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.*

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студенты гр. *АВТ-710*, *АВТФ* | *ассистент каф. ВТ* |
| *Аконечников Е.Н.* | *Михайленко Д.А.* |
| «\_\_\_» \_\_\_\_\_\_ 20\_\_г. | «\_\_\_» \_\_\_\_\_\_ 20\_\_г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *Перминов А.А.* | (подпись) |
| «\_\_\_» \_\_\_\_\_\_ 20\_\_г. |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| (подпись) |  |

Цель

1. Познакомиться с особенностями технологии Java и изучить синтаксис языка Java.

2. Изучить основные понятия и термины обработки событий в модели делегирования событий на Java.

3. Разработать учебную программу. Основная ее задача — разработка упрощенной имитации поведения объектов (все последующие лабораторные работы будут расширять это задание). Объекты реализуются через наследование: абстрактный класс + интерфейс → наследники.

Задание

Рабочий цикл программы:

* запускается процесс симуляции по клавише, генерируются объекты классов согласно заданию;
* симуляция завершается по другой клавише, выводится статистическая информация.

1. Для решения задачи:

* Разработать абстрактный класс объекта, согласно варианту индивидуального задания.
* Создать интерфейс IBehaviour, задающий поведение объекта (методы: move(), getx(), gety(), sety(), setx() и другие. Далее будут реализоваться алгоритмы движения объектов в окне программы).
* Реализовать иерархию классов, определяющих объекты по варианту и реализующие интерфейс IBehaviour.
* Создать класс Habitat (среда), определяющий размер рабочей области и хранящий массив объектов, с параметрами, заданными вариантом. Предусмотреть в классе метод Update, вызывающийся по таймеру и получающий на вход время, прошедшее от начала симуляции. В данном методе должны генерироваться новые объекты и помещаться в поле визуализации в случайном месте. Визуализация объекта - использовать готовые небольшие картинки;

Рабочее окно программы – область визуализации среды обитания объектов;

1. Симуляция должна запускаться по клавише **B** и останавливаться по клавише **E**. При остановке симуляции список очищается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише **T**;
2. По завершению симуляции область отображения объектов очищается и в поле визуализации выводится информация о количестве и типе сгенерированных объектов, а также время симуляции. Текст должен быть форматирован, т.е. выводиться с использованием разных шрифтов и цветов.
3. Параметры симуляции задаются в классе Habitat.

***Вариант 3***

Объект – аквариумная рыбка. Бывают 2 видов: золотая и гуппи. Золотые рыбки рождаются каждые N1 секунд с вероятностью P1. Гуппи рождаются каждые N2 секунд с вероятностью P2

Проектирование программы

Обсуждение основных идей алгоритма

Для программы создается семь классов:

Fish — абстрактный класс объекта, реализующий интерфейс IBehaviour.

Habitat — класс, в котором содержится массив рыбок и находит координаты рыбки, чтобы она попадала на экран.

* Update(double frameTime) — получает на вход время, прошедшее от начала симуляции. В данном методе генерируются новые объекты и помещаются в коллекцию объектов.

HabitatView — класс среды, определяющий параметры рабочей области (конструктор HabitatView ()) и описывающий весь пользовательский интерфейс. Определяет интерфейс KeyListener. Помимо этого, содержит следующие методы:

* Update() — получает на вход время, прошедшее от начала симуляции. В данном методе генерируются новые объекты и помещаются в поле визуализации в случайном месте.
* paint(Graphics g) — выводит изображение в поле визуализации.
* startSim () — запуск симуляции, сбрасываются счетчики объектов, создается элемент класса Habitat.
* stopSim () — завершает симуляцию.
* SimulationLoop — класс, наследуемый от класса TimerTask. Содержит метод run(), в котором определяется текущее время работы программы и вызывается метод update() класса Habitat.

Gold — класс, наследуемый от Fish и устанавливающий координаты золотой рыбки.

Guppy — класс, наследуемый от Fish и устанавливающий координаты картинки рыбки гуппи.

IBehaviour — реализуемый интерфейс, определяет основные параметры реализуемых его классов.

Main — основной класс работы программы, в котором создается объект класса Habitat и класс processKeyEvent который отвечает за обработку события — нажатие клавиши.

Пользовательский интерфейс

Управление пользовательским интерфейсом осуществляется через нажатие на клавиши клавиатуры. При первом запуске выводится информация о возможных действиях программы (рисунок 1).



Рисунок 1. Начальная информация

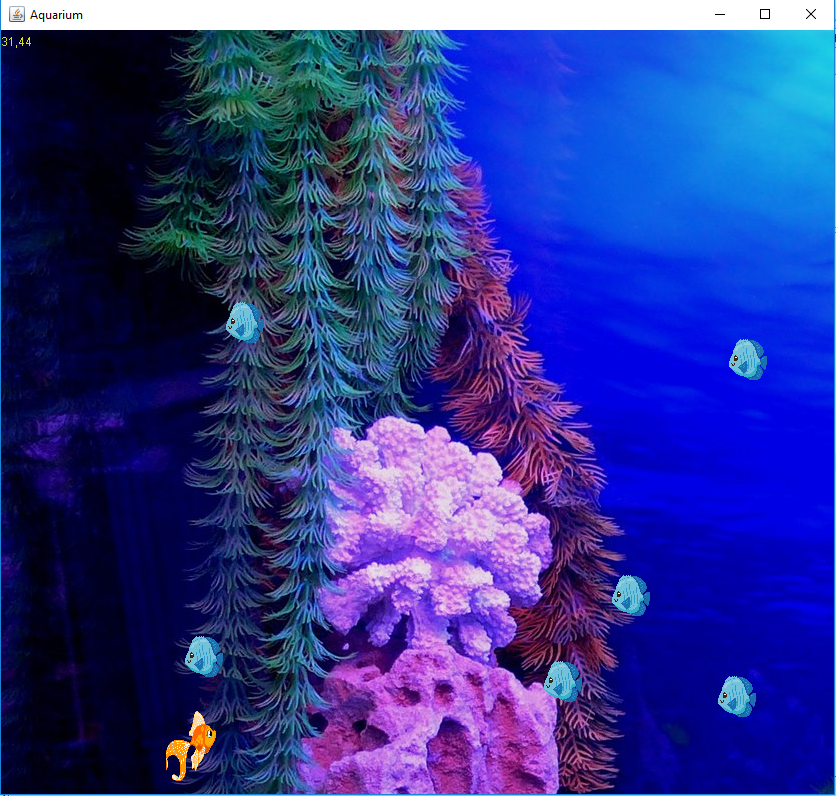


Рисунок 2. Графический интерфейс

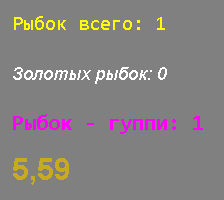


Рисунок 3. Вывод информации

Вывод:

В данной работе был изучен синтаксис языка Java, а также изучена графическая библиотека Swing. Для имитации был изучен встроенный класс Timer. Была разработана учебная программа.

Листинг программы

Habitat.java

package aquarium;

import java.util.LinkedList;

public class Habitat {

public LinkedList<Fish> ObjCollection = new LinkedList<>();

float timerGold = 0;

float timerGuppy = 0;

void Update(double timer){

timerGold += timer;

timerGuppy += timer;

if (timerGold > Gold.N1) {

if (Gold.isCreated()) addToCollections(0);

timerGold = 0;

}

if (timerGuppy > Guppy.N2) {

if (Guppy.isCreated()) addToCollections(1);

timerGuppy = 0;

}

}

public void addToCollections(int i){

int hi = HabitatView.heigthImg;

int wi = HabitatView.widthImg;

int upBoundWidth = HabitatView.Width - wi;

int upBoundHeight = HabitatView.Height - hi;

if(i == 0) ObjCollection.add(Fish.Sum, new Gold((int) ( Math.random() \* upBoundWidth ), (int) ( Math.random() \* upBoundHeight)));

else ObjCollection.add(Fish.Sum, new Guppy((int) ( Math.random() \* upBoundWidth ), (int) (Math.random() \* upBoundHeight)));

}

}

Main.java

package aquarium;

import java.awt.Dimension;

import java.awt.event.ComponentAdapter;

import java.awt.event.ComponentEvent;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import javax.swing.JFrame;

public class Main extends JFrame{

HabitatView view;

static Main frame;

Main(String name){

super(name);

view = new HabitatView();

setPreferredSize(new Dimension(1000, 800));

setMinimumSize(new Dimension(800, 800));

setVisible(true);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

add(view);

addComponentListener(new ComponentAdapter(){

public void componentResized(ComponentEvent e) {

HabitatView.Width = getWidth();

HabitatView.Height = getHeight();

}

});

addKeyListener((KeyListener) new processKeyEvent());

}

public static void main(String[] args) {

frame = new Main("Aquarium");

}

class processKeyEvent extends KeyAdapter {

public void keyReleased(KeyEvent e) {

super.keyReleased(e);

if (e.getKeyCode() == KeyEvent.VK\_T) {

view.show=!view.show;

}

if (e.getKeyCode() == KeyEvent.VK\_B) {

if (!view.simulating) view.startSim();

}

if (e.getKeyCode() == KeyEvent.VK\_E) {

view.stopSim();

}

}

}

}

Fish.java

package aquarium;

public abstract class Fish implements IBehaviour {

private int x;

private int y;

public static int Sum=0;

public Fish(){

x = 0;

y = 0;

Sum++;

}

public Fish(int x1, int y1){

x = x1;

y = y1;

Sum++;

}

public int getX(){

return x;

}

public int getY(){

return y;

}

public void setX(int x1){

x = x1;

}

public void setY(int y1){

y = y1;

}

}

Gold

package aquarium;

public class Gold extends Fish {

public static float P1 = 0.3f;

public static float N1 = 5;

public static int Sum1 = 0;

Gold(int x, int y){

super(x,y);

Sum1++;

}

public void move(){

}

static boolean isCreated() {

if (P1 >= Math.random()) return true;

else return false;

}

}

Guppy

package aquarium;

public class Guppy extends Fish{

public static float P2 = 0.5f;

public static float N2 = 3;

public static int Sum2 = 0;

Guppy(int x, int y){

super(x,y);

Sum2++;

}

static boolean isCreated() {

if (P2 >= Math.random())

return true;

else

return false;

}

public void move() {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

}

IBehaviour.java

package aquarium;

public interface IBehaviour {

void move();

int getx();

int gety();

void setx(int x);

void sety(int y);

}

HabitatView.java

package aquarium;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.File;

import java.io.IOException;

import java.util.Timer;

import java.util.TimerTask;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.imageio.ImageIO;

import javax.swing.JPanel;

public class HabitatView extends JPanel {

static Habitat habitat;

public static int widthImg = 50;

public static int heigthImg = 70;

public static double ElapsedTime;

static Timer timer;

private double deltaTime;

static boolean show = true;

static boolean simulating = false;

static boolean firstRun = true;

Image img1;

Image img2;

Image imgBG;

public static int Width;

public static int Height;

public HabitatView() {

try {

img1 = ImageIO.read(new File("Gold.png"));

img2 = ImageIO.read(new File("Guppy.png"));

imgBG = ImageIO.read(new File("BG.jpg"));

} catch (IOException ex) {

Logger.getLogger(HabitatView.class.getName()).log(Level.SEVERE, null, ex);

}

img1 = img1.getScaledInstance(widthImg, heigthImg, Image.SCALE\_DEFAULT);

img2 = img2.getScaledInstance(widthImg, heigthImg, Image.SCALE\_DEFAULT);

initComponents();

//startSim();

//stopSim();

repaint();

HabitatView.Width = this.getWidth();

HabitatView.Height = this.getHeight();

}

private void initComponents() {

}

//Слушатель нажатия клавиш

private void update(){

habitat.Update(deltaTime);

repaint();

}

//Отрисовка наших объектов и таймера при помощи буфера

public void paint(Graphics g) {

super.paint(g);

clearViewScreen(g);

if(simulating){

g.drawImage(imgBG, 0, 0, this);

//Отрисовывыаем объекты

for (Fish temp : habitat.ObjCollection)

if (temp instanceof Gold) g.drawImage(img1, temp.getX(), temp.getY(), this);

else if (temp instanceof Guppy) g.drawImage(img2, temp.getX(), temp.getY(), this);

//Отрисовывваем таймер

if (show){

g.setColor(new Color(222, 222, 30, 255));

g.drawString(String.format("%.2f", ElapsedTime), 0, 16);

}

//освобождаем ресуры

g.dispose();

}

else{

if (!firstRun) {

int midWidth = Width / 2 - 100;

int midHeight = Height / 2;

g.setFont(new Font("Consolas", Font.PLAIN, 20));

g.setColor(new Color(255, 255, 0, 255));

g.drawString(String.format("Рыбок всего: %d", Fish.Sum), midWidth, midHeight);

g.setFont(new Font("Arial", Font.ITALIC, 18));

g.setColor(new Color(255, 255, 255, 255));

g.drawString(String.format("Золотых рыбок: %d", Gold.Sum1), midWidth, midHeight + 50);

g.setFont(new Font("Consolas", Font.BOLD, 22));

g.setColor(new Color(255, 0, 255, 255));

g.drawString(String.format("Рыбок - гуппи: %d", Guppy.Sum2), midWidth, midHeight + 100);

g.setFont(new Font("Arial", Font.BOLD, 30));

g.setColor(new Color(200, 170, 30, 255));

g.drawString(String.format("%.2f", ElapsedTime), midWidth, midHeight + 150);

timer.cancel();

}

else {

g.setColor(Color.WHITE);

g.fillRect(0, 0, getWidth(), getHeight());

g.setFont(new Font("Consolas", Font.PLAIN, 20));

g.setColor(Color.GRAY);

g.drawString("Нажмите клавишу \"B\" для запуска", (getWidth()-31\*10) / 2 , getHeight() / 2);

}

}

}

private void clearViewScreen(Graphics g) {

g.setColor(Color.GRAY);

g.fillRect(0, 0, getWidth(), getHeight());

}

//вспмогательяные методы пауза. старт, стоп, продолжить

void startSim() {

firstRun = false;

simulating = true;

habitat = new Habitat();

timer = new Timer();

timer.schedule(new SimulationLoop(), 0, 10);

Fish.Sum = 0;

Gold.Sum1 = 0;

Guppy.Sum2 = 0;

}

void stopSim() {

simulating = false;

repaint();

}

//класс таймера

private class SimulationLoop extends TimerTask {

private double pauseTime;

private double startTime;

private double lastTime;

public SimulationLoop()

{

startTime = System.currentTimeMillis();

lastTime = startTime;

pauseTime = 0;

}

@Override

public void run() {

if (simulating) {

double currentTime = System.currentTimeMillis();

// Время, прошедшее от начала, в секундах

ElapsedTime = (currentTime - startTime - pauseTime) / 1000;

// Время, прошедшее с последнего обновления, в секундах

deltaTime = (currentTime - lastTime) / 1000;

lastTime = currentTime;

// Вызываем обновление

update();

}

else {

double currentTime = System.currentTimeMillis();

pauseTime = (currentTime - startTime - ElapsedTime \* 1000);

lastTime = currentTime;

}

}

}

}