**Министерство науки и высшего образования Российской Федерации**

**Новосибирский государственный технический университет**

**Кафедра вычислительной техники**

**Лабораторная работа №1**

**по дисциплине:** Технология программирования

**на тему:** Основы программирования на Java.

Обработка событий. Механизм делегирования событий.

Преподаватель: Михайленко Д.А.

Студент: Голиков А.Ю.

Группа: АВТ-808

Новосибирск 2020

**Цель**: познакомиться с особенностями технологии Java и изучить синтаксис языка. Изучить основные понятия и термины обработки событий в модели делегирования событий.

**Задание**: разработать программу. Основная задача – разработка упрощенной имитации поведения объектов (все последующие лабораторные работы будут расширять это задание). Объекты реализуются через наследование: абстрактный класс + интерфейс → наследники.

Рабочий цикл программы:

- запускается процесс симуляции по клавише, генерируются объекты классов согласно заданию;

- симуляция завершается по другой клавише, выводится статистическая информация.

Вариант №4

Объект – кролик. Бывают 2 видов: обыкновенный и альбинос. Обыкновенные кролики рождаются каждые N1 секунд с вероятностью P1. Альбиносы рождаются каждые N2 секунд, при условии, что их количество менее K% от общего числа кроликов, в противном случае – не рождаются вовсе.

**Ход работы:**

1. Был разработан абстрактный модель кролика BaseRabbit согласно варианту задания, содержащая в себе координаты точки на экране и картинку.

private Point coordinates;  
protected Image rabbitImage;

1. Разработаны наследники абстрактного класса BaseRabbit, каждый из которых определяет картинку своего вида. (NormalRabbit, WhiteRabbit)
2. Объявлен интерфейс AbstractFactory, содержащий методы:

Integer getAmountOfBirth();//получть количество рожденных объектов  
BaseRabbit birth(Integer time, Point coordinates) throws IOException;//Родить объект BaseRabbit, содержащий в себе ссылку на наследника  
void destroy();//уничтожить скписок объектов

1. Класс ConcreteFactory имплементирует интерфейс AbstractFactory, содержит в себе поля для симуляции объектов с конкретной логикой. (в данном случае кролики)

private Integer rabbitsAmount;//количество кроликов  
private Integer whiteRabbitsAmount;//количество белых кроликов  
private final Integer normalRabbitBirthTime;//время рождения обыкновенного кролика  
private final Integer whiteRabbitBirthTime;//время рождения белого кролика  
private final Float normalRabbitBirthProbability;//вероятность рождения обыкновенного кролика  
private final Float rabbitsPercent;//процент от кроликов

1. Разработан класс среды Habitat, отрисовывающий все объекты на экране, содержащий в себе методы:

void update(); //продолжить отрисовку среды

public void timerVisibility(Boolean isShown);//скрыть таймер

public void stop();//остановить отрисовку среды

1. Описан класс HabitatTask для запуска отрисовку среды с какой-то временной периодичностью.
2. Реализован класс Simulation, запускающий симуляцию, и перехватывающий клавиатурные события:

- При нажатии кнопки B, симуляция запускается. (присутствует проверка, что симуляция уже запущена)

- При нажатии кнопки T, скрывается или появляется таймер

-При нажатии кнопки E, симуляция завершается, выводится итоговая информация. (присутствует проверка, что симуляция еще не началась, или уже закнчена)

**Результат:**

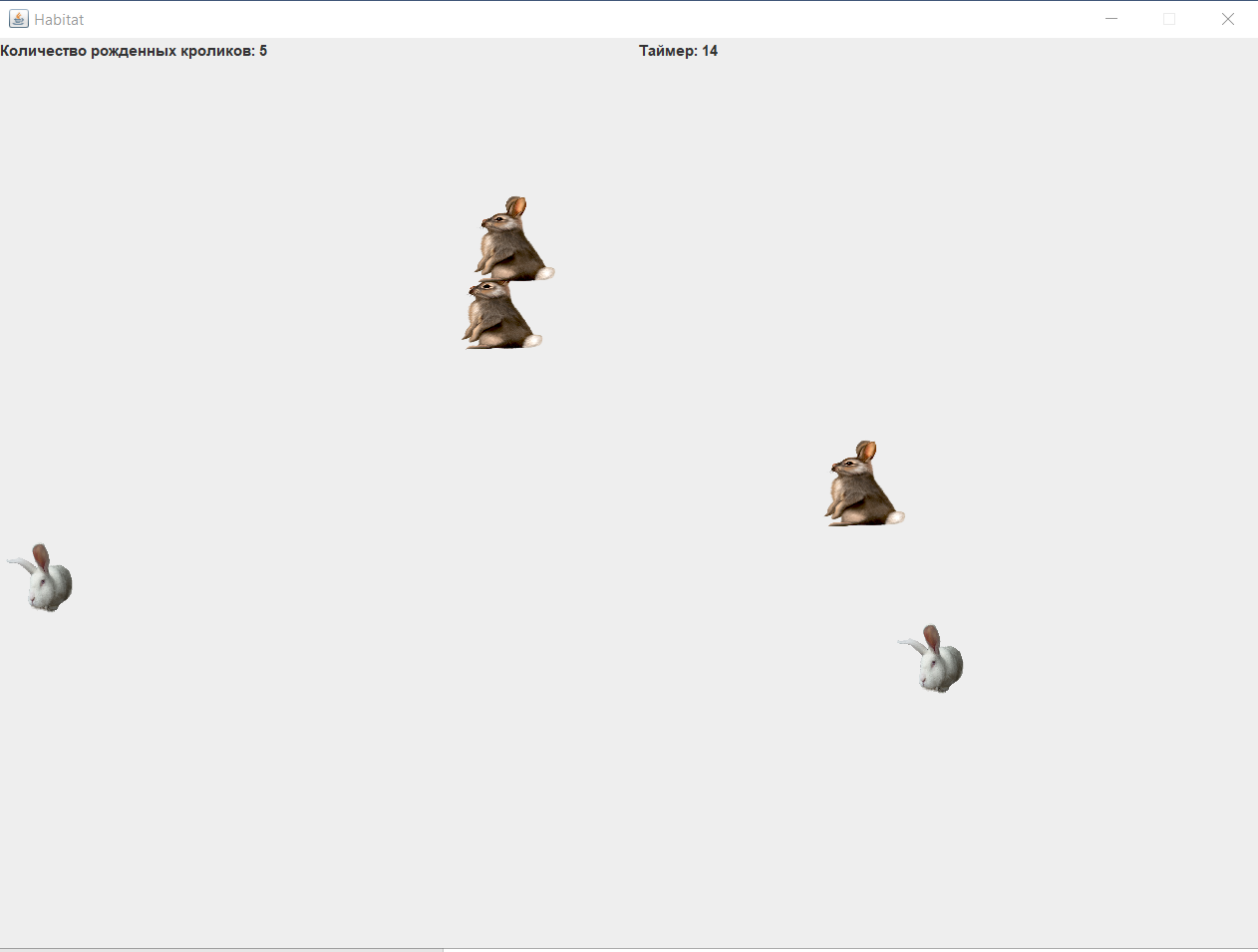


Рис. 1 – Во время симуляции

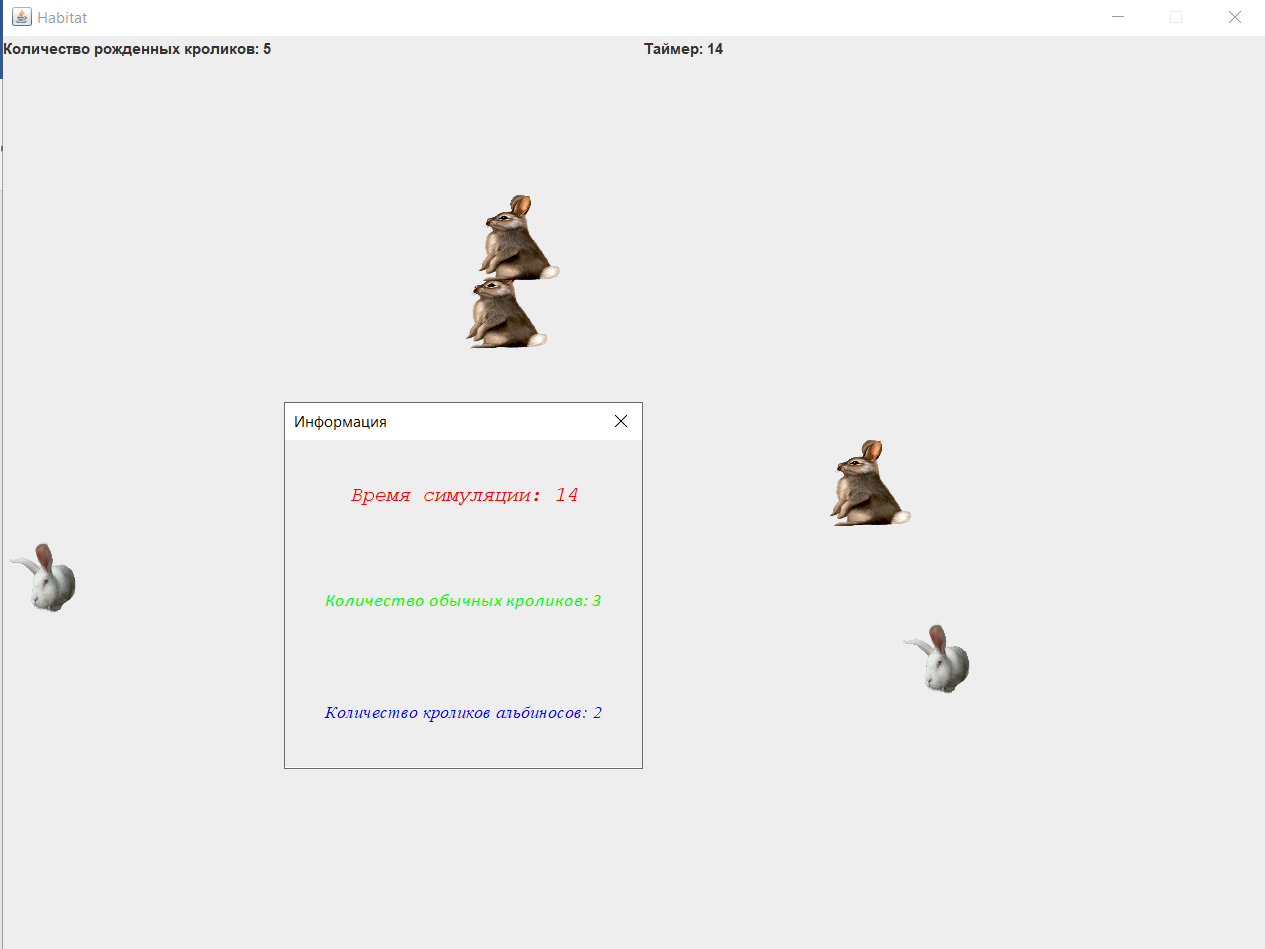


Рис. 2 – Завершение симуляции

**Вывод:** в данной работе была реализована программа симуляции рождения кроликов. Использован базовый синтаксис языка Java, и графические объекты библиотеки AWT и Swing. Была продемонстрирована возможность перехватывать события пользователя с клавиатуры в программе.

Код:

BaseRabbit

package Models.Abstract;  
  
import Behaviour.IBehaviour;  
  
import java.awt.\*;  
  
public abstract class BaseRabbit implements IBehaviour {  
 private Point coordinates;  
 protected Image rabbitImage;  
  
 protected BaseRabbit(Point coordinates) {  
 this.coordinates = coordinates;  
 }  
  
 public Image getRabbitImage() {  
 return rabbitImage;  
 }  
  
 public void setRabbitImage(Image rabbitImage) {  
 this.rabbitImage = rabbitImage;  
 }  
  
 public Point getCoordinates() {  
 return coordinates;  
 }  
  
 public void setCoordinates(Point coordinates) {  
 this.coordinates = coordinates;  
 }  
}

WhtieRabbit

package Models;  
  
import Models.Abstract.BaseRabbit;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.IOException;  
  
public class WhiteRabbit extends BaseRabbit {  
  
 public WhiteRabbit(Point coordinates) throws IOException {  
 super(coordinates);  
 this.rabbitImage = ImageIO.*read*(getClass().getResource("/WhiteRabbit.png"));  
 }  
}

NormalRabbit

package Models;  
  
import Models.Abstract.BaseRabbit;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.IOException;  
  
public class NormalRabbit extends BaseRabbit {  
 public NormalRabbit(Point coordinates) throws IOException {  
 super(coordinates);  
 this.rabbitImage = ImageIO.*read*(getClass().getResource("/Rabbit.png"));  
  
 }  
}

AbstractFactory

package Factory;  
  
import Models.Abstract.BaseRabbit;  
  
import java.awt.\*;  
import java.io.IOException;  
  
public interface AbstractFactory {  
 Integer getAmountOfBirth();//получть количество рожденных объектов  
 BaseRabbit birth(Integer time, Point coordinates) throws IOException;//Родить объект BaseRabbit, содержащий в себе ссылку на наследника  
 void destroy();//уничтожить скписок объектов  
}

ConcreteFactory

package Factory;  
  
import Models.Abstract.BaseRabbit;  
import Models.NormalRabbit;  
import Models.WhiteRabbit;  
  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
  
public class ConcreteFactory implements AbstractFactory {  
  
 private Integer rabbitsAmount;//количество кроликов  
 private Integer whiteRabbitsAmount;//количество белых кроликов  
 private final Integer normalRabbitBirthTime;//время рождения обыкновенного кролика  
 private final Integer whiteRabbitBirthTime;//время рождения белого кролика  
 private final Float normalRabbitBirthProbability;//вероятность рождения обыкновенного кролика  
 private final Float rabbitsPercent;//процент от кроликов  
  
 public ConcreteFactory(Integer normalRabbitBirthTime, Integer whiteRabbitBirthTime, Float normalRabbitBirthProbability, Float rabbitsPercent) {  
 rabbitsAmount = 0;  
 whiteRabbitsAmount = 0;  
 this.normalRabbitBirthTime = normalRabbitBirthTime;  
 this.whiteRabbitBirthTime = whiteRabbitBirthTime;  
 this.normalRabbitBirthProbability = normalRabbitBirthProbability;  
 this.rabbitsPercent = rabbitsPercent;  
 }  
  
 @Override  
 public Integer getAmountOfBirth() {  
 return rabbitsAmount;  
 }  
  
 @Override  
 public BaseRabbit birth(Integer time, Point coordinates) throws IOException {  
 if (time%normalRabbitBirthTime == 0) {  
 Random random = new Random();  
 float probability = random.nextFloat();  
 if (probability <= normalRabbitBirthProbability) {  
 BaseRabbit rabbit = new NormalRabbit(coordinates);  
 rabbitsAmount++;  
 return rabbit;  
 }  
 }  
 if (time%whiteRabbitBirthTime == 0) {  
 if(whiteRabbitsAmount<rabbitsAmount\*rabbitsPercent) {  
 BaseRabbit rabbit = new WhiteRabbit(coordinates);  
 rabbitsAmount++;  
 whiteRabbitsAmount++;  
 return rabbit;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
 rabbitsAmount = 0;  
 whiteRabbitsAmount = 0;  
 }  
}

Habitat

package Habitat;  
  
import DrawPanel.DrawRabbit;  
import Factory.AbstractFactory;  
import Factory.ConcreteFactory;  
import Models.Abstract.BaseRabbit;  
import Models.WhiteRabbit;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
public class Habitat extends JFrame {  
  
 private final JLabel rabbitsAmount;  
 private final JLabel timer;  
 private final AbstractFactory factory;  
 private final List<BaseRabbit> rabbits;  
 private Integer time;  
 private final DrawRabbit drawRabbit;  
 private Integer whiteRabbitsAmount;  
  
 public Habitat(String title, Integer normalRabbitBirthTime, Integer whiteRabbitBirthTime, Float normalRabbitBirthProbability, Float rabbitsPercent) {  
 super(title);  
 int width = 1024;  
 int height = 768;  
 setSize(width, height);  
 setResizable(false);  
 setLayout(null);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.time = 0;  
 rabbits = new ArrayList<>();  
 factory = new ConcreteFactory(normalRabbitBirthTime,whiteRabbitBirthTime,normalRabbitBirthProbability,rabbitsPercent);  
 rabbitsAmount = new JLabel("Количество рожденных кроликов: " + factory.getAmountOfBirth());  
 timer = new JLabel("Таймер: " + time);  
 JPanel labels = new JPanel();  
 labels.setLayout(new GridLayout(1,2));  
 labels.add(rabbitsAmount);  
 labels.add(timer);  
 labels.setBounds(0,0, width,20);  
 drawRabbit = new DrawRabbit(rabbits);  
 drawRabbit.setBounds(0,21, width, height -21);  
 add(labels);  
 add(drawRabbit);  
 whiteRabbitsAmount=0;  
 }  
  
 public void stop(){//остановить отрисовку среды  
 JLabel timerResult = new JLabel("Время симуляции: " + time);  
 timerResult.setFont(new Font("Courier New", Font.*ITALIC*,16));  
 timerResult.setForeground(Color.*RED*);  
 JLabel normalRabbitsResult = new JLabel("Количество обычных кроликов: " + (factory.getAmountOfBirth() - whiteRabbitsAmount));  
 normalRabbitsResult.setFont(new Font("Calibri", Font.*ITALIC*,15));  
 normalRabbitsResult.setForeground(Color.*GREEN*);  
 JLabel whiteRabbitsResult = new JLabel("Количество кроликов альбиносов: " + whiteRabbitsAmount);  
 whiteRabbitsResult.setFont(new Font("Times New Roman", Font.*ITALIC*,14));  
 whiteRabbitsResult.setForeground(Color.*BLUE*);  
 JDialog dialog = new JDialog(this, "Информация", true);  
 dialog.setLayout(null);  
 dialog.setResizable(false);  
 dialog.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 dialog.setSize(300, 300);  
 dialog.setLayout(new GridLayout(3,1));  
 timerResult.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(timerResult);  
 timerResult.setVisible(true);  
 normalRabbitsResult.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(normalRabbitsResult);  
 normalRabbitsResult.setVisible(true);  
 whiteRabbitsResult.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(whiteRabbitsResult);  
 whiteRabbitsResult.setVisible(true);  
 dialog.setVisible(true);  
 factory.destroy();  
 rabbits.clear();  
 time = 0;  
  
 rabbitsAmount.setText("Количество рожденных кроликов: " + factory.getAmountOfBirth());  
 timer.setText("Таймер: " + time);  
 drawRabbit.repaint();  
 }  
   
 public void timerVisibility(Boolean isShown){//скрыть таймер  
 if (isShown)  
 timer.setVisible(false);  
 else  
 timer.setVisible(true);  
 }  
  
 void update() {//продолжить симуляцию  
 time++;  
 timer.setText("Таймер: " + time);  
 Random coordinatesRandom = new Random();  
 int xCoordinate = coordinatesRandom.nextInt(drawRabbit.getWidth()-100);  
 int yCoordinate = coordinatesRandom.nextInt(drawRabbit.getHeight()-100);  
 Point coordinates = new Point(xCoordinate, yCoordinate);  
 try{  
 BaseRabbit rabbit = factory.birth(time, coordinates);  
 rabbitsAmount.setText("Количество рожденных кроликов: " + factory.getAmountOfBirth());  
 if(rabbit != null) {  
 if (rabbit.getClass() == WhiteRabbit.class)  
 whiteRabbitsAmount++;  
 rabbits.add(rabbit);  
 drawRabbit.repaint();  
 }  
 }  
 catch (Exception ex){  
 ex.printStackTrace();  
 }  
 }  
}

DrawRabbit

package DrawPanel;  
  
import Models.Abstract.BaseRabbit;  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.List;  
  
public class DrawRabbit extends JPanel {  
  
 private final List<BaseRabbit> rabbits;  
  
  
 public DrawRabbit(List<BaseRabbit> rabbits){  
 this.rabbits = rabbits;  
 }  
  
 @Override  
 protected void paintComponent(Graphics graphics) {  
 super.paintComponent(graphics);  
 if(rabbits.isEmpty())  
 return;  
 for (BaseRabbit rabbit : rabbits) {  
 graphics.drawImage(rabbit.getRabbitImage(),rabbit.getCoordinates().x,rabbit.getCoordinates().y,70,70,null);  
 }  
 }  
}

HabitatTask

package Habitat;  
  
import java.util.TimerTask;  
  
public class HabitatTask extends TimerTask {  
 private final Habitat habitat;  
  
 public HabitatTask(Habitat habitat){  
 this.habitat = habitat;  
 }  
  
 @Override  
 public void run() {  
 habitat.update();  
 }  
}

Simulaton

import Habitat.Habitat;  
import Habitat.HabitatTask;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.util.Timer;  
  
class Simulation implements KeyListener {  
 private Timer timer;  
 private HabitatTask habitatTask;  
 private final Habitat habitat;  
 private Boolean isStarted;  
 private Boolean isShown;  
  
 private Simulation() {  
 habitat = new Habitat("Habitat", 3,4,0.7f,0.7f);  
 habitat.setVisible(true);  
 habitat.addKeyListener(this);  
 isStarted = false;  
 isShown = true;  
 }  
  
 @Override  
 public void keyTyped(KeyEvent keyEvent) {  
 }  
  
 @Override  
 public void keyPressed(KeyEvent keyEvent) {  
 int key = keyEvent.getKeyCode();  
 if(key == KeyEvent.*VK\_B* && !isStarted) {  
 isStarted = true;  
 timer = new Timer();  
 habitatTask = new HabitatTask(habitat);  
 timer.schedule(habitatTask,0,1000);  
 }  
 if (key == KeyEvent.*VK\_E* && isStarted) {  
 timer.cancel();  
 habitatTask.cancel();  
 timer.purge();  
 habitat.stop();  
 isStarted = false;  
 }  
 if(key == KeyEvent.*VK\_T*) {  
 isShown = !isShown;  
 habitat.timerVisibility(isShown);  
 }  
 }  
  
 @Override  
 public void keyReleased(KeyEvent keyEvent) {  
  
 }  
  
 public static void main(String[] args) {  
 new Simulation();  
 }  
}