НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Программирование

Лабораторная работа № 3

Выполнил студент

Хвостова Ирина Леонидовна

Группа № Р3124

Преподаватель: Харитонова Анастасия Евгеньевна

г. Санкт-Петербург

2022

**Вариант 4254**

**Программа должна удовлетворять следующим требованиям:**

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

**Порядок выполнения работы:**

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

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

Однако на этой бутылочке не было ни черепа, ни костей, ни надписи "Яд!", и Алиса рискнула попробовать ее содержимое. А так как оно оказалось необыкновенно вкусным (на вкус - точь-в-точь смесь вишневого пирога, омлета, ананаса, жареной индюшки, тянучки и горячих гренков с маслом), она сама не заметила, как пузырек опустел.

Исходный код:

public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("\"Алиса в стране чудес\"");  
  
 Moonlighter alice = new Moonlighter("Алиса", 100, 100, 200);  
 Liquid flea = new Liquid(100, 100, 300);  
  
 alice.feel();  
 flea.potionSelection(alice);  
 flea.potionDelicious(alice);  
  
 Taste thisTaste = new Taste();  
 thisTaste.goodTaste();  
 alice.overLook();  
 flea.potionOver(alice);  
 }  
}

Блок интерфейсов:

* DefinitionOfTaste

public interface DefinitionOfTaste {  
 */\*\*  
 \* Определение вкуса  
 \*/* public void goodTaste();  
}

* Insects

public interface Insects {  
 */\*\*  
 \* Метод, который определяет выбор зелья  
 \** ***@Personage*** *p - персонаж  
 \** ***@return*** *true - в случае успеха, false - иначе  
 \*/* public boolean potionSelection(Personage p);  
 */\*\*  
 \* Метод, в котором зелье оказывается вкусным  
 \*/* public void potionDelicious(Personage p);  
  
 */\*\*  
 \* Метод, в котором зелье заканчивается  
 \*/* void potionOver(Personage p);  
}

Блок абстрактного класса:

* Personage

public abstract class Personage {  
 String name = "NoName";  
 String type = "Персонаж";  
 double x = 0;  
 double y = 0;  
 int force;  
 int height;  
 Mood mood = Mood.*NORMAL*;  
  
 @Override  
 public boolean equals(Object s) {  
 if (s == null) {  
 return false;  
 }  
 if (this == s) {  
 return true;  
 }  
 if (!(getClass() == s.getClass())){  
 return false;  
 }  
 else {  
 Personage tmp = (Personage) s;  
 return (tmp.name.equals(name) && force == tmp.force && mood == tmp.mood && tmp.type.equals(type));  
 }  
 }  
  
 @Override  
 public int hashCode(){  
 final int power = 239;  
 int hash = 0;  
 for (int i = 0; i < name.length(); ++i) {  
 hash = hash \* power + (int)name.charAt(i);  
 }  
  
 for (int i = 0; i < type.length(); ++i) {  
 hash = hash \* power + (int)type.charAt(i);  
 }  
  
  
 if (mood == Mood.*NORMAL*) {  
 hash \*= 2;  
 } else if (mood == Mood.*FURY*) {  
 hash \*= 3;  
 } else if (mood == Mood.*HAPPY*) {  
 hash \*= 4;  
 } else if (mood == Mood.*SAD*) {  
 hash \*= 5;  
 }  
  
 hash \*= force \* x \* y \* height;  
 return hash;  
 }  
  
 @Override  
 public String toString(){  
 return type + " " + name ;  
 }  
  
 */\*\*  
 \* Кто-то что-то почувствовал  
 \*/* public void feel(){  
 System.*out*.print(name + " почувствовал, что ");  
 }  
}

Блок enum:

* Drawings

enum Drawings {  
 *SKULL*,  
 *BONES*,  
 *POISON*}

* Savor

enum Savor {  
 *TASTELESS*,  
 *TASTY*,  
 *UNSAVORY*}

Блок других классов:

* Liquid

public class Liquid implements Insects{  
 String name = "Зелье";  
 String type = "Жидкость";  
 int force = 1;  
 double x = 0;  
 double y = 0;  
 double z = 0;  
  
 Liquid(double x, double y, double z) {  
 String name = "Зелье";  
 String type = "Жидкость";  
 int force = 9;  
 this.x = x;  
 this.y = y;  
 this.z = z;  
 }  
  
 @Override  
 public boolean potionSelection(Personage p) {  
 if (this.x == p.x && this.y == p.y && this.z >= p.height) {  
 System.*out*.println(this.name + " без надписей костей, черепов, яда.");  
 this.z = 6 \* p.height / 7;  
 return true;  
 }  
 else {  
 System.*out*.println(this.name + " с надписями костей, черепов, яда.");  
 this.z = 0;  
 return false;  
 }  
 }  
  
 @Override  
 public void potionDelicious(Personage p) {  
 p.force--;  
 System.*out*.println(this.name + " оказалось необыкновенно вкусным.");  
 }  
 @Override  
 public void potionOver(Personage p) {  
 p.force--;  
 System.*out*.println(this.name + " закончилось.");  
 }  
  
 @Override  
 public boolean equals(Object s) {  
 if (s == null) {  
 return false;  
 }  
 if (this == s) {  
 return true;  
 }  
 if (!(getClass() == s.getClass())){  
 return false;  
 }  
 else {  
 Moonlighter tmp = (Moonlighter) s;  
 return (tmp.name.equals(this.name) && this.force == tmp.force && this.type.equals(tmp.type));  
 }  
 }  
  
 @Override  
 public int hashCode(){  
 final int power = 239;  
 int hash = 0;  
 for (int i = 0; i < name.length(); ++i){  
 hash = hash \* power + (int)name.charAt(i);  
 }  
  
 for (int i = 0; i < type.length(); ++i){  
 hash = hash \* power + (int)type.charAt(i);  
 }  
  
 hash \*= force \* x \* y \* z;  
 return hash;  
 }  
  
 @Override  
 public String toString(){  
 return this.type + " " + this.name;  
 }  
}

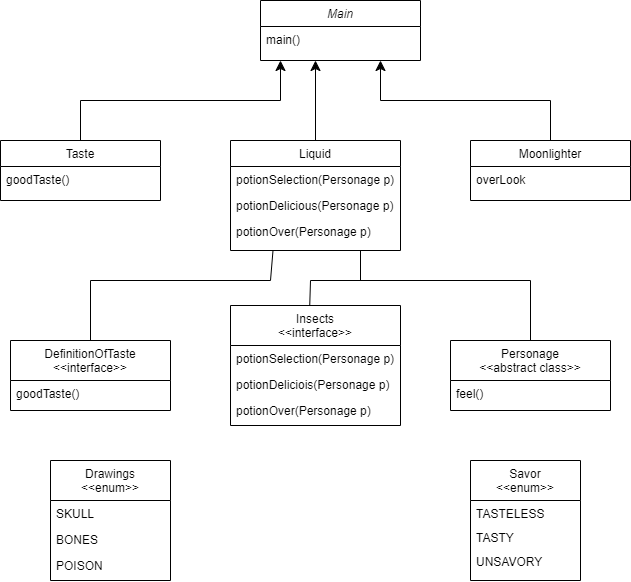
* Moonlighter

public class Moonlighter extends Personage {  
 String type = "Алиса";  
  
 Moonlighter(String name) {  
 this.name = name;  
 this.type = "Алиса";  
 }  
  
 Moonlighter(String name, double x, double y, int height) {  
 this.name = name;  
 this.type = "Алиса";  
 this.x = x;  
 this.y = y;  
 this.height = height;  
 }  
  
 public void overLook() {  
 System.*out*.println(this.name + " не заметила, как");  
 }  
  
  
}

* Taste

public class Taste implements DefinitionOfTaste {  
 Savor sound = Savor.*TASTELESS*;  
 String type = "Вкус";  
  
 @Override  
 public void goodTaste() {  
 this.sound = Savor.*TASTY*;  
 System.*out*.println("На вкус - точь-в-точь смесь вишневого пирога, омлета, ананаса, жареной индюшки, тянучки и горячих гренков с маслом");  
 }  
  
 @Override  
 public int hashCode(){  
 final int power = 239;  
 int hash = 0;  
  
 for (int i = 0; i < type.length(); ++i) {  
 hash = hash \* power + (int)type.charAt(i);  
 }  
  
 if (sound == Savor.*TASTELESS*) {  
 hash \*= 2;  
 } else if (sound == Savor.*TASTY*) {  
 hash \*= 3;  
 } else if (sound == Savor.*UNSAVORY*) {  
 hash \*= 4;  
 }  
 return hash;  
 }  
  
 @Override  
 public boolean equals(Object s) {  
 if (s == null) {  
 return false;  
 }  
 if (this == s) {  
 return true;  
 }  
 if (!(getClass() == s.getClass())){  
 return false;  
 }  
 else {  
 Taste tmp = (Taste) s;  
 return (sound == tmp.sound && tmp.type.equals(type));  
 }  
 }  
  
 @Override  
 public String toString(){  
 return type;  
 }  
}

Диаграмма классов реализованной объектной модели. Иерархия объектов.



**Результат работы программы:**

"Алиса в стране чудес"

Алиса попробовала Зелье без надписей костей, черепов, яда.

Зелье оказалось необыкновенно вкусным.

На вкус - точь-в-точь смесь вишневого пирога, омлета, ананаса, жареной индюшки, тянучки и горячих гренков с маслом

Алиса не заметила, как

Зелье закончилось.

**Вывод:**

В процессе выполнения лабораторной работы были получены навыки использования объектно-ориаентированного подхода программирования при использовании языка Java.