Министерство высшего образования и науки Российской Федерации

Новосибирский государственный технический университет

Лабораторная работа №1 по дисциплине  
«Технология программирования» на тему

***Основы программирования на Java.***

***Обработка событий. Механизм делегирования событий.***

Вариант №7

Факультет: АВТФ Преподаватель:

Группа: АВТ-810 Михайленко Д. А.

Студенты: Степаненко Владислав

Новосибирск

2020

#### Практические задания

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. Разобрать приведенные примеры **FontsList**, **LinesDraw, KeyCodes, Timer**, **DrawFig**.
4. 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.

1. Для решения задачи:

* 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.

При необходимости или желанию можно использовать другие паттерны.

***Вариант 7***

Список жилых домов города состоит из двух типов: капитальный, деревянный. Капитальные дома генерируются каждые N1 секунд с вероятностью P1. Деревянные дома генерируются каждые N2 секунд с вероятностью P2.

**Ход работы:**

Для создание домов разных типов реализованы отдельные классы-наследники (Деревянный – Wood/ Каменный - Stone)абстрактного класса House хранящего поля – координаты. Интерфейс AbstractFactory от которого наследуются “фабрики” для обоих типов домов, фабрики в свою очередь нужны для создания экземпляров каменных и деревянных домов. Каждый дом имеет свое изображение формата .png

В классе Habitat задаются параметры “среды обитания”, симуляции (вероятности и время появления домов). В конструкторе класса описан “слушатель” который будет считывать нажатия клавиатуры и выполнять соответствующие задачи, заданы параметр окна (фрейма), текстовая область JTextArea которая будет использоваться для вывода времени и статистики, массив всех домов. Время сэмулировано при помощи классов TimerTask и Timer, они выполняют метод update каждые 1000 мс. Метод update при каждом вызове проверяет условия среды обитания и при необходимости создает определенные типы домов со случайными координатами и увеличивает их счетчик.

При нажатии B начинается отсчет таймера и сама симуляция.

При нажатии T включается отображение таймера.

При нажатии E происходит остановка таймера и симуляции, очищение массива и области отрисовки, вывод статистики.

В классе main создается объект типа Habitat.

**Листинг программы:**

public interface AbstractFactory  
{  
 House createWood(int x, int y);  
 House createStone(int x, int y);  
}

public class ConcreteFactory implements AbstractFactory  
{  
 @Override  
 public House createStone(int x, int y) {  
 return new Stone(x,y);  
 }  
  
 @Override  
 public House createWood(int x, int y) {  
 return new Wood(x,y);  
 }  
}

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public class Habitat  
{  
 //private Image background;  
 private int width;  
 private int height;  
 private ConcreteFactory allFactory;  
 boolean simulation=false;  
 private Timer timer;  
 private float time;  
 House[] array;  
 private int i;  
  
 //условия: вероятности и время  
 private float t1,t2;  
 private double p1,p2;  
  
 Habitat()  
 {  
 this.height=720;  
 this.width=1280;  
 allFactory= new ConcreteFactory();  
 JFrame frame = new JFrame("Simulation");  
 frame.setSize(width,height);  
 frame.setVisible(true);  
 frame.setResizable(false);  
 i=0;t1=2;t2=3;p1=0.8;p2=0.7;  
 time=0;  
 timer=new Timer();  
 array = new House[1500];  
  
 JPanel panel= new JPanel();  
 JTextArea text =new JTextArea();  
 panel.add(text);  
 frame.add(panel);  
 //text.setPreferredSize(new Dimension(50,50));  
 text.setVisible(false);  
 text.setFont(new Font("Helvetica",Font.*BOLD*,14));  
  
 frame.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent code) {  
 super.keyPressed(code);  
 int key=code.getKeyCode();  
 switch(key)  
 {  
 case KeyEvent.*VK\_B*:  
 {  
 if (simulation==false)  
 {  
 simulation=true;  
 TimerTask tt = new TimerTask()  
 {  
 @Override  
 public void run() {  
 update(time,frame);time=time+1000; text.setText("Время:" + time/1000 );  
 }  
 };  
 timer.schedule(tt,0,1000);  
 }  
 }break;  
 case KeyEvent.*VK\_E*:  
 {  
 timer.cancel();  
 simulation=false;  
  
 text.setEditable(false);  
  
 text.setFont(new Font("TimesRoman", Font.*ITALIC*, 30));  
 text.setText("Всего домов: " + i +  
 "\nДеревянные: " + Wood.*counter* +  
 "\nКаменные: " + Stone.*counter* +  
 "\nВремя: " + (time/1000) + " секунд");  
 frame.repaint();  
 break;  
 }  
 case KeyEvent.*VK\_T*:  
 {  
 if (!text.isVisible())  
 { text.setVisible(true); }  
 else {text.setVisible(false);break; }  
  
 }  
 }  
 }  
 });  
  
 }  
 public void update (float time, JFrame frame)  
 {  
 if (time/1000 % t1 == 0 && time!=0 )  
 {  
 if (Math.*random*() < p1)  
 {  
 array[i]=allFactory.createStone((int)(Math.*random*()\*width),(int)(Math.*random*()\*height));  
 frame.getGraphics().drawImage(array[i].getImage(), array[i].getX(), array[i].getY(), null);  
 i++;  
 }  
 }  
 if (time/1000 % t2 == 0 && time!=0 ) {  
 if (Math.*random*() < p2) {  
 array[i] = allFactory.createWood((int) (Math.*random*() \* width), (int) (Math.*random*() \* height));  
 frame.getGraphics().drawImage(array[i].getImage(), array[i].getX(), array[i].getY(), null);  
 i++;  
 }  
 }  
 }  
}

import java.awt.\*;  
  
public abstract class House implements IBehaviour //все домики  
{  
 public int x; //координаты  
 public int y;  
 public abstract Image getImage();  
  
 @Override  
 public void setX(int x) {  
 this.x = x;  
 }  
 @Override  
 public void setY(int y) {  
 this.y = y;  
 }  
 @Override  
 public int getX() {  
 return x;  
 }  
 @Override  
 public int getY() {  
 return y;  
 }  
}

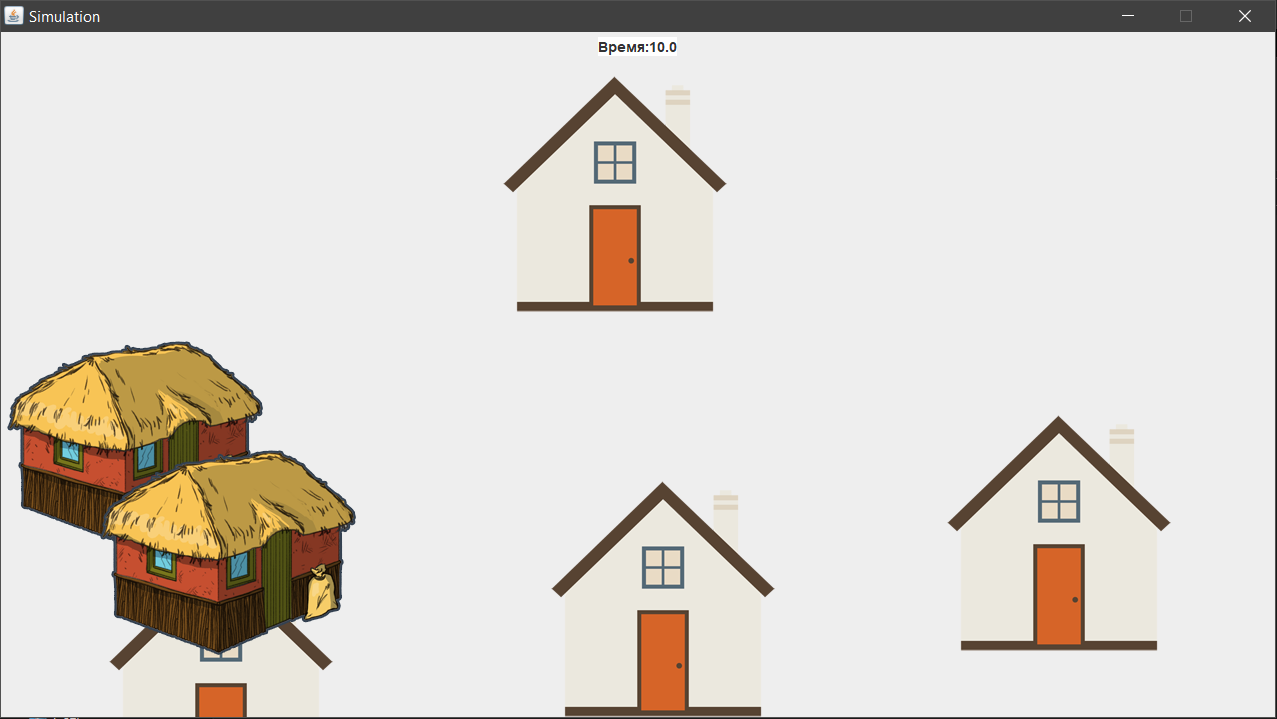
public interface IBehaviour  
{  
 int getX();  
 int getY();  
 void setX(int x);  
 void setY(int y);  
}

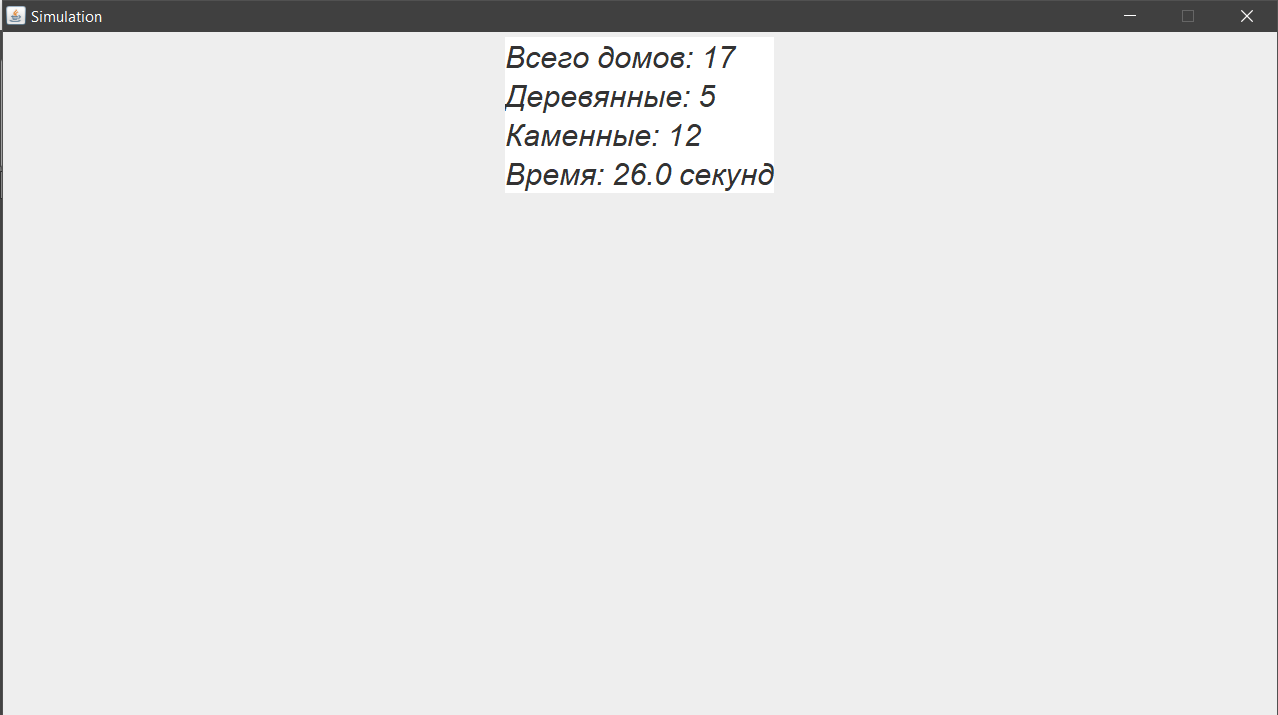
public class Main  
{  
 public static void main(String[] args)  
 {  
 Habitat habitat = new Habitat();  
 }  
  
}

import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.File;  
import java.io.IOException;  
  
public class Stone extends House //деревянные домики  
{  
 static int *counter*;  
 private Image image;  
 public Stone()  
 {  
 setX(0);  
 setY(0);  
 // получаем изображения  
 try  
 {  
 image = ImageIO.*read*(new File("stone.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 *counter*++;  
 }  
 public Stone(int x, int y)  
 {  
 setX(x);  
 setY(y);  
 // получаем изображения  
 try  
 {  
 image = ImageIO.*read*(new File("stone.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 *counter*++;  
 }  
 @Override  
 public Image getImage() {  
 {  
 // получаем изображения  
 try  
 {  
 image = ImageIO.*read*(new File("stone.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 return image;  
 }  
 }  
}

import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.File;  
import java.io.IOException;  
  
public class Wood extends House //деревянные домики  
{  
 static int *counter*;  
 private Image image;  
 public Wood()  
 {  
 setX(0);  
 setY(0);  
 // получаем изображения  
 try  
 {  
 image = ImageIO.*read*(new File("wood.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 *counter*++;  
 }  
 public Wood(int x, int y)  
 {  
 setX(x);  
 setY(y);  
  
 try  
 {  
 image = ImageIO.*read*(new File("wood.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 *counter*++;  
 }  
 @Override  
 public Image getImage() {  
 {  
 // получаем изображения  
 try  
 {  
 image = ImageIO.*read*(new File("wood.png"));  
 }  
 catch(IOException e)  
 {  
 e.printStackTrace();  
 }  
 return image;  
 }  
 }  
}

**Скриншоты работы:**





**Вывод:**

В результате выполнения лабораторной работы было проведено ознакомление особенностями технологии Java и начато изучение синтаксиса языка Java. Были изучены основные понятия и термины обработки событий в модели делегирования событий на Java.

Была разработана программа для упрощенной симуляции поведения объектов, которая будет расширяться и дополняться в дальнейшем. Было проведено ознакомление с паттернами программирования. Основными сложностями при выполнении работы стали работа с изображениями и таймерами.