Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Лабораторная работа №2 по дисциплине  
«Технология программирования» на тему

**Разработка графического интерфейса приложения.**

Вариант №7

Факультет: АВТФ Преподаватель:

Группа: АВТ-810 Михайленко Д. А.

Студенты: Степаненко Владислав

Новосибирск

2020

#### Практические задания

1. Познакомиться с основными графическими библиотеками Java - AWT и Swing и их основными компонентами. Изучить классы менеджеров компоновки, классы создания меню приложения, основных и диалоговых окон и т.д.
2. Для хранения генерируемых объектов использовать динамический массив объектов. Для обеспечения доступа всем элементам приложения и создания только одного экземпляра массива объектов используйте структурный паттерн Singleton.
3. При реализации шаблона Singleton используется класс Singleton. В этом классе определяется закрытый конструктор, имеется закрытая статическая переменная, в которой хранится ссылка на единственный экземпляр данного класса, а также определен статический метод доступа, возвращающий ссылку на этот экземпляр. Так же класс содержит объект коллекции и открытые методы доступа к ней. Таким образом можно в любом месте приложения получить ссылку на объект класса Singleton и доступ к коллекции объектов.
4. Доработать программу, созданную в лабораторной работе № 1:
   1. поделить рабочую область окна приложения на 2 части. Визуализация переносится в одну часть окна, панель управления в другую;
   2. добавить кнопки «Старт» и «Стоп» в панель управления. Они должны запускать и останавливать симуляцию соответственно. Если симуляция остановлена, то кнопка «Стоп» должна блокироваться. Если симуляция идет, то блокируется кнопка «Старт». Клавиши **B** и **E** должны функционировать по-прежнему;
   3. добавить переключатель «Показывать информацию», который разрешает отображение модального диалога из 7 пункта задания;
   4. добавить группу из 2 исключающих переключателей: «Показывать время симуляции» и «Скрывать время симуляции». Клавиша **T** должна функционировать по-прежнему;
   5. используя различные менеджеры компоновки, реализовать интерфейс пользователя согласно индивидуальному заданию;
   6. добавить в программу главное в меню и панель инструментов, в которых продублировать основные команды вашего интерфейса пользователя;
   7. при остановке симуляции должно появляться модальное диалоговое окно (при условии, что оно разрешено) с информацией о количестве и типе сгенерированных объектов, а также времени симуляции. Вся информация выводится в элементе TextArea, недоступном для редактирования. В диалоговом окне должно быть 2 кнопки: «ОК» и «Отмена». При нажатии на «ОК» симуляции останавливается, а при нажатии на «Отмена», соответственно продолжается;
   8. предусмотреть проверку данных вводимых пользователем. При вводе неверного значения обрабатывать исключительную ситуацию: выставлять значение по умолчанию и выводить диалоговое окно с сообщением об ошибке;
   9. Реализовать следующие элементы управления:
   * Периоды рождения объектов – текстовые поля;
   * Для задания вероятностей рождения объектов комбобокс и список (шаг значений 10%);
   * Дополнить интерфейс поясняющими метками.

**Ход работы:**

**1)**Был создан класс UserInterface, весь интерфейс размещен в нем

Окно программы было поделено на части при помощи компоновщика BorderLayout стандартного для JFrame

На западе расположена панель настроек, по центру расположена область симуляции

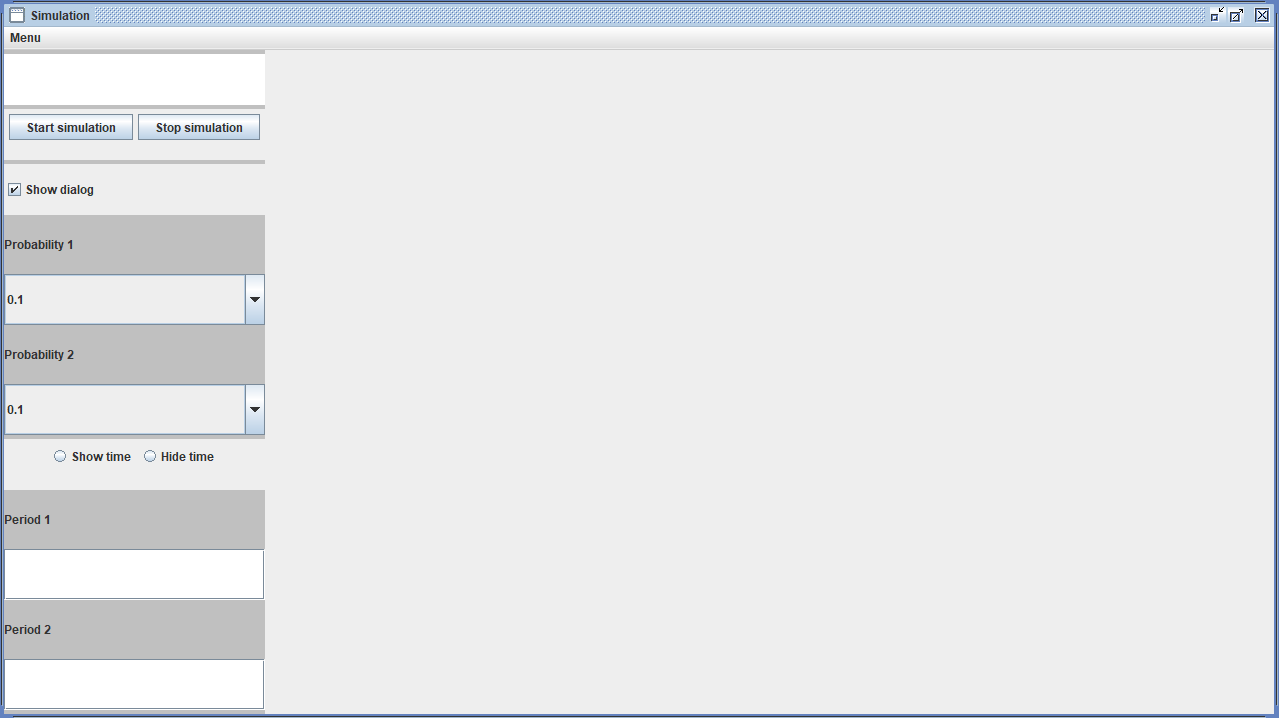


Рисунок - Окно программы

2)Все элементы на панели настроек размещены в столбец при помощи менеджера компоновки GridLayout (задан 1 столбец, 12 строк)

3) Добавлены кнопки Start simulation и Stop simulation при помощи JToggleButton

Если нажата одна их кнопок, вторая неактивна



Рисунок - Вкл\вкл симуляцию

4) Добавлен чекбокс показа диалогового окна (JCheckBox)

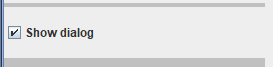


Рисунок – Чекбокс

5)Добавлены комбобоксы для выбора вероятностей появления элементов (JComboBox)

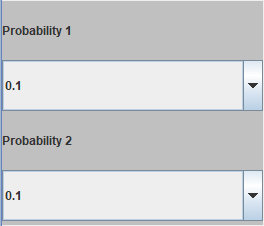


Рисунок – Комбобокс

6)Группа из JRadioButton для вкл\выкл таймера



Рисунок - Вкл\выкл таймера

7) Для задания периодов появления домов были добавлены текстовые поля, ввод значений осуществляется после нажатия Enter

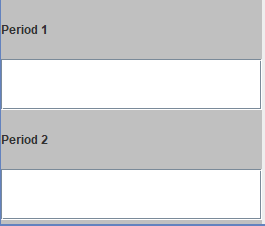


Рисунок - Текстовые поля

В полях реализована проверка вводимых данных, при неверно введеных устанавливаются значения по умолчанию, и выводится окно ошибки.

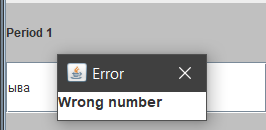


Рисунок - Окно об ошибке

8)Все основные функции продублированы в верхней панели инструментов

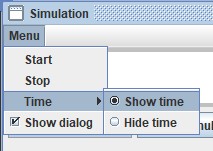


Рисунок - Панель инструментов

Элементы в панели включаются и выключаются соответственно своим аналогам на панели настроек.

Управление с клавиатуры активно при нажатии мыши на область симуляции.

9)При остановке симуляции, в зависимости от заданного значения показа диалога, формируется статистика по симуляции (окно JDialog)

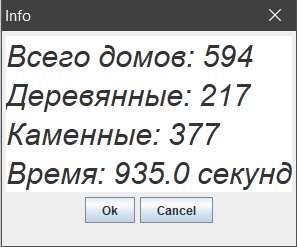


Рисунок – Статистика

При нажатии ОК – симуляция останавливается, Сancel – продолжается

**Листинг программы**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.Timer;  
import java.util.TimerTask;  
  
class Singleton  
{  
 private static Singleton *instance*;  
 private Singleton(){}  
 public static Singleton getInstance()  
 {  
 if(*instance* == null)  
 *instance* = new Singleton();  
 return *instance*;  
 }  
  
 private House[] array = new House[1000];  
  
 public void add(House house, int i)  
 {  
 array[i] = house;  
 }  
  
 public House getHouse(int i)  
 {  
 return array[i];  
 }  
}  
  
  
public class Habitat  
{  
 //private Image background;  
 private int width;  
 private int height;  
 private ConcreteFactory allFactory;  
 boolean simulation=false;  
 private Timer timer;  
 private float time;  
 Singleton array =Singleton.*getInstance*();  
 private int counter;  
  
 //условия: вероятности и время  
 private float t1,t2;  
 private double p1,p2;  
 public class UserInterface  
 {  
 private JFrame frame = new JFrame("Simulation");  
 JPanel simulation\_panel = new JPanel(); // разделение фрейма на область симуляциии и настроек  
 JPanel settings\_panel = new JPanel();  
 JTextArea timerInd = new JTextArea();  
 //панель инструментов  
 JMenuBar menubar = new JMenuBar();  
 JMenu menu = new JMenu("Menu");  
 JMenuItem startMenuItem = new JMenuItem("Start");  
 JMenuItem stopMenuItem = new JMenuItem("Stop");  
 JCheckBoxMenuItem showDialogMenuItem = new JCheckBoxMenuItem("Show dialog");  
 JMenu timeItemMenu = new JMenu("Time");  
 ButtonGroup timeItemMenuGroup = new ButtonGroup();  
 JRadioButtonMenuItem showTimeMenuItem = new JRadioButtonMenuItem("Show time");  
 JRadioButtonMenuItem hideTimeMenuItem = new JRadioButtonMenuItem("Hide time");  
  
 //кнопки  
 JPanel onPanel = new JPanel();  
 ButtonGroup onGroup = new ButtonGroup();  
 JToggleButton start\_button = new JToggleButton("Start simulation");  
 JToggleButton stop\_button = new JToggleButton("Stop simulation");  
 //чекбокс  
 JCheckBox showDialogBox = new JCheckBox("Show dialog");  
 //связанные переключатели  
 JPanel timeSwitchPanel = new JPanel();  
 ButtonGroup bg= new ButtonGroup();  
 JRadioButton showTimeButton= new JRadioButton("Show time");  
 JRadioButton hideTimeButton= new JRadioButton("Hide time");  
 //комбобокс  
 JComboBox cmb1 = new JComboBox();  
 JComboBox cmb2 = new JComboBox();  
 //текстовые зоны  
 JTextField period1 = new JTextField();  
 JTextField period2 = new JTextField();  
  
 UserInterface(){  
 frame.add(settings\_panel,BorderLayout.*WEST*);// настройки слева  
 frame.add(simulation\_panel,BorderLayout.*CENTER*);// симуляция справа  
  
 settings\_panel.setBackground(Color.*LIGHT\_GRAY*);  
 settings\_panel.add(timerInd);  
  
 timerInd.setVisible(true);  
 timerInd.setFont(new Font("Helvetica",Font.*BOLD*,14));  
 timerInd.setEditable(false);  
  
 //добавление элементов к верхней панели инструментов  
 menubar.add(menu);  
 menu.add(startMenuItem);  
 menu.add(stopMenuItem);  
 timeItemMenuGroup.add(showTimeMenuItem);  
 timeItemMenuGroup.add(hideTimeMenuItem);  
 menu.add(timeItemMenu);  
 timeItemMenu.add(showTimeMenuItem);  
 timeItemMenu.add(hideTimeMenuItem);  
 //menu.add(showTimeMenuItem);  
 //menu.add(hideTimeMenuItem);  
  
 menu.add(showDialogMenuItem);  
 frame.setJMenuBar(menubar);  
  
 //добавление кнопок  
 onPanel.add(start\_button);  
 onPanel.add(stop\_button);  
 onGroup.add(start\_button);  
 onGroup.add(stop\_button);  
  
 //связанные переключатели  
 timeSwitchPanel.add(showTimeButton);  
 timeSwitchPanel.add(hideTimeButton);  
 bg.add(showTimeButton);  
 bg.add(hideTimeButton);  
  
 //настройки комобоксов  
 cmb1.setEditable(false);  
 cmb2.setEditable(false);  
  
 for (int i = 1; i < 10;i=i+1 )  
 {  
 cmb1.addItem((double)i/10);  
 cmb2.addItem((double)i/10);  
 }  
 //добавление элементов к панели настроек  
 settings\_panel.setLayout(new GridLayout(12,1,0,4));  
 settings\_panel.add(onPanel);  
 settings\_panel.add(showDialogBox);  
  
 showDialogBox.setSelected(true);  
 //settings\_panel.add(text);  
 settings\_panel.add(new JLabel("Probability 1"));  
 settings\_panel.add(cmb1);  
 settings\_panel.add(new JLabel("Probability 2"));  
 settings\_panel.add(cmb2);  
 settings\_panel.add(timeSwitchPanel);  
 settings\_panel.add(new JLabel("Period 1"));  
 settings\_panel.add(period1);  
 settings\_panel.add(new JLabel("Period 2"));  
 settings\_panel.add(period2);  
  
 simulation\_panel.setVisible(true);  
 settings\_panel.setVisible(true);};  
  
 }  
 Habitat()  
 {  
 JFrame.*setDefaultLookAndFeelDecorated*(true);  
 UserInterface ui = new UserInterface();  
 this.height=720;  
 this.width=1280;  
 allFactory= new ConcreteFactory();  
 //JFrame frame = new JFrame("Simulation");  
 ui.frame.setSize(width,height);  
 ui.frame.setVisible(true);  
 // frame.setResizable(false);  
 counter =0;t1=2;t2=3;p1=0.8;p2=0.7;  
 time=0;  
  
  
 //слушатели  
 ui.simulation\_panel.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 ui.simulation\_panel.requestFocus();  
 //super.mouseClicked(e);  
 }  
 });  
 ui.start\_button.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 startSimulation(ui);  
 }  
  
 });  
 ui.startMenuItem.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e)  
 {  
 startSimulation(ui);  
 }  
  
 });  
  
 ui.stop\_button.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 pauseSimulation(ui);  
 }  
  
 });  
 ui.stopMenuItem.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 pauseSimulation(ui);  
 }  
  
 });  
  
 ui.showDialogMenuItem.setSelected(true);  
 ui.showDialogMenuItem.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.showDialogBox.setSelected(true);}  
 else  
 {ui.showDialogBox.setSelected(false);}  
 }  
 });  
 ui.showDialogBox.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.showDialogMenuItem.setSelected(true);}  
 else  
 {ui.showDialogMenuItem.setSelected(false);}  
 }  
 });  
  
 ui.showTimeMenuItem.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.showTimeButton.setSelected(true);ui.timerInd.setVisible(true);}  
 if (e.getStateChange()== ItemEvent.*DESELECTED*)  
 {ui.showTimeButton.setSelected(false);ui.timerInd.setVisible(false);}  
 }  
 });  
 ui.showTimeButton.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.showTimeMenuItem.setSelected(true);ui.timerInd.setVisible(true);}  
 if (e.getStateChange()== ItemEvent.*DESELECTED*)  
 {ui.showTimeMenuItem.setSelected(false);ui.timerInd.setVisible(false);}  
 }  
 });  
  
 ui.hideTimeMenuItem.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.hideTimeButton.setSelected(true);ui.timerInd.setVisible(false);}  
 if (e.getStateChange()== ItemEvent.*DESELECTED*)  
 {ui.hideTimeButton.setSelected(false);ui.timerInd.setVisible(true);}  
 }  
 });  
 ui.hideTimeButton.addItemListener(new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 if (e.getStateChange()== ItemEvent.*SELECTED*)  
 {ui.hideTimeMenuItem.setSelected(true);ui.timerInd.setVisible(false);}  
 if (e.getStateChange()== ItemEvent.*DESELECTED*)  
 {ui.hideTimeMenuItem.setSelected(false);ui.timerInd.setVisible(true);}  
 }  
 });  
  
 ui.cmb1.addItemListener(e -> p1= (double)ui.cmb1.getSelectedItem());  
 ui.cmb2.addItemListener(e -> p2= (double)ui.cmb2.getSelectedItem());  
 ui.period1.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int period;  
 try  
 {  
 period = Integer.*parseInt*(ui.period1.getText());  
 if (period > 0)  
 {t1=period;}  
 else throw new NumberFormatException();  
 }  
 catch (NumberFormatException ex)  
 {  
 t1=2;showError();  
 }  
 }  
 });  
 ui.period2.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int period;  
 try  
 {  
 period = Integer.*parseInt*(ui.period2.getText());  
 if (period > 0 )  
 t2=period;  
 else throw new NumberFormatException();  
 }  
 catch (NumberFormatException ex)  
 {  
 t2=3;showError();  
 }  
 }  
 });  
 ui.simulation\_panel.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)  
 startSimulation(ui);  
 }break;  
 case KeyEvent.*VK\_E*:  
 {  
 pauseSimulation(ui);  
 break;  
 }  
 case KeyEvent.*VK\_T*:  
 {  
 if (!ui.timerInd.isVisible())  
 { ui.timerInd.setVisible(true);  
 ui.showTimeMenuItem.setSelected(false);  
 ui.showTimeButton.setSelected(false);  
 ui.hideTimeMenuItem.setSelected(true);  
 ui.hideTimeButton.setSelected(true);}  
 else  
 {  
 ui.timerInd.setVisible(false);  
 ui.showTimeMenuItem.setSelected(true);  
 ui.showTimeButton.setSelected(true);  
 ui.hideTimeMenuItem.setSelected(false);  
 ui.hideTimeButton.setSelected(false);  
 break; }  
  
 }  
 }  
 }  
 });  
  
 }  
 private void pauseSimulation(UserInterface ui)  
 {  
 if (simulation==true)  
 {  
 timer.cancel();  
 simulation = false;  
 ui.stop\_button.setEnabled(false);  
 ui.start\_button.setEnabled(true);  
 if (ui.showDialogBox.isSelected())  
 {  
 showDialog(ui);  
 }  
 }  
 }  
 public void startSimulation(UserInterface ui)  
 {  
 if (simulation==false)  
 {  
  
 simulation=true;  
 ui.start\_button.setEnabled(false);  
 ui.stop\_button.setEnabled(true);  
 timer = new Timer();  
 TimerTask tt = new TimerTask()  
 {  
 @Override  
 public void run() {  
 update(time,ui.simulation\_panel);time=time+1000; ui.timerInd.setText("Время:" + time/1000 );  
 }  
 };  
 timer.schedule(tt,0,1000);  
 }  
 }  
 public void showError()  
 {  
 JDialog dialogError = new JDialog();  
 dialogError.setTitle("Error");  
 JTextArea errorInfo = new JTextArea();  
 errorInfo.setFont(new Font("Helvetica",Font.*BOLD*,14));  
 errorInfo.setText("Wrong number");  
 errorInfo.setEditable(false);  
 dialogError.add(errorInfo);  
 dialogError.setVisible(true);  
 errorInfo.setVisible(true);  
 dialogError.setSize(new Dimension(30,75));  
 }  
 public void showDialog(UserInterface ui)  
 {  
 JPanel dialogPanel = new JPanel();  
 JDialog dialog = new JDialog();  
 dialog.setTitle("Info");  
 JButton dialogCancel = new JButton("Cancel");  
 JButton dialogOk = new JButton("Ok");  
 JTextArea dialogText = new JTextArea();  
  
 dialogText.setEditable(false);  
 dialogText.setFont(new Font("TimesRoman", Font.*ITALIC*, 30));  
 dialogText.setText("Всего домов: " + counter +  
 "\nДеревянные: " + Wood.*counter* +  
 "\nКаменные: " + Stone.*counter* +  
 "\nВремя: " + (time/1000) + " секунд");  
  
 dialogPanel.add(dialogText);  
 dialogPanel.add(dialogOk);  
 dialogPanel.add(dialogCancel);  
 dialog.add(dialogPanel);  
 dialog.setSize(new Dimension(300, 250));  
  
  
 dialog.setResizable(false);  
 dialog.setVisible(true);  
  
 dialogOk.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 System.*out*.println("Button pressed");  
 dialog.dispose();  
 time = 0;  
 counter = 0;  
 Wood.*counter* = 0;  
 Stone.*counter* = 0;  
 ui.timerInd.setText("Время:" + time/1000 );  
 ui.frame.repaint();  
 }  
 });  
 dialogCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 ui.stop\_button.setEnabled(true);  
 ui.start\_button.setEnabled(false);  
 startSimulation(ui);  
 dialog.dispose();  
 }  
 });  
 }  
 public void update (float time, JPanel simulation\_panel)  
 {  
 System.*out*.println(p1);  
 System.*out*.println(t1);  
  
 if (time/1000 % t1 == 0 && time!=0 )  
 {  
 if (Math.*random*() < p1)  
 {  
 array.add(allFactory.createStone((int)(Math.*random*()\*simulation\_panel.getWidth()),(int)(Math.*random*()\*simulation\_panel.getHeight())),counter);  
 simulation\_panel.getGraphics().drawImage(array.getHouse(counter).getImage(),  
 array.getHouse(counter).getX(), array.getHouse(counter).getY(), null);  
 counter++;  
 }  
 }  
 if (time/1000 % t2 == 0 && time!=0 ) {  
 if (Math.*random*() < p2) {  
 array.add(allFactory.createWood((int) (Math.*random*() \* simulation\_panel.getWidth()), (int) (Math.*random*() \* simulation\_panel.getHeight())),counter);  
 simulation\_panel.getGraphics().drawImage(array.getHouse(counter).getImage(), array.getHouse(counter).getX(), array.getHouse(counter).getY(), null);  
 counter++;  
 }  
 }  
 }  
}