Санкт-Петербургский

Национальный исследовательский университет

информационных технологий,

механики и оптики

(НИУ ИТМО)

Факультет программной инженерии и компьютерной техники

**Дисциплина** Программирование

**Отчёт** по Лабораторной работе № 3

**Вариант** 712531

**Преподаватель** Пашнин А.Д.

Выполнил

Студент группы P3113

Бритков А.А.

« 22 » октября 2020 г.

Санкт-Петербург 2020

**Описание предметной области, по которой должна быть построена объектная модель:**

"А может быть, она осталась в дупле?" -- чуть не закричал Скуперфильд. Он уже хотел потребовать, чтоб остановили поезд, так как решил ехать обратно на поиски трости, но сообразил, что это обойдется ему намного дороже, чем купить новую трость. Поэтому он снова положил голову на подушку и попытался заснуть.

**Программа должна удовлетворять следующим требованиям:**

1. Доработанная модель должна соответствовать [принципам SOLID](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)).
2. Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).
3. В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().
4. Программа должна содержать как минимум один перечисляемый тип (enum).

**Порядок выполнения работы:**

1. Доработать объектную модель приложения.
2. Перерисовать диаграмму классов в соответствии с внесёнными в модель изменениями.
3. Согласовать с преподавателем изменения, внесённые в модель.
4. Модифицировать программу в соответствии с внесёнными в модель изменениями.

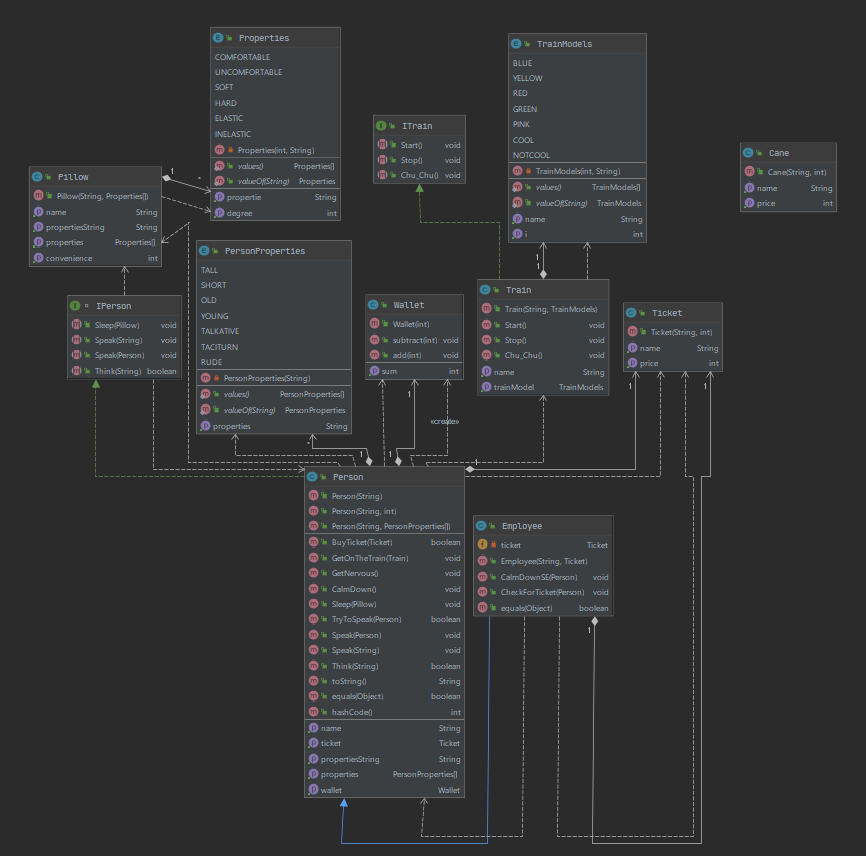
**Отчёт по работе должен содержать:**

1. Текст задания.
2. Диаграмма классов объектной модели.
3. Исходный код программы.
4. Результат работы программы.
5. Выводы по работе.

**Вопросы к защите лабораторной работы:**

1. Принципы объектно-ориентированного программирования SOLID и STUPID.
2. Класс Object. Реализация его методов по умолчанию.
3. Особенности реализации наследования в Java. Простое и множественное наследование.
4. Понятие абстрактного класса. Модификатор abstract.
5. Понятие интерфейса. Реализация интерфейсов в Java, методы по умолчанию. Отличия от абстрактных классов.
6. Перечисляемый тип данных (enum) в Java. Особенности реализации и использования.
7. Методы и поля с модификаторами static и final.
8. Перегрузка и переопределение методов. Коварианты возвращаемых типов данных.
9. Элементы функционального программирования в синтаксисе Java. Функциональные интерфейсы, лямбда-выражения. Ссылки на методы.

Диаграмма классов:



Исходный код программы:

public class Main {  
  
 public static void main(String[] args) {  
 Train train = new Train("поезд", TrainModels.*RED*);  
 Ticket ticket = new Ticket("билет", train.getTrainModel().getI());  
 Properties[] properties\_array = { Properties.*UNCOMFORTABLE*, Properties.*HARD* };  
 Pillow pillow = new Pillow("подушка", properties\_array);  
 Cane cane = new Cane("трость", 20);  
 Person person = new Person("Скуперфильд", 1000);  
 if (person.BuyTicket(ticket)){  
 return;  
 }  
 person.GetOnTheTrain(train);  
 Person employee = new Employee("проводник", ticket);  
 employee.CheckForTicket(person);  
 PersonProperties[] passengerProperties = {PersonProperties.*RUDE*, PersonProperties.*TACITURN*};  
 Person passenger = new Person("Сосед", passengerProperties);  
 if (person.TryToSpeak(passenger)){  
 person.Speak(passenger);  
 }  
 if (person.Think("Не забыл ли я свою трость в том дупле?")){  
 person.GetNervous();  
 employee.CalmDownSE(person);  
 person.CalmDown();  
 person.Sleep(pillow);  
 }  
 else{  
 person.Sleep(pillow);  
 }  
 }  
}  
  
  
  
  
  
public interface ITrain {  
 void Start();  
 void Stop();  
 void Chu\_Chu();  
}  
  
  
  
  
  
interface IPerson {  
 void Sleep(Pillow a);  
 void Speak(String topic);  
 void Speak(Person a);  
 boolean Think(String topic);   
 void CalmDownSE(Person a);  
 void CheckForTicket(Person a);

}  
  
  
  
  
import java.util.Objects;  
  
public class Person implements IPerson {  
 private String name = null;  
 private PersonProperties[] properties = {null};  
 private Wallet wallet = new Wallet(0);  
 private Ticket ticket = null;  
 public Person(String name){  
 this.name = name;  
 }  
 public Person(String name, int wallet){  
 this.name = name;  
 this.wallet.add(wallet);  
 }  
 public Person(String name, PersonProperties[] properties){  
 this.name = name;  
 this.properties = properties;  
 }  
  
 public Ticket getTicket() {  
 return ticket;  
 }  
  
 public Wallet getWallet() {  
 return wallet;  
 }  
  
 public PersonProperties[] getProperties(){  
 return properties;  
 }  
 public String getPropertiesString() {  
 String a = "";  
 for(int i = 0; i < properties.length; i++){  
 a += (i == properties.length - 1)?(properties[i].getProperties()):(properties[i].getProperties() + ", ");  
 }  
 return a;  
 }  
 public String getName(){  
 return name;  
 }  
 public boolean BuyTicket(Ticket a){  
 if (wallet.getSum() < a.getPrice()){  
 System.*out*.println("Объект " + name + " не может позволить себе купить билет стоимостью " + a.getPrice());  
 return true;  
 }  
 wallet.subtract(a.getPrice());  
 System.*out*.println("\*Объект " + name + " покупает билеты по цене " + a.getPrice() + "\*");  
 System.*out*.println("\*Общая сумма на руках у объекта " + name + " теперь составляет " + wallet.getSum() + '\*');  
 ticket = a;  
 return false;  
 }  
 public void GetOnTheTrain(Train a){  
 System.*out*.println("\*Объект " + name + " садится на поезд " + a.getTrainModel().getName() + '\*');  
 }  
 public void GetNervous(){  
 System.*out*.println("\*Объект " + name + " начинает нервничать\*");  
 }  
 public void CalmDown(){  
 System.*out*.println("\*Объект " + name + " успокаивается\*");  
 }  
 public void Sleep(Pillow a){  
 System.*out*.println("\*Объект " + name + " ложится на подушку с свойствами:" + a.getPropertiesString() + " - и пытаяется заснуть\*");  
 if (a.getConvenience() > 50) {  
 System.*out*.println("\*Объект " + name + " наелся и спит\*");  
 }  
 else {  
 System.*out*.println("\*Объект " + name + " не может заснуть\*");  
 }  
 }  
 public boolean TryToSpeak(Person a){  
 System.*out*.println("\*Объект " + name + " пытается завести разговор с объектом " + a.getName() + ", имеющим свойства:" + a.getPropertiesString() + '\*');  
 boolean flag = true;  
 for (PersonProperties b : a.getProperties()){  
 if (b.getProperties() == PersonProperties.*RUDE*.getProperties()){  
 flag = false;  
 }  
 if (b.getProperties() == PersonProperties.*TACITURN*.getProperties() && Math.*random*() <= 0.5){  
 flag = false;  
 }  
 }  
 if (flag){  
 System.*out*.println("\*Объект " + name + " успешно завёл разговор с объектом " + a.getName() + '\*');  
 }  
 else{  
 System.*out*.println("\*Объект " + name + " не смог завести разговор с объектом " + a.getName() + '\*');  
 }  
 return flag;  
 }  
 public void Speak(Person a){  
 System.*out*.println("\*Объект " + name + " разговаривает с объектом " + a.getName() + '\*');  
 }  
  
 public void Speak(String topic){  
 System.*out*.println(name + ": " + topic + '.');  
 }  
 public boolean Think(String topic){  
 System.*out*.println("\*Объект " + name + " рассуждает над тем, что " + topic + '\*');  
 if (Math.*random*() <= 0.5){  
 System.*out*.println("\*Объект " + name + " понял, что он прав\*");  
 return true;  
 }  
 System.*out*.println("\*Объект " + name + " так ни к чему и не пришёл\*");  
 return false;  
 }  
 @Override  
 public String toString(){  
 return getClass() + "{name='" + name + "', properties=" + getPropertiesString() + ", wallet=" + getWallet().getSum() + ", ticket=" + getTicket().getName() +"}";  
 }  
  
 @Override  
 public boolean equals(Object a){  
 if (this == a) {  
 return true;  
 }  
 if (!(a instanceof Person)){  
 return false;  
 }  
 Person person = (Person)a;  
 return (getName() == person.getName()) && (getTicket() == person.getTicket()) && (getWallet() == person.getWallet()) && (getPropertiesString() == person.getPropertiesString());  
 }  
 @Override  
 public int hashCode(){  
 return Objects.hash(getName(), getPropertiesString(), getWallet(), getTicket());  
 }  
}  
  
  
  
  
  
public class Employee extends Person {  
 private Ticket ticket;  
 public Employee(String name, Ticket ticket){  
 super(name);  
 this.ticket = ticket;  
 }  
  
 public void CalmDownSE(Person a){  
 System.*out*.println("\*Объект " + getName() + " успокаивает объект " + a.getName() + '\*');  
 }  
 public void CheckForTicket(Person a){  
 System.*out*.println("\*Объект " + getName() + " проверяет объекта " + a.getName() + " на наличие билета\*");  
 if (a.getTicket() == null){  
 System.*out*.println("\*Объект " + getName() + " просит объект " + a.getName() + " заплатить за билет\*");  
 if (a.BuyTicket(ticket)) {  
 System.*out*.println("\*Объект " + getName() + " высаживает объект " + a.getName() + " с поезда\*");  
 }  
 }  
 else{  
 System.*out*.println("\*Объект " + a.getName() + " имеет при себе билет\*");  
 }  
 }  
 @Override  
 public boolean equals(Object a){  
 if (this == a) {  
 return true;  
 }  
 if (!(a instanceof Employee)){  
 return false;  
 }  
 Employee person = (Employee)a;  
 return (getName() == person.getName()) && (getTicket() == person.getTicket()) && (getWallet() == person.getWallet()) && (getPropertiesString() == person.getPropertiesString());  
 }  
}  
  
  
  
  
  
public enum Properties {  
 *COMFORTABLE*(50, "Удобный"),  
 *UNCOMFORTABLE*(10, "Неудобный"),  
 *SOFT*(20, "Мягкий"),  
 *HARD*(10, "Жёсткий"),  
 *ELASTIC*(30, "Упругий"),  
 *INELASTIC*(10, "Неупругий");  
 private int degree;  
 private String propertie;  
 Properties(int degree, String propertie){  
 this.degree = degree;  
 this.propertie = propertie;  
 }  
  
 public int getDegree() {  
 return degree;  
 }  
  
 public String getPropertie() {  
 return propertie;  
 }  
}  
  
  
  
  
public enum PersonProperties {  
 *TALL*("Высокий"),  
 *SHORT*("Низкий"),  
 *OLD*("Старый"),  
 *YOUNG*("Молодой"),  
 *TALKATIVE*("Разговорчивый"),  
 *TACITURN*("Неразговорчивый"),  
 *RUDE*("Грубый");  
 private String properties;  
 PersonProperties(String properties){  
 this.properties = properties;  
 }  
  
 public String getProperties() {  
 return properties;  
 }  
}  
  
  
  
  
public enum TrainModels {  
 *BLUE*(120, "Blue"),  
 *YELLOW*(110, "Yellow"),  
 *RED*(100, "Red"),  
 *GREEN*(130, "Green"),  
 *PINK*(140, "Pink"),  
 *COOL*(2281337, "Cool"),  
 *NOTCOOL*(10, "NotCool");  
 private int i;  
 private String name;  
  
 TrainModels(int i, String name){  
 this.i = i;  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getI() {  
 return i;  
 }  
}  
  
  
  
  
public class Pillow {  
 private Properties[] properties;  
 private String name;  
 private int convenience;  
 public Pillow(String name, Properties[] properties){  
 this.name = name;  
 this.properties = properties;  
 for (int i = 0; i < properties.length; i++){  
 this.convenience += properties[i].getDegree();  
 }  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getConvenience() {  
 return convenience;  
 }  
  
 public Properties[] getProperties() {  
 return properties;  
 }  
  
 public String getPropertiesString() {  
 String a = "";  
 for (int i = 0; i < properties.length; i++){  
 a += (i == properties.length - 1)?(properties[i].getPropertie()):(properties[i].getPropertie() + ", ");  
 }  
 return a;  
 }  
}  
  
  
  
  
public class Train implements ITrain {  
 private String name;  
 private TrainModels TrainModel;  
 public Train(String name, TrainModels TrainModel){  
 this.name = name;  
 this.TrainModel = TrainModel;  
 }  
 public String getName(){  
 return name;  
 }  
 public TrainModels getTrainModel(){  
 return TrainModel;  
 }  
 public void Start(){  
 System.*out*.println("\*Поезд поехал\*");  
 }  
 public void Stop(){  
 System.*out*.println("\*Поезд остановился\*");  
 }  
 public void Chu\_Chu(){  
 System.*out*.println("\*Чух-Чух!\*");  
 }  
}  
  
  
  
  
public class Wallet {  
 private int sum;  
 public Wallet(int sum){  
 this.sum = sum;  
 }  
  
 public int getSum() {  
 return sum;  
 }  
 public void subtract(int a){  
 sum -= a;  
 }  
 public void add(int a){  
 sum += a;  
 }  
}  
  
  
  
  
public class Ticket {  
 private String name;  
 private int price;  
 public Ticket(String name, int price){  
 this.name = name;  
 this.price = price;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public String getName() {  
 return name;  
 }  
}  
  
  
  
  
public class Cane {  
 private String name;  
 private int price;  
 public Cane(String name, int price){  
 this.name = name;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
}

Результат работы программы:

\*Объект Скуперфильд покупает билеты по цене 100\*

\*Общая сумма на руках у объекта Скуперфильд теперь составляет 900\*

\*Объект Скуперфильд садится на поезд Red\*

\*Объект проводник проверяет объекта Скуперфильд на наличие билета\*

\*Объект Скуперфильд имеет при себе билет\*

\*Объект Скуперфильд пытается завести разговор с объектом Сосед, имеющим свойства:Грубый, Неразговорчивый\*

\*Объект Скуперфильд не смог завести разговор с объектом Сосед\*

\*Объект Скуперфильд рассуждает над тем, что Не забыл ли я свою трость в том дупле?\*

\*Объект Скуперфильд так ни к чему и не пришёл\*

\*Объект Скуперфильд ложится на подушку с свойствами:Неудобный, Жёсткий - и пытаяется заснуть\*

\*Объект Скуперфильд не может заснуть\*

Вывод: Я начал лучше ориентироваться в java, понял, для чего нужны классы типов: enum, interface и т.д. -, научился создавать UML диаграммы в intellij idea.