Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Лабораторная работа № 4**

По дисциплине «Программирование»

Вариант 34569,5

Выполнила:

Студентка группы P3117

Русакова Е.Д.

Преподаватель:

Письмак А.Е.

Санкт-Петербург

2022

Оглавление

[Задание: 2](#_Toc128398565)

[Диаграмма классов реализованной объектной модели: 2](#_Toc128398566)

[Исходный код программы: 2](#_Toc128398567)

[Main.java 2](#_Toc128398568)

[INamable.java 3](#_Toc128398569)

[IContain.java 3](#_Toc128398570)

[Actions.java 3](#_Toc128398571)

[Features.java 4](#_Toc128398572)

[Entity.java 4](#_Toc128398573)

[Human.java 5](#_Toc128398574)

[Things.java 6](#_Toc128398575)

[KitchenThing.java 7](#_Toc128398576)

[Dishes.java 8](#_Toc128398577)

[Результат работы программы: 8](#_Toc128398578)

[Вывод: 9](#_Toc128398579)

# Задание:

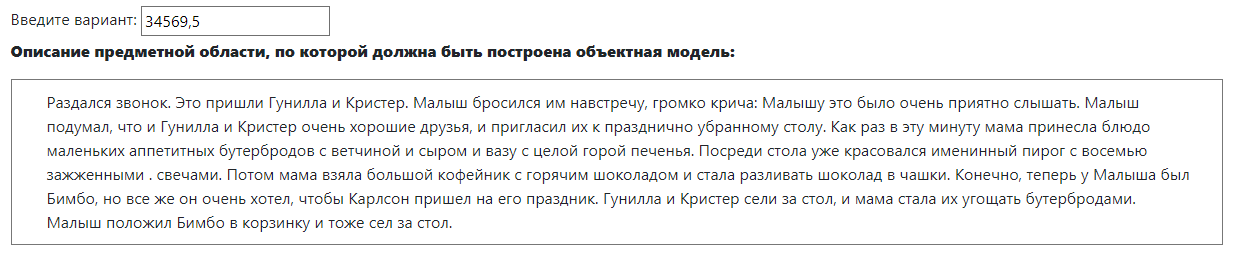
**Программа должна удовлетворять следующим требованиям:**

1. В программе должны быть реализованы 2 собственных класса исключений (checked и unchecked), а также обработка исключений этих классов.
2. В программу необходимо добавить использование локальных, анонимных и вложенных классов (static и non-static).

**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.

Задание для варианта 34569,5:



# Диаграмма классов реализованной объектной модели:

# Исходный код программы:

## Main.java

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Things Bell = **new** Things("звонок");

Bell.setActions(Actions.RING);

Human Malish = **new** Human("Малыш");

Human Gunilla = **new** Human("Гунилла");

Human Christer = **new** Human("Кристер");

Human Mom = **new** Human("Мама");

Gunilla.setActions(Actions.COME);

Christer.setActions(Actions.COME);

Malish.setActions(Actions.MEET);

**try** {

Malish.scream("???");

} **catch** (PhraseException e){

System.out.println(e.getMessage());

}

Malish.setFeatures(Features.PLEASANT);

Malish.setActions(Actions.THINK);

Gunilla.setFeatures(Features.GOODFRIEND);

Christer.setFeatures(Features.GOODFRIEND);

Things Table = **new** Things("стол");

Things Pie = **new** Things("пирог");

Things Cups = **new** Things("чашки");

Dishes Dish = **new** Dishes("блюдо", "бутерброды");

Dishes Vase = **new** Dishes("ваза", "печенье");

Dishes CoffeePot = **new** Dishes("кофейник", "горячий шоколад");

Malish.invite(Table, Gunilla);

Malish.invite(Table, Christer);

Table.setFeatures(Features.FESTIVE);

Mom.bring(Dish);

Dish.contFeatures(Features.SMALL);

Dish.contFeatures(Features.APPETIZING);

Dish.contFeatures(Features.FILLING);

Mom.bring(Vase);

Pie.flaunt(Table);

Pie.setFeatures(Features.BIRTHDAY);

IContain Candles = **new** IContain() {

@Override

**public** String getContent() {

**return** "с зажжеными свечами";

}

};

System.out.println(Pie.getName() + " " + Candles.getContent());

Mom.take(CoffeePot);

Mom.spill(CoffeePot, Cups);

Human.Dog Bimbo = **new** Human.Dog(Malish, "Бимбо");

Bimbo.setOwner();

Malish.setActions(Actions.WANT);

**class** CartoonCharacter **extends** Human{

**public** CartoonCharacter(String name){

**super**(name);

}

**private** Actions action;

**public** **void** come(){

**this**.action = Actions.COME;

System.out.println(getName() + " " + **this**.action.getActions() + " на праздник");

}

}

CartoonCharacter Carlson = **new** CartoonCharacter("Карлсон");

Carlson.come();

Gunilla.sit(Table);

Christer.sit(Table);

Mom.treat(Dish);

Things Basket = **new** Things("корзинка");

Bimbo.put(Basket);

Malish.sit(Table);

}

}

## INamable.java

**public** **interface** INamable {

**public** String getName();

}

## IContain.java

**public** **interface** IContain {

**public** String getContent();

}

## Actions.java

**public** **enum** Actions {

THINK("подумал что"),

INVITE("пригласил к"),

BRING("принесла"),

FLAUNT("красовался на"),

TAKE("взяла"),

SPILL("разливает"),

SIT\_DOWN("сел за"),

SCREAM("закричал"),

COME("пришел"),

MEET("бросился на встречу"),

WANT("очень хотел, чтобы"),

RING("раздался"),

TREAT("стала угощать");

**private** String action;

Actions(String action){

**this**.action = action;

}

**public** String getActions(){

**return** action;

}

@Override

**public** String toString(){

**return** **this**.toString();

}

}

## Features.java

**public** **enum** Features {

GOODFRIEND("хороший друг"),

FESTIVE("празднично убран"),

FILLING("с ветчиной и сыром"),

SMALL("маленькие"),

APPETIZING("аппетитные"),

BIRTHDAY("именинный"),

PLEASANT("было очень приятно");

**private** String feature;

Features(String feature){

**this**.feature = feature;

}

**public** String getFeatures() {

**return** feature;

}

@Override

**public** String toString() {

**return** **this**.toString();

}

}

## Entity.java

**public** **abstract** **class** Entity **implements** INamable{

**private** **final** String name;

**public** Entity(String name){

**this**.name = name;

}

**public** String getName(){

**return** name;

}

@Override

**public** String toString() {

**return** "name: " + name;

}

@Override

**public** **int** hashCode(){

**return** name.hashCode();

}

@Override

**public** **boolean** equals(Object o){

**if** (**this** == o) {

**return** **true**;

}

**if** (o == **null** || o.getClass() != getClass()){

**return** **false**;

}

Entity check = (Entity) o;

**return** hashCode() == check.hashCode();

}

}

## Human.java

**public** **class** Human **extends** Entity{

**public** Human (String name){

**super**(name);

}

**private** Actions action;

**public** Actions getAction() {

**return** action;

}

**public** **void** setActions(Actions action){

**this**.action = action;

System.out.println(getName() + " " + **this**.action.getActions());

}

**public** **void** setActionsQuiet(Actions action){

**this**.action = action;

}

**public** **void** bring(Dishes dish){

**this**.setActionsQuiet(Actions.BRING);

System.out.println(getName() + " " + **this**.action.getActions() + " " + dish.getName() + " с " + dish.getContent());

}

**public** **void** take(Dishes dish){

**this**.setActionsQuiet(Actions.TAKE);

System.out.println(getName() + " " + **this**.action.getActions() + " " + dish.getName() + " с " + dish.getContent());

}

**public** **void** invite(Things thing, Human human){

**this**.setActionsQuiet(Actions.INVITE);

System.out.println(getName() + " " + **this**.action.getActions() + " " +

thing.getName() + " " + human.getName());

}

**public** **void** spill(Dishes dish,Things thing){

**this**.setActionsQuiet(Actions.SPILL);

System.out.println(getName() + " " + **this**.action.getActions() + " " +

dish.getContent() + " в " + thing.getName());

}

**public** **void** sit(Things thing){

**this**.setActionsQuiet(Actions.SIT\_DOWN);

System.out.println(getName() + " " + **this**.action.getActions() + " " + thing.getName());

}

**public** **void** scream(String phrase) **throws** PhraseException{

**this**.setActionsQuiet(Actions.SCREAM);

**if**(phrase != "") {

System.out.println(getName() + " " + **this**.action.getActions() + " " + phrase);

} **else** {

**throw** **new** PhraseException(getName() + " промолчал, вместо крика");

}

}

**public** **void** treat(Dishes dish){

**this**.setActionsQuiet(Actions.TREAT);

System.out.println(getName() + " " + **this**.action.getActions() + " " + dish.getContent());

}

**private** Features feature;

**public** Features getFeatures() {

**return** feature;

}

**public** **void** setFeatures(Features feature){

**this**.feature = feature;

System.out.println(getName() + " " + **this**.feature.getFeatures());

}

**public** **static** **class** Dog {

**private** Human owner;

**private** String name;

**public** Human getOwner(){

**return** owner;

}

**public** String getName(){

**return** name;

}

**public** Dog(Human owner, String name){

**this**.name = name;

**this**.owner = owner;

}

**public** **void** setOwner(){

System.out.println("У " + owner.getName() + " есть " + getName());

}

**public** **void** put(Things thing){

System.out.println(owner.getName() + " положил " + getName()

+ " в " + thing.getName());

}

}

@Override

**public** **boolean** equals(Object o){

**if** (**this** == o){

**return** **true**;

}

**if**( o == **null** || o.getClass() != getClass()) {

**return** **false**;

}

Human check = (Human) o;

**return** hashCode() == check.hashCode();

}

}

## Things.java

**public** **class** Things **extends** Entity{

**public** Things(String name){

**super**(name);

}

**public** Actions action;

**public** Actions getAction(){

**return** action;

}

**public** **void** setActions(Actions action){

**this**.action = action;

System.out.println(getName() + " " + **this**.action.getActions() );

}

**public** **void** setActionQuiet(Actions action){

**this**.action = action;

}

**public** **void** flaunt(Things thing){

**this**.setActionQuiet(Actions.FLAUNT);

System.out.println(getName() + " " + **this**.action.getActions() + " " + thing.getName());

}

**private** Features feature;

**public** Features getFeatures(){

**return** feature;

}

**public** **void** setFeatures(Features feature){

**this**.feature = feature;

System.out.println(getName() + " " + **this**.feature.getFeatures());

}

@Override

**public** **boolean** equals(Object o){

**if** (**this** == o){

**return** **true**;

}

**if**( o == **null** || o.getClass() != getClass()) {

**return** **false**;

}

Things check = (Things) o;

**return** hashCode() == check.hashCode();

}

}

## KitchenThing.java

**public** **abstract** **class** kitchenThing **implements** INamable, IContain{

**private** **final** String name;

**private** String content;

**public** kitchenThing(String name, String content){

**this**.name = name;

**this**.content = content;

}

**public** String getName(){

**return** name;

}

**public** String getContent(){

**return** content;

}

@Override

**public** String toString() {

**return** "name: " + name;

}

@Override

**public** **int** hashCode(){

**return** name.hashCode() + content.hashCode();

}

@Override

**public** **boolean** equals(Object o){

**if** (**this** == o) {

**return** **true**;

}

**if** (o == **null** || o.getClass() != getClass()){

**return** **false**;

}

kitchenThing check = (kitchenThing) o;

**return** hashCode() == check.hashCode();

}

}

## Dishes.java

**public** **class** Dishes **extends** KitchenThing{

**public** Dishes (String name, String content){

**super**(name, content);

}

**private** Features feature;

**public** Features getFeatures(){

**return** feature;

}

**public** **void** contFeatures(Features feature){

**this**.feature = feature;

System.out.println(getContent() + " " + **this**.feature.getFeatures());

}

@Override

**public** **boolean** equals(Object o){

**if** (**this** == o){

**return** **true**;

}

**if**( o == **null** || o.getClass() != getClass()) {

**return** **false**;

}

Dishes check = (Dishes) o;

**return** hashCode() == check.hashCode();

}

}

## PhraseException.java

**public** **class** PhraseException **extends** Exception{

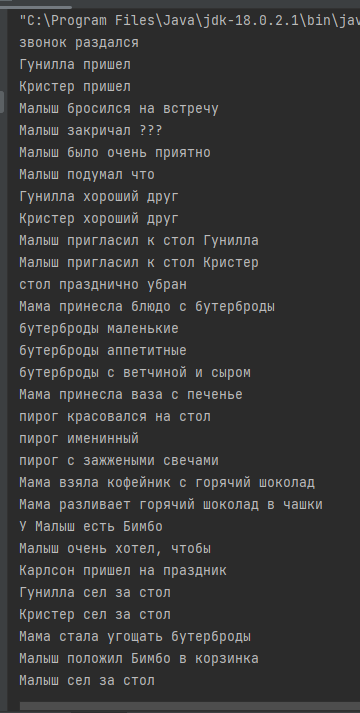
**public** PhraseException(String message){

**super**(message);

}

}

# Результат работы программы:



# Вывод:

При выполнении лабораторной работы я познакомилась с анонимными, вложенными и локальными классами, проверяемыми и непроверяемыми исключениями в java.