**Министерство науки и высшего образования Российской Федерации**

**Новосибирский государственный технический университет**

**Кафедра вычислительной техники**

**Лабораторная работа №3**

**по дисциплине:** Технология программирования

**на тему:** Классы-коллекции

Преподаватель: Михайленко Д.А.

Студент: Голиков А.Ю.

Группа: АВТ-808

Новосибирск 2020

**Цель**: изучить особенности реализации классов-коллекций в Java.

**Задание**: доработать программу, созданную в лабораторной работе № 2:

1) добавить генерируемым объектам понятия «время рождения» и «время жизни». Время рождения устанавливается в момент генерации объекта, и по значению соответствует времени, прошедшему от начала симуляции. Время жизни – время, через которое объект должен исчезнуть, считая от времени рождения;

2) вынести установку параметров времени жизни объектов в пользовательский интерфейс. Для каждого типа объекта должно задаваться собственное время. Рекомендуется использовать текстовые поля, но следуют помнить о проверке на ввод некорректных данных;

3) добавить генерируемым объектам уникальные целочисленные идентификаторы (случайные числа), которые назначаются при генерации объекта. Для хранения сгенерированных идентификаторов используйте коллекцию удобную для поиска по варианту;

4) использовать коллекции по варианту. При генерации объекта происходит добавление его в коллекцию (в класс добавить поле идентификатора), а во вторую коллекцию: идентификаторы существующих объектов, в третью идентификатор + время рождения. При возникновении события по таймеру обойдите коллекцию и удалите все объекты, время жизни которых истекло, а также все данные во вспомогательных коллекциях; Для обеспечения доступа к коллекциям используйте структурный паттерн Singleton.

5) добавьте в панель управления кнопку «Текущие объекты». По нажатию на эту кнопку появляется модальное диалоговое окно, содержащее список всех «живых» объектов на момент нажатия со временем их рождения (время рождения – ключ). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения. Типы коллекций задаются вариантом.

Вариант №4

**Ход работы:**

1. Был обновлен абстрактный класс BaseRabbit, добавлены целочисленные поля время рождения» и «время жизни», а также поле id типа uuid.

private UUID id;  
private Integer birthTime;  
private Integer deathTime;

1. Были добавлены текстовые поля, для ввода времени смерти каждого типа кроликов, а также кнопка, при нажатии на которую можно увидеть список живых объектов симуляции(симуляция ставится на паузу).
2. AliveRabbitsDialog – модальное окно, содержащее информацию о симуляции (id живого объекта : время рождения)
3. Был модифицирован класс Singleton, для содержания объектов симуляции была использована коллекция Vector, для соответствия Id:Время жизни объекта – HashMap, для содержания id живых объектов – TreeSet.

**Результат:**

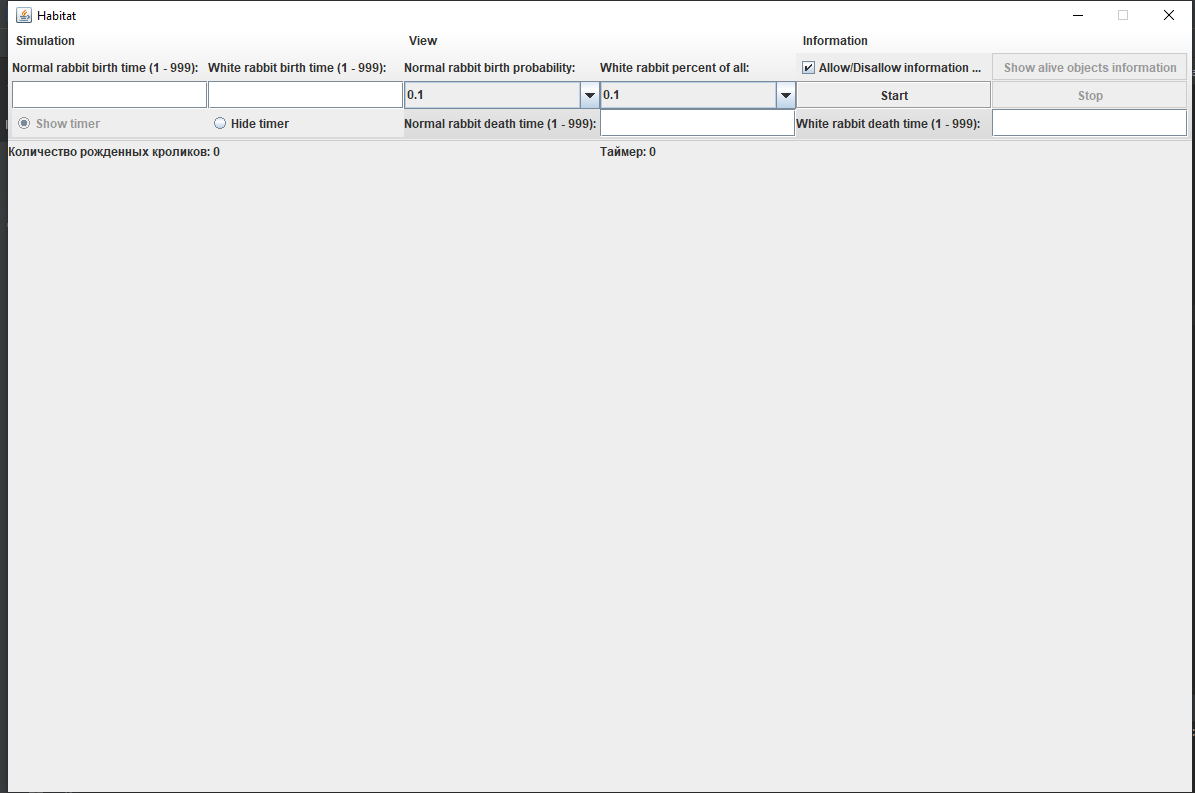


Рис. 1 – Перед симуляцией

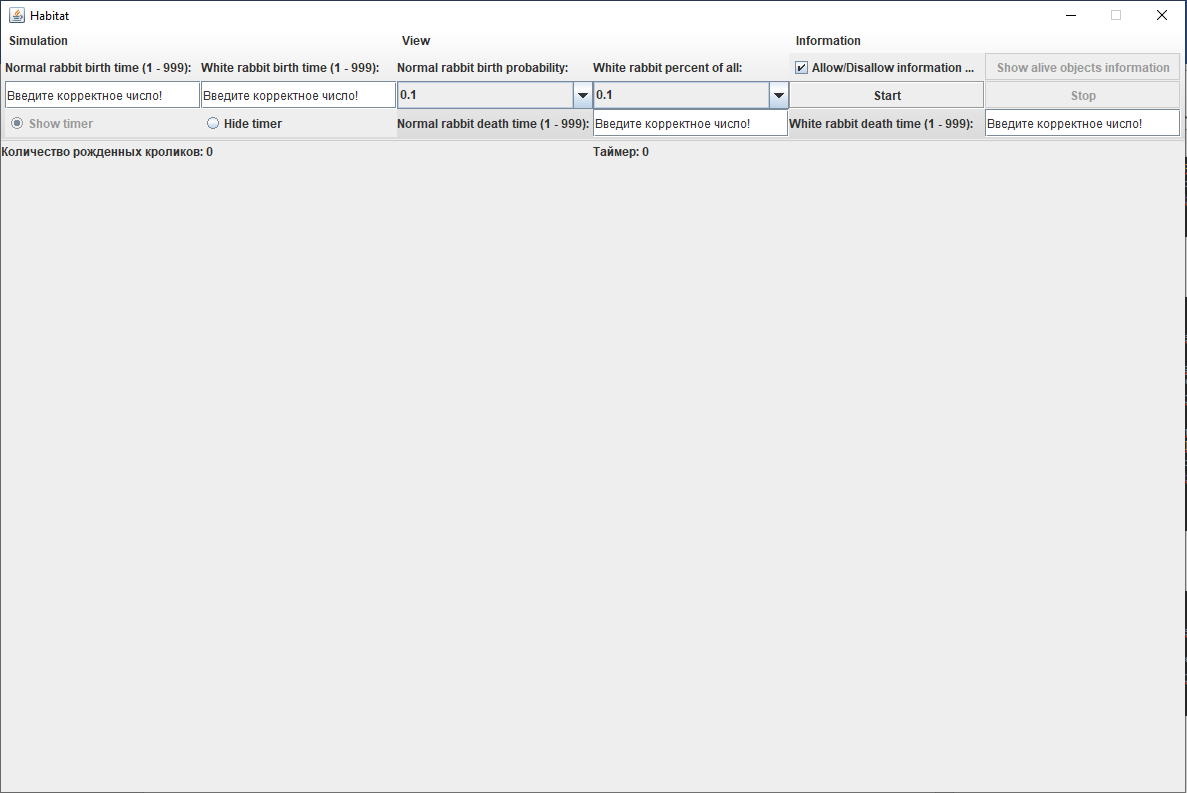


Рис. 2 – Ввод неверных значений

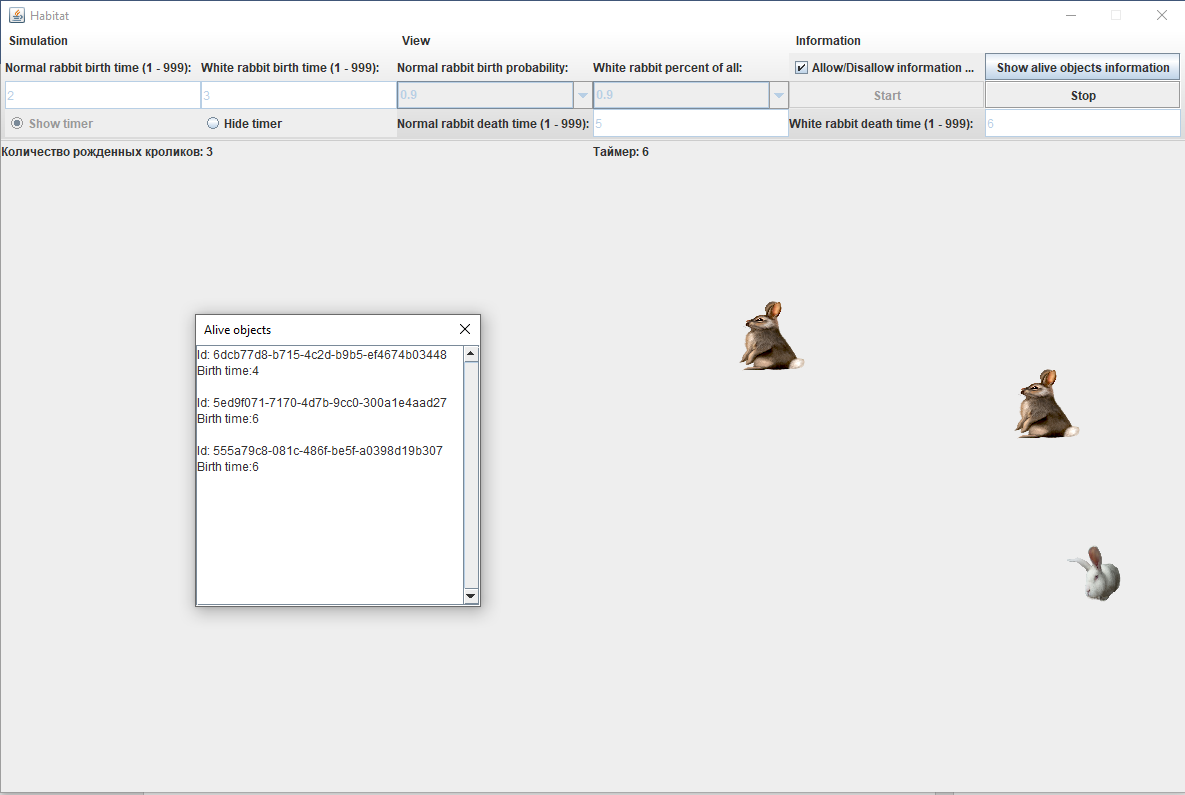


Рис. 3 – При нажатии на кнопку «Информация о живых объектах»

**Вывод:** в данной работе была улучшена программа симуляции рождения кроликов, изучены стандартные типы коллекций языка Java, необходимых для разработки.

Код:

BaseRabbit

package Models.Abstract;  
import java.awt.\*;  
import java.util.UUID;  
  
public abstract class BaseRabbit {  
 private UUID id;  
 private Point coordinates;  
 protected Image rabbitImage;  
 private Integer birthTime;  
 private Integer deathTime;  
  
 protected BaseRabbit(Point coordinates, Integer birthTime, Integer deathTime) {  
 this.coordinates = coordinates;  
 this.birthTime = birthTime;  
 this.deathTime = deathTime;  
 id = UUID.*randomUUID*();  
 }  
  
 public Integer getBirthTime() {  
 return birthTime;  
 }  
  
 public Integer getDeathTime() {  
 return deathTime;  
 }  
  
 public UUID getId() {  
 return id;  
 }  
  
 public Image getRabbitImage() {  
 return rabbitImage;  
 }  
  
 public Point getCoordinates() {  
 return 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 {  
 BaseRabbit birthNormalRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException;//Родить объект BaseRabbit, содержащий в себе ссылку на наследника  
 BaseRabbit birthWhiteRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException;  
}

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 {  
 @Override  
 public BaseRabbit birthNormalRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException {  
 return new NormalRabbit(coordinates, birthTime, deathTime);  
 }  
  
 @Override  
 public BaseRabbit birthWhiteRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException {  
 return new WhiteRabbit(coordinates, birthTime, deathTime);  
 }  
  
}

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  
 private final JTextField normalRabbitDeathTime;  
 private final JTextField whiteRabbitDeathTime;  
 private final JButton showAliveObjectsInformation;  
  
 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);  
  
 var normalRabbitDeathTimeLabel = new JLabel("Normal rabbit death time (1 - 999):");  
 normalRabbitDeathTime = new JTextField();  
 var whiteRabbitDeathTimeLabel = new JLabel("White rabbit death time (1 - 999):");  
 whiteRabbitDeathTime = new JTextField();  
  
 showAliveObjectsInformation = new JButton("Show alive objects information");  
 showAliveObjectsInformation.setEnabled(false);  
  
 showAliveObjectsInformation.addActionListener(actionListener);  
  
 add(normalRabbitBirthTimeLabel);  
 add(whiteRabbitBirthTimeLabel);  
  
 add(normalRabbitBirthProbabilityLabel);  
 add(whiteRabbitPercentLabel);  
  
  
 add(isInformationPanelAllowed);  
 add(showAliveObjectsInformation);  
 add(normalRabbitBirthTime);  
 add(whiteRabbitBirthTime);  
  
 add(normalRabbitBirthProbability);  
 add(rabbitsPercent);  
  
 add(startSimulation);  
 add(stopSimulation);  
  
 add(timerAllowed);  
 add(timerNotAllowed);  
  
 add(normalRabbitDeathTimeLabel);  
 add(normalRabbitDeathTime);  
 add(whiteRabbitDeathTimeLabel);  
 add(whiteRabbitDeathTime);  
 }  
  
 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);  
 whiteRabbitDeathTime.setEnabled(false);  
 normalRabbitDeathTime.setEnabled(false);  
 showAliveObjectsInformation.setEnabled(true);  
 }  
  
 public Boolean checkBirthTimeSimulationProperties() {  
 Boolean check;  
 check = checkTextFieldsForCorrectness(normalRabbitBirthTime);  
 if(!check){  
 check = checkTextFieldsForCorrectness(whiteRabbitBirthTime);  
 return false;  
 }  
 check = checkTextFieldsForCorrectness(whiteRabbitBirthTime);  
 return check;  
 }  
  
 public Boolean checkDeathTimeSimulationProperties() {  
 Boolean check;  
 check = checkTextFieldsForCorrectness(normalRabbitDeathTime);  
 if(!check){  
 check = checkTextFieldsForCorrectness(whiteRabbitDeathTime);  
 return false;  
 }  
 check = checkTextFieldsForCorrectness(whiteRabbitDeathTime);  
 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);  
 normalRabbitDeathTime.setEnabled(true);  
 whiteRabbitDeathTime.setEnabled(true);  
 showAliveObjectsInformation.setEnabled(false);  
 }  
  
 public JTextField getNormalRabbitDeathTime() {  
 return normalRabbitDeathTime;  
 }  
  
 public JTextField getWhiteRabbitDeathTime() {  
 return whiteRabbitDeathTime;  
 }  
  
 public JButton getShowAliveObjectsInformation() {  
 return showAliveObjectsInformation;  
 }  
}

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 Models.NormalRabbit;  
import Models.WhiteRabbit;  
import Panels.\*;  
import Factory.AbstractFactory;  
import Factory.ConcreteFactory;  
import Models.Abstract.BaseRabbit;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
import java.io.IOException;  
import java.util.ArrayList;  
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 Boolean isInformationPanelAllowed;  
  
 private Integer normalRabbitBirthTime;  
 private Integer whiteRabbitBirthTime;  
 private Float normalRabbitBirthProbability;  
 private Float rabbitPercent;  
 private Integer normalRabbitDeathTime;  
 private Integer whiteRabbitDeathTime;  
  
 private Integer whiteRabbitsAmount;  
 private Integer normalRabbitsAmount;  
 private Integer rabbitsAmount;  
 private Integer deathTime = 3;  
  
  
 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*);  
 whiteRabbitsAmount = 0;  
 normalRabbitsAmount = 0;  
 rabbitsAmount=0;  
 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);  
  
 }  
  
 public void stop(){//остановить отрисовку среды  
 if(isInformationPanelAllowed) {  
 EndSimulationDialog endSimulationDialog = new EndSimulationDialog(this,"Информация",time, normalRabbitsAmount, whiteRabbitsAmount);  
 endSimulationDialog.viewInformation();  
 }  
 whiteRabbitsAmount=0;  
 normalRabbitsAmount=0;  
 rabbitsAmount=0;  
 rabbits.*clearRabbits*();  
 time = 0;  
  
 informationPanel.setRabbitsAmount(rabbitsAmount);  
 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++;  
 clearDeathRabbits(time);  
 informationPanel.setTimer(time);  
 Random coordinatesRandom = new Random();  
  
  
 try{  
 if(time%normalRabbitBirthTime==0)  
 {  
 int xCoordinate = coordinatesRandom.nextInt(drawRabbit.getWidth()-100);  
 int yCoordinate = coordinatesRandom.nextInt(drawRabbit.getHeight()-100);  
  
 Point coordinates = new Point(xCoordinate, yCoordinate);  
 BaseRabbit rabbit = birthNormalRabbit(coordinates,time,normalRabbitDeathTime);  
 if(rabbit != null) {  
 rabbits.*addRabbit*(rabbit);  
 }  
 }  
 if (time%whiteRabbitBirthTime==0)  
 {  
 int xCoordinate = coordinatesRandom.nextInt(drawRabbit.getWidth()-100);  
 int yCoordinate = coordinatesRandom.nextInt(drawRabbit.getHeight()-100);  
  
 Point coordinates = new Point(xCoordinate, yCoordinate);  
 BaseRabbit rabbit = birthWhiteRabbit(coordinates,time,whiteRabbitDeathTime);  
 if(rabbit != null) {  
 rabbits.*addRabbit*(rabbit);  
 }  
  
 }  
 informationPanel.setRabbitsAmount(rabbitsAmount);  
 drawRabbit.repaint();  
 }  
 catch (Exception ex){  
 ex.printStackTrace();  
 }  
 }  
  
  
 private BaseRabbit birthNormalRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException {  
 Random random = new Random();  
 float probability = random.nextFloat();  
 if (probability <= normalRabbitBirthProbability) {  
 BaseRabbit rabbit = factory.birthNormalRabbit(coordinates,birthTime,deathTime);  
 rabbitsAmount++;  
 normalRabbitsAmount++;  
 return rabbit;  
 }  
 return null;  
 }  
  
 private BaseRabbit birthWhiteRabbit(Point coordinates, Integer birthTime, Integer deathTime) throws IOException {  
 if(whiteRabbitsAmount<rabbitsAmount\*rabbitPercent) {  
 BaseRabbit rabbit = factory.birthWhiteRabbit(coordinates,birthTime,deathTime);  
 rabbitsAmount++;  
 whiteRabbitsAmount++;  
 return rabbit;  
 }  
 return null;  
 }  
  
 private void clearDeathRabbits(Integer time){  
 var listToDelete = new ArrayList<BaseRabbit>();  
 for (var rabbit: rabbits.*getRabbits*()) {  
  
 if(rabbit instanceof WhiteRabbit && time - rabbit.getBirthTime() == whiteRabbitDeathTime)  
 {  
 listToDelete.add(rabbit);  
 whiteRabbitsAmount--;  
 rabbitsAmount--;  
 }  
 if(rabbit instanceof NormalRabbit && time - rabbit.getBirthTime() == normalRabbitDeathTime)  
 {  
 listToDelete.add(rabbit);  
 normalRabbitsAmount--;  
 rabbitsAmount--;  
 }  
 }  
 for (var rabbit: listToDelete) {  
 rabbits.*clearRabbit*(rabbit);  
 }  
 }  
  
 public JButton getStartSimulationButton(){  
 return menuPanel.getSimulationButton();  
 }  
  
 public void disableSimulationProperties(){  
 menuPanel.disableSimulationProperties();  
 }  
  
 public Boolean checkBirthSimulationProperties() {  
 return menuPanel.checkBirthSimulationProperties();  
 }  
  
 public Boolean checkDeathSimulationProperties() {  
 return menuPanel.checkDeathSimulationProperties();  
 }  
  
 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, Integer normalRabbitDeathTime, Integer whiteRabbitDeathTime) {  
 this.normalRabbitBirthTime = normalRabbitBirthTime;  
 this.whiteRabbitBirthTime = whiteRabbitBirthTime;  
 this.normalRabbitBirthProbability = normalRabbitBirthProbability;  
 this.rabbitPercent = rabbitPercent;  
 this.normalRabbitDeathTime = normalRabbitDeathTime;  
 this.whiteRabbitDeathTime = whiteRabbitDeathTime;  
 }  
  
 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();  
 }  
  
 public Integer getNormalRabbitDeathTime() {  
 return menuPanel.getNormalRabbitDeathTime();  
 }  
  
 public Integer getWhiteRabbitDeathTime() {  
 return menuPanel.getWhiteRabbitDeathTime();  
 }  
  
 public JButton getShowAliveObjectsInformation(){  
 return menuPanel.getShowAliveObjectsInformation();  
 }  
  
 public void showAliveObjectsInformation() {  
 AliveRabbitsDialog aliveRabbitsDialog = new AliveRabbitsDialog(this,"Alive objects");  
 aliveRabbitsDialog.viewInformation();  
 }  
}

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());  
 }  
  
 if(e.getSource() == habitat.getShowAliveObjectsInformation()){  
 timer.cancel();  
 habitatTask.cancel();  
 timer.purge();  
  
 habitat.showAliveObjectsInformation();  
 timer = new Timer();  
 habitatTask = new HabitatTask(habitat);  
 timer.schedule(habitatTask,0,1000);  
  
 }  
 habitat.requestFocus();  
 }  
  
 private void startSimulation(){  
 habitat.disableSimulationProperties();  
 var isBirthPropertiesCorrect = habitat.checkBirthSimulationProperties();  
 var isDeathPropertiesCorrect = habitat.checkDeathSimulationProperties();  
 if(isBirthPropertiesCorrect && isDeathPropertiesCorrect){  
 Integer normalRabbitBirthTime = habitat.getNormalRabbitBirthTime();  
 Integer whiteRabbitBirthTime = habitat.getWhiteRabbitBirthTime();  
 Float normalRabbitBirthProbability = habitat.getNormalRabbitBirthProbability();  
 Float rabbitPercent = habitat.getRabbitPercent();  
 Integer normalRabbitDeathTime = habitat.getNormalRabbitDeathTime();  
 Integer whiteRabbitDeathTime = habitat.getWhiteRabbitDeathTime();  
  
 habitat.setSimulationProperties(normalRabbitBirthTime, whiteRabbitBirthTime, normalRabbitBirthProbability, rabbitPercent, normalRabbitDeathTime, whiteRabbitDeathTime);  
 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;  
  
 }  
}

Singletone:

package Habitat.RabbitList;  
  
import Models.Abstract.BaseRabbit;  
  
import java.util.\*;  
  
public final class Singleton {  
 private static Singleton *instance*;  
 private final List<BaseRabbit> rabbits;  
 private final Set<UUID> rabbitsIds;  
 private final Map<UUID,Integer> rabbitMap;  
  
 private Singleton() {  
 rabbits = new Vector<>();  
 rabbitsIds = new TreeSet<>();  
 rabbitMap=new HashMap<>();  
 }  
  
 public static void addRabbit(BaseRabbit rabbit) {  
 *instance*.rabbits.add(rabbit);  
 *instance*.rabbitsIds.add(rabbit.getId());  
 *instance*.rabbitMap.put(rabbit.getId(),rabbit.getBirthTime());  
 }  
  
 public static void clearRabbits(){  
 if (!*instance*.rabbits.isEmpty())  
 {  
 *instance*.rabbits.clear();  
 *instance*.rabbitsIds.clear();  
 *instance*.rabbitMap.clear();  
 }  
  
 }  
  
 public static List<BaseRabbit> getRabbits(){  
 return *instance*.rabbits;  
 }  
  
 public static Map<UUID, Integer> getRabbitsInformation(){  
 return *instance*.rabbitMap;  
 }  
  
 public static void clearRabbit(BaseRabbit rabbit){  
 *instance*.rabbits.remove(rabbit);  
 *instance*.rabbitsIds.remove(rabbit.getId());  
 *instance*.rabbitMap.remove(rabbit.getId());  
 }  
  
  
 public static Singleton getInstance(){  
 if (*instance* == null) {  
 *instance* = new Singleton();  
 }  
 return *instance*;  
 }  
}

AliveRabbitsDialog:

package Panels;  
  
import Habitat.RabbitList.Singleton;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class AliveRabbitsDialog extends JDialog {  
 private final JTextArea results;  
  
 public AliveRabbitsDialog(JFrame frame, String title) {  
 super(frame, title, true);  
 results = new JTextArea();  
  
 setLayout(new BoxLayout(getContentPane(), BoxLayout.*X\_AXIS*));  
 var rabbits = Singleton.*getInstance*();  
 for (var rabbitId: rabbits.*getRabbitsInformation*().keySet()) {  
 results.append("Id: " + rabbitId +"\n"+"Birth time:"+rabbits.*getRabbitsInformation*().get(rabbitId) + "\n\n");  
 }  
 results.setEditable(false);  
 setResizable(false);  
 setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 setSize(new Dimension(300,300));  
 JScrollPane scrollPane = new JScrollPane(results,JScrollPane.*VERTICAL\_SCROLLBAR\_ALWAYS*,JScrollPane.*HORIZONTAL\_SCROLLBAR\_NEVER*);  
 add(scrollPane);  
 setFocusable(true);  
 }  
  
 public void viewInformation(){  
 results.setVisible(true);  
 setVisible(true);  
 }  
}