Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет ИТМО"
Факультет Программной инженерии и компьютерной техники

Лабораторная работа №4 по курсу "Программирование"

Группа: **P3131**
Студент: Друян Эдуард Андреевич

Санкт-Петербург
2021 г.

Задание

Текст задания

Доработать программу из лабораторной работы #3, обновив реализацию объектной модели в соответствии с новой версией описания предметной области.

Предметная область

В ответ на это Пилюлькин только таинственно улыбнулся. Когда Медуница и весь обслуживающий персонал вернулись в больницу, то сразу же обнаружили исчезновение Ворчуна. Они бросились в кладовую, и тут же была обнаружена пропажа двух комплектов одежды. В кладовой осталась только одежда Бульки.

Таким образом выяснился план побега, который был задуман Ворчуном и доктором Пилюлькиным. По этому плану доктор Пилюлькин должен был бежать в голом виде через окно. Злоумышленники рассчитывали, что весь персонал больницы бросится за ним в погоню, — тогда Ворчун свободно проникнет в кладовую и похитит одежду, как свою, так и Пилюлькина. План оправдался во всех деталях.

Медуница ещё долго разыскивала Ворчуна с похищенной им одеждой, и, пока шли поиски, Ворчун сидел, притаившись, в зарослях лопуха. Хотя сидение в лопухах не такое уж весёлое дело, но Ворчун был вне себя от радости, что вырвался на свободу. Он с наслаждением глядел на прозрачное синее небо, на свежую зелёную травку. На лице его даже появилась улыбка. Он дал сам себе клятву никогда в жизни не ворчать больше и быть довольным всем на свете, если только не попадёт снова в больницу.

Наконец Ворчун увидел, что Медуница скрылась в больнице. Тогда он потихоньку вылез из своей засады, разыскал Пилюлькина и отдал ему одежду.

Пилюлькин бросился обнимать своего приятеля. Они оба очень сдружились, пока находились в больнице. Пилюлькин быстро оделся. Растеряйка, Авоська, Винтик и другие малыши окружили Ворчуна и стали поздравлять его с благополучным возвращением из больницы. Все были удивлены его весёлым видом.

Требования к программе

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 nested`** и **`inner`**).

Порядок выполнения

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.

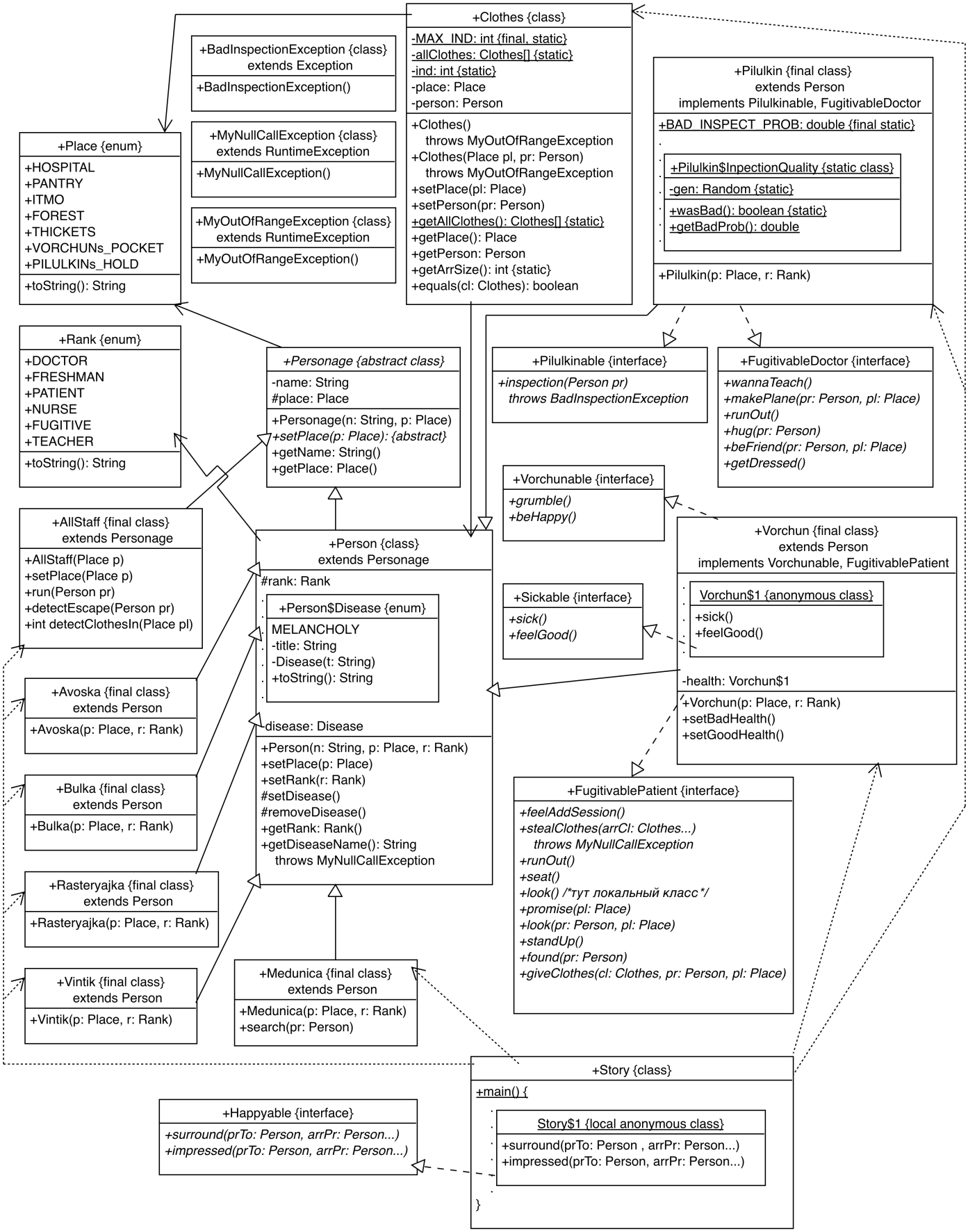
Требования к отчету

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

Вопросы к защите

1. Обработка исключительных ситуаций, три типа исключений.
2. Вложенные, локальные и анонимные классы.
3. Механизм рефлексии (reflection) в Java. Класс Class.

Диаграмма классов объектной модели



Исходный код программы

additions

Clothes.java

```
1.   package lib.additions;
2.
3.   import lib.enums.*;
4.   import lib.pers.*;
5.   import lib.exceptions.*;
6.
7.   final public class Clothes {
8.       private static final int MAX_IND = 20;
9.       private static Clothes[] allClothes = new Clothes[MAX_IND];
10.      private static int ind = 0;
11.
12.      private Place place;
13.      private Person person;
14.
15.      public Clothes() throws MyOutOfRangeException {
16.          place = null;
17.          person = null;
18.          if (ind == MAX_IND) {
19.              throw new MyOutOfRangeException();
20.          }
21.          else { allClothes[ind++] = this; }
22.      }
23.
24.      public Clothes(Place pl, Person pr)
25.      throws MyOutOfRangeException {
26.          place = pl;
27.          person = pr;
28.          if (ind == MAX_IND) {
29.              throw new MyOutOfRangeException();
30.          }
31.          else { allClothes[ind++] = this; }
32.          System.out.println("Одежда персоны " +
33.              pr.getName() + " попадает в " +
34.              "место: " + pl.toString());
35.      }
36.
37.      public void setPlace(Place pl) {
38.          place = pl;
39.          System.out.println("Одежда персоны " +
40.              person.getName() + " попадает в " +
41.              "место: " + place.toString());
42.      }
43.
44.      public void setPerson(Person pr) {
45.          System.out.println("Одежда переходит" +
46.              " от персоны " + person.getName() +
47.              " к персоне " + pr.getName());
48.          person = pr;
49.      }
50.
51.      public static Clothes[] getAllClothes() { return allClothes; }
52.      public Place getPlace() { return place; }
53.      public Person getPerson() { return person; }
54.      public static int getArrSize() { return ind; }
```

Clothes.java

```
55.      @Override
56.      public boolean equals(Object obj) {
57.          if ( !(obj instanceof Clothes) ) {
58.              return false;
59.          }
60.          if (place != ((Clothes)obj).getPlace()) {
61.              return false;
62.          }
63.          if (person != ((Clothes)obj).getPerson()) {
64.              return false;
65.          }
66.          return true;
67.      }
68.  }
```

enums

Place.java

```
1.   package lib.enums;
2.
3.   public enum Place {
4.       HOSPITAL("больница \"Укол-Комарика\""),
5.       PANTRY("кладовая"),
6.       ITMO("ИТМО"),
7.       FOREST("лес"),
8.       THICKETS("заросли лопуха"),
9.       VORCHUNs_POCKET("карман Ворчуна"),
10.      PILULKINs_HOLD("владение Пилюлькина");
11.
12.      private String title;
13.
14.      private Place(String t) { title = t; }
15.
16.      @Override
17.      public String toString() { return title; }
18.  }
19.
```

Rank.java

```
1.   package lib.enums;
2.
3.   public enum Rank {
4.       DOCTOR("доктор"),
5.       FRESHMAN("малыш-первач"),
6.       PATIENT("пациент"),
7.       NURSE("медсестра"),
8.       FUGITIVE("беглец"),
9.       TEACHER("преподаватель");
10.      private String title;
11.      private Rank(String t) { title = t; }
12.      @Override
13.      public String toString() { return title; }
14.  }
```

exceptions

BadInspectionException.java

```
1.   package lib.exceptions;
2.
3.   import java.lang.Exception;
4.
5.   // Пилюлькин не смог ничего найти,
6.   // хотя болезнь у персоны есть.
7.   // Причина такого поведения остается загадкой.
8.   // Обработка в Story.
9.   public class BadInspectionException extends Exception {
10.      public BadInspectionException() {
11.          super(" ::::::::::::::::::::::::::::::::\n" +
12.              " Плохой осмотр\n" +
13.              " ::::::::::::::::::::::::::::::::");
14.      }
15.  }
```

MyNullCallException.java

```
1.   package lib.exceptions;
2.
3.   import java.lang.RuntimeException;
4.
5.   // Попытка вызвать метод от null.
6.   // Может быть выброшен при осмотре,
7.   // если болезнь не назначена
8.   // и если одежда никому не принадлежит.
9.   // Обработка в Story.
10.  public class MyNullCallException extends RuntimeException {
11.      public MyNullCallException() {
12.          super(" //////////////////////////////////\n " +
13.              " Попытка вызова метода от \"null\"\n" +
14.              " //////////////////////////////////");
15.      }
16.  }
```

MyOutOfRangeException.java

```
1.   package lib.exceptions;
2.
3.   import java.lang.RuntimeException;
4.
5.   // Массив одежды ограничен размером: 20.
6.   // На всякий случай пропишу исключение.
7.   // Это исключение решил не обрабатывать,
8.   // что сделать могу, т.к. unchecked.
9.   public class MyOutOfRangeException extends RuntimeException {
10.      public MyOutOfRangeException() {
11.          super("\n ################################\n" +
12.              " Выход за предел массива\n" +
13.               " ################################\n");
14.      }
15.  }
```

interf

FugitivableDoctor.java

```
1.   package lib.interf;
2.   import lib.enums.*;
3.   import lib.pers.*;
4.   public interface FugitivableDoctor {
5.       public abstract void wannaTeach();
6.       public abstract void makePlane(Person pr, Place pl);
7.       public abstract void runOut();
8.       public abstract void hug(Person pr);
9.       public abstract void beFriend(Person pr, Place pl);
10.      public abstract void getDressed();
11.  }
```

Happyable.java

```
1.   package lib.interf;
2.   import lib.enums.*;
3.   import lib.pers.*;
4.   public interface Happyable {
5.       public abstract void surround(Person prTo, Person... arrPr);
6.       public abstract void impressed(Person prTo, Person... arrPr);
7.   }
```

Pilulkinable.java

```
1.   package lib.interf;
2.   import lib.enums.*;
3.   import lib.pers.*;
4.   import lib.exceptions.*;
5.   public interface Pilulkinable {
6.       public abstract void inspection(Person pr)
7.           throws BadInspectionException;
8.   }
```

Sickable.java

```
1.   package lib.interf;
2.   import lib.enums.*;
3.   public interface Sickable {
4.       public abstract void sick();
5.       public abstract void feelGood();
6.   }
```

Vorchunable.java

```
1.   package lib.interf;
2.   import lib.enums.*;
3.   public interface Vorchunable {
4.       public abstract void grumble();
5.       public abstract void beHappy();
6.   }
```

pers

Personage.java

```
1.   package lib.pers;
2.   import lib.enums.*;
3.   public abstract class Personage {
4.       private String name;
5.       protected Place place;
6.
7.       public Personage(String n, Place p) {
8.           name = n;
9.           place = p;
10.      }
11.
12.      public abstract void setPlace(Place p);
13.
14.      public String getName() { return name; }
15.      public Place getPlace() { return place; }
16.  }
```

Person.java

```
1.   package lib.pers;
2.
3.   import lib.enums.*;
4.   import lib.exceptions.*;
5.
6.   public class Person extends Personage {
7.       protected Rank rank;
8.
9.       // вложенный класс
10.      public enum Disease {
11.          MELANCHOLY("меланхолия");
12.          private String title;
13.          private Disease(String t) { title = t; }
14.          @Override
15.          public String toString() { return title; }
16.      }
17.
18.      private Disease disease = null;
19.
20.      public Person(String n, Place p, Rank r) {
21.          super(n, p);
22.          System.out.println(getName() +
23.              " прибывает в место: " + p.toString());
24.          rank = r;
25.          System.out.println(getName() +
26.              " - " + r.toString());
27.      }
28.
29.      @Override
30.      public void setPlace(Place p) {
31.          place = p;
32.          System.out.println(getName() +
33.              " прибывает в место: " + p.toString());
34.      }
35.
36.      public void setRank(Rank r) {
37.          rank = r;
```

Person.java

```
38.           System.out.println(getName() + " - " + r.toString());
39.       }
40.
41.       protected void setDisease() { disease = Disease.MELANCHOLY; }
42.       protected void removeDisease() { disease = null; }
43.
44.       public Rank getRank() { return rank; }
45.
46.       public String getDiseaseName()
47.       throws MyNullCallException {
48.           if (disease == null) {
49.               throw new MyNullCallException();
50.           }
51.           return disease.toString();
52.       }
53.   }
```

AllStaff.java

```
1.    package lib.pers;
2.
3.    import lib.enums.*;
4.    import lib.additions.Clothes;
5.    import lib.pers.Person;
6.    import java.util.ArrayList;
7.
8.    final public class AllStaff extends Personage {
9.        public AllStaff(Place p) {
10.           super("Весь персонал", p);
11.           System.out.println(getName() +
12.               " прибывает в место: " + p.toString());
13.       }
14.
15.       @Override
16.       public void setPlace(Place p) {
17.           place = p;
18.           System.out.println("Весь персонал" +
19.               " прибывает в место: " + p.toString());
20.       }
21.
22.       public void run(Person pr) {
23.           System.out.println("Весь персонал" +
24.               " бросается в погоню за персоной "+
25.               pr.getName());
26.       }
27.       public void detectEscape(Person pr) {
28.           System.out.println("Весь персонал" +
29.               " обнаруживает исчезновение" +
30.               " персоны " + pr.getName());
31.       }
32.       public int detectClothesIn(Place pl) {
33.           int cnt = 0;
34.           ArrayList<Person> arrPr = new ArrayList<Person>();
35.           Clothes[] arrCl = Clothes.getAllClothes();
36.           for (int i = 0;
37.           i < arrCl.length && arrCl[i] != null; ++i) {
38.               if (pl == arrCl[i].getPlace()) {
39.                   ++cnt;
40.                   arrPr.add(arrCl[i].getPerson());
41.               }
42.           }
43.           System.out.println("Весь персонал" +
44.               " обнаруживает одежду" +
```

AllStaff.java

```
45.                 " в количестве " + cnt);
46.             System.out.print(" Принадлежность одежды:");
47.             for (Person pr : arrPr) {
48.                 System.out.print(" " + pr.getName());
49.             }
50.             System.out.println();
51.             return cnt;
52.         }
53.     }
```

Avoska.java

```
1.   package lib.pers;
2.
3.   import lib.enums.*;
4.
5.   final public class Avoska extends Person {
6.       public Avoska(Place p, Rank r) {
7.           super("Авоська", p, r);
8.       }
9.   }
```

Bulka.java

```
1.   package lib.pers;
2.
3.   import lib.enums.*;
4.
5.   final public class Bulka extends Person {
6.       public Bulka(Place p, Rank r) {
7.           super("Булька", p, r);
8.       }
9.   }
```

Rasteryajka.java

```
1.   package lib.pers;
2.
3.   import lib.enums.*;
4.
5.   final public class Rasteryajka extends Person {
6.       public Rasteryajka(Place p, Rank r) {
7.           super("Растеряйка", p, r);
8.       }
9.   }
```

Vintik.java

```
1.   package lib.pers;
2.
3.   import lib.enums.*;
4.
5.   final public class Vintik extends Person {
6.       public Vintik(Place p, Rank r) {
7.           super("Растеряйка", p, r);
8.       }
9.   }
```

Medunica.java

```
1.  package lib.pers;
2.
3.  import lib.enums.*;
4.
5.  final public class Medunica extends Person {
6.      public Medunica(Place p, Rank r) {
7.          super("Медуница", p, r);
8.      }
9.      public void search(Person pr) {
10.         System.out.println("Медуница " +
11.             "долго разыскивает персону " +
12.             pr.getName());
13.     }
14. }
```

Pilulkin.java

```
1.  package lib.pers;
2.
3.  import lib.enums.*;
4.  import lib.additions.Clothes;
5.  import lib.interf.*;
6.  import java.util.Random;
7.  import lib.exceptions.*;
8.
9.  final public class Pilulkin extends Person
10. implements Pilulkinable, FugitivableDoctor {
11.     public static final double BAD_INSPECT_PROB = 0.20;
12.
13.     // статический вложенный класс
14.     public static class InpectionQuality {
15.         private static Random gen = new Random();
16.
17.         public static boolean wasBad() {
18.             if (gen.nextDouble() < BAD_INSPECT_PROB) {
19.                 System.out.println("Пилюлькин старался, " +
20.                     "но ничего не нашел\nИстории конец, " +
21.                     "кто слушал - молодец");
22.                 return true;
23.             }
24.             else { return false; }
25.         }
26.
27.         public static double getBadProb()
28.             { return BAD_INSPECT_PROB; }
29.     }
30.
31.     public Pilulkin(Place p, Rank r) {
32.         super("Пилюлькин", p, r);
33.     }
34.
35.     // Pilulkinable
36.     @Override
37.     public void inspection(Person pr)
38.     throws BadInspectionException, MyNullCallException {
39.         System.out.println("Пилюлькин осматривает " +
40.             "персону " + pr.getName());
41.
42.         if (InpectionQuality.wasBad()) {
43.             throw new BadInspectionException();
44.         }
45.
46.         String diseaseName;
```

Pilulkin.java

```
47.             try {
48.                 diseaseName = pr.getDiseaseName();
49.             } catch (MyNullCallException e) {
50.                 System.out.println(pr.getName() + " здоров");
51.                 throw e;
52.             }
53.
54.             System.out.print("Результат осмотра: ");
55.             System.out.println("недуг персоны " +
56.                 pr.getName() +
57.                 " – " + diseaseName);
58.             pr.setRank(Rank.PATIENT);
59.         }
60.
61.         // FugitivableDoctor
62.         @Override
63.         public void wannaTeach() {
64.             System.out.println("Пилюлькин ощущает" +
65.                 " безудержное рвение стать преподавателем");
66.         }
67.         @Override
68.         public void makePlane(Person pr, Place pl) {
69.             System.out.println("Пилюлькин и " +
70.                 pr.getName() + " придумали план побега" +
71.                 " из места: " + pl.toString());
72.         }
73.         @Override
74.         public void runOut() {
75.             System.out.println("Пилюлькин сбегает" +
76.                 " в голом виде через окно");
77.         }
78.         @Override
79.         public void hug(Person pr) {
80.             System.out.println("Пилюлькин бросился" +
81.                 " обнимать персону " + pr.getName());
82.         }
83.         @Override
84.         public void beFriend(Person pr, Place pl) {
85.             System.out.println("Пилюлькин и " +
86.                 pr.getName() + " сдружились в месте: " +
87.                 pl.toString());
88.         }
89.         @Override
90.         public void getDressed() {
91.             Clothes[] arrCl = Clothes.getAllClothes();
92.             boolean isTherePilulkinsClothes = false;
93.             for (int i = 0; i < arrCl.length && arrCl[i] != null; ++i) {
94.                 if (arrCl[i].getPlace() == Place.PILULKINs_HOLD) {
95.                     isTherePilulkinsClothes = true;
96.                     break;
97.                 }
98.             }
99.             if (isTherePilulkinsClothes) {
100.                System.out.println("Пилюлькин одевается");
101.            }
102.            else {
103.                System.out.println("Пилюлькин хотел одеться, но не смог");
104.            }
105.        }
106. }
```

Vorchun.java

```
1.  package lib.pers;
2.
3.  import lib.enums.*;
4.  import lib.additions.Clothes;
5.  import lib.interf.*;
6.  import lib.exceptions.*;
7.
8.  final public class Vorchun extends Person
9.  implements Vorchunable, FugitivablePatient {
10.     public Vorchun(Place p, Rank r) {
11.         super("Ворчун", p, r);
12.     }
13.
14.     // анонимный класс
15.     private Sickable health = new Sickable() {
16.         @Override
17.         public void sick() {
18.             setDisease();
19.             System.out.println(getName() + " чем-то болеет");
20.         }
21.
22.         @Override
23.         public void feelGood() {
24.             removeDisease();
25.             System.out.println(getName() +
26.                 " чувствует себя здорово");
27.         }
28.     };
29.
30.     public void setBadHealth() { health.sick(); }
31.     public void setGoodHealth() { health.feelGood(); }
32.
33.     // Vorchunable
34.     @Override
35.     public void grumble() {
36.         System.out.println(getName() +
37.             " порой ворчит");
38.     }
39.     @Override
40.     public void beHappy() {
41.         System.out.println(getName() +
42.             " обрел счастье и улыбается");
43.     }
44.
45.     // FugitivablePatient
46.     @Override
47.     public void feelAddSession() {
48.         System.out.println(getName() +
49.             " ощущает приближение допсы");
50.     }
51.     @Override
52.     public void stealClothes(Clothes... arrCl)
53.     throws MyNullCallException {
54.         if (arrCl.length == 0) {
55.             System.out.println(getName() + " одежды " +
56.                 "не нашел, но искал и хотел украсть");
57.             return;
58.         }
59.         for (Clothes cl : arrCl) {
60.             if (cl.getPerson()==null || cl.getPlace()==null) {
61.                 throw new MyNullCallException();
62.             }
63.             System.out.println(getName() +
64.                 " похищает одежду персоны " +
65.                 cl.getPerson().getName());
```

Vorchun.java

```
66.                 cl.setPlace(Place.VORCHUNs_POCKET);
67.             }
68.         }
69.         @Override
70.         public void runOut() {
71.             System.out.println(getName() + " сбегает спокойно");
72.         }
73.         @Override
74.         public void seat() {
75.             System.out.println(getName() + " сидит, притаившись");
76.         }
77.         @Override
78.         public void look() {
79.             final String sky = "прозрачное синее небо";
80.             final String grass = "свежую зеленую травку";
81.
82.             // локальный класс
83.             class ToLook {
84.                 public String onSmth() {
85.                     return new String(" с наслаждением" +
86.                         " глядит на " + sky +
87.                         ", на " + grass);
88.                 }
89.             }
90.
91.             System.out.println(getName() + (new ToLook()).onSmth());
92.         }
93.         @Override
94.         public void promise(Place pl) {
95.             System.out.println(getName() + " дал клятву" +
96.                 " никогда не ворчать и быть счастливым,\n"
97.                 + " если только не попадет в место: " +
98.                 pl.toString());
99.         }
100.        @Override
101.        public void look(Person pr, Place pl) {
102.            System.out.println(getName() +
103.                " видит, что персона " +
104.                pr.getName() + " скрывается в месте " +
105.                pl.toString());
106.        }
107.        @Override
108.        public void standUp() {
109.            System.out.println(getName() +
110.                " потихоньку выходит из засады");
111.        }
112.        @Override
113.        public void found(Person pr) {
114.            System.out.println(getName() +
115.                " разыскал персону " +
116.                pr.getName());
117.        }
118.        @Override
119.        public void giveClothes(Clothes cl, Person pr, Place pl) {
120.            Clothes[] arrCl = Clothes.getAllClothes();
121.            boolean isThereClothes = false;
122.            for (int i = 0; i < arrCl.length && arrCl[i] != null; ++i) {
123.                if (arrCl[i].getPlace() == Place.VORCHUNs_POCKET &&
124.                cl == arrCl[i]) {
125.                    isThereClothes = true;
126.                    break;
127.                }
128.            }
129.            if (isThereClothes) {
130.                System.out.println(getName() +
```

Vorchun.java

```
131.                 " отдал одежду персоны " +
132.                 cl.getPerson().getName() +
133.                 " персоне " + pr.getName());
134.             cl.setPlace(pl);
135.         }
136.         else {
137.             System.out.println("Пилюлькин хотел одеться, но не смог");
138.         }
139.     }
140. }
```

RunMake

clearAll.sh

```
rm -R ../../lib/*
rmdir ../../lib/
```

makeAll.sh

```
javac -d ../../
-cp ../../ ../*/*.java
```

runLab4.sh

```
java -cp ../../
lib.story.Story
```

story

Story.java

```
1.   package lib.story;
2.
3.   import lib.enums.*;
4.   import lib.pers.*;
5.   import lib.additions.*;
6.   import lib.interf.Happyable;
7.   import lib.exceptions.*;
8.   import java.util.Random;
9.
10.  import java.lang.reflect.*;
11.
12.  public class Story {
13.      public static final double OK_PROB = 0.75;
14.      public static void main(String[] args) {
15.          System.out.println("--------------------");
16.          Random gen = new Random();
17.
18.          // выброс ислючения без обработки
19.          if (gen.nextDouble() > OK_PROB) {
20.              checkUnchecked();
21.          }
22.
23.          AllStaff als = new AllStaff(Place.HOSPITAL);
24.          Pilulkin pil = new Pilulkin(Place.HOSPITAL, Rank.DOCTOR);
25.          Medunica med = new Medunica(Place.HOSPITAL, Rank.NURSE);
26.          Bulka blk = new Bulka(Place.HOSPITAL, Rank.PATIENT);
27.          Clothes clPil = new Clothes(Place.PANTRY, pil);
28.          Clothes clBlk = new Clothes(Place.PANTRY, blk);
29.          Vorchun vrc = new Vorchun(Place.ITMO, Rank.FRESHMAN);
30.
31.          // если болезнь не будет установлена,
32.          // то выбрасывается исключение
33.          if (gen.nextDouble() < OK_PROB) {
34.              vrc.setBadHealth();
```

Story.java

```
35.             vrc.grumble();
36.         }
37.
38.         System.out.println();
39.         // осмотр может выбросить 2 исключения
40.         vrc.setPlace(Place.HOSPITAL);
41.         try {
42.             pil.inspection(vrc);
43.         } catch(BadInspectionException e) {
44.             System.out.println(e.getMessage());
45.             return;
46.         } catch(MyNullCallException e) {
47.             System.out.println(e.getMessage());
48.         }
49.         Clothes clVrc = new Clothes(Place.PANTRY, vrc);
50.         System.out.println();
51.
52.         pil.wannaTeach();
53.         vrc.feelAddSession();
54.         pil.makePlane(vrc, Place.HOSPITAL);
55.         System.out.println();
56.
57.         pil.setRank(Rank.FUGITIVE);
58.         vrc.setRank(Rank.FUGITIVE);
59.         pil.runOut();
60.         pil.setPlace(Place.FOREST);
61.         als.run(pil);
62.         vrc.setPlace(Place.PANTRY);
63.
64.         // поля в классе Clothes инициализируется null
65.         // unchecked => если не писать try-catch,
66.         // компилятор ошибки не выдает
67.         if (gen.nextDouble() < OK_PROB) {
68.             vrc.stealClothes(clVrc, clPil);
69.         }
70.         else {
71.             try {
72.                 Clothes cl = new Clothes();
73.                 vrc.stealClothes(cl);
74.             }
75.             catch(MyNullCallException e) {
76.                 System.out.println(e.getMessage());
77.             }
78.         }
79.
80.         vrc.runOut();
81.         vrc.setPlace(Place.FOREST);
82.         System.out.println();
83.
84.         als.setPlace(Place.PANTRY);
85.         als.detectEscape(vrc);
86.         als.detectClothesIn(Place.PANTRY);
87.         System.out.println();
88.
89.         med.search(vrc);
90.         vrc.setPlace(Place.THICKETS);
91.         vrc.seat();
92.         vrc.look();
93.         vrc.beHappy();
94.         vrc.promise(Place.HOSPITAL);
95.         System.out.println();
96.
97.         vrc.look(med, Place.HOSPITAL);
98.         vrc.standUp();
99.         vrc.setPlace(Place.FOREST);
```

Story.java

```
100.          vrc.found(pil);
101.          vrc.giveClothes(clPil, pil, Place.PILULKINs_HOLD);
102.          System.out.println();
103.
104.          pil.hug(vrc);
105.          pil.beFriend(vrc, Place.HOSPITAL);
106.          pil.getDressed();
107.          pil.setRank(Rank.TEACHER);
108.          vrc.setRank(Rank.FRESHMAN);
109.          pil.setPlace(Place.ITMO);
110.          vrc.setPlace(Place.ITMO);
111.          System.out.println();
112.
113.          Avoska avs = new Avoska(Place.ITMO, Rank.FRESHMAN);
114.          Vintik vnt = new Vintik(Place.ITMO, Rank.FRESHMAN);
115.          Rasteryajka rsk = new Rasteryajka(Place.ITMO, Rank.FRESHMAN);
116.          // локальный, анонимный класс
117.          Happyable happy = new Happyable() {
118.              @Override
119.              public void surround(Person prTo, Person... arrPr) {
120.                  System.out.print(arrPr[0].getName());
121.                  for (int i = 1; i < arrPr.length; ++i) {
122.                      System.out.print(", " +
123.                          arrPr[i].getName());
124.                  }
125.                  System.out.println(" окружили и поздравили" +
126.                      " с возвращением персону " + prTo.getName());
127.              }
128.              @Override
129.              public void impressed(Person prTo, Person... arrPr) {
130.                  System.out.print(arrPr[0].getName());
131.                  for (int i = 1; i < arrPr.length; ++i) {
132.                      System.out.print(", " +
133.                          arrPr[i].getName());
134.                  }
135.                  System.out.println(" удивлены веселым видом" +
136.                      " персоны " + prTo.getName());
137.              }
138.          };
139.          happy.surround(vrc, avs, rsk, vnt);
140.          happy.impressed(vrc, avs, rsk, vnt);
141.
142.          System.out.println("--------------------");
143.
144.          // вложенный класс
145.          System.out.println("\n       innerClass:\t"+
146.              Person.Disease.class.getName() +"\n");
147.          // статический класс
148.          System.out.println("staticNestedClass:\t"+
149.              Pilulkin.InpectionQuality.class.getName() +"\n");
150.          // локальный, анонимный класс
151.          System.out.println("locAnonymousClass:\t"+
152.              happy.getClass().getName() +"\n");
153.      }
154.
155.      public static void checkUnchecked() {
156.          final int tempSize = 19;
157.          Clothes[] tempArrCl = new Clothes[tempSize];
158.          for (int i = 0; i < tempSize; ++i) {
159.              tempArrCl[i] = new Clothes(
160.                  Place.THICKETS,
161.                  new Avoska(Place.ITMO, Rank.FRESHMAN)
162.              );
163.          }
164.          System.out.println("\nClothes size: " +
```

Story.java

```
165.             Clothes.getArrSize()  + "\n");
166.     }
167. }
```

Результат работы программы

Output (without caught exceptions)

```
1.   --------------------
2.   Весь персонал прибывает в место: больница "Укол-Комарика"
3.   Пилюлькин прибывает в место: больница "Укол-Комарика"
4.   Пилюлькин - доктор
5.   Медуница прибывает в место: больница "Укол-Комарика"
6.   Медуница - медсестра
7.   Булька прибывает в место: больница "Укол-Комарика"
8.   Булька - пациент
9.   Одежда персоны Пилюлькин попадает в место: кладовая
10.  Одежда персоны Булька попадает в место: кладовая
11.  Ворчун прибывает в место: ИТМО
12.  Ворчун - малыш-первач
13.  Ворчун чем-то болеет
14.  Ворчун порой ворчит
15.
16.  Ворчун прибывает в место: больница "Укол-Комарика"
17.  Пилюлькин осматривает персону Ворчун
18.  Результат осмотра: недуг персоны Ворчун - меланхолия
19.  Ворчун - пациент
20.  Одежда персоны Ворчун попадает в место: кладовая
21.
22.  Пилюлькин ощущает безудержное рвение стать преподавателем
23.  Ворчун ощущает приближение допсы
24.  Пилюлькин и Ворчун придумали план побега из места: больница "Укол-Комарика"
25.
26.  Пилюлькин - беглец
27.  Ворчун - беглец
28.  Пилюлькин сбегает в голом виде через окно
29.  Пилюлькин прибывает в место: лес
30.  Весь персонал бросается в погоню за персоной Пилюлькин
31.  Ворчун прибывает в место: кладовая
32.  Ворчун похищает одежду персоны Ворчун
33.  Одежда персоны Ворчун попадает в место: карман Ворчуна
34.  Ворчун похищает одежду персоны Пилюлькин
35.  Одежда персоны Пилюлькин попадает в место: карман Ворчуна
36.  Ворчун сбегает спокойно
37.  Ворчун прибывает в место: лес
38.
39.  Весь персонал прибывает в место: кладовая
40.  Весь персонал обнаруживает исчезновение персоны Ворчун
41.  Весь персонал обнаруживает одежду в количестве 1
42.   Принадлежность одежды: Булька
43.
44.  Медуница долго разыскивает персону Ворчун
45.  Ворчун прибывает в место: заросли лопуха
46.  Ворчун сидит, притаившись
47.  Ворчун с наслаждением глядит на прозрачное синее небо, на свежую зеленую травку
48.  Ворчун обрел счастье и улыбается
```

Output (without caught exceptions)

```
49.  Ворчун дал клятву никогда не ворчать и быть счастливым,
50.   если только не попадет в место: больница "Укол-Комарика"
51.
52.  Ворчун видит, что персона Медуница скрывается в месте больница "Укол-Комарика"
53.  Ворчун потихоньку выходит из засады
54.  Ворчун прибывает в место: лес
55.  Ворчун разыскал персону Пилюлькин
56.  Ворчун отдал одежду персоны Пилюлькин персоне Пилюлькин
57.  Одежда персоны Пилюлькин попадает в место: владение Пилюлькина
58.
59.  Пилюлькин бросился обнимать персону Ворчун
60.  Пилюлькин и Ворчун сдружились в месте: больница "Укол-Комарика"
61.  Пилюлькин одевается
62.  Пилюлькин - преподаватель
63.  Ворчун - малыш-первач
64.  Пилюлькин прибывает в место: ИТМО
65.  Ворчун прибывает в место: ИТМО
66.
67.  Авоська прибывает в место: ИТМО
68.  Авоська - малыш-первач
69.  Винтик прибывает в место: ИТМО
70.  Винтик - малыш-первач
71.  Растеряйка прибывает в место: ИТМО
72.  Растеряйка - малыш-первач
73.  Авоська, Растеряйка, Винтик окружили и поздравили с возвращением персону Ворчун
74.  Авоська, Растеряйка, Винтик удивлены веселым видом персоны Ворчун
75.  --------------------
76.
77.         innerClass: lib.pers.Person$Disease
78.
79.  staticNestedClass: lib.pers.Pilulkin$InpectionQuality
80.  locAnonymousClass: lib.story.Story$1
```

Output (with BadInspectionException)

```
--------------------
Весь персонал прибывает в место: больница "Укол-Комарика"
Пилюлькин прибывает в место: больница "Укол-Комарика"
Пилюлькин - доктор
Медуница прибывает в место: больница "Укол-Комарика"
Медуница - медсестра
Булька прибывает в место: больница "Укол-Комарика"
Булька - пациент
Одежда персоны Пилюлькин попадает в место: кладовая
Одежда персоны Булька попадает в место: кладовая
Ворчун прибывает в место: ИТМО
Ворчун - малыш-первач

Ворчун прибывает в место: больница "Укол-Комарика"
Пилюлькин осматривает персону Ворчун
Пилюлькин старался, но ничего не нашел
Истории конец, кто слушал - молодец
 ::::::::::::::::::::::::::::::
 Плохой осмотр
 ::::::::::::::::::::::::::::::
```

Выводы

Изучил и закрепил на практике следующие знания и навыки: обработка исключительных ситуаций, три типа исключений; вложенные, локальные и анонимные классы; механизм рефлексии (**reflection**) в **Java**, класс **Class**.