Университет ИТМО

Направление СППО

Лабораторная работа №3 по Программированию

Преподаватель: Горбунов Михаил Витальевич

Выполнил: Потапова Вера Сергеевна
Группа: P3110
Вариант: 10401

Санкт-Петербург
2020

Задание:

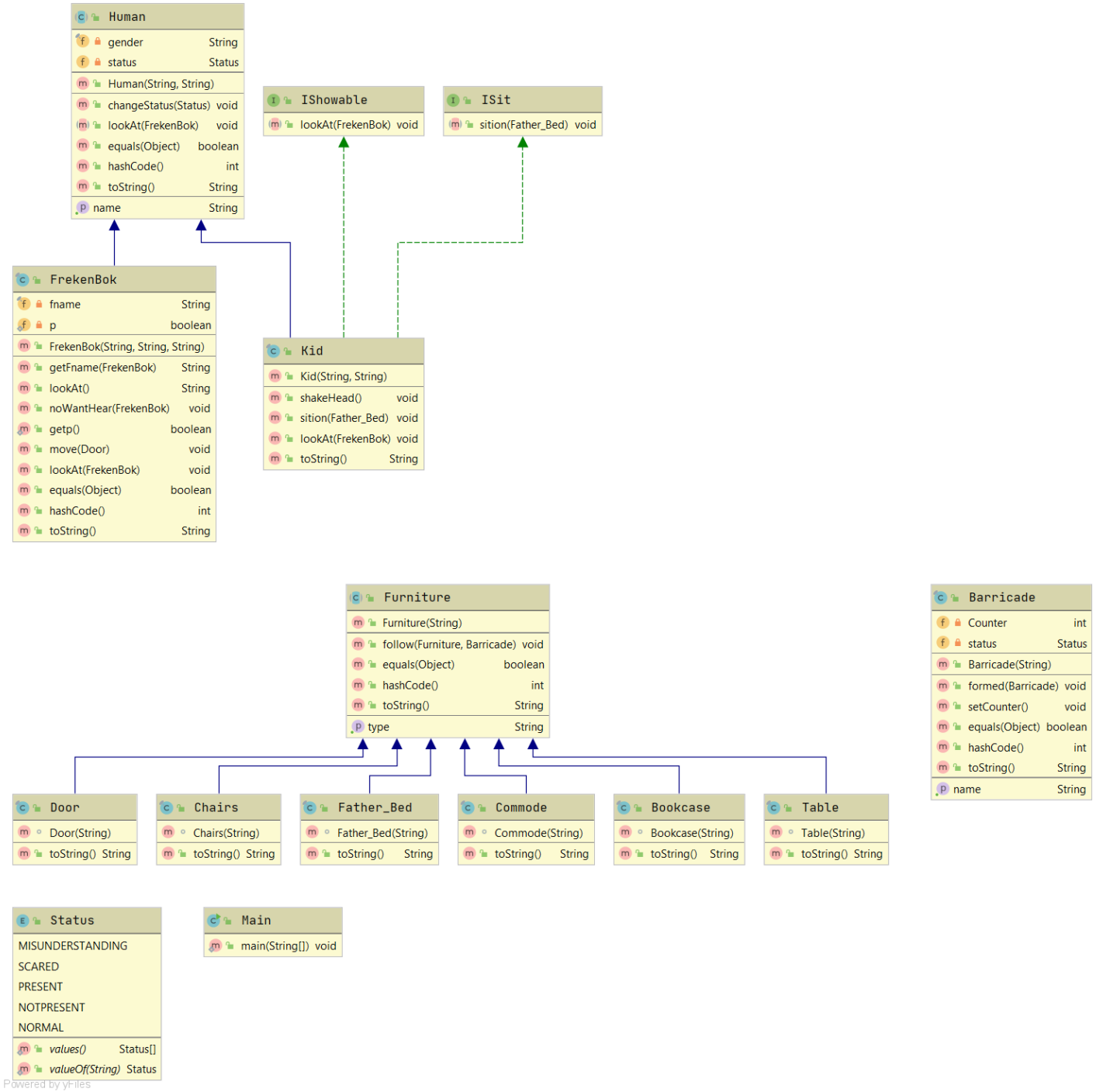
Описание предметной области, по которой должна быть построена объектная модель:

> Малыш никак не мог этого понять, он сел на папину кровать, поглядел на перепуганную фрекен Бок и покачал головой. Но сейчас фрекен Бок и слышать не хотела о Фриде. Она продолжала придвигать всю мебель к двери -- за комодом последовали стол, стулья и этажерка. Перед столом образовалась уже настоящая баррикада.

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).

Диаграмма классов:



Код:

Main.java

```
public class Main {
    public static void main(String[] args) {
        Kid kid = new Kid( name: "Malysh",  gender: "male");
        FrekenBok hildur = new FrekenBok( name: "Freken Bok",  gender: "female",  _fname: "Hildur");
        FrekenBok frida = new FrekenBok( name: "Freken Bok",  gender: "female",  _fname: "Frida");

        Father_Bed bed = new Father_Bed( type: "father's bed");
        Door door = new Door( type: "door");
        Commode commode = new Commode( type: "chest of drawers");
        Chairs chairs = new Chairs( type: "chairs");
        Bookcase bookcase = new Bookcase( type: "bookcase");
        Table table = new Table( type: "table");
        Barricade barricade = new Barricade( name: "barricade");
        kid.changeStatus(Status.MISUNDERSTANDING);
        kid.sition(bed);
        hildur.changeStatus(Status.SCARED);
        kid.lookAt(hildur);
        kid.shakeHead();
        hildur.noWantHear(frida);
        hildur.move(door);
        commode.follow(door, barricade);
        table.follow(door, barricade);
        chairs.follow(door, barricade);
        bookcase.follow(door, barricade);
        barricade.formed(barricade);
    }
}
```

ISit.java

```
public interface ISit {
    void sition (Father_Bed bed);
}
```

IShowable.java

```
public interface IShowable {
    void lookAt (FrekenBok hildur);
}
```

Human.java

```java
import java.util.Objects;

public abstract class Human  {
    private final String name;
    private final String gender;

    private Status status = Status.NORMAL;
    public Human(String name, String gender){
        this.name = name;
        this.gender = gender;
    }
    public String getName(){ return name; }
    public void changeStatus(Status status){
        String msg = "";
        if (status == Status.NORMAL){
            msg = "normal";
        }else if (status == Status.MISUNDERSTANDING){
            msg = "misunderstanding";
        }else if (status == Status.SCARED){
            msg = "scared";
        }
        System.out.println(toString() + " change status to " + msg + '.');
        this.status = status;
    }

    public abstract void lookAt(FrekenBok hildur);

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Human human = (Human) o;
        return Objects.equals(name, human.name) &&
                Objects.equals(gender, human.gender) &&
                status == human.status;
    }

    @Override
    public int hashCode() { return Objects.hash(name, gender, status); }

    @Override
    public String toString() {
        return "Human{" +
                "name='" + name + '\'' +
                ", gender='" + gender + '\'' +
                ", status=" + status +
                '}';
    }
}
```

Kid.java

```
public final class Kid extends Human implements ISit, IShowable{

    public Kid (String name, String gender){
        super(name, gender);
        System.out.println("Human - " + name + " was successfully created.");
    }

    public void shakeHead() { System.out.println(getName() + " shook his head."); }

    @Override
    public void sition(Father_Bed bed) { System.out.println(getName() + " sat on the " + bed.getType() + '.'); }

    @Override
    public void lookAt(FrekenBok hildur) { System.out.println(getName() + " looked at " + hildur.lookAt() + '.'); }

    @Override
    public String toString() { return getName(); }

}
```

FrekenBok.java

```
import java.util.Objects;

public final class FrekenBok extends Human {
    private final String fname;
    private static boolean p;

    public FrekenBok(String name, String gender, String _fname){
        super(name, gender);
        fname = _fname;
        System.out.println("Human - " + name + ' ' + _fname + " was successfully created.");
    }
    public String getFname(FrekenBok k) { return k.fname; }

    public String lookAt() { return ("scared " + getName()); }

    public void noWantHear(FrekenBok k){
        System.out.println("Now " + getName() + " didn't want to hear about " + getFname(k) + '.');
    }

    public static boolean getp() { return p; }

    public void move (Door door){
            p = true;
            System.out.println(getName() + " continued to move all the furniture to the " + door.getType() + ".");
    }
```

```
    @Override
    public void lookAt(FrekenBok hildur) {

    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        FrekenBok frekenBok = (FrekenBok) o;
        return Objects.equals(fname, frekenBok.fname);
    }

    @Override
    public int hashCode() { return Objects.hash(fname); }

    @Override
    public String toString() { return "Freken Bok"; }

}
```

Status.java

```
public enum Status {
    MISUNDERSTANDING,
    SCARED,
    PRESENT,
    NOTPRESENT,
    NORMAL
}
```

Furniture.java

```
import java.util.Objects;

public abstract class Furniture {
    private final String type;
    public Furniture(String type) { this.type = type; }
    public String getType(){ return type; }
    public void follow(Furniture x, Barricade b){
        if (FrekenBok.getp()){
            b.setCounter();
            System.out.println(getType() + " followed to " + x.getType() + ".");
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Furniture furniture = (Furniture) o;
        return Objects.equals(type, furniture.type);
    }

    @Override
    public int hashCode() { return Objects.hash(type); }

    @Override
    public String toString() {
        return "Furniture{" +
                "type='" + type + '\'' +
                '}';
    }
}
```

Bookcase.java

```java
public final class Bookcase extends Furniture{
    Bookcase (String type) { super(type); }

    @Override
    public String toString() { return "bookcase"; }
}
```

Chairs.java

```java
public final class Chairs extends Furniture{
    Chairs (String type) { super(type); }

    @Override
    public String toString() { return "chairs"; }
}
```

Commode.java

```java
public final class Commode extends Furniture{
    Commode (String type) { super(type); }

    @Override
    public String toString() { return "commode"; }
}
```

Door.java

```java
public final class Door extends Furniture {
    Door (String type) { super(type); }

    @Override
    public String toString() { return "door"; }
}
```

Father_Bed.java

```java
public final class Father_Bed extends Furniture{
    Father_Bed(String type) { super(type); }

    @Override
    public String toString() { return "father's bed"; }
}
```

Tadle.java

```java
public final class Table extends Furniture{
    Table(String type) { super(type); }

    @Override
    public String toString() { return "table"; }
}
```

Barricade.java

```java
import java.util.Objects;

public final class Barricade {
    private int Counter;
    private final String name;
    private Status status;

    public Barricade (String name) { this.name = name; }

    public String getName(){return name;}

    public void formed(Barricade b){
        String msg = "";
        if (b.Counter > 3){
            b.status = Status.PRESENT;
        } else {
            b.status = Status.NOTPRESENT;
        }
        if (b.status == Status.NOTPRESENT){
            msg = "fake";
        }
        if (b.status == Status.PRESENT){
            msg = "present";
        }
        System.out.println("A " + msg + " " + getName() + " was formed.");
    }

    public void setCounter() { Counter += 1; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Barricade barricade = (Barricade) o;
        return Objects.equals(name, barricade.name) &&
                status == barricade.status;
    }

    @Override
    public int hashCode() { return Objects.hash(name, status); }

    @Override
    public String toString() {
        return "Barricade{" +
                "name='" + name + '\'' +
                ", status=" + status +
                '}';
    }
}
```

Результат работы программы:

```
Human - Malysh was successfully created.
Human - Freken Bok Hildur was successfully created.
Human - Freken Bok Frida was successfully created.
Malysh change status to misunderstanding.
Malysh sat on the father's bed.
Freken Bok change status to scared.
Malysh looked at scared Freken Bok.
Malysh shook his head.
Now Freken Bok didn't want to hear about Frida.
Freken Bok continued to move all the furniture to the door.
chest of drawers followed to door.
table followed to door.
chairs followed to door.
bookcase followed to door.
A present barricade was formed.
```

Вывод:

В ходе выполнения данной лабораторной работы я научилась создавать объектно-ориентированную модель на языке Java. Узнала, что такое интерфейсы и абстрактные классы и в чём их различие. Поняла, в чём заключаются принципы SOLID. А так же научилась пользоваться системой сборки Gradle.