ThingInterface.java 5](#__RefHeading___Toc909_570131292)

[PersonInterface.java 5](#__RefHeading___Toc911_570131292)

[WeatherInterface.java 5](#__RefHeading___Toc913_570131292)

[WeatherEnum.java 5](#__RefHeading___Toc915_570131292)

[WeatherAbstract.java 6](#__RefHeading___Toc917_570131292)

[Package core 6](#__RefHeading___Toc919_570131292)

[Weather.java 6](#__RefHeading___Toc921_570131292)

[Transport.java 9](#__RefHeading___Toc923_570131292)

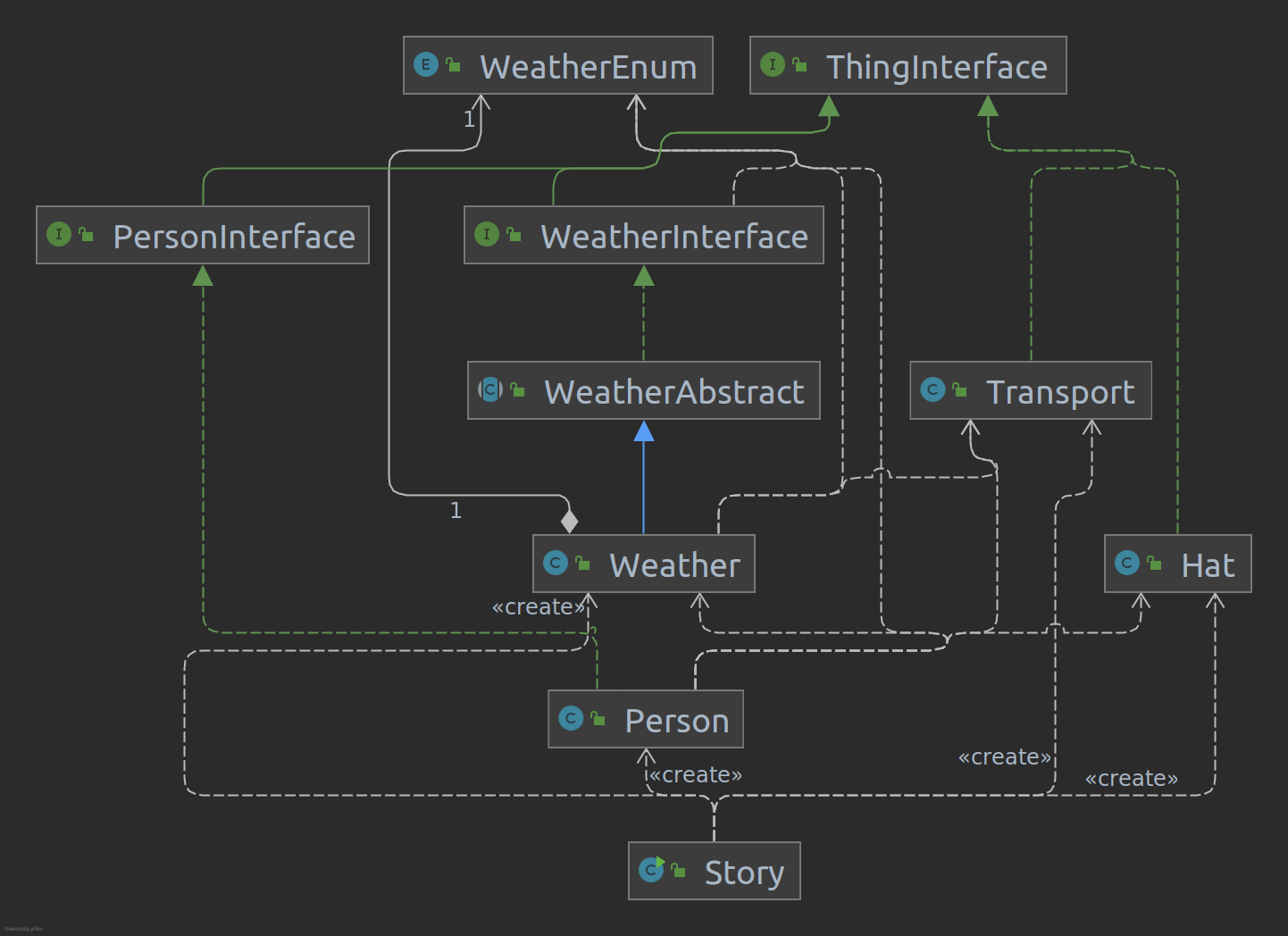
[Hat.java 11](#__RefHeading___Toc925_570131292)

[Person.java 12](#__RefHeading___Toc927_570131292)

[Результат работы 17](#__RefHeading___Toc929_570131292)

[Вывод 17](#__RefHeading___Toc931_570131292)

# Диаграмма классов



# Описание предметной области

Незнайка выглянул из корзины и увидел внизу облака, которые закрывали землю. Но Незнайка и этому не поверил. Он сидел на своем месте и крепко держал руками на голове шляпу. Он думал, что шляпа может свалиться с него, раз он вверх ногами сидит. Ветер быстро гнал шар над облаками, но скоро все заметили, что шар стал опускаться.

# Исходный код

## Package run

### Story.java

package run;

import core.\*;

public class Story {

public static void main(String[] args) {

Person Dunno = new Person();

Transport aTransport = new Transport();

Weather aWeather = new Weather();

Dunno.lookDown(aWeather, aTransport);

Hat aHat = new Hat();

Dunno.hold(aHat);

Dunno.thinkAbout(aHat);

aWeather.moveTransport(aTransport);

try {

Thread.sleep(5000);

} catch (InterruptedException ex) {

Thread.currentThread().interrupt();

}

aTransport.moveDown();

}

}

## Package util

### ThingInterface.java

package util;

public interface ThingInterface {

String getName();

}

### PersonInterface.java

package util;

public interface PersonInterface extends ThingInterface {

boolean isSitting();

boolean isUpsideDown();

boolean isBelieving();

}

### WeatherInterface.java

package util;

public interface WeatherInterface extends ThingInterface {

WeatherEnum getWeather();

boolean isWindy();

void setWind(boolean windy);

}

### WeatherEnum.java

package util;

public enum WeatherEnum {

CLOUDY, RAINY, SUNNY, SNOWY

}

### WeatherAbstract.java

package util;

public abstract class WeatherAbstract implements WeatherInterface {

private boolean windy;

public WeatherAbstract(boolean windy) {

this.windy = windy;

}

@Override

public boolean isWindy() {

return windy;

}

@Override

public void setWind(boolean windy) {

this.windy = windy;

}

}

## Package core

### Weather.java

package core;

import util.WeatherAbstract;

import util.WeatherEnum;

public class Weather extends WeatherAbstract {

private String name;

private final WeatherEnum TYPE = WeatherEnum.CLOUDY;

public Weather() {

super(true);

name = "Ветер";

joinStory();

}

public Weather(String name) {

super(true);

this.name = name;

joinStory();

}

public Weather(String name, boolean windy) {

super(windy);

this.name = name;

joinStory();

}

private void joinStory() {

if (isWindy()) {

System.out.println("Ветренная погода '" + name + "' началась.");

} else {

System.out.println("Погода '" + name + "' началась.");

}

}

public void moveTransport(Transport obj) {

if (isWindy()) {

System.out.println("Погода " + name + " несет транспорт '" + obj.getName() + "'.");

obj.move();

}

}

@Override

public String getName() {

return name;

}

@Override

public WeatherEnum getWeather() {

return TYPE;

}

@Override

public String toString() {

if (isWindy()) {

return "Ветренная погода '" + name + "'";

} else {

return "Погода '" + name + "'";

}

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj instanceof Weather) {

return (name.equals(((Weather) obj).getName()) && isWindy() == ((Weather) obj).isWindy());

} else {

return false;

}

}

@Override

public int hashCode() {

if (isWindy()) {

return name.hashCode() + name.length();

} else {

return name.hashCode();

}

}

}

### Transport.java

package core;

import util.ThingInterface;

public class Transport implements ThingInterface {

private String name;

private boolean moving;

private boolean flying;

public Transport() {

name = "Шар";

moving = false;

flying = true;

joinStory();

}

public Transport(String name, boolean flying) {

this.name = name;

moving = false;

this.flying = flying;

joinStory();

}

private void joinStory() {

System.out.println("Транспорт '" + name + "' возник.");

}

public void move() {

if (!moving) {

moving = true;

System.out.println("Транспорт '" + name + "' начал быстро двигаться.");

} else {

System.out.println("Транспорт '" + name + "' уже и так двигается.");

}

}

public boolean isMoving() {

return moving;

}

public boolean isFlying() {

return flying;

}

public void moveDown() {

System.out.println("Все заметили, что '" + name + "' начал опускаться.");

}

@Override

public String getName() {

return name;

}

@Override

public String toString() {

return "Транспорт '" + name + "'";

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj instanceof Transport) {

return name.equals(((Transport) obj).name) && moving == ((Transport) obj).isMoving();

}

return false;

}

@Override

public int hashCode() {

if (moving) {

return name.hashCode() + name.length();

}

return name.hashCode();

}

}

### Hat.java

package core;

import util.ThingInterface;

public class Hat implements ThingInterface {

private String name;

public Hat() {

name = "Шляпа";

joinStory();

}

public Hat(String name) {

this.name = name;

joinStory();

}

private void joinStory() {

System.out.println("Головной убор '" + name + "' возник.");

}

@Override

public String getName() {

return name;

}

}

### Person.java

package core;

import util.PersonInterface;

import util.WeatherEnum;

public class Person implements PersonInterface {

private String name;

private boolean believes;

private boolean sitting;

private boolean upsidedown;

public Person() {

name = "Незнайка";

believes = false;

sitting = true;

upsidedown = true;

joinStory();

}

public Person(String name) {

this.name = name;

believes = false;

sitting = true;

upsidedown = true;

joinStory();

}

public Person(String name, boolean believes, boolean sitting, boolean upsidedown) {

this.name = name;

this.believes = believes;

this.sitting = sitting;

this.upsidedown = upsidedown;

joinStory();

}

private void joinStory() {

System.out.println("Человек по имени '" + name + "' присоединился к истории.");

}

public void lookDown(Weather weather, Transport transport) {

if (transport.isFlying()) {

if (weather.getWeather().equals(WeatherEnum.CLOUDY)) {

System.out.println("Человек по имени '" + name + "' смотрит вниз из " + transport

+ " и видит облака, закрывающие землю.");

believe();

} else {

System.out.println("Человек по имени '" + name + "' смотрит вниз из " + transport + " и видит землю.");

}

} else {

System.out.println("Человек по имени '" + name + "' смотрит вниз из " + transport

+ " и не видит ничего, так как транспорт не летит.");

}

}

public void hold(Hat hat) {

if (sitting) {

System.out.println(

name + " сидит на своем месте и крепко держит руками головной убор " + hat.getName() + ".");

} else {

System.out.println(name + " стоит и крепко держит руками головной убор " + hat.getName() + ".");

}

}

public void thinkAbout(Hat hat) {

if (upsidedown) {

System.out.println(name + " думает, что головной убор " + hat.getName()

+ " может свалиться с него, ведь он сидит вверх ногами.");

} else {

System.out.println(

name + " ни о чем не думает и не понимает, зачем держит головной убор " + hat.getName() + ".");

}

}

private void believe() {

if (believes) {

System.out.println(name + " этому верит.");

} else {

System.out.println(name + " этому не верит.");

}

}

@Override

public boolean isSitting() {

return sitting;

}

@Override

public boolean isBelieving() {

return believes;

}

@Override

public boolean isUpsideDown() {

return upsidedown;

}

@Override

public String getName() {

return name;

}

@Override

public String toString() {

return "Человек '" + name + "'";

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj instanceof Person) {

return (name.equals(((Person) obj).getName()) && isBelieving() == ((Person) obj).isBelieving()

&& isSitting() == ((Person) obj).isSitting() && isUpsideDown() == ((Person) obj).isUpsideDown());

} else {

return false;

}

}

@Override

public int hashCode() {

if (isBelieving()) {

return name.hashCode() + name.length();

} else {

return name.hashCode();

}

}

}

# Результат работы

Человек по имени 'Незнайка' присоединился к истории.

Транспорт 'Шар' возник.

Ветренная погода 'Ветер' началась.

Человек по имени 'Незнайка' смотрит вниз из Транспорт 'Шар' и видит облака, закрывающие землю.

Незнайка этому не верит.

Головной убор 'Шляпа' возник.

Незнайка сидит на своем месте и крепко держит руками головной убор Шляпа.

Незнайка думает, что головной убор Шляпа может свалиться с него, ведь он сидит вверх ногами.

Погода Ветер несет транспорт 'Шар'.

Транспорт 'Шар' начал быстро двигаться.

Все заметили, что 'Шар' начал опускаться.

# Вывод

Во время выполнения данной лабораторной работы я научился применять принципы SOLID на практике, изучил понятия абстрактного класса, интерфейса и перечисления, а так же столкнулся с основными плюсом и минусом SOLID: очень сложно начать проект без предварительной разработки архитектуры и представления необходимых зависимостей, однако гораздо легче расширять и дополнять готовый проект, в котором используются принципы SOLID.