Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Лабораторная работа №4

по дисциплине

«Программирование»

Вариант – 10765.9

Выполнил: Данилов Павел Юрьевич Р3110

Преподаватель: Горбунов Михаил Витальевич

Санкт-Петербург

2020 г.

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

Бутылка между тем наклонилась от тряски больше, и вода полилась из нее тонкой струйкой. С удовольствием глотая эту сладкую, пахучую, приятно щиплющую за язык жидкость, Скуперфильд прикидывал в уме, во сколько обошлась бы ему газированная вода, если бы понадобилось уплатить за нее. Эту сумму он вычитал из суммы, затраченной на покупку пропавшей трости, и испытывал удовольствие оттого, что сумма пропажи как бы становилась меньше. Бутылка тем временем наклонялась больше, благодаря чему газированная вода текла не переставая. В соответствии с этим текли и мысли в голове Скуперфильда. Постепенно увлекшись, он стал мечтать о том, как было бы хорошо, если бы при каждой железнодорожной поездке ему удавалось выпить хотя бы бутылку газированной воды бесплатно. Разделив стоимость пропавшей трости на стоимость бутылки газированной воды с сиропом, он вычислил количество железнодорожных поездок, которые пришлось бы совершить, чтоб вернуть сумму денег, затраченных на покупку трости. Занимаясь этими приятными расчетами, Скуперфильд постепенно забыл о своих огорчениях и пришел в хорошее настроение. Как раз в этот момент бутылка окончательно опрокинулась и, полетев вниз, стукнула Скуперфильда по лбу. Потрогав ушибленный лоб, он убедился, что на этот раз отделался шишкой. Чувствуя, что боль от удара понемногу проходит, он успокоился и наконец заснул. Поезд между тем мчался вперед. Колеса мерно постукивали. Время тоже не стояло на месте. Когда Скуперфильд заснул, было далеко за полночь. Не прошло и двух часов, как впереди засветились огни Брехенвиля. Колеса застучали на стрелках. Поезд постепенно замедлил ход и вскоре остановился. Скуперфильд, однако, продолжал спать. Проводник забыл его разбудить и вспомнил об этом, лишь когда поезд уже отошел от станции. Чтоб избежать неприятных объяснений, он решил пока не будить Скуперфильда, а принялся тормошить его, как только поезд остановился на следующей станции, которая имела какое-то странное название -- "Паноптикум". В ответ на это Скуперфильд только отмахивался рукой и продолжал храпеть, словно не к нему обращались. Видя, что поезд скоро отойдет и от этой станции, проводник рассердился не на шутку и закричал Скуперфильду прямо в ухо: Услыхав, что ему придется за что-то платить, но не разобрав за что, Скуперфильд на минутку очнулся и, соскочив со скамьи, осовело уставился на проводника. Воспользовавшись этим, проводник схватил его за шиворот, подтащил к выходу и вытолкнул на перрон. Вернувшись обратно, он поднял валявшуюся на полу газету, достал из-под лавки цилиндр, набитый всякой всячиной, и, подойдя к двери, сунул все это в руки ошалевшему Скуперфильду. Скуперфильд хотел о чем-то спросить и уже раскрыл рот, но поезд как раз в это мгновение тронулся, и он так и остался на перроне с разинутым ртом. Незнайка и Козлик даже не слыхали, что произошло ночью. Они спали достаточно крепко, так как в предыдущую ночь им не удалось как следует выспаться из-за кинокошмаров. Уже давно рассвело, а они продолжали спать и, наверно, проехали бы Сан-Комарик, если бы проводник не разбудил их. Видя, что Незнайка и Козлик даже не пошевелились, он принялся стучать по их полкам стальными щипцами, которыми пользовался для пробивки билетов. Услышав стук, Незнайка и Козлик проснулись.

Программа должна удовлетворять следующим требованиям:

1. В программе должны быть реализованы 2 собственных класса исключений (checked и unchecked), а также обработка исключений этих классов.
2. В программу необходимо добавить использование локальных, анонимных и вложенных классов (static и non-static).

Отчет должен содержать:

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

Диаграмма классов объектной модели

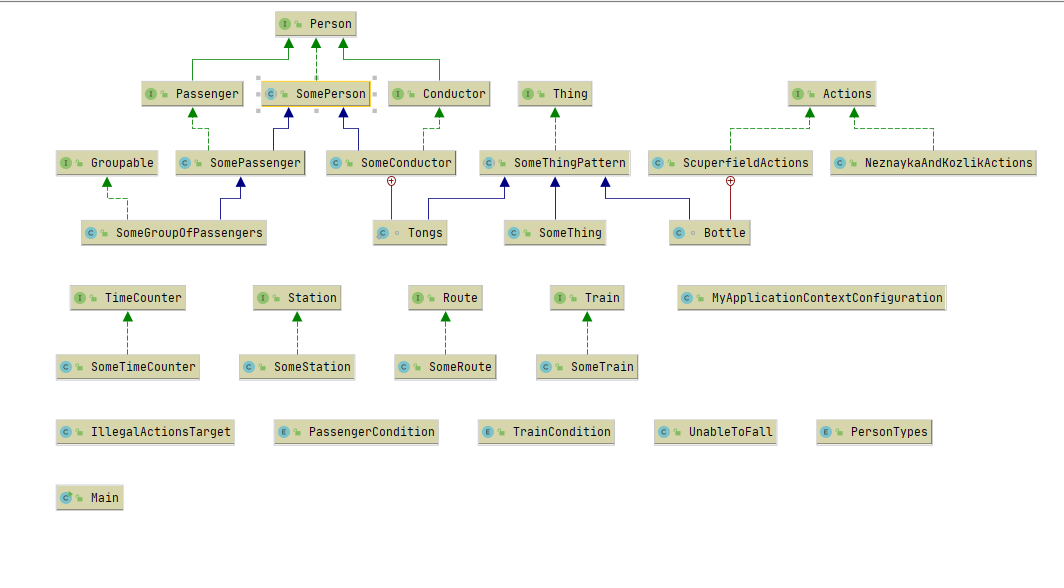
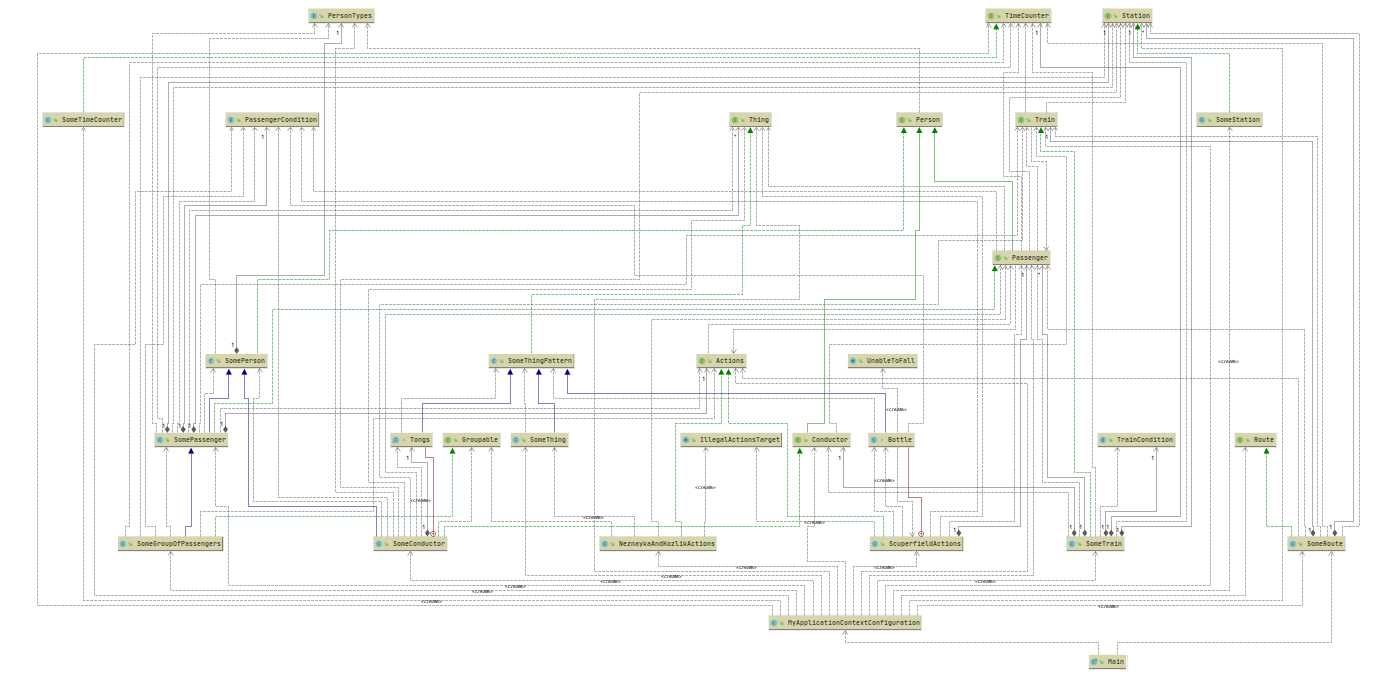
****

Диаграмма классов объектной модели с указанием связей классов:

****Исходный код

Исходный код можно найти по [ссылке](https://github.com/47iq/ITMO/tree/master/Programming/Lab4)

**public interface** Actions {  
 **void** completeActions(Passenger passenger) **throws** UnableToFall;  
}

**public interface** Conductor **extends** Person {  
 **void** checkPassengersOut(Train train);  
 **void** doubleCheck(Train train);  
}

**public interface** Groupable {  
}

**public class** IllegalActionsTarget **extends** RuntimeException {  
 **public** IllegalActionsTarget(String message) {  
 **super** (message);  
 }  
}

**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
**public class** Main {  
 **public static void** main(String[] args) **throws** UnableToFall {  
 ApplicationContext ctx = **new** AnnotationConfigApplicationContext(MyApplicationContextConfiguration.**class**);  
 SomeRoute route = ctx.getBean(SomeRoute.**class**);  
 route.runFullRoute();  
 }  
}

**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Configuration;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
@Configuration  
**public class** MyApplicationContextConfiguration {  
 @Bean  
 **public** Conductor conductor(){  
 **return new** SomeConductor(**"Conductor"**, 1.0);  
 }  
  
 @Bean  
 **public** Train train(){  
 Train train = **new** SomeTrain(station0(), conductor(), timeCounter());  
 train.addPassenger(passenger());  
 train.addPassenger(passenger2());  
 **return** train;  
 }  
  
 @Bean  
 **public** TimeCounter timeCounter(){  
 **return new** SomeTimeCounter(3);  
 }  
  
 @Bean  
 **public** Route route(){  
 **return new** SomeRoute(train(), stations());  
 }  
  
 @Bean  
 **public** List<Station> stations(){  
 List<Station> stations = **new** ArrayList<>();  
 stations.add(station1());  
 stations.add(station2());  
 stations.add(station3());  
 **return** stations;  
 }  
  
 @Bean  
 **public** Thing thing1() {  
 **return new** SomeThing(**"newspaper"**);  
 }  
  
 @Bean  
 **public** Thing thing2() {  
 **return new** SomeThing(**"top hat with some stuff"**);  
 }  
  
 @Bean List<Thing> baggage(){  
 List<Thing> baggage = **new** ArrayList<>();  
 baggage.add(thing1());  
 baggage.add(thing2());  
 **return** baggage;  
 }  
  
 @Bean  
 **public** Station station0() {  
 **return new** SomeStation(**"some station"**, 2);  
 }  
 @Bean  
 **public** Station station1() {  
 **return new** SomeStation(**"Brehenville"**, 1);  
 }  
  
 @Bean  
 **public** Station station2() {  
 **return new** SomeStation(**"Panopticon"**, 1);  
 }  
  
 @Bean  
 **public** Station station3() {  
 **return new** SomeStation(**"San-Komarik"**, 10);  
 }  
  
 @Bean  
 **public** Passenger passenger(){  
 **return new** SomePassenger(**"Scuperfield"**, PassengerCondition.***REGULAR\_AWAKE***, station1(), 10, baggage(), actions());  
 }  
  
 @Bean Actions actions(){  
 **return new** ScuperfieldActions();  
 }  
  
 @Bean Actions actions2(){  
 **return new** NeznaykaAndKozlikActions();  
 }  
  
 @Bean  
 **public** Passenger passenger2(){  
 **return new** SomeGroupOfPassengers(**"Neznayka and Kozlik"**, PassengerCondition.***SATISFIED***, station3(), 5, actions2());  
 }  
}

**public class** NeznaykaAndKozlikActions **implements** Actions{  
 NeznaykaAndKozlikActions() { }  
  
 **public void** completeActions(Passenger passenger) {  
 **if**(!passenger.getName().equals(**"Neznayka and Kozlik"**) || !(passenger **instanceof** Groupable))  
 **throw new** IllegalActionsTarget(**"Either passenger must be instance of \"Groupable\" or "** +  
 **"passengerGroup name must be \"Neznayka and Kozlik\"!"**);  
 System.***out***.println(passenger + **" are watching "** + **new** SomeThing(**"Horror movies"**){});  
 }  
}

**import** java.util.List;  
  
**public interface** Passenger **extends** Person {  
 PassengerCondition getCondition();  
 Station getDestination();  
 **void** leave(Train train);  
 **void** sleep();  
 **boolean** isAsleep();  
 **void** setCondition(PassengerCondition condition);  
 **double** getDepthOfSleep();  
 **void** commentOnLeavingTrain(Station station);  
 **void** fallAsleepTime(TimeCounter counter);  
 List<Thing> getBaggage();  
 Actions getActions();  
}

**public enum** PassengerCondition {  
 ***ASLEEP***{  
 **public** String toString() {  
 **return "asleep"**;  
 }  
 },  
 ***REGULAR\_AWAKE*** {  
 *//REGULAR\_AWAKE = !ASLEEP with no emotions* **public** String toString() {  
 **return "awake"**;  
 }  
 },  
 ***SHOCKED***{  
 **public** String toString() {  
 **return "shocked"**;  
 }  
 },  
 ***WOUNDED\_IN\_FOREHEAD***{  
 **public** String toString() {  
 **return "having a big bump on the head"**;  
 }  
 },  
 ***LIGHT\_WOUNDED\_IN\_FOREHEAD***{  
 **public** String toString() {  
 **return "having a small bump on the head"**;  
 }  
 },  
 ***SATISFIED***{  
 **public** String toString() {  
 **return "satisfied"**;  
 }  
 },  
 ***HAPPY***{  
 **public** String toString() {  
 **return "happy"**;  
 }  
 }  
}

**public interface** Person {  
 String getName();  
 **void** setName(String name);  
 **void** say(String phrase);  
 **void** think(String thought);  
 PersonTypes getType();  
}

**public enum** PersonTypes {  
 ***PASSENGER*** {  
 **public** String toString() {  
 **return "Passenger"**;  
 }  
 },  
 ***CONDUCTOR*** {  
 **public** String toString() {  
 **return "Conductor"**;  
 }  
 }  
}

**public interface** Route {  
 **void** runFullRoute() **throws** UnableToFall;  
}

**import** java.util.Objects;  
  
**public class** ScuperfieldActions **implements** Actions{  
 **private** Passenger **passenger**;  
  
 **public** ScuperfieldActions() {  
 }  
  
 **class** Bottle **extends** SomeThingPattern{  
 **private final** String **liquid**;  
 **private boolean IsSpilling** = **false**;  
 **private int pitch** = 0;  
 **private boolean hasFallen** = **false**;  
 **private** String **liquidDescription**;  
  
 Bottle(String name, String liquid, **double** price, **int** pitch, String liquidDescription) **throws** UnableToFall {  
 **super**(name, price);  
 **this**.**liquid** = liquid;  
 **this**.**liquidDescription** = liquidDescription;  
 **for**(**int** i = 0; i < pitch; i++)  
 tilt();  
 }  
  
 **public void** tilt() **throws** UnableToFall {  
 System.***out***.println(**this** + **" tilts."**);  
 **pitch**++;  
 **if**(**pitch** > 2)  
 fallOnPersonsHead();  
 **else if**(**pitch** > 1)  
 spillOnPerson();  
 **else if**(**pitch** == 1)  
 spill();  
 }  
  
 **public void** fallOnPersonsHead() **throws** UnableToFall{  
 **if**(**hasFallen**)  
 **throw new** UnableToFall(**this** + **" has already fallen!"**);  
 String s = **""**;  
 System.***out***.println(**this** + **" falls on "** + **passenger** + **"'s head"** + s + **"."**);  
 System.***out***.println(**this** + **" hits "** + **passenger** + **"'s forehead."**);  
 **double** hitPower = Math.*random*() \* 10;  
 **if**(hitPower < 5)  
 **passenger**.setCondition(PassengerCondition.***LIGHT\_WOUNDED\_IN\_FOREHEAD***);  
 **else  
 passenger**.setCondition(PassengerCondition.***WOUNDED\_IN\_FOREHEAD***);  
 **hasFallen** = **true**;  
 }  
  
 **public void** spill() {  
 System.***out***.println(getLiquid() + **" spills from "** + **this** + **"."**);  
 **IsSpilling** = **true**;  
 }  
  
 **public void** spillOnPerson(){  
 **if**(**IsSpilling**) {  
 System.***out***.println(getLiquid() + **" from "** + **this** + **" spills on "** + **passenger** + **"."**);  
 System.***out***.println(**passenger** + **" drinks "** + **liquidDescription** + **" "** + getLiquid() + **" from "** + **this** + **"."**);  
 }  
 }  
  
 **public** String getLiquid() {  
 **return liquid**;  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** Bottle)) **return false**;  
 Bottle bottle = (Bottle) that;  
 **return** Objects.*equals*(bottle.getName(), getName())  
 && Objects.*equals*(bottle.getLiquid(), **liquid**) && Objects.*equals*(bottle.getPrice(), getPrice())  
 && Boolean.*compare*(bottle.**IsSpilling**, **IsSpilling**) == 1;  
 }  
  
 **public** String toString() {  
 **return** getName();  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(getName(), getPrice(), getLiquid(), **IsSpilling**);  
 }  
 }  
  
 **public void** completeActions(Passenger passenger) **throws** UnableToFall {  
 **this**.**passenger** = passenger;  
 **if**(!passenger.getName().equals(**"Scuperfield"**))  
 **throw new** IllegalActionsTarget(**"You can only complete ScuperfieldActions for passengers named Scuperfield."**);  
  
 **class** Cane **extends** SomeThingPattern{  
 Cane(String name, **double** price) {  
 **super**(name, price);  
 System.***out***.println(**"//Previously "** + passenger + **" lost his "** + **this** + **" which costs "** +  
 (**int**) **this**.getPrice() + **"."**);  
 }  
 }  
  
 Cane lostCane = **new** Cane(**"Cane"**, 10000);  
 Bottle bottle = **new** Bottle(**"Bottle"**, **"Soda"**, 10, 2, **"sweet, great smelling, "** +  
 **"pleasantly pinching mouth"**);  
 makeCalculations(bottle, lostCane);  
 bottle.tilt();  
 }  
  
 **private void** makeCalculations(Thing thing, Thing secondThing) {  
 calcPrice(thing);  
 sub(thing.getPrice(), secondThing.getPrice());  
 calculateNum(thing.getPrice(), secondThing.getPrice());  
 }  
  
 **private void** calcPrice(Thing thing) {  
 **double** res = thing.getPrice();  
 **passenger**.think(**"I would have to pay "** + (**int**) res + **" for this "** + thing);  
 }  
  
 **private void** sub(**double** price, **double** price0) {  
 **double** res = price0 - price;  
 **passenger**.think(**"The sum of my loss has lowered. It's only "** + (**int**) res + **" now!"**);  
 **passenger**.setCondition(PassengerCondition.***SATISFIED***);  
 }  
  
 **private void** calculateNum(**double** price, **double** price0) {  
 **double** res = price0 / price;  
 **if** (price0 % price > 0)  
 res++;  
 String MIDDLE = **"trips"**;  
 **if**(res == 1)  
 MIDDLE = **"trip"**;  
 **passenger**.think(**"I have to take only "** + (**int**) res + **" train "** + MIDDLE + **" to get my money back!"**);  
 **passenger**.setCondition(PassengerCondition.***HAPPY***);  
 }  
}

**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Objects;  
  
**public class** SomeConductor **extends** SomePerson **implements** Conductor {  
 **private double forgetfulness**;  
 **private final** Tongs **tongs** = **new** Tongs();  
  
 **static class** Tongs **extends** SomeThingPattern {  
 **public** Tongs() {  
 **super**(**"tongs"**);  
 }  
 }  
  
 **public** SomeConductor(String s, **double** forgetfulness) {  
 **super**(s);  
 **super**.setType(PersonTypes.***CONDUCTOR***);  
 **this**.**forgetfulness** = forgetfulness;  
 }  
  
 **public void** checkPassengersOut(Train train) {  
 Station station = train.getStation();  
 Station prevStation = train.getPrevStation();  
 ArrayList<Passenger> passengers= train.getPassengers();  
 ArrayList<Passenger> out = **new** ArrayList<>();  
 **for**(Passenger passenger: passengers) {  
 **if**(passenger.getDestination().equals(station) && Math.*random*() > **forgetfulness** || passenger.getDestination().equals(prevStation)){  
 out.add(passenger);  
 }  
 }  
 **for**(Passenger passenger: out) {  
 remindToLeave(passenger, train);  
 }  
 }  
  
 **public void** doubleCheck(Train train) {  
 Station station = train.getStation();  
 ArrayList<Passenger> passengers= train.getPassengers();  
 **int** counter = 0;  
 System.***out***.print(**this** + **" checks if he forgot to tell anybody to leave the train.\n"**);  
 **for**(Passenger passenger: passengers) {  
 String START = **"he"**;  
 String MIDDLE = **"his"**;  
 **if**(passenger **instanceof** Groupable){  
 START = **"they"**;  
 MIDDLE = **"their"**;  
 }  
 **if**(passenger.getDestination().equals(station)){  
 System.***out***.print(**"Oh no! "** + **this** + **" forgot to tell "** + passenger + **" to leave.\n"**);  
 counter++;  
 System.***out***.print(**this** + **" decides to wait till the next station and not to say "** + passenger  
 + **" that "** + START +**" skipped "** + MIDDLE + **" station, because "** + **this** +**" wants to avoid explanations.\n"**);  
 **this**.**forgetfulness** = 0;  
 System.***out***.print(**this** + **" tries to get more concentrated. He won't forget about any passenger from now on.\n"**);  
 }  
 }  
 **if**(counter == 0)  
 System.***out***.print(**this** + **" didn't forget anyone.\n"**);  
 }  
  
 **private void** remindToLeave(Passenger passenger, Train train) {  
 say(passenger + **", you have to leave now!"**);  
 **if**(!passenger.getCondition().equals(PassengerCondition.***ASLEEP***)) {  
 passenger.leave(train);  
 }  
 **else** {  
 String MIDDLE = **"is"**;  
 **if**(passenger **instanceof** Groupable)  
 MIDDLE = **"are"**;  
 System.***out***.println(passenger + **" didn't hear "** + **this** + **" cause "** + passenger + **" "** + MIDDLE + **" sleeping."**);  
 shake(passenger, train);  
 }  
 }  
  
 **private void** throwBaggageOut(Passenger passenger) {  
 List<Thing> baggage = passenger.getBaggage();  
 List<Thing> out = **new** ArrayList<>();  
 **if**(!baggage.isEmpty()) {  
 **for**(Thing thing: baggage) {  
 System.***out***.println(**this** + **" throws out "** + passenger + **"'s "** + thing + **"."**);  
 out.add(thing);  
 }  
 **for**(Thing thing: out) {  
 baggage.remove(thing);  
 }  
 }  
 }  
  
 **private void** shake(Passenger passenger, Train train) {  
 System.***out***.println(**this** + **" shakes "** + passenger + **"."**);  
 **double** depthOfSleep = passenger.getDepthOfSleep();  
 **if**(depthOfSleep < 3) {  
 System.***out***.print(**this** + **" wakes "** + passenger + **" up.\n"**);  
 passenger.setCondition(PassengerCondition.***REGULAR\_AWAKE***);  
 }  
 **else** {  
 String BEGINNING = **""**;  
 String ENDING = **""**;  
 String MIDDLE;  
 **if**(passenger **instanceof** Groupable){  
 BEGINNING += (passenger + **" are "**);  
 ENDING += **"their shelves"**;  
 MIDDLE = **"they"**;  
 }  
 **else** {  
 BEGINNING += (passenger + **" is "**);  
 ENDING += **"his shelf"**;  
 MIDDLE = **"he"**;  
 }  
 **if**(depthOfSleep < 6) {  
 System.***out***.println(BEGINNING + **"still sleeping. "** + **this** + **" takes out "** + **tongs** + **" and starts knocking on "** + ENDING + **"."**);  
 passenger.setCondition(PassengerCondition.***REGULAR\_AWAKE***);  
 passenger.leave(train);  
 }  
 **else** {  
 System.***out***.println(BEGINNING + **"still sleeping. "** + **this** + **" realizes that "** + MIDDLE +  
 **" won't wake up and gets angry. "** + **this** + **" shouts into "** + passenger + **"'s ear."**);  
 passenger.setCondition(PassengerCondition.***REGULAR\_AWAKE***);  
 kickOutOfTrain(passenger, train);  
 throwBaggageOut(passenger);  
 }  
 }  
 }  
  
 **private void** kickOutOfTrain(Passenger passenger, Train train) {  
 System.***out***.println(**this** + **" kicks "** + passenger + **" out of the train."**);  
 passenger.setCondition(PassengerCondition.***SHOCKED***);  
 passenger.leave(train);  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeConductor)) **return false**;  
 SomeConductor conductor = (SomeConductor) that;  
 **return** Double.*compare*(conductor.**forgetfulness**, **forgetfulness**) == 0  
 && Objects.*equals*(getName(), conductor.getName()) && Objects.*equals*(getType(), conductor.getType());  
 }  
  
 **public** String toString() {  
 **return** getName();  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(getName(), getType(), **forgetfulness**);  
 }  
}

**public class** SomeGroupOfPassengers **extends** SomePassenger **implements** Groupable {  
 SomeGroupOfPassengers(String names, PassengerCondition condition, Station destination, **double** depthOfSleep,  
 Actions actions) {  
 **super**(names, condition, destination, depthOfSleep, actions);  
 }  
  
 **public** String toString() {  
 **return super**.getType() + **"s "** + **super**.getName();  
 }  
  
 **public void** say(String s) {  
 System.***out***.print(**this** + **" say: \""** + s + **"\"\n"**);  
 }  
  
 **public void** sleep() {  
 System.***out***.print(**this**+ **" are sleeping. "**);  
 say(**"Zzz.."**);  
 }  
  
 **protected void** commentOnSetCondition(PassengerCondition condition){  
 System.***out***.print(**this** + **" are now "** + condition + **". \n"**);  
 }  
  
 **public void** commentOnLeavingTrain(Station station) {  
 System.***out***.print(**this** + **" leave the train at "** + station + **".\n"**);  
 **if**(station != getDestination()) {  
 **if**(!getCondition().equals(PassengerCondition.***SHOCKED***))  
 say(**"Hey! That's not our station!"**);  
 **else** System.***out***.println(**this** + **" want to say something but are too shocked to open their mouths."**);  
 }  
 }  
  
 **public void** fallAsleepTime(TimeCounter counter) {  
 **if**(!**super**.getCondition().equals(PassengerCondition.***ASLEEP***)) {  
 System.***out***.print(**this** + **" fall asleep because the time is "** + counter + **"\n"**);  
 **super**.setCondition(PassengerCondition.***ASLEEP***);  
 }  
 }  
  
}

**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Objects;  
  
**public class** SomePassenger **extends** SomePerson **implements** Passenger {  
 **private** PassengerCondition **condition**;  
 **private final** Station **destination**;  
 **private final double depthOfSleep**;  
 **private** List<Thing> **baggage** = **new** ArrayList<>();  
 **private** Actions **actions**;  
  
 **public** SomePassenger(String name, PassengerCondition condition, Station destination, **double** depthOfSleep,  
 Actions actions) {  
 **super**(name);  
 **super**.setType(PersonTypes.***PASSENGER***);  
 **this**.**destination** = destination;  
 **this**.**condition** = condition;  
 **this**.**depthOfSleep** = depthOfSleep;  
 **this**.**actions** = actions;  
 }  
  
 **public** SomePassenger(String name, PassengerCondition condition, Station destination, **double** depthOfSleep,  
 List<Thing> baggage, Actions actions) {  
 **super**(name);  
 **super**.setType(PersonTypes.***PASSENGER***);  
 **this**.**destination** = destination;  
 **this**.**condition** = condition;  
 **this**.**depthOfSleep** = depthOfSleep;  
 **this**.**baggage** = baggage;  
 **this**.**actions** = actions;  
 }  
  
 **public** Station getDestination() {  
 **return destination**;  
 }  
  
 **public void** leave(Train train) {  
 train.removePassenger(**this**);  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomePassenger)) **return false**;  
 SomePassenger passenger = (SomePassenger) that;  
 **return** Objects.*equals*(passenger.**destination**, **destination**) && Objects.*equals*(getName(), passenger.getName())  
 && Objects.*equals*(getType(), passenger.getType())  
 && Objects.*equals*(getCondition(), passenger.getCondition());  
 }  
  
 **public void** commentOnLeavingTrain(Station station) {  
 System.***out***.print(**this** + **" leaves the train at "** + station + **".\n"**);  
 **if**(station != **destination**) {  
 **if**(!**condition**.equals(PassengerCondition.***SHOCKED***))  
 say(**"Hey! That's not my station!"**);  
 **else** System.***out***.println(**this** + **" wants to say something but is too shocked to open his mouth."**);  
 }  
 }  
  
 **public void** sleep() {  
 System.***out***.print(**this**+ **" is sleeping. "**);  
 say(**"Zzz.."**);  
 }  
  
 **public void** setCondition(PassengerCondition condition) {  
 **this**.**condition** = condition;  
 commentOnSetCondition(condition);  
 }  
  
 **protected void** commentOnSetCondition(PassengerCondition condition){  
 System.***out***.print(**this** + **" is now "** + condition + **". \n"**);  
 }  
  
 **public void** fallAsleepTime(TimeCounter counter) {  
 **if**(!**condition**.equals(PassengerCondition.***ASLEEP***)) {  
 System.***out***.print(**this** + **" falls asleep because the time is "** + counter + **"\n"**);  
 setCondition(PassengerCondition.***ASLEEP***);  
 }  
 }  
  
 **public** List<Thing> getBaggage() {  
 **return baggage**;  
 }  
  
 **public** Actions getActions() {  
 **return actions**;  
 }  
  
 **public double** getDepthOfSleep() {  
 **return this**.**depthOfSleep**;  
 }  
  
 **public** PassengerCondition getCondition() {  
 **return condition**;  
 }  
  
 **public boolean** isAsleep() {  
 **return** (getCondition() == PassengerCondition.***ASLEEP***);  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(getCondition(), getName(), getType(), **destination**);  
 }  
}

**abstract public class** SomePerson **implements** Person {  
 **private** String **name**;  
 **private** PersonTypes **type**;  
  
 **public** SomePerson(String name){  
 **this**.**name** = name;  
 }  
  
 **public void** say(String s) {  
 System.***out***.print(**this** + **" says: \""** + s + **"\"\n"**);  
 }  
  
 **public void** think(String s) {  
 System.***out***.print(**this** + **" thinks: \""** + s + **"\"\n"**);  
 }  
  
 **protected void** setType(PersonTypes type) {  
 **this**.**type** = type;  
 }  
  
 **public** String toString() {  
 **return type** + **" "** + **name**;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** PersonTypes getType() {  
 **return type**;  
 }  
}

**import** java.util.List;  
**import** java.util.Objects;  
  
**public class** SomeRoute **implements** Route {  
 **private final** List<Station> **stations**;  
 **private final** Train **train**;  
  
 **public** SomeRoute(Train train, List<Station> stations) {  
 **this**.**stations** = stations;  
 **this**.**train** = train;  
 }  
  
 **private void** timeToSleepCheck() {  
 List<Passenger> passengers = **train**.getPassengers();  
 TimeCounter timeCounter = **train**.getTimeCounter();  
 **if**(timeCounter.getHour() < 6)  
 **for**(Passenger passenger: passengers)  
 passenger.fallAsleepTime(timeCounter);  
 **for**(Passenger passenger : passengers)  
 **if**(passenger.isAsleep())  
 passenger.sleep();  
 }  
  
 **public void** runFullRoute() **throws** UnableToFall {  
 **int** ind = 0;  
 **for**(Station station: **stations**) {  
 **if**(ind == 0 && !**train**.getPassengers().isEmpty()) {  
 **train**.noCheckStart();  
 **for**(Passenger passenger: **train**.getPassengers()) {  
 Actions actions = passenger.getActions();  
 **if**(actions != **null**)  
 actions.completeActions(passenger);  
 }  
 timeToSleepCheck();  
 **train**.stopAt(station);  
 }  
 **else if** (!**train**.getPassengers().isEmpty()) {  
 **train**.start();  
 timeToSleepCheck();  
 **train**.stopAt(station);  
 }  
 **else** {  
 **train**.lastStart();  
 }  
 ind++;  
 }  
 **train**.lastStart();  
 }  
  
 **public int** hashCode(){  
 **return** Objects.*hash*(**stations**, **train**);  
 }  
  
 **public** String toString() {  
 **return "The route is: "** + **stations** + **" with a "** + **train**;  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeRoute)) **return false**;  
 SomeRoute route = (SomeRoute)that;  
 **return** Objects.*equals*(route.**stations**, **stations**)  
 && Objects.*equals*(route.**train**, **train**);  
 }  
}

**import** java.util.Objects;  
  
**public class** SomeStation **implements** Station{  
 **private final** String **name**;  
 **private final int stageTime**;  
  
 **public** SomeStation(String name, **int** stageTime) {  
 **this**.**name** = name;  
 **this**.**stageTime** = stageTime;  
 }  
  
 **public** String getName(){  
 **return name**;  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeStation)) **return false**;  
 SomeStation station = (SomeStation) that;  
 **return** Objects.*equals*(**name**, station.**name**);  
 }  
  
 **public int** getStageTime() {  
 **return stageTime**;  
 }  
  
 **public** String toString() {  
 **return name**;  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(**name**);  
 }  
}

**public class** SomeThing **extends** SomeThingPattern {  
 **public** SomeThing(String name) {  
 **super**(name, 300);  
 }  
}

**import** java.util.Objects;  
  
**public abstract class** SomeThingPattern **implements** Thing{  
 **private final** String **name**;  
 **private final double price**;  
  
 **public** SomeThingPattern(String name, **double** price){  
 **this**.**name** = name;  
 **this**.**price** = price;  
 **if**(price <= 0) **throw new** IllegalArgumentException();  
 }  
  
 **public** SomeThingPattern(String name){  
 **this**.**name** = name;  
 **price** = 300;  
 }  
  
 **public double** getPrice() {  
 **return this**.**price**;  
 }  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeThingPattern)) **return false**;  
 SomeThingPattern someThing = (SomeThingPattern) that;  
 **return** Objects.*equals*(**name**, someThing.**name**);  
 }  
  
 **public** String toString() {  
 **return this**.**name**;  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(**name**);  
 }  
}

**import** java.util.Objects;  
  
**public class** SomeTimeCounter **implements** TimeCounter{  
 **private int currentHour**;  
 SomeTimeCounter(**int** currentHour) {  
 **this**.**currentHour** = currentHour;  
 }  
  
 **public int** getHour() {  
 **return this**.**currentHour**;  
 }  
  
 **public void** addHours(**int** number) {  
 **currentHour** = (**currentHour** + number) % 24;  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeTimeCounter)) **return false**;  
 SomeTimeCounter counter = (SomeTimeCounter) that;  
 **return** Objects.*equals*(**currentHour**, counter.**currentHour**);  
 }  
  
 **public** String toString() {  
 String out = **""**;  
 **if**(**currentHour** < 12) {  
 out += **currentHour** + **" am."**;  
 }  
 **else** {  
 out += (**currentHour** - 12) + **" pm."**;  
 }  
 **return** out;  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(**currentHour**);  
 }  
}

**import** java.util.ArrayList;  
**import** java.util.Objects;  
  
**public class** SomeTrain **implements** Train {  
 **private** Station **station**;  
 **private** Station **prevStation**;  
 **private** TrainCondition **condition**;  
 **private final** Conductor **conductor**;  
 **private final** TimeCounter **timeCounter**;  
 **private final** ArrayList<Passenger> **passengers** = **new** ArrayList<>();  
  
 **public** SomeTrain(Station station, Conductor conductor, TimeCounter timeCounter) {  
 **this**.**station** = station;  
 **condition** = TrainCondition.***STAYING***;  
 **this**.**conductor** = conductor;  
 System.***out***.print(**this**.toString());  
 **this**.**timeCounter** = timeCounter;  
 }  
  
 **public void** start() {  
 noCheckStart();  
 **conductor**.doubleCheck(**this**);  
 **condition** = TrainCondition.***MOVING***;  
 }  
  
 **public void** lastStart() {  
 System.***out***.print(**"Train starts moving and goes away from "** + **station** + **".\n"**);  
 **condition** = TrainCondition.***MOVING***;  
 System.***out***.print(**"The train disappears in the distance...\n"**);  
 }  
  
 **public void** noCheckStart() {  
 System.***out***.print(**"Train starts moving and goes away from "** + **station** + **".\n"**);  
 **condition** = TrainCondition.***MOVING***;  
 }  
  
 **public** TimeCounter getTimeCounter() {  
 **return timeCounter**;  
 }  
  
 **public void** stopAt(Station station) {  
 **condition** = TrainCondition.***STAYING***;  
 **timeCounter**.addHours(**this**.**station**.getStageTime());  
 **prevStation** = **this**.**station**;  
 **this**.**station** = station;  
 System.***out***.print(**"Train stops at "** + **this**.**station** + **" at "** + **timeCounter** + **"\n"**);  
 **conductor**.checkPassengersOut(**this**);  
 }  
  
 **public** Station getStation(){  
 **return station**;  
 }  
  
  
 **public** Station getPrevStation(){  
 **return prevStation**;  
 }  
  
 **public** ArrayList<Passenger> getPassengers() {  
 **return passengers**;  
 }  
  
 **public void** addPassenger(Passenger passenger) {  
 **passengers**.add(passenger);  
 }  
  
 **public void** removePassenger(Passenger passenger) {  
 **passengers**.remove(passenger);  
 String beginning = **"Current station is"**;  
 **if**(passenger.getDestination().equals(**station**))  
 System.***out***.print( beginning + **" "** + passenger + **"'s destination station.\n"**);  
 **else** System.***out***.print( beginning +**"n't "** + passenger + **"'s destination station\n"**);  
 passenger.commentOnLeavingTrain(**station**);  
 }  
  
 **public boolean** equals(Object that) {  
 **if** (**this** == that) **return true**;  
 **if** (!(that **instanceof** SomeTrain)) **return false**;  
 SomeTrain train = (SomeTrain) that;  
 **return** Objects.*equals*(**station**, train.**station**) && Objects.*equals*(**prevStation**, train.**prevStation**)  
 && Objects.*equals*(**condition**, train.**condition**) && Objects.*equals*(**passengers**, train.**passengers**);  
 }  
  
 **public** String toString() {  
 **final** String BEGINNING = **"Train is "** + **condition**;  
 **final** String ENDING = **station** + **". The train's conductor's name is "** + **conductor**.getName() + **".\n"**;  
 **if**(**condition**.equals(TrainCondition.***MOVING***))  
 **return** BEGINNING + **" from "** + ENDING;  
 **else  
 return** BEGINNING + **" at "** + ENDING;  
 }  
  
 **public int** hashCode() {  
 **return** Objects.*hash*(**station**, **prevStation**, **condition**, **passengers**);  
 }  
}

**public interface** Station {  
 String getName();  
 **int** getStageTime();  
}

**public interface** Thing {  
 String getName();  
 **double** getPrice();  
}

**public interface** TimeCounter {  
 **int** getHour();  
 **void** addHours(**int** number);  
}

**import** java.util.ArrayList;  
  
**public interface** Train {  
 **void** start();  
 **void** lastStart();  
 **void** noCheckStart();  
 **void** stopAt(Station station);  
 Station getStation();  
 Station getPrevStation();  
 **void** addPassenger(Passenger passenger);  
 **void** removePassenger(Passenger passenger);  
 ArrayList<Passenger> getPassengers();  
 TimeCounter getTimeCounter();  
}

**public enum** TrainCondition{  
 ***STAYING*** {  
 **public** String toString() {  
 **return "staying"**;  
 }  
 },  
 ***MOVING*** {  
 **public** String toString() {  
 **return "moving"**;  
 }  
 }  
}

**public class** UnableToFall **extends** Exception{  
 UnableToFall(String message) {  
 **super**(message);  
 }  
}

Output

Train is staying at some station. The train's conductor's name is Conductor.

Train starts moving and goes away from some station.

//Previously Passenger Scuperfield lost his Cane which costs 10000.

Bottle tilts.

Soda spills from Bottle.

Bottle tilts.

Soda from Bottle spills on Passenger Scuperfield.

Passenger Scuperfield drinks sweet, great smelling, pleasantly pinching mouth Soda from Bottle.

Passenger Scuperfield thinks: "I would have to pay 10 for this Bottle"

Passenger Scuperfield thinks: "The sum of my loss has lowered. It's only 9990 now!"

Passenger Scuperfield is now satisfied.

Passenger Scuperfield thinks: "I have to take only 1000 train trips to get my money back!"

Passenger Scuperfield is now happy.

Bottle tilts.

Bottle falls on Passenger Scuperfield's head.

Bottle hits Passenger Scuperfield's forehead.

Passenger Scuperfield is now having a big bump on the head.

Passengers Neznayka and Kozlik are watching Horror movies

Passenger Scuperfield falls asleep because the time is 3 am.

Passenger Scuperfield is now asleep.

Passengers Neznayka and Kozlik fall asleep because the time is 3 am.

Passengers Neznayka and Kozlik are now asleep.

Passenger Scuperfield is sleeping. Passenger Scuperfield says: "Zzz.."

Passengers Neznayka and Kozlik are sleeping. Passengers Neznayka and Kozlik say: "Zzz.."

Train stops at Brehenville at 5 am.

Train starts moving and goes away from Brehenville.

Conductor checks if he forgot to tell anybody to leave the train.

Oh no! Conductor forgot to tell Passenger Scuperfield to leave.

Conductor decides to wait till the next station and not to say Passenger Scuperfield that he skipped his station, because Conductor wants to avoid explanations.

Conductor tries to get more concentrated. He won't forget about any passenger from now on.

Passenger Scuperfield is sleeping. Passenger Scuperfield says: "Zzz.."

Passengers Neznayka and Kozlik are sleeping. Passengers Neznayka and Kozlik say: "Zzz.."

Train stops at Panopticon at 6 am.

Conductor says: "Passenger Scuperfield, you have to leave now!"

Passenger Scuperfield didn't hear Conductor cause Passenger Scuperfield is sleeping.

Conductor shakes Passenger Scuperfield.

Passenger Scuperfield is still sleeping. Conductor realizes that he won't wake up and gets angry. Conductor shouts into Passenger Scuperfield's ear.

Passenger Scuperfield is now awake.

Conductor kicks Passenger Scuperfield out of the train.

Passenger Scuperfield is now shocked.

Current station isn't Passenger Scuperfield's destination station

Passenger Scuperfield leaves the train at Panopticon.

Passenger Scuperfield wants to say something but is too shocked to open his mouth.

Conductor throws out Passenger Scuperfield's newspaper.

Conductor throws out Passenger Scuperfield's top hat with some stuff.

Train starts moving and goes away from Panopticon.

Conductor checks if he forgot to tell anybody to leave the train.

Conductor didn't forget anyone.

Passengers Neznayka and Kozlik are sleeping. Passengers Neznayka and Kozlik say: "Zzz.."

Train stops at San-Komarik at 7 am.

Conductor says: "Passengers Neznayka and Kozlik, you have to leave now!"

Passengers Neznayka and Kozlik didn't hear Conductor cause Passengers Neznayka and Kozlik are sleeping.

Conductor shakes Passengers Neznayka and Kozlik.

Passengers Neznayka and Kozlik are still sleeping. Conductor takes out tongs and starts knocking on their shelves.

Passengers Neznayka and Kozlik are now awake.

Current station is Passengers Neznayka and Kozlik's destination station.

Passengers Neznayka and Kozlik leave the train at San-Komarik.

Train starts moving and goes away from San-Komarik.

The train disappears in the distance...

**Выводы:** в результате выполнения этой работы я понял основы работы с reflection API, ознакомился с системой исключений и их иерархией, понял, когда и зачем применяются различные вложенные и локальные классы.