**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: Быков Д.А.

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 11

Объекты – домашние животные. Бывают 2 видов: кошки и собаки. Кошки генерируются каждые N1 секунд с вероятностью P1. Собаки генерируются каждые N2 секунд с вероятностью P2.

**Ход работы**

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

private Integer xCord;  
private Integer yCord;  
protected Image animalImage;

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

BaseAnimal birthCat(Integer xCord,Integer yCord) throws IOException;  
BaseAnimal birthDog(Integer xCord,Integer yCord) throws IOException;

1. Класс AnimalFactory имплементирует интерфейс AbstractFactory, содержит в себе симуляцию объектов с конкретной логикой. (в данном случае животные)

public class AnimalFactory implements AbstractFactory{  
 @Override  
 public BaseAnimal birthCat(Integer xCord, Integer yCord) throws IOException {  
 return new Cat(xCord, yCord);  
 }  
  
 @Override  
 public BaseAnimal birthDog(Integer xCord, Integer yCord) throws IOException {  
 return new Dog(xCord, yCord);  
 }  
}

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

private void changeTimerVisibility()

private void update()

private void stop()

1. Описан класс DrawAnimalPanel для запуска отрисовку среды с какой-то временной периодичностью.
2. Реализован метод 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 закочена)

**Результат:**

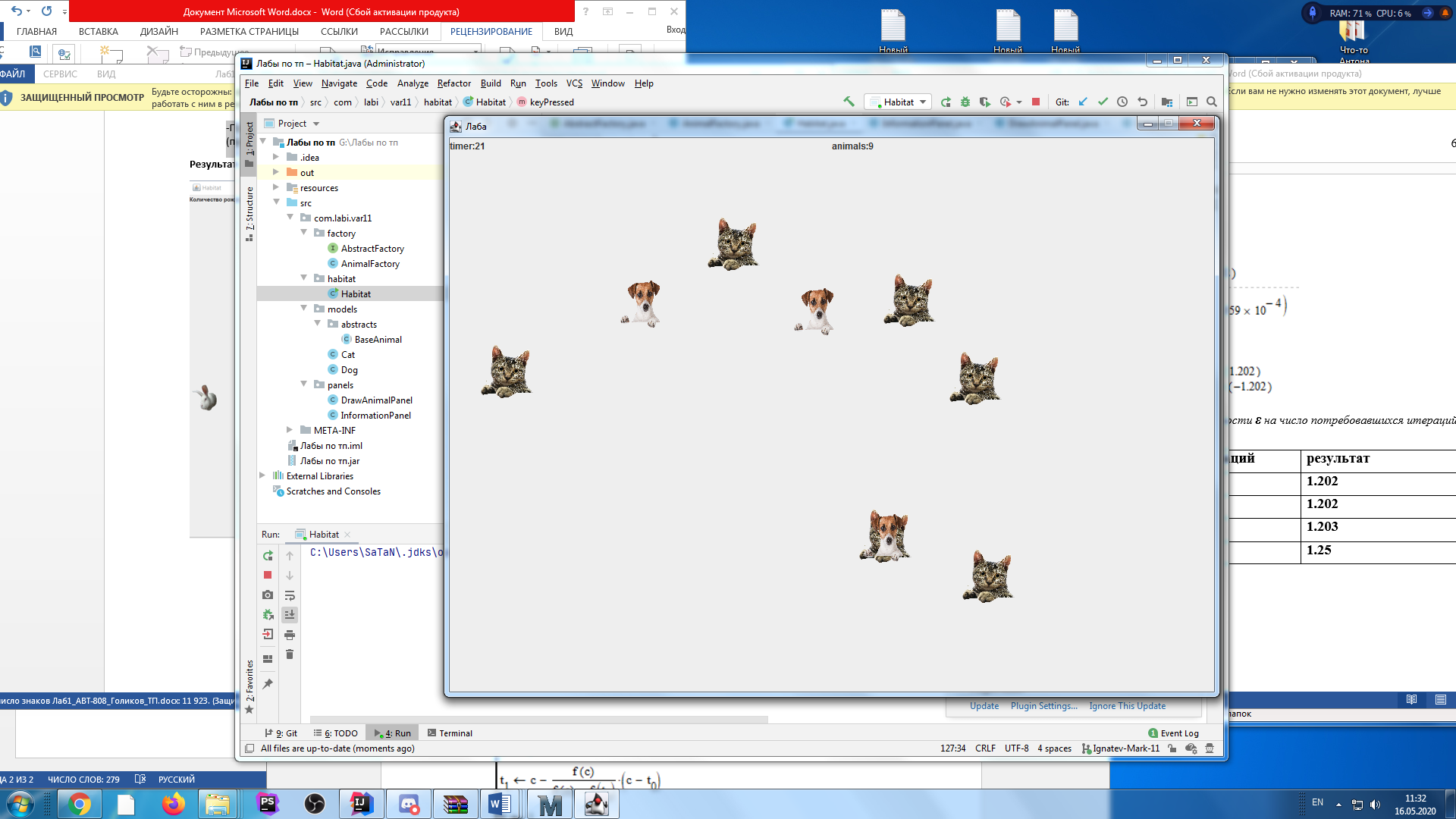


Рис. 1 – Во время симуляции

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

Код:

BaseAnimal

package com.labi.var11.models.abstracts;  
  
import java.awt.\*;  
  
public abstract class BaseAnimal {  
 private Integer xCord;  
 private Integer yCord;  
 protected Image animalImage;  
  
 protected BaseAnimal(Integer xCord, Integer yCord){  
 this.xCord=xCord;  
 this.yCord=yCord;  
 }  
  
 public Integer getX() {  
 return xCord;  
 }  
  
 public void setX(Integer xCord) {  
 this.xCord = xCord;  
 }  
  
 public Integer getY() {  
 return yCord;  
 }  
  
 public void setY(Integer yCord) {  
 this.yCord = yCord;  
 }  
  
 public Image getAnimalImage() {  
 return animalImage;  
 }  
}

Cat

package com.labi.var11.models;  
  
import com.labi.var11.models.abstracts.BaseAnimal;  
  
import javax.imageio.ImageIO;  
import java.io.IOException;  
  
public class Cat extends BaseAnimal {  
  
 public Cat(Integer xCord, Integer yCord) throws IOException {  
 super(xCord, yCord);  
 this.animalImage = ImageIO.*read*(getClass().getResource("/cat.png"));  
 }  
}

Dog

package com.labi.var11.models;  
  
import com.labi.var11.models.abstracts.BaseAnimal;  
  
import javax.imageio.ImageIO;  
import java.io.IOException;  
  
public class Dog extends BaseAnimal {  
  
 public Dog(Integer xCord, Integer yCord) throws IOException {  
 super(xCord, yCord);  
 this.animalImage = ImageIO.*read*(getClass().getResource("/dog.png"));  
 }  
}

AbstractFactory

package com.labi.var11.factory;  
  
import com.labi.var11.models.abstracts.BaseAnimal;  
  
import java.io.IOException;  
  
public interface AbstractFactory {  
 BaseAnimal birthCat(Integer xCord,Integer yCord) throws IOException;  
 BaseAnimal birthDog(Integer xCord,Integer yCord) throws IOException;  
}

AnimalFactory

package com.labi.var11.factory;  
  
import com.labi.var11.models.Cat;  
import com.labi.var11.models.Dog;  
import com.labi.var11.models.abstracts.BaseAnimal;  
  
import java.io.IOException;  
  
public class AnimalFactory implements AbstractFactory{  
 @Override  
 public BaseAnimal birthCat(Integer xCord, Integer yCord) throws IOException {  
 return new Cat(xCord, yCord);  
 }  
  
 @Override  
 public BaseAnimal birthDog(Integer xCord, Integer yCord) throws IOException {  
 return new Dog(xCord, yCord);  
 }  
}

Habitat

package com.labi.var11.habitat;  
  
import com.labi.var11.factory.AbstractFactory;  
import com.labi.var11.factory.AnimalFactory;  
import com.labi.var11.models.abstracts.BaseAnimal;  
import com.labi.var11.panels.DrawAnimalPanel;  
import com.labi.var11.panels.InformationPanel;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public class Habitat extends JFrame implements KeyListener {  
 private AbstractFactory animalFactory;  
 private Integer catGenTime;  
 private Integer dogGenTime;  
 private Double catGenProbability;  
 private Double dogGenProbability;  
 private Integer width;  
 private Integer height;  
 private Integer time;  
 private List<BaseAnimal> animals;  
 private DrawAnimalPanel drawAnimalPanel;  
 private Boolean isStarted;  
 private Timer t;  
 private TimerTask tt;  
 private InformationPanel informationPanel;  
 private Integer animalAmount;  
 private Boolean isVisible;  
  
  
 public Habitat(Integer width, Integer height) {  
 super("Лаба");  
 this.width = width;  
 this.height = height;  
 setSize(width, height);  
 setResizable(false);  
 setLayout(new BoxLayout(getContentPane(), BoxLayout.*Y\_AXIS*));  
 setVisible(true);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 addKeyListener(this);  
 catGenTime = 3;  
 dogGenTime = 4;  
 catGenProbability = 0.9;  
 dogGenProbability = 0.8;  
 time = 0;  
 animalFactory = new AnimalFactory();  
 animals = new ArrayList<BaseAnimal>();  
 drawAnimalPanel = new DrawAnimalPanel(animals);  
 informationPanel = new InformationPanel();  
 informationPanel.setMaximumSize(new Dimension(width, 20));  
 drawAnimalPanel.setMaximumSize(new Dimension(width, height - 130));  
 add(informationPanel);  
 add(drawAnimalPanel);  
 isStarted=false;  
 animalAmount = 0;  
 isVisible = true;  
 }  
  
 private void changeTimerVisibility(){  
 isVisible =! isVisible;  
 informationPanel.setTimerVisibility(isVisible);  
  
 }  
  
  
 private void stop(){  
 time=0;  
 informationPanel.setTimer(time);  
 animals.clear();  
 drawAnimalPanel.repaint();  
 animalAmount = 0;  
 informationPanel.setAnimalAmount(animalAmount);  
  
 }  
  
 private void update(){  
 time++;  
 informationPanel.setTimer(time);  
 Random random = new Random();  
 Integer xWidth = random.nextInt(width-100);  
 Integer yHeight = random.nextInt(height-100);  
 try {  
 if (time % catGenTime == 0) {  
 Double randomProbability = random.nextDouble();  
 if (randomProbability < catGenProbability) {  
 BaseAnimal animal = animalFactory.birthCat(xWidth, yHeight);  
 animals.add(animal);  
 animalAmount++;  
 informationPanel.setAnimalAmount(animalAmount);  
 drawAnimalPanel.repaint();  
 }  
 }  
 if (time % dogGenTime == 0) {  
 Double randomProbability = random.nextDouble();  
 if (randomProbability < dogGenProbability) {  
 BaseAnimal animal = animalFactory.birthDog(xWidth, yHeight);  
 animals.add(animal);  
 animalAmount++;  
 informationPanel.setAnimalAmount(animalAmount);  
 drawAnimalPanel.repaint();  
 }  
 }  
 }  
 catch (Exception exc){  
 exc.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
 new Habitat(1024,768);  
 }  
  
 @Override  
 public void keyTyped(KeyEvent e) {  
  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 int key = e.getKeyCode();  
  
 if(key == KeyEvent.*VK\_B* && !isStarted) {  
 isStarted = true;  
 t = new Timer();  
 tt = new TimerTask() {  
  
 @Override  
 public void run() {  
 update();  
 }  
 };  
  
 t.schedule(tt,0,1000);  
 }  
 if (key == KeyEvent.*VK\_E* && isStarted) {  
 t.cancel();  
 tt.cancel();  
 t.purge();  
 stop();  
 isStarted = false;  
 }  
 if(key == KeyEvent.*VK\_T*) {  
 changeTimerVisibility();  
 }  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
  
 }  
}

DrawAnimalPanel

package com.labi.var11.panels;  
  
import com.labi.var11.models.abstracts.BaseAnimal;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.util.List;  
  
public class DrawAnimalPanel extends JPanel {  
 private final List<BaseAnimal> animalList;  
  
 public DrawAnimalPanel(List<BaseAnimal> animalList) {  
 this.animalList = animalList;  
 }  
 @Override  
 protected void paintComponent(Graphics graphics) {  
 super.paintComponent(graphics);  
 if(animalList.isEmpty())  
 return;  
 for (BaseAnimal animal : animalList) {  
 graphics.drawImage(animal.getAnimalImage(),animal.getX(),animal.getY(),70,70,null);  
 }  
 }  
  
  
}

InfomationPanel

package com.labi.var11.panels;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class InformationPanel extends JPanel {  
 private JLabel animalAmount;  
 private JLabel timer;  
  
  
 public InformationPanel() {  
 setLayout(new GridLayout(1,2));  
 timer = new JLabel("timer:0");  
 animalAmount = new JLabel("animals:0");  
 add(timer);  
 add(animalAmount);  
 setVisible(true);  
  
 }  
  
 public void setTimer(Integer time){  
 timer.setText("timer:"+time);  
  
  
 }  
  
 public void setAnimalAmount(Integer animals){  
 animalAmount.setText("animals:"+animals);  
 }  
  
 public void setTimerVisibility(Boolean isVisible){  
 timer.setVisible(isVisible);  
 }  
}