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 №3
по курсу "Программирование"

Группа: **P3131**
Студент: Друян Эдуард Андреевич
Вариант: **3108**
GitHub

Санкт-Петербург
2021 г.

Текст задания

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

Медуница еще долго разыскивала Ворчуна с похищенной им одеждой, и, пока шли поиски, Ворчун сидел, притаившись, в зарослях лопуха. Хотя сидение в лопухах не такое уж веселое дело, но Ворчун был вне себя от радости, что вырвался на свободу. Он с наслаждением глядел на прозрачное синее небо, на свежую зеленую 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 попадет снова в больницу.

Программа должна удовлетворять следующим требованиям:

1. Доработанная модель должна соответствовать принципам SOLID.
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`).

SOLID концепция заключается в следующем:

1. Принцип единственной ответственности: "У класса не должно быть более одной причины для изменения". Другими словами, у каждого класса должна быть только одна ответственность.

2. Принцип открытости-закрытости: "Программные объекты ... должны быть открыты для расширения, но закрыты для модификации".

3. Принцип подстановки Лискова: "Функции, использующие указатели или ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом". См. Также проектирование по контракту.

4. Принцип разделения интерфейсов: "Многие клиентские интерфейсы лучше, чем один интерфейс общего назначения".

5. Принцип инверсии зависимостей: "Положитесь на абстракции, а не на конкреции".

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.

Отчет 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.

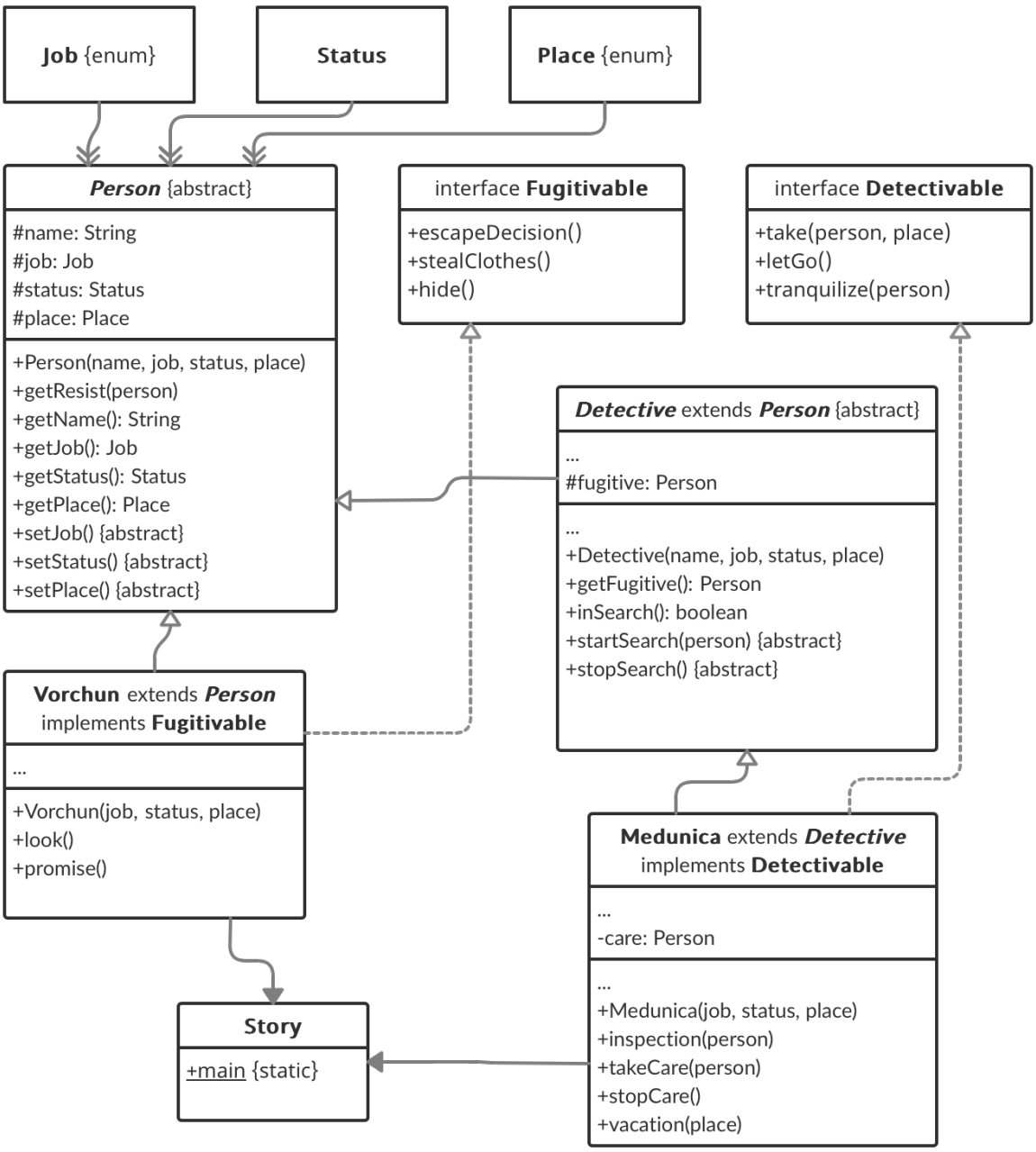
TASK

Медуница еще долго разыскивала Ворчуна с похищенной им одеждой,
и, пока шли поиски, Ворчун сидел, притаившись, в зарослях лопуха.
Хотя сидение в лопухах не такое уж веселое дело,
но Ворчун был вне себя от радости, что вырвался на свободу.
Он с наслаждением глядел на прозрачное синее небо,
на свежую зеленую 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 попадет снова в больницу.

REQUIREMENTS

- SOLID
- 2 interfaces
- 1 abstract class
- 1 enum
- equals()
- toString()
- hashCode()

Диаграмма классов объектной модели



Исходный код программы

ENUMS

Job.java

```
1.   package prog1.lab3.enums;
2.   import java.util.Random;
3.
4.   public enum Job {
5.       DOCTOR("врач"),
6.       FRESHMAN("первач"),
7.       PATIENT("пациент"),
8.       RESISTANT_FRESHMAN("стрессоустойчивый первач");
9.
10.      private String title;
11.
12.      Job(String t) { title = t; }
13.
14.      @Override
15.      public String toString() { return title; }
16.  }
```

Status.java

```
1.   package prog1.lab3.enums;
2.   import java.util.Random;
3.
4.   public enum Status {
5.       SERENITY("чувствует легкую безмятежность"),
6.       SICK("ворчит, недосыпает, страдает меланхолией"),
7.       ALERT("тревога оттого, что сбежавшего может укусить волчок за бочок"),
8.       TAN("хочет красивый загар"),
9.       HAPPY("вне себя от радости, даже улыбается");
10.
11.      private String title;
12.
13.      Status(String t) { title = t; }
14.
15.      @Override
16.      public String toString() { return title; }
17.  }
```

Place.java

```java
package prog1.lab3.enums;
import java.util.Random;

public class Place {
    public static final String h1 = "в больнице BloodyHospital";
    public static final String h2 = "в больницу BloodyHospital";
    public static final String i1 = "в ИТМО на ИВТ";
    public static final String i2 = "в ИТМО на ИВТ";
    public static final String f1 = "в лесочке";
    public static final String f2 = "в лесочек";
    public static final String t1 = "в зарослях лопуха";
    public static final String t2 = "в заросли лопуха";
    public static final String e1 = "в Египте";
    public static final String e2 = "в Египет";
    private String where;
    private String to;

    public Place(String w, String t) {
        where = w;
        to = t;
    }
    public String getWhere() { return where; }
    public String getTo() { return to; }

    public static Place HOSPITAL = new Place(h1, h2);
    public static Place ITMO = new Place(i1, i2);
    public static Place FOREST = new Place(f1, f2);
    public static Place THICKETS = new Place(t1, t2);
    public static Place EGYPT = new Place(e1, e2);

    @Override
    public String toString() { return to; }
    @Override
    public int hashCode() {
        return 1 * to.hashCode() + 2 * where.hashCode();
    }
    @Override
    public boolean equals(Object p) {
        if (p == null) { return false; }
        if (this == p) { return true; }
        if ( !(p instanceof Place) ) { return false; }
        return ( (Place)p ).to.equals(to)
            && ( (Place)p ).where.equals(where);
    }

}
```

ABSTRACT-CLASSES

Person.java

```java
package prog1.lab3.abstract_classes;
import prog1.lab3.enums.*;
import java.lang.StringBuilder;

public abstract class Person {
    public static final double GETRESIST_PROBABILITY = 0.70;

    protected String name;
    protected Job job;
    protected Status status;
    protected Place place;

    public Person(String n, Job j, Status s, Place p) {
        name = n;
        job = j;
        status = s;
        place = p;
    }
    public void getResist(Person person) {
        System.out.println(this.getName()
+ " ощущает сопротивление от персоны " + person.getName() + ".");
    }
    public String getName() { return name; }
    public Job getJob() { return job; }
    public Status getStatus() { return status; }
    public Place getPlace() { return place; }

    public abstract void setJob(Job j);
    public abstract void setStatus(Status s);
    public abstract void setPlace(Place p);
}
```

Detective.java

```java
package prog1.lab3.abstract_classes;
import prog1.lab3.enums.*;
import prog1.lab3.abstract_classes.Person;
import java.lang.StringBuilder;

public abstract class Detective extends Person {
    public static final double FINDING_PROBABILITY = 0.50;

    protected Person fugitive = null;

    public Detective(String n, Job j, Status s, Place p) {
        super(n, j, s, p);
    }
    public Person getFugitive() { return fugitive; }
    public boolean inSearch() {
        if (fugitive == null) { return false; }
        return true;
    }

    public abstract void startSearch(Person fugitive);
    public abstract void stopSearch();
}
```

INTERFACES

Detectivable.java

```java
package prog1.lab3.interfaces;
import prog1.lab3.abstract_classes.Person;
import prog1.lab3.enums.*;

public interface Detectivable {
    public void take(Person fugitive, Place p);
    public void letGo();
    public void tranquilize(Person fugitive);
}
```

Fugitivable.java

```java
package prog1.lab3.interfaces;
import prog1.lab3.abstract_classes.Person;
import prog1.lab3.enums.*;

public interface Fugitivable {
    public void escapeDecision();
    public void stealClothes();
    public void hide();
}
```

HEROES

Medunica.java

```
1.  package prog1.lab3.heroes;
2.  import prog1.lab3.enums.*;
3.  import prog1.lab3.interfaces.*;
4.  import prog1.lab3.abstract_classes.*;
5.  import java.lang.StringBuilder;
6.  import java.io.PrintStream;
7.  import java.util.Random;
8.
9.  public class Medunica extends Detective implements Detectivable {
10.     public static PrintStream so = System.out;
11.     public static final double LETGO_PROBABILITY = 0.50;
12.     public static final double VACATION_PROBABILITY = 0.60;
13.
14.     private Person care;
15.
16.     public Medunica(Job j, Status s, Place p) {
17.         super("Медуница", j, s, p);
18.         so.print(super.name + " ");
19.         so.println("занимает должность: "
20.             + super.job.toString() + " "
21.             + super.place.getWhere() + ".");
22.         so.println("Статус Медуницы: "
23.             + super.status.toString() + ".");
24.         care = null; // можно удалить
25.     }
26.
27.     public void inspection(Person person) {
28.         so.println("Медуница проводит осмотр персоны "
29.             + person.getName() + ".");
30.         if (person.getStatus() == Status.SICK) {
31.             so.println("Медуница диагностирует меланхолию у персоны "
32.                 + person.getName() + ".");
33.             this.takeCare(person);
34.         }
35.         else if (person.getStatus() == Status.HAPPY) {
36.             so.println("Медуница замечает безмерное счастье "
37.                 + "у персоны " + person.getName() + ".");
38.         }
39.     }
40.     public void takeCare(Person person) {
41.         so.println("Медуница позаботится о персоне "
42.             + person.getName() + ".");
43.         care = person;
44.     }
45.     public void stopCare() {
46.         if (care == null) { return; }
47.         so.println("Медуница прекращает заботу о персоне "
48.             + care.getName() + ".");
49.         care = null;
50.     }
51.     public void vacation(Place p) {
52.         so.println("Медуница решается на отдых "
53.             + p.getWhere() + ".");
54.         this.setStatus(status.TAN);
55.         this.setPlace(p);
56.     }
```

```
57.      @Override
58.      public void setJob(Job j) {
59.          so.println("Новая должность Медуницы: "
60.              + j.toString() + ".");
61.          super.job = j;
62.      }
63.      @Override
64.      public void setStatus(Status s) {
65.          so.println("Новый статус Медуницы: "
66.              + s.toString() + ".");
67.          super.status = s;
68.      }
69.      @Override
70.      public void setPlace(Place p) {
71.          so.println("Медуница прибывает "
72.              + p.getTo() + ".");
73.          super.place = p;
74.      }
75.
76.      @Override
77.      public void startSearch(Person fugitive) {
78.          if (super.fugitive != null) { return; }
79.          so.println("Медуница начинает поиск персоны "
80.              + fugitive.getName() + ".");
81.          super.fugitive = fugitive;
82.          this.setStatus(Status.ALERT);
83.          this.setPlace(Place.FOREST);
84.      }
85.      @Override
86.      public void stopSearch() {
87.          if (!this.inSearch()) { return; }
88.
89.          Random generator = new Random();
90.          double chance = generator.nextDouble();
91.          if (chance < super.FINDING_PROBABILITY) {
92.              so.println("Медуница после долгих поисков "
93.                  + "все-таки нашла персону "
94.                  + super.fugitive.getName() + ".");
95.
96.              chance = generator.nextDouble();
97.              if (chance < LETGO_PROBABILITY) {
98.                  this.letGo();
99.                  this.stopCare();
100.
101.                 this.setPlace(Place.HOSPITAL);
102.                 super.fugitive.setPlace(Place.ITMO);
103.                 super.fugitive.setJob(Job.RESISTANT_FRESHMAN);
104.             }
105.             else {
106.                 this.take(super.fugitive, Place.HOSPITAL);
107.
108.                 chance = generator.nextDouble();
109.                 if (chance < Person.GETRESIST_PROBABILITY) {
110.                     super.getResist(super.fugitive);
111.                     this.tranquilize(super.fugitive);
112.                 }
113.
114.                 this.setPlace(Place.HOSPITAL);
115.                 this.setStatus(Status.SERENITY);
116.                 super.fugitive.setPlace(Place.HOSPITAL);
117.                 super.fugitive.setStatus(Status.SICK);
118.             }
119.         }
```

```
120.         else {
121.             so.println("Медуница бросила поиски персоны "
122.                 + super.fugitive.getName() + ".");
123.             this.stopCare();
124.             this.setPlace(Place.HOSPITAL);
125.
126.             chance = generator.nextDouble();
127.             if (chance < VACATION_PROBABILITY)
128.                 { this.vacation(Place.EGYPT); }
129.
130.             super.fugitive.setPlace(Place.ITMO);
131.             super.fugitive.setJob(Job.RESISTANT_FRESHMAN);
132.         }
133.         super.fugitive = null;
134.     }
135.
136.     @Override
137.     public void take(Person fugitive, Place p) {
138.         so.println("Медуница забирает персону "
139.             + fugitive.getName() + " "
140.             + p.getTo() + ".");
141.     }
142.     @Override
143.     public void letGo() {
144.         so.println("Медуница отпускает персону "
145.             + fugitive.getName() + ".");
146.     }
147.     @Override
148.     public void tranquilize(Person fugitive) {
149.         so.println("Медуница всаживает транквилизатор в персону "
150.             + fugitive.getName() + ".");
151.     }
152. }
```

Vorchun.java

```java
package prog1.lab3.heroes;
import prog1.lab3.enums.*;
import prog1.lab3.interfaces.*;
import prog1.lab3.abstract_classes.Person;
import java.lang.StringBuilder;
import java.io.PrintStream;

public class Vorchun extends Person implements Fugitivable {
    public static PrintStream so = System.out;

    public Vorchun(Job j, Status s, Place p) {
        super("Ворчун", j, s, p);
        so.print(super.name + " ");
        so.println("занимает должность: "
            + super.job.toString() + " "
            + super.place.getWhere() + ".");
        so.println("Статус Ворчуна: "
            + super.status.toString() + ".");
    }
    public void look() {
        so.println("Ворчун с наслаждением глядит "
            + "на свежую зеленую траву и "
            + "на прозрачное синее небо.");
        this.setStatus(Status.HAPPY);
    }
    public void promise() {
        so.println("Ворчун пообещал себе "
        + "никогда не ворчать и быть довольным "
        + "всем на свете, если только вновь "
        + "не окажется в больнице BloodyHospital.");
    }

    @Override
    public void setJob(Job j) {
        so.println("Новая должность Ворчуна: "
            + j.toString() + ".");
        super.job = j;
    }
    @Override
    public void setStatus(Status s) {
        so.println("Новый статус Ворчуна: "
            + s.toString() + ".");
        super.status = s;
    }
    @Override
    public void setPlace(Place p) {
        so.println("Ворчун прибывает " + p.getTo() + ".");
        super.place = p;
    }

    public void escapeDecision() {
        so.println("Ворчун решается на побег.");
    }
    public void stealClothes() {
        so.println("Ворчун крадет одежду.");
    }
    public void hide() {
        so.println("Ворчун присаживается и притаивается.");
    }
}
```

Результат работы программы

Один, из возможных вариантов:

```
--------------------
Медуница занимает должность: врач в больнице BloodyHospital.
Статус Медуницы: чувствует легкую безмятежность.
Ворчун занимает должность: первач в ИТМО на ИВТ.
Статус Ворчуна: ворчит, недосыпает, страдает меланхолией.
Ворчун прибывает в больницу BloodyHospital.
Медуница проводит осмотр персоны Ворчун.
Медуница диагностирует меланхолию у персоны Ворчун.
Медуница позаботится о персоне Ворчун.
Новая должность Ворчуна: пациент.

Ворчун решается на побег.
Ворчун крадет одежду.
Ворчун прибывает в лесочек.
Медуница начинает поиск персоны Ворчун.
Новый статус Медуницы: тревога оттого, что сбежавшего может укусить волчок за
бочок.
Медуница прибывает в лесочек.
Ворчун прибывает в заросли лопуха.
Ворчун присаживается и притаивается.
Ворчун с наслаждением глядит на свежую зеленую траву и на прозрачное синее небо.
Новый статус Ворчуна: вне себя от радости, даже улыбается.

Ворчун пообещал себе никогда не ворчать и быть довольным всем на свете, если
только вновь не окажется в больнице BloodyHospital.
Медуница после долгих поисков все-таки нашла персону Ворчун.
Медуница забирает персону Ворчун в больницу BloodyHospital.
Медуница ощущает сопротивление от персоны Ворчун.
Медуница всаживает транквилизатор в персону Ворчун.
Медуница прибывает в больницу BloodyHospital.
Новый статус Медуницы: чувствует легкую безмятежность.
Ворчун прибывает в больницу BloodyHospital.
Новый статус Ворчуна: ворчит, недосыпает, страдает меланхолией.
--------------------
```

Выводы по работе

Закрепил знания о принципах объектно-ориентированного программирования SOLID и STUPID, о классе Object и о реализации его методов по умолчанию, об особенностях реализации наследования в Java, о простом и множественном наследовании, о понятие абстрактного класса, о модификаторе abstract, о понятии интерфейса и его реализации в Java, о методах по умолчанию, о перечисляемом типе данных (enum) в Java, о модификаторах static и final, о перегрузке и переопределении методов.