**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: Кокорин А.С.

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 №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.

**Ход работы:**

1. Была разработана абстрактная модель **BaseFish** согласно варианту задания, содержащая в себе координаты точки на экране и картинку.

private Point cords;  
protected Image img;

1. Разработаны наследники абстрактного класса **BaseFish**, каждый из которых определяет картинку своего вида. (**GuppyFish**, **GoldFish**)
2. Объявлен интерфейс **AbstractFactory**, содержащий методы:

public interface AbstractFactory {  
 Integer getAmountOfBirth();  
 Integer getGoldFishAmountBirth();  
 Integer getGuppyFishAmountBirth();  
 BaseFish birth(Integer time, Point cords) throws IOException;  
  
  
  
 void destroy();

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

private Integer guppyFishAmount;   
private Integer fishAmount;  
private Integer goldFishAmount;  
private Integer goldFishBirth;  
private Integer guppyFishBirth;  
private Float goldFishBirthProbability;  
private Float guppyFishBirthProbability;

1. Разработан класс среды Habitat, который отрисовывает все объекты содержащиеся в методе.

Метод **update** – продолжает работу

Метод **timerVis** – скрывает таймер

Метод **stop** – останавливает процесс симуляции

void update();

public void timerVis(Boolean isShown);

public void stop();

1. Описан класс **Task** для запуска отрисовки среды с какой-то временной периодичностью.
2. Был создан интерфейс IBehaviour который задаёт поведение объекта.
3. public interface IBehaviour {  
    void setCords(int x) ;  
    void setImg(int y);  
    int getImg();  
    int getCords() ;  
   }
4. 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 закончена)

**Результат:**

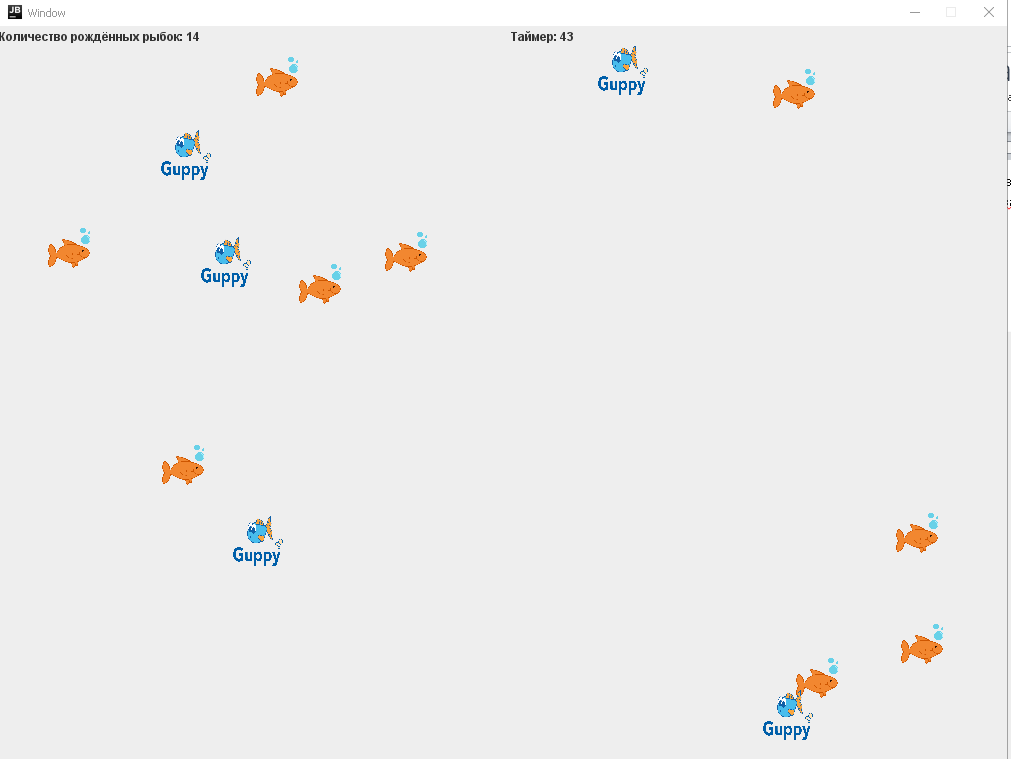


Рис. 1 – Во время симуляции

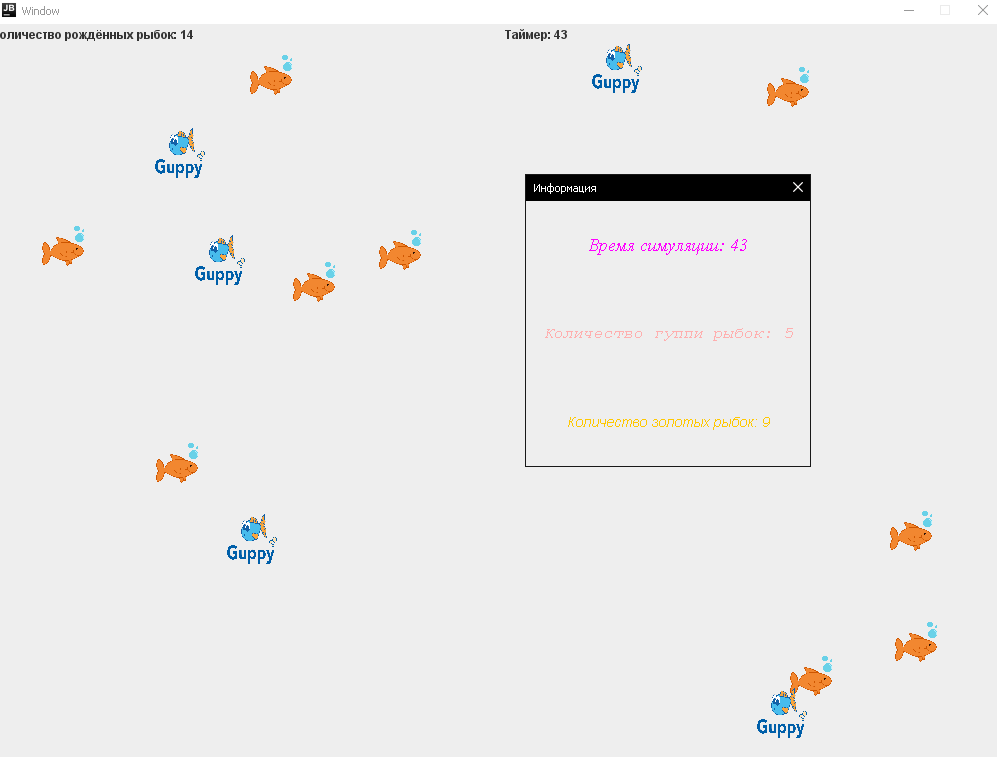


Рис. 2 – Завершение симуляции

**Вывод:** В данной работе был использован базовый синтаксис Java, а так же графические библиотеки Swing и AWT. Была реализована программа симуляции рождения рыбок, а так же была продемонстрирована обработка событий с клавиатуры.

**Код:**

**BaseFish**

package com.nstu.tp.models.abstracts;  
  
import java.awt.\*;  
  
public abstract class BaseFish {  
 private Point cords;  
 protected Image img;  
  
 public BaseFish(Point cords) {  
 this.cords = cords;  
 }  
  
 public Point getCords() {  
 return cords;  
 }  
  
 public void setCords(Point cords) {  
 this.cords = cords;  
 }  
  
 public Image getImg() {  
 return img;  
 }  
  
 public void setImg(Image img) {  
 this.img = img;  
 }  
  
}

**GoldFish**

package com.nstu.tp.models;  
  
import com.nstu.tp.models.abstracts.BaseFish;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.File;  
import java.io.IOException;  
import java.net.URL;  
  
public class GoldFish extends BaseFish {  
 public GoldFish(Point cords) throws IOException {  
 super(cords);  
 this.img = ImageIO.*read*(getClass().getResource("/GoldFish.jpg"));  
 }  
  
 @Override  
 public String toString() {  
 return "GoldFish";  
 }  
}

**GuppyFish**

package com.nstu.tp.models;  
  
import com.nstu.tp.models.abstracts.BaseFish;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.IOException;  
  
public class GuppyFish extends BaseFish {  
 public GuppyFish(Point cords) throws IOException {  
 super(cords);  
 this.img = ImageIO.*read*(getClass().getResource("/GuppyFish.png"));  
 }  
  
 @Override  
 public String toString() {  
 return "GuppyFish";  
 }  
}

**AbstractFactory**

package com.nstu.tp.factory;  
  
import com.nstu.tp.models.GoldFish;  
import com.nstu.tp.models.abstracts.BaseFish;  
  
import java.awt.\*;  
import java.io.IOException;  
  
public interface AbstractFactory {  
 Integer getAmountOfBirth();  
 Integer getGoldFishAmountBirth();  
 Integer getGuppyFishAmountBirth();  
 BaseFish birth(Integer time, Point cords) throws IOException;  
  
  
  
 void destroy();   
  
  
  
}

**ConcreteFactory**

package com.nstu.tp.factory;  
  
import com.nstu.tp.models.GoldFish;  
import com.nstu.tp.models.GuppyFish;  
import com.nstu.tp.models.abstracts.BaseFish;  
  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
  
public class ConcreteFactory implements AbstractFactory {  
 private Integer guppyFishAmount;  
 private Integer fishAmount;  
 private Integer goldFishAmount;  
 private Integer goldFishBirth;  
 private Integer guppyFishBirth;  
 private Float goldFishBirthProbability;  
 private Float guppyFishBirthProbability;  
  
 public ConcreteFactory(Integer goldFishBirth, Integer guppyFishBirth, Float goldFishBirthProbability, Float guppyFishBirthProbability) {  
 guppyFishAmount =0;  
 fishAmount = 0;  
 goldFishAmount =0;  
 this.goldFishBirth = goldFishBirth;  
 this.guppyFishBirth = guppyFishBirth;  
 this.goldFishBirthProbability = goldFishBirthProbability;  
 this.guppyFishBirthProbability = guppyFishBirthProbability;  
 }  
  
 @Override  
 public Integer getAmountOfBirth() {  
 return fishAmount;  
 }  
  
 public Integer getGoldFishAmountBirth() {  
 return goldFishAmount;  
 }  
 public Integer getGuppyFishAmountBirth() {  
 return guppyFishAmount;  
 }  
  
 @Override  
 public BaseFish birth(Integer time, Point cords) throws IOException {  
 // if (time % goldFishBirth == 0 && time % guppyFishBirth == 0){  
  
  
 if (time % goldFishBirth == 0){  
 Random random = new Random();  
 Float birthProbability = random.nextFloat();  
 if (birthProbability <= goldFishBirthProbability){  
 BaseFish fish = new GoldFish(cords);  
 fishAmount++;  
 goldFishAmount++;  
 return fish;  
  
 }  
 }  
 if (time % guppyFishBirth == 0){  
 Random random = new Random();  
 Float birthProbability = random.nextFloat();  
 if (birthProbability <= guppyFishBirthProbability){  
 BaseFish fish = new GuppyFish(cords);  
 guppyFishAmount++;  
 fishAmount++;  
 return fish;  
 }  
 }  
  
  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
 fishAmount = 0;  
 goldFishAmount = 0;  
 guppyFishAmount = 0;  
  
 }  
}

**Habitat**

package com.nstu.tp.habitat;  
  
import com.nstu.tp.DrawFish.DrawFish;  
import com.nstu.tp.factory.AbstractFactory;  
import com.nstu.tp.factory.ConcreteFactory;  
import com.nstu.tp.models.GoldFish;  
import com.nstu.tp.models.abstracts.BaseFish;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
  
import java.util.Random;  
  
public class Habitat extends JFrame {  
 BaseFish fishy[];  
 private final JLabel fishAmount;  
 private final JLabel timer;  
 private final AbstractFactory factory;  
 private Integer i=0;  
 private Integer Mass = 100;  
 private Integer time;  
 private final DrawFish drawFish;  
 private Integer guppyFishAmount;  
 private Integer goldFishAmount;  
  
  
 public Habitat(String title, Integer goldFishBirth, Integer guppyFishBirth, Float goldFishBirthProbability, Float guppyFishBirthProbability) {  
 super(title);  
 int width = 1024;  
 int height = 768;  
 setSize(width, height);  
 setResizable(false);  
 setLayout(null);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 this.time = 0;  
 fishy = new BaseFish[Mass];  
 factory = new ConcreteFactory(goldFishBirth, guppyFishBirth, goldFishBirthProbability, guppyFishBirthProbability);  
 fishAmount = new JLabel("Колличество рождённых рыбок:" + factory.getAmountOfBirth());  
 timer = new JLabel("Время" + time);  
 JPanel label = new JPanel();  
 label.setLayout(new GridLayout(1, 2));  
 label.add(fishAmount);  
 label.add(timer);  
 label.setBounds(0, 0, width, 20);  
 drawFish = new DrawFish(fishy,Mass);  
 drawFish.setBounds(0, 15, width, height - 15);  
 add(label);  
 add(drawFish);  
 goldFishAmount = 0;  
 guppyFishAmount = 0;  
  
 }  
 public void stop(){  
 JLabel timerRes = new JLabel("Время симуляции: " + time);  
 timerRes.setFont(new Font("Times New Roman", Font.*ITALIC*,18));  
 timerRes.setForeground(Color.*magenta*);  
 JLabel guppyFishResult = new JLabel("Количество гуппи рыбок: " + (factory.getGuppyFishAmountBirth()));  
 guppyFishResult.setFont(new Font("courier new", Font.*ITALIC*,16));  
 guppyFishResult.setForeground(Color.*pink*);  
 JLabel goldFishResult = new JLabel("Количество золотых рыбок: " + (factory.getGoldFishAmountBirth()));  
 goldFishResult.setFont(new Font("Comic Sans", Font.*ITALIC*,14));  
 goldFishResult.setForeground(Color.*orange*);  
 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));  
 timerRes.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(timerRes);  
 timerRes.setVisible(true);  
 guppyFishResult.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(guppyFishResult);  
 guppyFishResult.setVisible(true);  
 goldFishResult.setHorizontalAlignment(SwingConstants.*CENTER*);  
 dialog.add(goldFishResult);  
 goldFishResult.setVisible(true);  
 dialog.setVisible(true);  
 factory.destroy();  
 for (int j=0;j<Mass;j++)  
 {fishy[j]=null;}  
 i = 0;  
 time = 0;  
  
 fishAmount.setText("Количество рожденных рыбок: " + factory.getAmountOfBirth());  
 timer.setText("Таймер: " + time);  
 drawFish.repaint();  
  
 }  
 public void timerVis(Boolean isShown){  
 if (isShown)  
 timer.setVisible(false);  
 else  
 timer.setVisible(true);  
 }  
  
 void update() {  
 time++;  
 timer.setText("Таймер: " + time);  
 Random coordinatesRandom = new Random();  
 int xCoordinate = coordinatesRandom.nextInt(drawFish.getWidth()-100);  
 int yCoordinate = coordinatesRandom.nextInt(drawFish.getHeight()-100);  
 Point coordinates = new Point(xCoordinate, yCoordinate);  
 try{  
 BaseFish baseFish = factory.birth(time, coordinates);  
 fishAmount.setText("Количество рождённых рыбок: " + factory.getAmountOfBirth());  
  
 if(baseFish != null) {  
 if (baseFish.getClass() == GoldFish.class)  
 goldFishAmount++;  
  
  
 fishy[i] = baseFish;  
 i++;  
 drawFish.repaint();  
 }  
  
 }  
 catch (Exception ex){  
 ex.printStackTrace();  
 }  
 }  
}

**DrawFish**

package com.nstu.tp.DrawFish;  
  
import com.nstu.tp.models.abstracts.BaseFish;  
import java.awt.\*;  
import javax.swing.\*;  
  
  
  
 public class DrawFish extends JPanel {  
  
  
 BaseFish fish[];  
 int Mass;  
  
  
 public DrawFish(BaseFish fish[], int Mass)  
 {  
 this.fish = fish;  
 this.Mass=Mass;  
  
 }  
  
 @Override  
 protected void paintComponent(Graphics g)  
 {  
 super.paintComponent(g);  
 if (fish[0] == null) {  
 return;  
 }  
  
 for (int i=0; fish[i] != null; i++)  
  
 if (i < Mass)  
 g.drawImage(fish[i].getImg(),fish[i].getCords().x,fish[i].getCords().y,50,50,null);  
  
  
 }  
 }

**Task**

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

**Simulation**

import com.nstu.tp.habitat.Habitat;  
import com.nstu.tp.habitat.Task;  
  
  
import java.awt.event.KeyEvent;  
import java.awt.event.KeyListener;  
import java.util.Timer;  
  
  
class Simulation implements KeyListener {  
 private final Habitat habitat;  
 private Timer timer;  
 private Task Task;  
 private Boolean isStarted;  
 private Boolean isShown;  
  
private Simulation(){  
 habitat = new Habitat("Window", 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();  
 Task = new Task(habitat);  
 timer.schedule(Task,0,1000);  
 }  
 if (key == KeyEvent.*VK\_E* && isStarted) {  
 timer.cancel();  
 Task.cancel();  
 timer.purge();  
 habitat.stop();  
 isStarted = false;  
 }  
 if(key == KeyEvent.*VK\_T*) {  
 isShown = !isShown;  
 habitat.timerVis(isShown);  
  
 }  
  
  
 }  
  
 @Override  
 public void keyReleased(KeyEvent keyEvent) {  
  
 }  
  
  
 public static void main(String[] args) {  
 new Simulation();  
 }  
}