**Министерство науки и высшего образования Российской Федерации**

**Новосибирский государственный технический университет**

**Кафедра вычислительной техники**

**Лабораторная работа №2**

**по дисциплине:** Технология программирования

**на тему:** Разработка графического интерфейса приложения

Преподаватель: Михайленко Д.А.

Студент: Голиков А.Ю.

Группа: АВТ-808

Новосибирск 2020

**Цель**: 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. Познакомиться с паттерном Singleton.

**Задание**: 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.

Вариант №4

**Ход работы:**

1. Был разработан класс Singleton по одноимённому паттерну проектирования, содержащий в себе статичное поле и статичный метод получения состояния объекта, а также сам список.

private static Singleton *instance*;  
public List<BaseRabbit> rabbits;

1. Главный JFrame – Habitat поделен на три панели:
   1. MenuPanel – содержит в себе меню и панель инструментов
   2. InformationPanel – панель, отображающая таймер и количество объектов
   3. DrawRabbitPanel – панель, отвечающая за отрисовку списка объектов
2. InformationDialog – диалоговое окно, показывающее информацию после окончания симуляции(можно отключить в панели инструментов и в меню)
3. MenuPanel делится на два компонента:
   1. MenuComponent – наследник JMenuBar, содержит в себе пункты меню
      1. private final JMenu simulationMenu;//пункт меню бара  
         private final JMenu viewMenu;//пункт меню бара  
         private final JMenu informationMenu;//пункт меню бара  
         private final JMenuItem simulationStartability;//item меню  
         private final JMenuItem timerVisibility;//item меню  
         private final JMenuItem informationViewability;//item меню
   2. ToolBarComponent – наследник JToolbar, содержит в себе панель инструментов
      1. private final JButton startSimulation;//начать симуляцию  
         private final JButton stopSimulation;//закончить симуляцию  
         private final JCheckBox isInformationPanelAllowed;//чекбокс показа диалогового окна  
         private final ButtonGroup timerPanelVisability;//группа кнопок  
         private final JRadioButton timerAllowed;//показ таймера  
         private final JRadioButton timerNotAllowed;//убрать таймр  
         private final JTextField normalRabbitBirthTime;//поле для получения времени рождения  
         private final JTextField whiteRabbitBirthTime;//поле для получения времени рождения  
         private final JComboBox normalRabbitBirthProbability;//вероятность рождения  
         private final JComboBox rabbitsPercent;// процент от общего числа кроликов
4. В классе Simulation обрабатываются события пользователя с меню и панелью инструментов.

**Результат:**

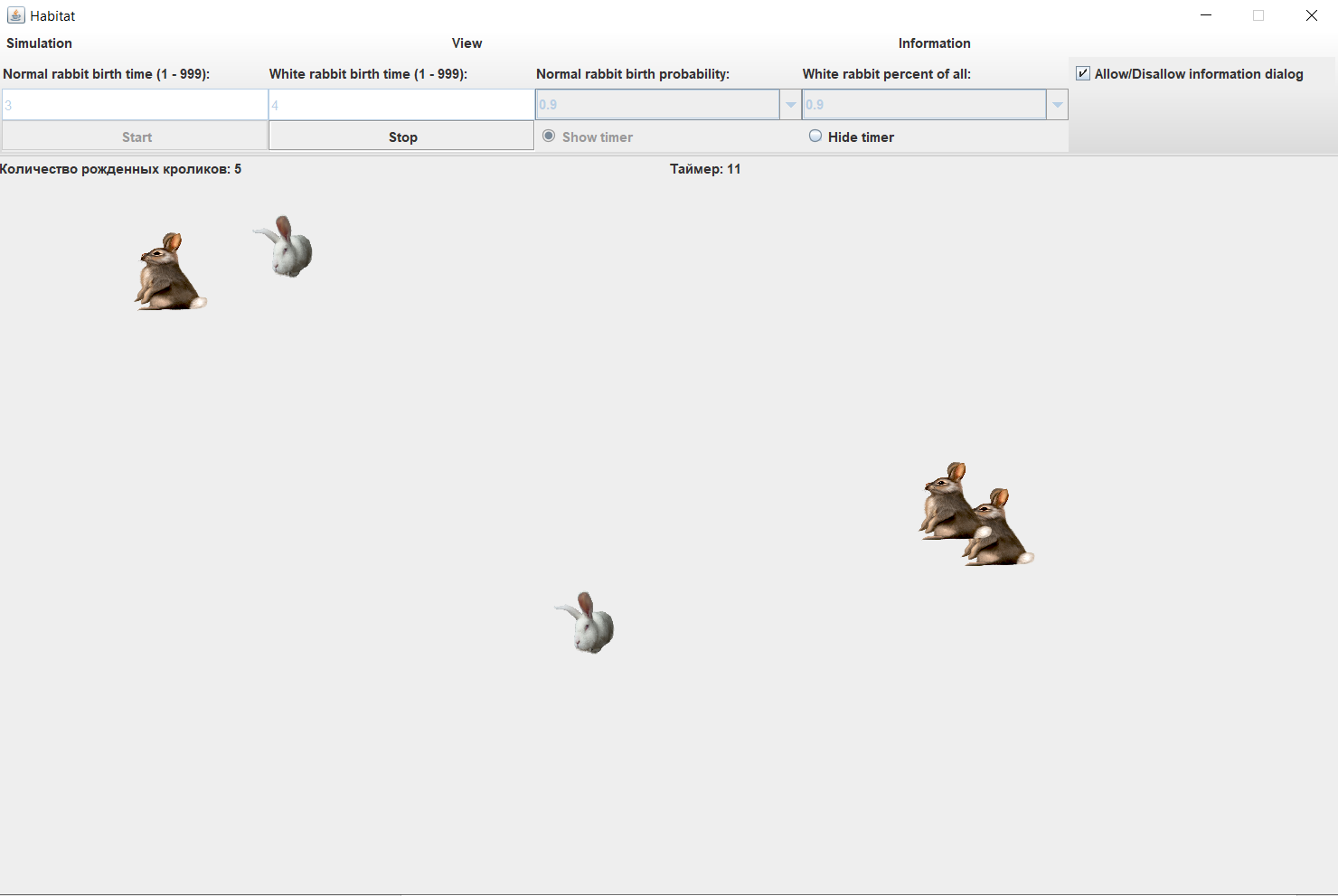


Рис. 1 – Во время симуляции

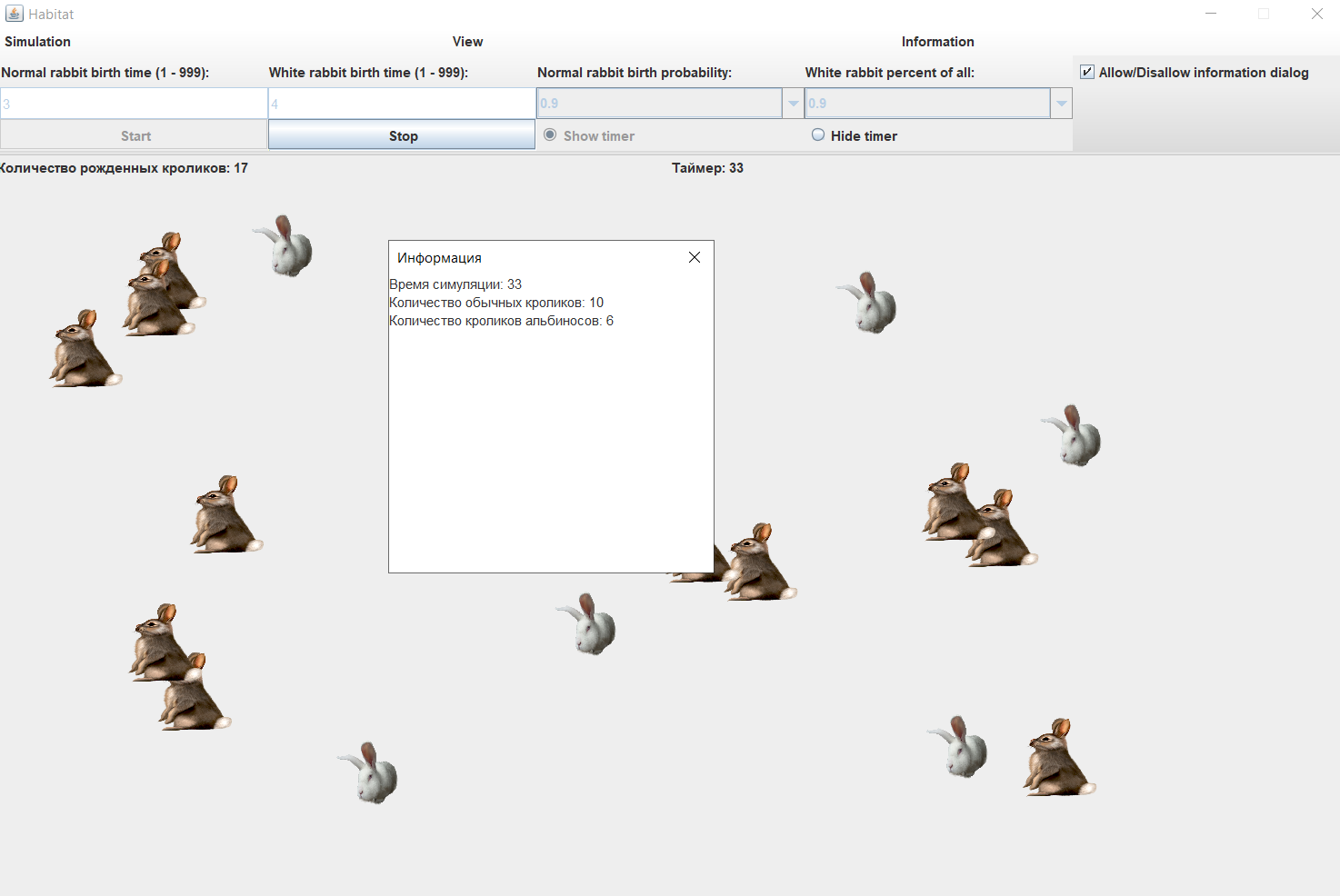


Рис. 2 – Завершение симуляции

**Вывод:** в данной работе была улучшена программа симуляции рождения кроликов, изучены компоненты библиотеки Swing и AWT. Также, был применен паттерн проектирования Singleton.

Код:

BaseRabbit

package Models.Abstract;  
  
import Behaviour.IBehaviour;  
  
import java.awt.\*;  
  
public abstract class BaseRabbit implements IBehaviour {  
 private Point coordinates;  
 protected Image rabbitImage;  
  
 protected BaseRabbit(Point coordinates) {  
 this.coordinates = coordinates;  
 }  
  
 public Image getRabbitImage() {  
 return rabbitImage;  
 }  
  
 public void setRabbitImage(Image rabbitImage) {  
 this.rabbitImage = rabbitImage;  
 }  
  
 public Point getCoordinates() {  
 return coordinates;  
 }  
  
 public void setCoordinates(Point coordinates) {  
 this.coordinates = coordinates;  
 }  
}

WhtieRabbit

package Models;  
  
import Models.Abstract.BaseRabbit;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.IOException;  
  
public class WhiteRabbit extends BaseRabbit {  
  
 public WhiteRabbit(Point coordinates) throws IOException {  
 super(coordinates);  
 this.rabbitImage = ImageIO.*read*(getClass().getResource("/WhiteRabbit.png"));  
 }  
}

NormalRabbit

package Models;  
  
import Models.Abstract.BaseRabbit;  
  
import javax.imageio.ImageIO;  
import java.awt.\*;  
import java.io.IOException;  
  
public class NormalRabbit extends BaseRabbit {  
 public NormalRabbit(Point coordinates) throws IOException {  
 super(coordinates);  
 this.rabbitImage = ImageIO.*read*(getClass().getResource("/Rabbit.png"));  
  
 }  
}

AbstractFactory

package Factory;  
  
import Models.Abstract.BaseRabbit;  
  
import java.awt.\*;  
import java.io.IOException;  
  
public interface AbstractFactory {  
 Integer getAmountOfBirth();//получть количество рожденных объектов  
  
 BaseRabbit birth(Integer time, Point coordinates) throws IOException;//Родить объект BaseRabbit, содержащий в себе ссылку на наследника  
  
 void destroy();//уничтожить скписок объектов  
  
 void setSimulationProperties(Integer normalRabbitBirthTime, Integer whiteRabbitBirthTime, Float normalRabbitBirthProbability, Float rabbitsPercent);  
}

ConcreteFactory

package Factory;  
  
import Models.Abstract.BaseRabbit;  
import Models.NormalRabbit;  
import Models.WhiteRabbit;  
  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
  
public class ConcreteFactory implements AbstractFactory {  
  
 private Integer rabbitsAmount;//количество кроликов  
 private Integer whiteRabbitsAmount;//количество белых кроликов  
 private Integer normalRabbitBirthTime;//время рождения обыкновенного кролика  
 private Integer whiteRabbitBirthTime;//время рождения белого кролика  
 private Float normalRabbitBirthProbability;//вероятность рождения обыкновенного кролика  
 private Float rabbitsPercent;//процент от кроликов  
  
 public ConcreteFactory() {  
 rabbitsAmount = 0;  
 whiteRabbitsAmount = 0;  
 }  
  
 @Override  
 public Integer getAmountOfBirth() {  
 return rabbitsAmount;  
 }  
  
 @Override  
 public BaseRabbit birth(Integer time, Point coordinates) throws IOException {  
 if (time%normalRabbitBirthTime == 0) {  
 Random random = new Random();  
 float probability = random.nextFloat();  
 if (probability <= normalRabbitBirthProbability) {  
 BaseRabbit rabbit = new NormalRabbit(coordinates);  
 rabbitsAmount++;  
 return rabbit;  
 }  
 }  
 if (time%whiteRabbitBirthTime == 0) {  
 if(whiteRabbitsAmount<rabbitsAmount\*rabbitsPercent) {  
 BaseRabbit rabbit = new WhiteRabbit(coordinates);  
 rabbitsAmount++;  
 whiteRabbitsAmount++;  
 return rabbit;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 public void destroy() {  
 rabbitsAmount = 0;  
 whiteRabbitsAmount = 0;  
 }  
  
 @Override  
 public void setSimulationProperties(Integer normalRabbitBirthTime, Integer whiteRabbitBirthTime, Float normalRabbitBirthProbability, Float rabbitsPercent) {  
 this.normalRabbitBirthTime = normalRabbitBirthTime;  
 this.whiteRabbitBirthTime = whiteRabbitBirthTime;  
 this.normalRabbitBirthProbability = normalRabbitBirthProbability;  
 this.rabbitsPercent = rabbitsPercent;  
 }  
}

ToolBarComponent

package Panels.Components;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyEvent;  
  
public class ToolBarComponent extends JToolBar {  
 private final JButton startSimulation;//начать симуляцию  
 private final JButton stopSimulation;//закончить симуляцию  
 private final JCheckBox isInformationPanelAllowed;//чекбокс показа диалогового окна  
 private final ButtonGroup timerPanelVisability;//группа кнопок  
 private final JRadioButton timerAllowed;//показ таймера  
 private final JRadioButton timerNotAllowed;//убрать таймр  
 private final JTextField normalRabbitBirthTime;//поле для получения времени рождения  
 private final JTextField whiteRabbitBirthTime;//поле для получения времени рождения  
 private final JComboBox normalRabbitBirthProbability;//вероятность рождения  
 private final JComboBox rabbitsPercent;// процент от общего числа кроликов  
  
  
 public ToolBarComponent(ActionListener actionListener) {  
 setFloatable(false);  
 setFocusable(false);  
 setLayout(new GridLayout(3,5));  
  
 startSimulation = new JButton("Start");  
 startSimulation.setMnemonic(KeyEvent.*VK\_B*);  
 startSimulation.addActionListener(actionListener);  
  
 stopSimulation = new JButton("Stop");  
 stopSimulation.setMnemonic(KeyEvent.*VK\_E*);  
 stopSimulation.setEnabled(false);  
 stopSimulation.addActionListener(actionListener);  
  
 isInformationPanelAllowed = new JCheckBox("Allow/Disallow information dialog", true);  
 isInformationPanelAllowed.addActionListener(actionListener);  
  
 timerAllowed = new JRadioButton("Show timer");  
 timerAllowed.setSelected(true);  
 timerAllowed.setEnabled(false);  
  
 timerNotAllowed = new JRadioButton("Hide timer");  
  
  
 timerPanelVisability = new ButtonGroup();  
 timerPanelVisability.add(timerAllowed);  
 timerPanelVisability.add(timerNotAllowed);  
  
 timerAllowed.addActionListener(actionListener);  
 timerNotAllowed.addActionListener(actionListener);  
  
 JLabel normalRabbitBirthTimeLabel = new JLabel("Normal rabbit birth time (1 - 999): ");  
 JLabel whiteRabbitBirthTimeLabel = new JLabel("White rabbit birth time (1 - 999): ");  
  
 JLabel normalRabbitBirthProbabilityLabel = new JLabel("Normal rabbit birth probability: ");  
 JLabel whiteRabbitPercentLabel = new JLabel("White rabbit percent of all: ");  
  
  
  
 normalRabbitBirthTime = new JTextField();  
 whiteRabbitBirthTime = new JTextField();  
  
 Float[] probabilityArray = {0.1f, 0.2f, 0.3f, 0.4f, 0.5f, 0.6f, 0.7f, 0.8f, 0.9f, 1f};  
 normalRabbitBirthProbability = new JComboBox(probabilityArray);  
 Float[] rabbitsPercentArray = {0.1f, 0.2f, 0.3f, 0.4f, 0.5f, 0.6f, 0.7f, 0.8f, 0.9f, 1f};  
 rabbitsPercent = new JComboBox(rabbitsPercentArray);  
  
  
 add(normalRabbitBirthTimeLabel);  
 add(whiteRabbitBirthTimeLabel);  
  
 add(normalRabbitBirthProbabilityLabel);  
 add(whiteRabbitPercentLabel);  
  
  
 add(isInformationPanelAllowed);  
  
 add(normalRabbitBirthTime);  
 add(whiteRabbitBirthTime);  
  
 add(normalRabbitBirthProbability);  
 add(rabbitsPercent);  
 add(new JLabel(""));  
 add(startSimulation);  
 add(stopSimulation);  
  
 add(timerAllowed);  
 add(timerNotAllowed);  
  
 }  
  
 public JButton getStartSimulation() {  
 return startSimulation;  
 }  
  
 public JButton getStopSimulation() {  
 return stopSimulation;  
 }  
  
 public JCheckBox getIsInformationPanelAllowed() {  
 return isInformationPanelAllowed;  
 }  
  
 public JRadioButton getTimerAllowed() {  
 return timerAllowed;  
 }  
  
 public JRadioButton getTimerNotAllowed() {  
 return timerNotAllowed;  
 }  
  
 public JTextField getNormalRabbitBirthTime() {  
 return normalRabbitBirthTime;  
 }  
  
 public JTextField getWhiteRabbitBirthTime() {  
 return whiteRabbitBirthTime;  
 }  
  
 public JComboBox getNormalRabbitBirthProbability() {  
 return normalRabbitBirthProbability;  
 }  
  
 public JComboBox getRabbitsPercent() {  
 return rabbitsPercent;  
 }  
  
 public void disableSimulationProperties(){  
 startSimulation.setEnabled(false);  
 normalRabbitBirthTime.setEnabled(false);  
 whiteRabbitBirthTime.setEnabled(false);  
 normalRabbitBirthProbability.setEnabled(false);  
 rabbitsPercent.setEnabled(false);  
 }  
  
 public Boolean checkSimulationProperties() {  
 Boolean check;  
 check = checkTextFieldsForCorrectness(normalRabbitBirthTime);  
 if(!check){  
 check = checkTextFieldsForCorrectness(whiteRabbitBirthTime);  
 return false;  
 }  
 check = checkTextFieldsForCorrectness(whiteRabbitBirthTime);  
 return check;  
 }  
  
 private Boolean checkTextFieldsForCorrectness(JTextField checkTextField){  
 try{  
 Integer checkNumber = Integer.*parseInt*(checkTextField.getText());  
 if (!(checkNumber>=1 && checkNumber<=999)) {  
 checkTextField.setText("Введите корректное число!");  
 return false;  
 }  
 }  
 catch (NumberFormatException e){  
 checkTextField.setText("Введите корректное число!");  
 return false;  
 }  
 return true;  
 }  
  
 public void enableStopButton() {  
 stopSimulation.setEnabled(true);  
 }  
  
 public void enableSimulationProperties() {  
 startSimulation.setEnabled(true);  
 normalRabbitBirthTime.setEnabled(true);  
 whiteRabbitBirthTime.setEnabled(true);  
 normalRabbitBirthProbability.setEnabled(true);  
 rabbitsPercent.setEnabled(true);  
 stopSimulation.setEnabled(false);  
 }  
  
}

MenuComponent

package Panels.Components;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
  
public class MenuComponent extends JMenuBar{  
 private final JMenu simulationMenu;//пункт меню бара  
 private final JMenu viewMenu;//пункт меню бара  
 private final JMenu informationMenu;//пункт меню бара  
 private final JMenuItem simulationStartability;//item меню  
 private final JMenuItem timerVisibility;//item меню  
 private final JMenuItem informationViewability;//item меню  
  
  
 public MenuComponent(ActionListener actionListener){  
 setLayout(new GridLayout(1,3));  
 setFocusable(false);  
 simulationStartability = new JMenuItem("Start/Stop");  
 timerVisibility = new JMenuItem("Show/Hide");  
 informationViewability = new JMenuItem("Allow/Disallow");  
  
 simulationStartability.addActionListener(actionListener);  
 timerVisibility.addActionListener(actionListener);  
 informationViewability.addActionListener(actionListener);  
  
 simulationMenu = new JMenu("Simulation");  
 viewMenu = new JMenu("View");  
 informationMenu = new JMenu("Information");  
  
 simulationMenu.add(simulationStartability);  
 viewMenu.add(timerVisibility);  
 informationMenu.add(informationViewability);  
  
 add(simulationMenu);  
 add(viewMenu);  
 add(informationMenu);  
 }  
  
 public JMenuItem getSimulationStartability() {  
 return simulationStartability;  
 }  
  
 public JMenuItem getTimerVisibility() {  
 return timerVisibility;  
 }  
  
 public JMenuItem getInformationViewability() {  
 return informationViewability;  
 }  
}

InformationDialog

package Panels;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class InformationDialog extends JDialog {  
  
 private final JTextArea results;  
  
 public InformationDialog(JFrame frame, String title, Integer time, Integer normalRabbitsAmount, Integer whiteRabbitsAmount) {  
 super(frame, title, false);  
 setLayout(new BorderLayout());  
 results = new JTextArea();  
 String timerResult = "Время симуляции: " + time + "\n";  
  
 String normalRabbitsResult = "Количество обычных кроликов: " + normalRabbitsAmount + "\n";  
  
 String whiteRabbitsResult = "Количество кроликов альбиносов: " + whiteRabbitsAmount + "\n";  
 results.append(timerResult);  
 results.append(normalRabbitsResult);  
 results.append(whiteRabbitsResult);  
 results.setEditable(false);  
 setResizable(false);  
 setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 setSize(new Dimension(300,300));  
 add(results);  
  
 this.setModal(true);  
  
 }  
  
 public void viewInformation(){  
 results.setVisible(true);  
 setVisible(true);  
 }  
}

InformationPanel

package Panels;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class InformationPanel extends JPanel {  
 private final JLabel rabbitsAmount;  
 private final JLabel timer;  
  
 public InformationPanel() {  
 setLayout(new GridLayout(1,2));  
 rabbitsAmount = new JLabel("Количество рожденных кроликов: 0");  
 timer = new JLabel("Таймер: 0");  
 add(rabbitsAmount);  
 add(timer);  
 }  
  
 public void setRabbitsAmount(Integer rabbitsAmount) {  
 this.rabbitsAmount.setText("Количество рожденных кроликов: " + rabbitsAmount);  
 }  
  
 public void setTimer(Integer timer) {  
 this.timer.setText("Таймер: " + timer);  
 }  
  
 public void setTimerVisibility(Boolean isVisible) {  
 timer.setVisible(isVisible);  
 }  
  
}

MenuPanel

package Panels;  
  
import Panels.Components.MenuComponent;  
import Panels.Components.ToolBarComponent;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
  
public class MenuPanel extends JPanel {  
  
 private final MenuComponent menu;  
 private final ToolBarComponent toolBar;  
  
 public MenuPanel(Integer width, ActionListener actionListener) {  
 setLayout(new BoxLayout(this,BoxLayout.*Y\_AXIS*));  
 menu = new MenuComponent(actionListener);  
 toolBar = new ToolBarComponent(actionListener);  
 menu.setMaximumSize(new Dimension(width, 20));  
 toolBar.setMaximumSize(new Dimension(width, 90));  
 add(menu);  
 add(toolBar);  
  
 }  
  
 public JButton getSimulationButton(){  
 return toolBar.getStartSimulation();  
 }  
  
 public void disableSimulationProperties(){  
 toolBar.disableSimulationProperties();  
 }  
  
 public Boolean checkSimulationProperties() {  
 return toolBar.checkSimulationProperties();  
 }  
  
 public Integer getNormalRabbitBirthTime() {  
 return Integer.*parseInt*(toolBar.getNormalRabbitBirthTime().getText());  
 }  
  
 public Integer getWhiteRabbitBirthTime() {  
 return Integer.*parseInt*(toolBar.getWhiteRabbitBirthTime().getText());  
 }  
  
 public Float getNormalRabbitBirthProbability() {  
 return (Float) toolBar.getNormalRabbitBirthProbability().getSelectedItem();  
 }  
  
 public Float getRabbitPercent() {  
 return (Float) toolBar.getRabbitsPercent().getSelectedItem();  
 }  
  
 public void enableStopButton() {  
 toolBar.enableStopButton();  
 }  
  
 public void enableSimulationProperties() {  
 toolBar.enableSimulationProperties();  
 }  
  
 public JButton getStopSimulationButton() {  
 return toolBar.getStopSimulation();  
 }  
  
 public JRadioButton getHideTimerRadioButton() {  
 return toolBar.getTimerNotAllowed();  
 }  
  
 public JRadioButton getShowTimerRadioButton() {  
 return toolBar.getTimerAllowed();  
 }  
  
 public JCheckBox getInformationDialogSelecter() {  
 return toolBar.getIsInformationPanelAllowed();  
 }  
  
 public JMenuItem getMenuStartStopButton() {  
 return menu.getSimulationStartability();  
 }  
  
 public JMenuItem getMenuShowHideTimerButton() {  
 return menu.getTimerVisibility();  
 }  
  
 public JMenuItem getMenuShowHideInformationDialog() {  
 return menu.getInformationViewability();  
 }  
}

HabitatFrame

package Habitat;  
  
import Habitat.RabbitList.Singleton;  
import Panels.DrawRabbitPanel;  
import Factory.AbstractFactory;  
import Factory.ConcreteFactory;  
import Models.Abstract.BaseRabbit;  
import Models.WhiteRabbit;  
import Panels.InformationDialog;  
import Panels.InformationPanel;  
import Panels.MenuPanel;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.util.Random;  
  
public class HabitatFrame extends JFrame {  
  
 private final AbstractFactory factory;  
 private final Singleton rabbits;  
 private Integer time;  
 private final DrawRabbitPanel drawRabbit;  
 private final MenuPanel menuPanel;  
 private final InformationPanel informationPanel;  
 private Integer whiteRabbitsAmount;  
 private Boolean isInformationPanelAllowed;  
  
 public HabitatFrame(String title, ActionListener actionListener) {  
 super(title);  
 int width = 1200;  
 int height = 800;  
 setSize(width, height);  
 setResizable(false);  
 setFocusable(true);  
 setLayout(new BoxLayout(getContentPane(), BoxLayout.*Y\_AXIS*));  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 this.time = 0;  
 rabbits = Singleton.*getInstance*();  
 factory = new ConcreteFactory();  
  
 isInformationPanelAllowed = true;  
  
 menuPanel = new MenuPanel(width, actionListener);  
 menuPanel.setMaximumSize(new Dimension(width, 110));  
 informationPanel = new InformationPanel();  
 informationPanel.setMaximumSize(new Dimension(width, 20));  
 drawRabbit = new DrawRabbitPanel();  
 drawRabbit.setMaximumSize(new Dimension(width, height - 130));  
  
 add(menuPanel);  
 add(informationPanel);  
 add(drawRabbit);  
  
 whiteRabbitsAmount=0;  
 }  
  
 public void stop(){//остановить отрисовку среды  
 if(isInformationPanelAllowed) {  
 InformationDialog informationDialog = new InformationDialog(this,"Информация",time,factory.getAmountOfBirth() - whiteRabbitsAmount,whiteRabbitsAmount);  
 informationDialog.viewInformation();  
 }  
 factory.destroy();  
 rabbits.*clearRabbits*();  
 time = 0;  
  
 informationPanel.setRabbitsAmount(factory.getAmountOfBirth());  
 informationPanel.setTimer(time);  
 drawRabbit.repaint();  
 }  
   
 public void timerVisibility(Boolean isShown){//скрыть таймер  
 if (isShown)  
 informationPanel.setTimerVisibility(false);  
 else  
 informationPanel.setTimerVisibility(true);  
 }  
  
 void update() {//продолжить симуляцию  
 time++;  
 informationPanel.setTimer(time);  
 Random coordinatesRandom = new Random();  
 int xCoordinate = coordinatesRandom.nextInt(drawRabbit.getWidth()-100);  
 int yCoordinate = coordinatesRandom.nextInt(drawRabbit.getHeight()-100);  
  
 Point coordinates = new Point(xCoordinate, yCoordinate);  
  
 try{  
 BaseRabbit rabbit = factory.birth(time, coordinates);  
 informationPanel.setRabbitsAmount(factory.getAmountOfBirth());  
 if(rabbit != null) {  
 if (rabbit.getClass() == WhiteRabbit.class)  
 whiteRabbitsAmount++;  
 rabbits.*addRabbit*(rabbit);  
 drawRabbit.repaint();  
 }  
 }  
 catch (Exception ex){  
 ex.printStackTrace();  
 }  
 }  
  
 public JButton getStartSimulationButton(){  
 return menuPanel.getSimulationButton();  
 }  
  
 public void disableSimulationProperties(){  
 menuPanel.disableSimulationProperties();  
 }  
  
 public Boolean checkSimulationProperties() {  
 return menuPanel.checkSimulationProperties();  
 }  
  
 public Integer getNormalRabbitBirthTime() {  
 return menuPanel.getNormalRabbitBirthTime();  
 }  
  
 public Integer getWhiteRabbitBirthTime() {  
 return menuPanel.getWhiteRabbitBirthTime();  
 }  
  
 public Float getNormalRabbitBirthProbability() {  
 return menuPanel.getNormalRabbitBirthProbability();  
 }  
  
 public Float getRabbitPercent() {  
 return menuPanel.getRabbitPercent();  
 }  
  
 public void setSimulationProperties(Integer normalRabbitBirthTime, Integer whiteRabbitBirthTime, Float normalRabbitBirthProbability, Float rabbitPercent) {  
 factory.setSimulationProperties(normalRabbitBirthTime, whiteRabbitBirthTime, normalRabbitBirthProbability, rabbitPercent);  
 }  
  
 public void enableStopButton() {  
 menuPanel.enableStopButton();  
 }  
  
 public void enableSimulationProperties() {  
 menuPanel.enableSimulationProperties();  
 }  
  
 public JButton getStopSimulationButton() {  
 return menuPanel.getStopSimulationButton();  
 }  
  
 public JRadioButton getHideTimerRadioButton() {  
 return menuPanel.getHideTimerRadioButton();  
 }  
  
 public JRadioButton getShowTimerRadioButton() {  
 return menuPanel.getShowTimerRadioButton();  
 }  
  
 public JCheckBox getInformationDialogSelecter() {  
 return menuPanel.getInformationDialogSelecter();  
 }  
  
 public void changeInformationPanelVisibility() {  
 isInformationPanelAllowed=!isInformationPanelAllowed;  
 }  
  
 public JMenuItem getMenuStartStopButton() {  
 return menuPanel.getMenuStartStopButton();  
 }  
  
 public JMenuItem getMenuShowHideTimerButton() {  
 return menuPanel.getMenuShowHideTimerButton();  
 }  
  
 public JMenuItem getMenuShowHideInformationDialog() {  
 return menuPanel.getMenuShowHideInformationDialog();  
 }  
}

Singleton

package Habitat.RabbitList;  
  
import Models.Abstract.BaseRabbit;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public final class Singleton {  
 private static Singleton *instance*;  
 public final List<BaseRabbit> rabbits;  
  
 private Singleton() {  
 rabbits = new ArrayList<>();  
 }  
  
 public static void addRabbit(BaseRabbit rabbit) {  
 *instance*.rabbits.add(rabbit);  
 }  
  
 public static void clearRabbits(){  
 if (!*instance*.rabbits.isEmpty())  
 *instance*.rabbits.clear();  
 }  
  
 public static List<BaseRabbit> getRabbits(){  
 return *instance*.rabbits;  
 }  
  
 public static Singleton getInstance(){  
 if (*instance* == null) {  
 *instance* = new Singleton();  
 }  
 return *instance*;  
 }  
}

DrawRabbitPanel

package Panels;  
  
import Habitat.RabbitList.Singleton;  
import Models.Abstract.BaseRabbit;  
import javax.swing.\*;  
import java.awt.\*;  
  
public class DrawRabbitPanel extends JPanel {  
  
 private final Singleton rabbits;  
  
 public DrawRabbitPanel() {  
 rabbits = Singleton.*getInstance*();  
 }  
  
  
 @Override  
 protected void paintComponent(Graphics graphics) {  
 super.paintComponent(graphics);  
 if(rabbits.*getRabbits*().isEmpty())  
 return;  
 for (BaseRabbit rabbit : rabbits.*getRabbits*()) {  
 graphics.drawImage(rabbit.getRabbitImage(),rabbit.getCoordinates().x,rabbit.getCoordinates().y,70,70,null);  
 }  
 }  
}

HabitatTask

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

Simulation

import Habitat.HabitatFrame;  
import Habitat.HabitatTask;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.util.Timer;  
  
class Simulation extends KeyAdapter implements ActionListener{  
 private Timer timer;  
 private HabitatTask habitatTask;  
 private final HabitatFrame habitat;  
 private Boolean isStarted;  
 private Boolean isShown;  
  
 private Simulation() {  
 habitat = new HabitatFrame("Habitat",this);  
 habitat.setVisible(true);  
 habitat.addKeyListener(this);  
 isStarted = false;  
 isShown = true;  
 }  
  
 @Override  
 public void keyPressed(KeyEvent keyEvent) {  
 int key = keyEvent.getKeyCode();  
 if(key == KeyEvent.*VK\_B* && !isStarted) {  
 startSimulation();  
 }  
 if (key == KeyEvent.*VK\_E* && isStarted) {  
 stopSimulation();  
 }  
 if(key == KeyEvent.*VK\_T*) {  
 changeTimerVisibility();  
 }  
 }  
  
 public static void main(String[] args) {  
 new Simulation();  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if(e.getSource() == habitat.getStartSimulationButton()){  
 startSimulation();  
 }  
  
 if(e.getSource() == habitat.getStopSimulationButton()){  
 stopSimulation();  
 }  
  
 if(e.getSource() == habitat.getHideTimerRadioButton() || e.getSource() == habitat.getShowTimerRadioButton()) {  
 changeTimerVisibility();  
 }  
  
 if(e.getSource() == habitat.getInformationDialogSelecter()) {  
 habitat.changeInformationPanelVisibility();  
 }  
  
 if(e.getSource() == habitat.getMenuStartStopButton()) {  
 if(!isStarted) {  
 startSimulation();  
 }  
 else {  
 stopSimulation();  
 }  
 }  
  
 if(e.getSource() == habitat.getMenuShowHideTimerButton()) {  
 changeTimerVisibility();  
 }  
  
 if(e.getSource() == habitat.getMenuShowHideInformationDialog()) {  
 habitat.changeInformationPanelVisibility();  
 habitat.getInformationDialogSelecter().setSelected(!habitat.getInformationDialogSelecter().isSelected());  
 }  
  
 habitat.requestFocus();  
  
 }  
  
 private void startSimulation(){  
 habitat.disableSimulationProperties();  
 if(habitat.checkSimulationProperties()){  
 Integer normalRabbitBirthTime = habitat.getNormalRabbitBirthTime();  
 Integer whiteRabbitBirthTime = habitat.getWhiteRabbitBirthTime();  
 Float normalRabbitBirthProbability = habitat.getNormalRabbitBirthProbability();  
 Float rabbitPercent = habitat.getRabbitPercent();  
 habitat.setSimulationProperties(normalRabbitBirthTime, whiteRabbitBirthTime, normalRabbitBirthProbability, rabbitPercent);  
 startSimulationTask();  
 habitat.enableStopButton();  
 }  
 else {  
 habitat.enableSimulationProperties();  
 }  
 }  
  
 private void stopSimulation(){  
 stopSimulationTask();  
 habitat.enableSimulationProperties();  
 }  
  
 private void startSimulationTask(){  
 isStarted = true;  
 timer = new Timer();  
 habitatTask = new HabitatTask(habitat);  
 timer.schedule(habitatTask,0,1000);  
 }  
  
 private void stopSimulationTask(){  
 timer.cancel();  
 habitatTask.cancel();  
 timer.purge();  
 habitat.stop();  
 isStarted = false;  
 }  
  
 private void changeTimerVisibility(){  
 habitat.timerVisibility(isShown);  
 if(isShown){  
 habitat.getShowTimerRadioButton().setEnabled(true);  
 habitat.getHideTimerRadioButton().setEnabled(false);  
 habitat.getHideTimerRadioButton().setSelected(true);  
  
 }  
 else {  
 habitat.getShowTimerRadioButton().setEnabled(false);  
 habitat.getHideTimerRadioButton().setEnabled(true);  
 habitat.getShowTimerRadioButton().setSelected(true);  
 }  
 isShown = !isShown;  
  
 }  
}