Министерство науки и высшего образования РФ

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Кафедра Вычислительной техники

Лабораторная работа 1

“Основы программирования на Java. Обработка событий. Механизм делегирования событий”

по дисциплине “Технологии программирования”

Преподаватель: Михайленко Д.А.

Студент: Полнарев С.В.

Группа: АВТ-808

Новосибирск 2020

**Цель работы**

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.

**Задание**

Вариант 6

Рабочий коллектив компании состоит из разработчиков и менеджеров. Разработчики генерируются каждые N1 секунд с вероятностью P1. Менеджеры генерируются каждые N2 секунд при условии, что их количество менее K% от общего числа разработчиков, в противном случае – не генерируются.

- Разработать абстрактный класс объекта, согласно варианту индивидуального задания.

- Создать интерфейс IBehaviour, задающий поведение объекта (далее будут реализоваться алгоритмы движения объектов в окне программы).

- Реализовать иерархию классов, определяющих объекты по варианту и реализующие интерфейс IBehaviour.

- Создать интерфейс AbstractFactory для создания объектов классов.

- Реализовать от него класс ConcreteFactory, возвращающий нужный объект, согласно варианту индивидуального задания.

- По желанию можно использовать паттерн Factory Method.

- 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 – схематично, плюсом будет, если объект будет похож на оригинал (можно 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 уничтожается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише T;

2. По завершению симуляции в поле визуализации должна выводиться 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.

4. При необходимости или желанию можно использовать другие паттерны.

**Ход работы**

1. Были разработаны классы:

* AbstractEmployee, содержащий в себе

private BufferedImage img;

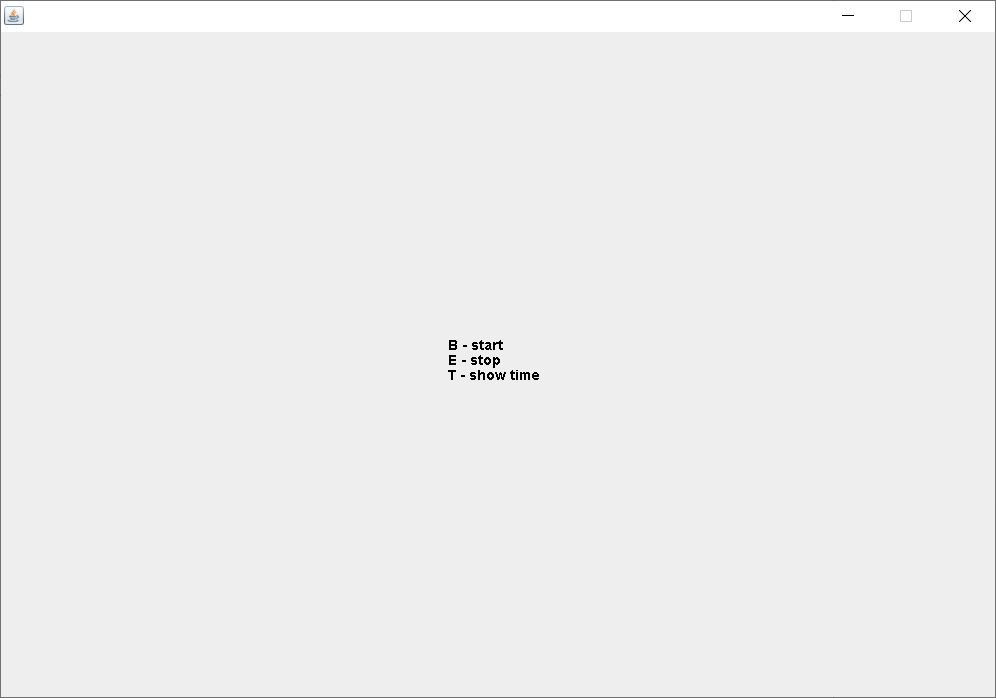
private Point coords;

а также геттеры и сеттеры.

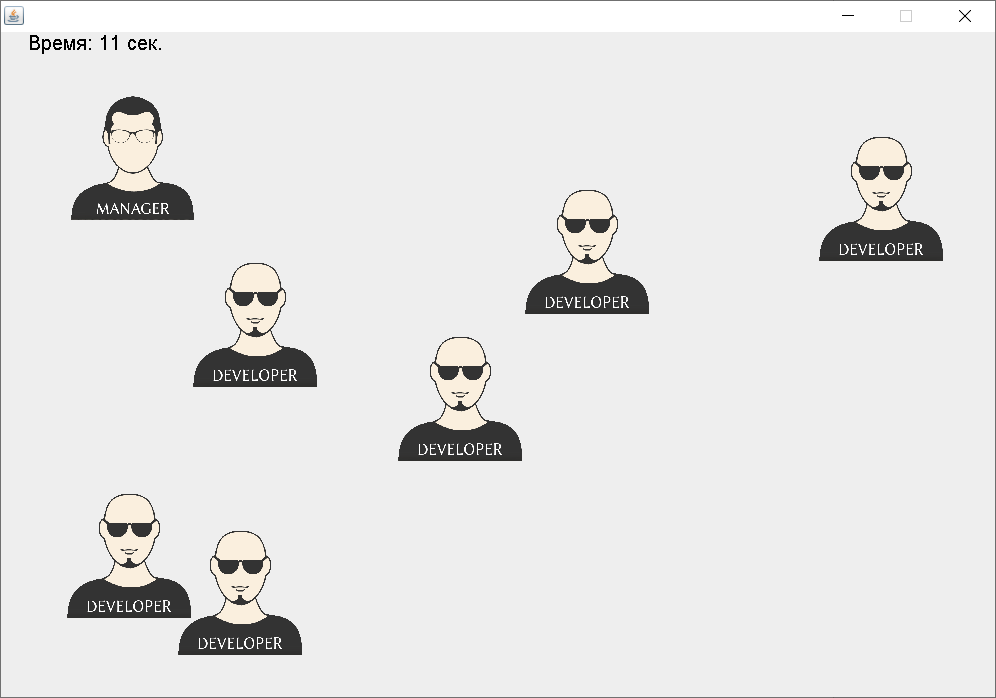
* Наследники класса AbstractEmployee – Developer и Manager

1. Был разработан класс Habitat, отвечающий за отрисовку объектов и контроль симуляции. В этом классе реализована двойная буферизация с использованием BufferStrategy, предотвращающая мерцания при обновлении экрана.
2. Абстрактный класс AbstractFactory, содержащий в себе метод getEmployee(), и реализующий этот метод класс ConcreteFactory. Класс ConcreteFactory также загружает изображения, необходимые для создания классов Developer и Manager и последующей их отрисовки.
3. Создан задающий размер рабочей области и запускающий приложение класс App

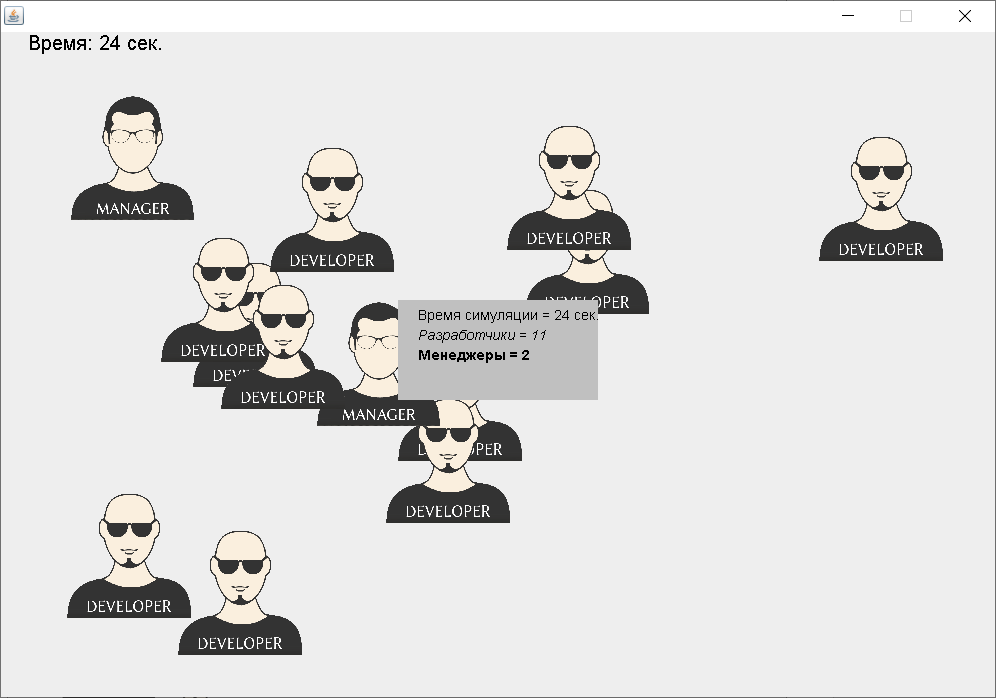
**Демонстрация работы программы**



*Рисунок 1. “Начальный экран”*



*Рисунок 2. “Работа программы”*



*Рисунок 3. “Остановка программы”*

**Выводы:**

В результате выполнения работы мы познакомились с базовыми типами Java, изучили основные понятия и термины в модели обработки событий, а также научились создавать пользовательский интерфейс программы.

**Приложение**

**Листинг программы**

**Habitat.java**

package tehprog;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferStrategy;

import java.util.LinkedList;

import java.util.Timer;

import java.util.TimerTask;

public class Habitat extends JFrame {

private Timer timer = null;

package tehprog;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferStrategy;

import java.util.LinkedList;

import java.util.Timer;

import java.util.TimerTask;

public class Habitat extends JFrame {

private Timer timer = null;

private ConcreteFactory factory = new ConcreteFactory();

private LinkedList<AbstractEmployee> employees= new LinkedList<>();

private long simTime;

private boolean toggleTime = false;

private BufferStrategy screenBuffer;

private Dimension infoTab = new Dimension(200,100);

private Dimension avatarDim = new Dimension(125,125);

public Habitat(int w, int h) {

super();

initGUI(new Dimension(w,h));

}

public void start(){

if(timer != null)

return;

employees.clear();

factory.reset();

simTime = 0;

timer = new Timer();

timer.schedule(new TimerTask() {

@Override

public void run() {

AbstractEmployee employee = factory.getEmployee(simTime,

new Dimension(getWidth()-avatarDim.width,

getHeight()-avatarDim.height));

if(employee != null)

employees.add(employee);

simTime++;

update();

}

}, 0, 1000);

}

public void stop(){

if(timer != null){

timer.cancel();

timer = null;

drawInfo(screenBuffer.getDrawGraphics());

}

}

public void update(){

Graphics g = screenBuffer.getDrawGraphics();

g.setColor(Color.WHITE);

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

for(AbstractEmployee e : employees)

g.drawImage(e.getImg(), e.getCoords().x, e.getCoords().y, avatarDim.width, avatarDim.height, null);

if(toggleTime) {

g.setColor(Color.BLACK);

g.setFont(g.getFont().deriveFont(20.0f));

g.drawString("Время: " + simTime + " сек.", 30, 50);

}

g.dispose();

screenBuffer.show();

}

private void initGUI(Dimension dim){

setResizable(false);

setSize(dim);

addKeyListener(new KeyAdapter() {

@Override

public void keyReleased(KeyEvent e) {

super.keyTyped(e);

if(e.getKeyChar() == 'b' || e.getKeyChar() == 'B')

start();

else if(e.getKeyChar() == 'e' || e.getKeyChar() == 'E')

stop();

else if(e.getKeyChar() == 't' || e.getKeyChar() == 'T'){

toggleTime = !toggleTime;

update();

}

}

});

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

createBufferStrategy(2);

screenBuffer = getBufferStrategy();

Graphics t = screenBuffer.getDrawGraphics();

t.setFont(new Font("Arial",Font.BOLD, 14));

t.drawString("B - start",getWidth()/2-50, getHeight()/2);

t.drawString("E - stop",getWidth()/2-50, getHeight()/2 + 15);

t.drawString("T - show time",getWidth()/2-50, getHeight()/2 + 30);

screenBuffer.show();

}

private void drawInfo(Graphics g){

g.setColor(Color.LIGHT\_GRAY);

g.fillRect(getWidth()/2 - infoTab.width/2, getHeight()/2 - infoTab.height/2,

infoTab.width, infoTab.height);

g.setColor(Color.BLACK);

g.setFont(g.getFont().deriveFont(14.0f));

g.drawString("Время симуляции = "+ simTime +" сек.",

(getWidth()/2 - infoTab.width/2)+20, (getHeight()/2 - infoTab.height/2) +20);

g.setFont(new Font("Arial",Font.ITALIC, 14));

g.drawString("Разработчики = "+ factory.getDeveloperSpawned(),

(getWidth()/2 - infoTab.width/2)+20, (getHeight()/2 - infoTab.height/2) +40);

g.setFont(new Font("Arial",Font.BOLD, 14));

g.drawString("Менеджеры = "+ factory.getManagerSpawned(),

(getWidth()/2 - infoTab.width/2)+20, (getHeight()/2 - infoTab.height/2) +60);

g.dispose();

screenBuffer.show();

}

}

**AbstractFactory.java**

package tehprog;

import java.awt.\*;

public interface AbstractFactory {

AbstractEmployee getEmployee(long time, Dimension s);

}

**ConcreteFactory.java**

package tehprog;

import javax.imageio.ImageIO;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.IOException;

public class ConcreteFactory implements AbstractFactory{

private int managerTime= 5;

private int managerProb = 100;

private int managerLimit = 20;

private long managerLastSec;

private int developerTime=3;

private int developerProb = 70;

private long developerLastSec;

private BufferedImage developerImg;

private BufferedImage managerImg;

private int managerSpawned;

private int developerSpawned;

public ConcreteFactory(){

try {

managerImg = ImageIO.read(getClass().getResource("/res/manager.png"));

} catch (IOException e) {

System.out.println("manager img error");

}

try {

developerImg = ImageIO.read(getClass().getResource("/res/developer.png"));

} catch (IOException e) {

System.out.println("programmer img error");

}

}

public AbstractEmployee getEmployee(long time, Dimension spawnZone) {

if(time - developerLastSec >= developerTime) {

developerLastSec = time;

if(Math.random()\*100 < developerProb) {

developerSpawned++;

return new Developer(developerImg, new Point((int) (Math.random() \* spawnZone.width),

(int) (Math.random() \* spawnZone.height)));

}

}

if(time - managerLastSec >= managerTime){

managerLastSec = time;

if(managerSpawned < developerSpawned \* managerLimit/100 && Math.random()\*100 < managerProb) {

managerSpawned++;

return new Manager(managerImg, new Point((int) (Math.random() \* spawnZone.width),

(int) (Math.random() \* spawnZone.height)));

}

}

return null;

}

public int getDeveloperSpawned() {

return developerSpawned;

}

public int getManagerSpawned() {

return managerSpawned;

}

public void reset(){

managerSpawned = 0;

developerSpawned = 0;

developerLastSec = 0;

managerLastSec = 0;

}

}

**App.java**

package tehprog;

public class App {

public static void main(String [] a){

Habitat h = new Habitat(1000,700);

}

}

**Manager.java**

package tehprog;

import java.awt.\*;

import java.awt.image.BufferedImage;

public class Manager extends AbstractEmployee{

public Manager(BufferedImage img, Point coords) {

super(img, coords);

}

}

**Developer.java**

package tehprog;

import java.awt.\*;

import java.awt.image.BufferedImage;

public class Developer extends AbstractEmployee{

public Developer(BufferedImage img, Point coords) {

super(img, coords);

}

}

**AbstractEmployee.java**

package tehprog;

import java.awt.\*;

import java.awt.image.BufferedImage;

public abstract class AbstractEmployee{

private BufferedImage img;

private Point coords;

public AbstractEmployee() {

}

public AbstractEmployee(BufferedImage img, Point coords) {

this.img = img;

this.coords = coords;

}

public Point getCoords() {

return coords;

}

public void setCoords(Point coords) {

this.coords = coords;

}

public Image getImg() {

return img;

}

public void setImg(BufferedImage img) {

this.img = img;

}

}