НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Программирование

Лабораторная работа № 4

Выполнил студент

Мысов Михаил Сергеевич

Группа № R3137

Преподаватель: Райла Мартин

г. Санкт-Петербург

2021

**Вариант: 373095**

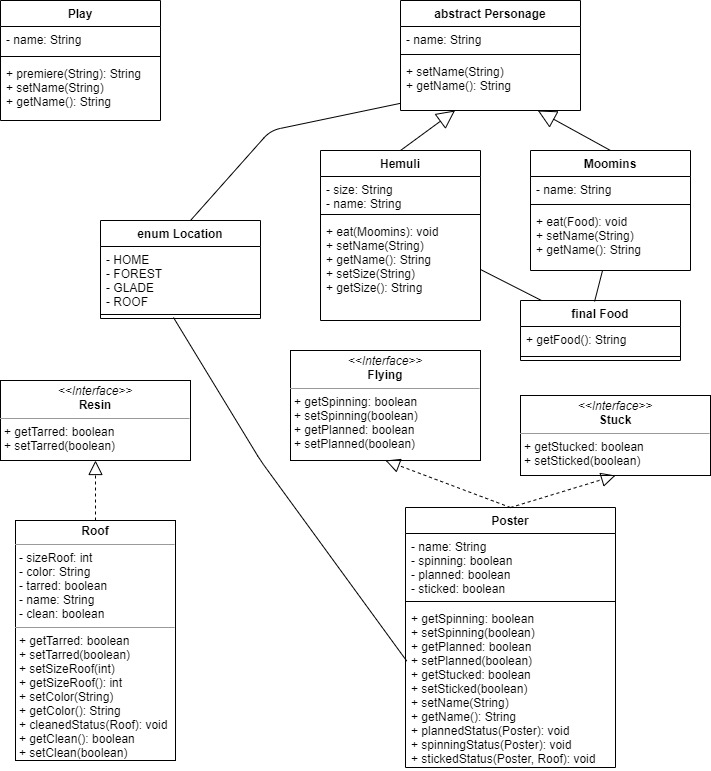
**Задание:**

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

О премьере пьесы.

Пока маленькая Хемулиха угощала гостей чаем у себя дома, театральная афиша продолжала кружиться над лесом. Она спланировала на лесную полянку и прилипла к крыше, которую только что просмолили.

**Диаграмма классов реализованной модели:**

****

**Исходный код программы:**

1. **Main**

public class Main {  
  
 public static void main(String[] args) {  
  
 Play play = new Play("Пьеса");  
 Food food = new Food();  
 Hemuli hemuli = new Hemuli();  
 Moomins moomins = new Moomins();  
 Poster poster = new Poster();  
 Roof roof = new Roof("крыша", 5, "yellow", true, true);  
  
 hemuli.setName("Хемулиха");  
 hemuli.setSize("Маленькая");  
  
 moomins.setName("Мумми");  
  
 poster.setName("Театральная афиша");  
  
 play.premiere();  
  
 hemuli.eat(hemuli, moomins);  
 moomins.eat(food, moomins);  
  
 poster.spinningStatus(poster);  
 poster.plannedStatus(poster);  
 poster.stickedStatus(poster, roof);  
 roof.cleanedStatus(roof,poster);  
 roof.tarredStatus(roof);  
 }  
}

1. **Flying:**

public interface Flying {  
 boolean getSpinning();  
 void setSpinning(boolean spinning);  
  
 boolean getPlanned();  
 void setPlanned(boolean planned);  
}

1. **Food**

public final class Food {  
 public String getFood() {  
 return "вкусный чай";  
 }  
}

1. **Hemuli**

public class Hemuli extends Personage {  
 private String size;  
 private String name;  
  
 public String getSize() {  
 return size;  
 }  
  
 public void setSize(String size) {  
 this.size = size;  
 }  
  
 @Override  
 public String getName() {  
 return this.name;  
 }  
  
 @Override  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void eat(Hemuli h, Moomins m) {  
 System.*out*.print(h.getSize() + " " + h.getName() + " успешно угостила " + m.getName() + " " + Location.*HOME*.getLocation() + ". ");  
 }  
  
   
}

1. **Location**

public enum Location {  
 *HOME* ("у себя дома"),  
 *FOREST*("над лесом"),  
 *GLADE*("на лесную полянку"),  
 *ROOF*("к крыше");  
  
 private String location;  
  
 Location (String location){  
 this.location = location;  
 }  
  
 public String getLocation(){  
 return location;  
 }  
}

1. **Moomins**

public class Moomins extends Personage {  
 private String name;  
  
 @Override  
 public String getName() {  
 return this.name;  
 }  
  
 @Override  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void eat(Food f, Moomins m) {  
  
 System.*out*.println(m.getName() + " выпил " + f.getFood() + ".");  
 }  
}

1. **Personage**

public abstract class Personage {  
 protected String name;  
  
 public abstract String getName();  
  
 public abstract void setName(String name);  
  
}

1. **Play**

public class Play {  
 private String name;  
  
 Play(String name){  
 this.name = name;  
 }  
  
 public String getName() {  
 return this.name;  
 }  
  
 public void premiere() {  
 System.*out*.println(name + " началась.");  
 }  
}

1. **Poster**

import java.util.Objects;  
  
public class Poster implements Flying, Stick {  
  
 private String name;  
 private boolean spinning = true;  
 private boolean planned = true;  
 private boolean sticked = true;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public boolean getSpinning() {  
 return spinning;  
 }  
  
 @Override  
 public void setSpinning(boolean spinning) {  
 this.spinning = spinning;  
 }  
  
 @Override  
 public boolean getPlanned() {  
 return planned;  
 }  
  
 @Override  
 public void setPlanned(boolean planned) {  
 this.planned = planned;  
 }  
  
 @Override  
 public boolean getSticked() {  
 return sticked;  
 }  
  
 @Override  
 public void setSticked(boolean sticked) {  
 this.sticked = sticked;  
 }  
  
 public void spinningStatus(Poster p) {  
 if (p.getSpinning()) {  
 System.*out*.println(p.getName() + " кружится " + Location.*FOREST*.getLocation() + ".");  
 }  
 else {  
 System.*out*.println(p.getName() + " не кружится " + Location.*FOREST*.getLocation() + ".");  
 }  
 }  
  
 public void plannedStatus(Poster p) {  
 if (p.getPlanned()) {  
 System.*out*.print(p.getName() + " планирует " + Location.*GLADE*.getLocation());  
 spinning = false;  
 }  
 else {  
 System.*out*.print(p.getName() + " остаётся кружиться " + Location.*FOREST*.getLocation());  
 }  
 }  
  
 public void stickedStatus(Poster p, Roof r) {  
 if (p.getSticked()) {  
 System.*out*.println(" и прилипает " + Location.*ROOF*.getLocation() + ".");  
 planned = false;  
 r.setClean(false);  
 }  
 else {  
 System.*out*.println(p.getName() + " остаётся кружиться " + Location.*FOREST*.getLocation() + ".");  
 }  
 }  
  
 @Override  
 public String toString() {  
 return "Poster{" +  
 "name='" + name + '\'' +  
 ", spinning=" + spinning +  
 ", planned=" + planned +  
 ", sticked=" + sticked +  
 '}';  
 }  
  
 @Override  
 public int hashCode() {  
 return this.name.hashCode();  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Poster poster = (Poster) o;  
 return spinning == poster.spinning &&  
 planned == poster.planned &&  
 sticked == poster.sticked &&  
 Objects.*equals*(name, poster.name);  
 }  
}

1. **Resin**

public interface Resin {  
 boolean getTarred();  
 void setTarred(boolean Tarred);  
}

1. **Roof**

import java.util.Objects;  
  
public class Roof implements Resin {  
  
 private String name;  
 private int sizeRoof;  
 private String color;  
 private boolean tarred;  
 private boolean clean;  
  
 Roof(String name, int sizeRoof, String color, boolean tarred, boolean clean) {  
 this.name = name;  
 this.sizeRoof = sizeRoof;  
 this.color = color;  
 this.tarred = tarred;  
 this.clean = clean;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getSizeRoof() {  
 return sizeRoof;  
 }  
  
 public void setSizeRoof(int sizeRoof) {  
 this.sizeRoof = sizeRoof;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 @Override  
 public boolean getTarred() {  
 return tarred;  
 }  
  
 @Override  
 public void setTarred(boolean tarred) {  
 this.tarred = tarred;  
 }  
  
 public boolean getClean() {  
 return clean;  
 }  
  
 public void setClean(boolean clean) {  
 this.clean = clean;  
 }  
  
 public void cleanedStatus(Roof r, Poster p) {  
 if (r.getClean()) {  
 System.*out*.println(Location.*ROOF*.getLocation() + " ничего не прилипло... ");  
 }  
 else {  
 System.*out*.println("\* " + Location.*ROOF*.getLocation() + " прилипла " + p.getName() + " \*");  
 }  
 }  
 public void tarredStatus(Roof r) {  
 if (r.getTarred()) {  
 System.*out*.print("В данный момент " + r.getName() + " просмолена.");  
 }  
 else {  
 System.*out*.print("В данный момент " + r.getName() + " не просмолена.");  
 }  
  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Roof roof = (Roof) o;  
 return sizeRoof == roof.sizeRoof &&  
 tarred == roof.tarred &&  
 clean == roof.clean &&  
 Objects.*equals*(name, roof.name) &&  
 Objects.*equals*(color, roof.color);  
 }  
  
 @Override  
 public int hashCode() {  
 return this.name.hashCode();  
 }  
  
 @Override  
 public String toString() {  
 return "Roof{" +  
 "name='" + name + '\'' +  
 ", sizeRoof=" + sizeRoof +  
 ", color='" + color + '\'' +  
 ", tarred=" + tarred +  
 ", clean=" + clean +  
 '}';  
 }  
}

1. **Stick**

public interface Stick {  
 boolean getSticked();  
 void setSticked(boolean sticked);  
}

**Вывод программы:**

*Пьеса началась.*

*Маленькая Хемулиха успешно угостила Мумми у себя дома. Мумми выпил вкусный чай.*

*Театральная афиша кружится над лесом.*

*Театральная афиша планирует на лесную полянку и прилипает к крыше.*

*\* к крыше прилипла Театральная афиша \**

*В данный момент крыша просмолена.*

**Выводы:**

Мы изучили

* принципы объектно- ориентированного программирования SOLID
* понятие абстрактного класса
* реализация интерфейсов в Java
* особенности реализации наследования в Java
* перечисляемый тип данных (enum)
* элементы функционального программирования в синтаксисе Java