Федеральное государственное автономное образовательное учреждение высшего

образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчет**

**По лабораторной работе №3**

По дисциплине «Программирование»

Вариант 210037.

Выполнил: Чураков Александр Алексеевич, группа P3131

Преподаватель: Письмак Алексей Евгеньевич, преподаватель факультета ПИиКТ.

Санкт-Петербург

~ 2023 ~

Оглавление

[Задание 3](#_Toc150507029)

[Диаграмма классов реализованной объектной модели 4](#_Toc150507030)

[Исходный код 4](#_Toc150507031)

[Перечисления 4](#_Toc150507032)

[BodyParts 4](#_Toc150507033)

[FindTime 4](#_Toc150507034)

[Pots 4](#_Toc150507035)

[Stuff 4](#_Toc150507036)

[Персонажи 4](#_Toc150507037)

[Абстрактный персонаж 4](#_Toc150507038)

[Кристофер Робин 4](#_Toc150507039)

[Кенга 4](#_Toc150507040)

[Крошка Ру 4](#_Toc150507041)

[Пятачок 4](#_Toc150507042)

[Тигра 4](#_Toc150507043)

[Тигр 4](#_Toc150507044)

[Интерфейсы 4](#_Toc150507045)

[Части тела 4](#_Toc150507046)

[Времена 4](#_Toc150507047)

[Банки 4](#_Toc150507048)

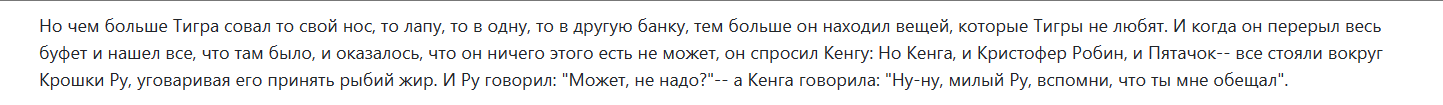
[Вещи 5](#_Toc150507049)

[Результат работы 5](#_Toc150507050)

[Вывод 5](#_Toc150507051)

# Задание

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



«Но чем больше Тигра совал то свой нос, то лапу, то в одну, то в другую банку, тем больше он находил вещей, которые Тигры не любят. И когда он перерыл весь буфет и нашел все, что там было, и оказалось, что он ничего этого есть не может, он спросил Кенгу: Но Кенга, и Кристофер Робин, и Пятачок-- все стояли вокруг Крошки Ру, уговаривая его принять рыбий жир. И Ру говорил: "Может, не надо?"-- а Кенга говорила: "Ну-ну, милый Ру, вспомни, что ты мне обещал".»

**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.

**Отчёт 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.

# Диаграмма классов реализованной объектной модели

# Исходный код

## Перечисления

### BodyParts

*package* src.enums;  
  
*public enum* BodyParts {  
 NOSE("нос"), PAW("лапу");  
  
 *private final* String name;  
  
 BodyParts(String name) {  
 *this*.name = name;  
 }  
  
 *public* String getName() {  
 *return* name;  
 }  
  
 *public* BodyParts[] getVals() {  
 *return* values();  
 }  
}

### FindTime

*package* src.enums;  
  
*public enum* FindTime {  
 PAST("находил"), PRESENT("нашел");  
  
 *private final* String name;  
  
 FindTime(String name) {  
 *this*.name = name;  
 }  
  
 *public* String getName() {  
 *return* name;  
 }  
}

### Pots

*package* src.enums;  
  
*public enum* Pots {  
 ONE("одну"),  
 OTHER("другую");  
  
 *private final* String name;  
  
 Pots(String name) {  
 *this*.name = name;  
 }  
  
 *public* String getName() {  
 *return* name;  
 }  
  
 *public* Pots[] getVals() {  
 *return* Pots.values();  
 }  
}

### Stuff

*package* src.enums;  
  
*public enum* Stuff {  
 THINGS("вещей"), ALL("все");  
  
 *private final* String name;  
  
 Stuff(String name) {  
 *this*.name = name;  
 }  
  
 *public* String getName() {  
 *return* name;  
 }  
}

## Персонажи

### Абстрактный персонаж

*package* src.heroes.abstractClasses;  
  
  
*public abstract class* AbstractHero {  
 *protected* String name;  
 *protected* String roditName;  
  
 *public* AbstractHero(String name) {  
 *this*.name = name;  
 }  
  
 *public* AbstractHero(String name, String roditName) {  
 *this*.name = name;  
 *this*.roditName = roditName;  
 }  
  
 *public* String getName() {  
 *return* name;  
 }  
  
 *public* String getRoditName() {  
 *return* roditName;  
 }  
}

### Кристофер Робин

*package* src.heroes;  
  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.Stand;  
  
*import* java.util.Objects;  
  
*public class* ChristoferRobin *extends* AbstractHero *implements* Stand {  
 *public* ChristoferRobin(String name) {  
 *super*(name);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 ChristoferRobin that = (ChristoferRobin) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "ChristoferRobin{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}

### Кенга

*package* src.heroes;  
  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.SpeakAble;  
*import* src.interfaces.Stand;  
  
*import* java.util.Objects;  
  
*public class* Kenga *extends* AbstractHero *implements* SpeakAble, Stand {  
 *public* Kenga(String name) {  
 *super*(name);  
 }  
  
 *public* Kenga(String name, String roditName) {  
 *super*(name, roditName);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 Kenga that = (Kenga) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "Кенгу";  
 }  
  
 *@Override  
 public void* speak(String str) {  
 System.out.print(" - а Кенга говорит: \"" + str + "\".");  
 }  
}

### Крошка Ру

*package* src.heroes;  
  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.SpeakAble;  
*import* src.interfaces.Stand;  
  
*import* java.util.Objects;  
  
*public class* KroshkaRu *extends* AbstractHero *implements* SpeakAble, Stand {  
 *public* KroshkaRu(String name) {  
 *super*(name);  
 }  
  
 *public* KroshkaRu(String name, String roditName) {  
 *super*(name, roditName);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 KroshkaRu that = (KroshkaRu) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "KroshkaRu{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
  
 *@Override  
 public void* speak(String str) {  
 System.out.print(" И " + *this*.name + " говорит: \"" + str + "\"");  
 }  
}

### Пятачок

*package* src.heroes;  
  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.Stand;  
  
*import* java.util.Objects;  
  
*public class* Piglet *extends* AbstractHero *implements* Stand {  
 *public* Piglet(String name) {  
 *super*(name);  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 Piglet that = (Piglet) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "Piglet{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}

### Тигра

*package* src.heroes;  
  
*import* src.enums.BodyParts;  
*import* src.enums.FindTime;  
*import* src.enums.Pots;  
*import* src.enums.Stuff;  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.AskAble;  
*import* src.interfaces.FindAble;  
*import* src.interfaces.StickAble;  
  
*import* java.util.Objects;  
  
*public class* Tigra *extends* Tiger *implements* StickAble, FindAble, AskAble {  
 *public* Tigra(String name) {  
 *super*(name);  
 }  
  
 *@Override  
 public void* find(FindTime time, Stuff stuff) {  
 System.out.print(" тем больше " + *this*.name + " " + time.getName() + " " + stuff.getName());  
 }  
  
 *@Override  
 public void* stick(BodyParts bp, Pots pots) {  
 System.out.print("Но чем больше " + *this*.name + " совал свой ");  
 *for* (BodyParts v : bp.getVals()) {  
 System.out.print(v.getName() + " ");  
 }  
  
 *for* (Pots pt : pots.getVals()) {  
 System.out.print(" то в ");  
 System.out.print(pt.getName());  
 }  
 System.out.print(" банку");  
 }  
  
 *public void* dontLike() {  
 System.out.print("есть не может, ");  
 }  
  
 *@Override  
 public void* ask(AbstractHero obj) {  
 System.out.print(" он спросил " + obj.getRoditName() + ": ");  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 Tigra that = (Tigra) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "Tigra{" + name + "}";  
 }  
  
 *public void* dig(Stuff stuff) {  
 System.out.print(" И когда он перерыл " + stuff.getName() + " весь буфет,");  
 }  
  
  
}

### Тигр

*package* src.heroes;  
  
*import* src.enums.FindTime;  
*import* src.enums.Stuff;  
*import* src.heroes.abstractClasses.AbstractHero;  
*import* src.interfaces.FindAble;  
  
*import* java.util.Objects;  
  
*public class* Tiger *extends* AbstractHero *implements* FindAble {  
 *public* Tiger(String name) {  
 *super*(name);  
 }  
  
 *public void* hate() {  
 System.out.print(" которые " + *this*.name + " не любят.");  
 }  
  
 *@Override  
 public void* find(FindTime time, Stuff stuff) {  
 System.out.print(time.getName() + " " + stuff.getName() + " что там было");  
 }  
  
 *@Override  
 public int* hashCode() {  
 *return super*.hashCode() + *this*.getName().hashCode();  
 }  
  
 *@Override  
 public boolean* equals(Object o) {  
 *if* (*this* == o) *return true*;  
 *if* (o == *null* || getClass() != o.getClass()) *return false*;  
 Tiger that = (Tiger) o;  
 *return* Objects.equals(name, that.name);  
 }  
  
 *@Override  
 public* String toString() {  
 *return* "Tiger{" +  
 "name='" + name + '\'' +  
 ", roditName='" + roditName + '\'' +  
 '}';  
 }  
}

## Интерфейсы

### AskAble

*package* src.interfaces;  
  
*import* src.heroes.abstractClasses.AbstractHero;  
  
*public interface* AskAble {  
 *void* ask(AbstractHero whom);  
}

### FindAble

*package* src.interfaces;  
  
*import* src.enums.FindTime;  
*import* src.enums.Stuff;  
  
*public interface* FindAble {  
 *void* find(FindTime time, Stuff stuff);  
}

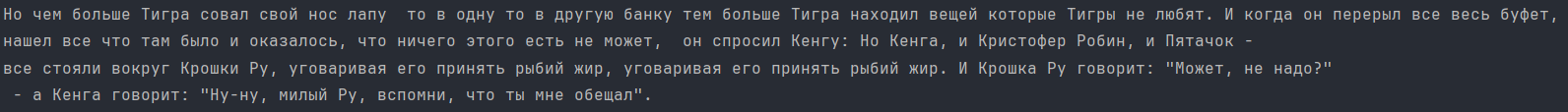
### SpeakAble

*package* src.interfaces;  
  
*public interface* SpeakAble {  
 *void* speak(String whatToSay);  
}

### StickAble

*package* src.interfaces;  
  
*import* src.enums.BodyParts;  
*import* src.enums.Pots;  
  
*public interface* StickAble {  
 *void* stick(BodyParts obj, Pots pots);  
}

# Результат работы



# Вывод