Министерство науки и высшего образования РФ

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Кафедра Вычислительной техники

Новосибирск 2020

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АВТ-814  Студент: Альжанов В.А. | Преподаватель:  Васюткина Ирина Александровна |

Цель работы

Лабораторная работа 3

“Классы-коллекции”

по дисциплине “Технологии программирования”

1. Изучить особенности реализации классов-коллекций в Java.
2. Доработать программу, созданную в лабораторной работе № 2.

Задание

Вариант 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;
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 (тип объекта, время рождения, идентификатор). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения. Типы коллекций задаются вариантом.

Коллекция для хранения объектов: ArrayList

Коллекция для хранения и поиска уникальных идентификаторов: HashSet

Коллекция для хранения времени рождения объектов: TreeMap

Демонстрация работы программы

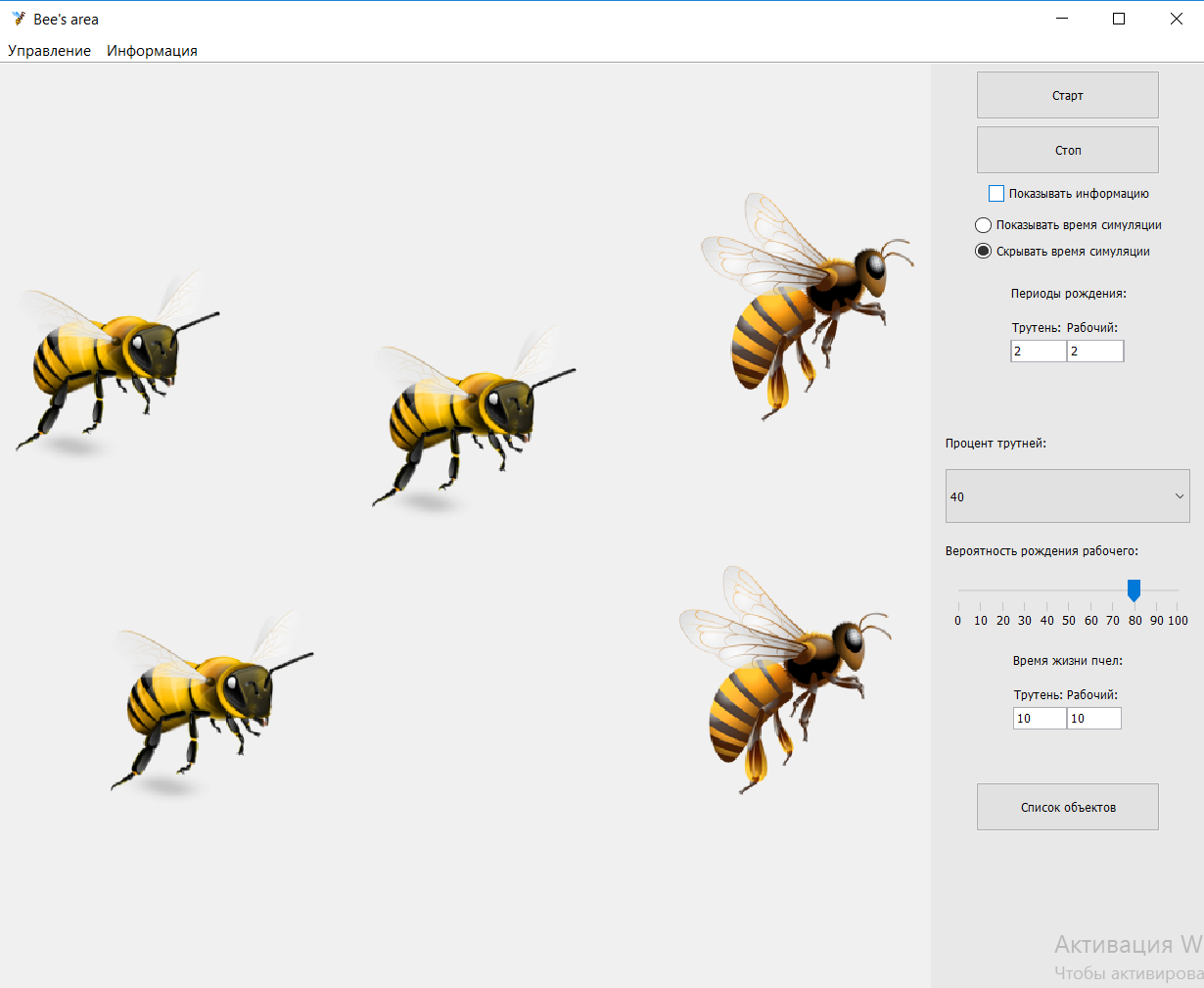


Рис. 1 – Результат выполнения программы

Описание блоков программы

В **abstract class Bee** были добавленыполя **int id** и **long timeOfBirth**. Первое поле предназначено для хранения уникальных идентификаторов сгенерированных объектов, а второе – для хранения времени рождения объекта.

В классе **GUI** на **JPanel control** была прикреплена кнопка **JButton currentObjectsButton**, вызывающая модальное диалоговое окно, в котором в не редактируемом текстовом поле отображается информация о типе объекта, уникальном идентификаторе и времени его рождения. Получение времени рождения осуществляется с использованием ключа-идентификатора через обращение к коллекции **HashMap birthdays**.

В классе **Habitat** появились поля **droneTimeOfLife, WorkerTimeOfLife**, хранящие время жизни объектов соответствующих классов. Также были добавлены коллекции **ArrayList<Bee> array**, которая пришла на замену массиву **Pet[] array**, **HashSet<Integer> identifiers**, которая хранит в себе идентификаторы сгенерированных объектов, **TreeMap<Integer, Long> birthdays**, предназначенная для хранения времени рождения сгенерированных объектов.

В методе **update()** появился цикл, в котором происходит проверка на тип объекта и сравнение прошедшего с начала симуляции времени со временем его жизни. При выполнении всех условий происходит удаление объекта и его данных из соответствующих коллекций.

for (int i = 0; i < Habitat.*array*.size(); i++) {  
 if (Habitat.*array*.get(i) instanceof Drone && (time - Habitat.*array*.get(i).timeOfBirth) >= *droneTimeOfLife*) {  
 *array*.remove(i);  
 *birthdays*.remove(i);  
 *identifiers*.remove(i);  
 --i;  
 --*iK*;  
 System.*out*.println("Drone has deleted.");  
 } else if (Habitat.*array*.get(i) instanceof Worker && (time - Habitat.*array*.get(i).timeOfBirth) >= *WorkerTimeOfLife*) {  
 *array*.remove(i);  
 *birthdays*.remove(i);  
 *identifiers*.remove(i);  
 --i;  
 System.*out*.println("Worker has deleted.");  
 }  
}

При генерации теперь происходит запись уникального идентификатора и времени рождения в соответствующие коллекции.

id = new Random().nextInt();  
Drone drone = new Drone(droneImg, x, y, id, time);  
*array*.add(drone);  
*identifiers*.add(id);  
*birthdays*.put(id, drone.timeOfBirth);

При запуске симуляции происходит очистка всех трех коллекций.

Habitat.*array*.clear();  
 Habitat.*identifiers*.clear();  
 Habitat.*birthdays*.clear();

Выводы

В результате выполнения работы мы изучили особенности реализации классов-коллекций в Java.

Приложение

**Листинг программы**

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

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.Timer;  
import java.util.TimerTask;  
import java.util.concurrent.atomic.AtomicLong;  
  
class GUI extends JFrame {  
 private boolean isTimerRunning = false,  
 isTimeVisible = false,  
 isStatisticsVisible = false,  
 isShowModalDialog = false;  
 private AtomicLong timeSum = new AtomicLong(0);  
 private Timer timer;  
 private long time;  
  
 private void start() {  
 Habitat.*array*.clear();  
 Habitat.*identifiers*.clear();  
 Habitat.*birthdays*.clear();  
 Habitat.*numberOfDrones* = 0;  
 Habitat.*numberOfWorkers* = 0;  
 timer = new Timer();  
 isTimerRunning = true;  
 time = 0; // аннулируем время которое передаем в update()  
 timeSum.set(0); // аннулируем время которое отображается на экране  
 }  
  
 private int stop() {  
 Object[] options = {  
 "ОК",  
 "Отмена"};  
 TextArea statTextArea = new TextArea("Время симуляции: " + timeSum + " с \nИтого пчел: " +  
 (Habitat.*numberOfDrones* + Habitat.*numberOfWorkers*) +  
 "\nТрутней: " + Habitat.*numberOfDrones* + "\nРабочих: " + Habitat.*numberOfWorkers*);  
 statTextArea.setEditable(false);  
 statTextArea.setFont(new Font("Arial", Font.*PLAIN*, 16));  
 statTextArea.setPreferredSize(new Dimension(200, 100));  
 return JOptionPane.*showOptionDialog*(null,  
 statTextArea,  
 "Результаты симуляции",  
 JOptionPane.*YES\_NO\_OPTION*,  
 JOptionPane.*INFORMATION\_MESSAGE*,  
 null,  
 options,  
 null);  
 }  
  
 private void \_continue() {  
 timer = new Timer();  
 isTimerRunning = true;  
 }  
  
 GUI() {  
 super("Bee's area");  
 final int DEFAULT\_X = 1000; // первоначальные размеры фрейма  
 final int DEFAULT\_Y = 1000;  
 this.setSize(DEFAULT\_X, DEFAULT\_Y);  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // завершение программы при закрытии фрейма  
 this.setMinimumSize(new Dimension(450, 600));  
 this.setIconImage(new ImageIcon("Icon.png").getImage());  
 this.setLocationRelativeTo(null); // фрейм в центре экрана  
 this.setFocusable(true); // фокус на фрейме при запуске  
 try {  
 UIManager.*setLookAndFeel*(UIManager.*getSystemLookAndFeelClassName*());  
 } catch (ClassNotFoundException | IllegalAccessException | InstantiationException | UnsupportedLookAndFeelException e) {  
 e.printStackTrace();  
 }  
  
 // глобальный контейнер  
 JPanel scene = new JPanel();  
 add(scene);  
 scene.setLayout(new BorderLayout());  
  
 // панель управления симуляцией  
 JPanel control = new JPanel();  
 Dimension controlSize = new Dimension(225, getHeight());  
 control.setPreferredSize(controlSize);  
 control.setLayout(new FlowLayout(FlowLayout.*CENTER*));  
 control.setBackground(new Color(232, 232, 232));  
 scene.add(control, BorderLayout.*EAST*);  
  
 // панель визуализации объектов  
 int visualX = getWidth() - controlSize.width - 17;  
 int visualY = getHeight();  
 VPanel visual = new VPanel(visualX, visualY);  
 scene.add(visual, BorderLayout.*WEST*);  
  
 // время симуляции  
 JLabel timeModule = new JLabel("Время: " + timeSum + " с");  
 timeModule.setFont(new Font("Times New Roman", Font.*PLAIN*, 25));  
 visual.add(timeModule);  
 timeModule.setVisible(false);  
  
 // статистика симуляции  
 JLabel statistics = new JLabel();  
 statistics.setFont(new Font("Arial", Font.*PLAIN*, 25));  
 statistics.setBackground(visual.getBackground());  
 visual.add(statistics);  
  
 // создание рабочей области  
 Habitat habitat = new Habitat(visualX, visualY);  
  
 class SimulationTask extends TimerTask {  
 private long time;  
 private Habitat simulation;  
  
 SimulationTask(Habitat simulation, long time) {  
 this.simulation = simulation;  
 this.time = time;  
 }  
  
 @Override  
 public void run() {  
 timeSum.set(timeSum.get() + 1);  
 simulation.update(time); // обновление рабочей области  
 time++;  
 repaint();  
 timeModule.setText("Время: " + timeSum.get() + " с");  
 }  
 }  
  
 this.addKeyListener(new KeyAdapter() {  
 Timer timer;  
 long time;  
 SimulationTask updating = null;  
  
 @Override  
 public void keyPressed(KeyEvent evt) {  
  
 char event = evt.getKeyChar();  
 if (event == 'и' || event == 'B' || event == 'И') event = 'b';  
 if (event == 'у' || event == 'E' || event == 'У') event = 'e';  
 if (event == 'е' || event == 'T' || event == 'Е') event = 't';  
  
 switch (event) {  
 case 'b':  
 if (!isTimerRunning) { // без условия таймер бы ускорялся  
 repaint(); // очистка фрейма перерисовкой  
 if (isStatisticsVisible) {  
 statistics.setVisible(false);  
 isStatisticsVisible = false;  
 }  
  
 Habitat.*numberOfDrones* = 0;  
 Habitat.*numberOfWorkers* = 0;  
 timer = new Timer();  
 isTimerRunning = true;  
 time = 0; // аннулируем время которое передаем в update()  
 timeSum.set(0); // аннулируем время которое отображается на экране  
 updating = new SimulationTask(habitat, time);  
 timer.schedule(updating, 0, 1000);  
 }  
 break;  
 case 'e':  
 if (isTimerRunning) {  
 timer.cancel();  
 isTimerRunning = false;  
 }  
 if (!isStatisticsVisible) {  
 if (isTimeVisible) {  
 timeModule.setVisible(false);  
 isTimeVisible = false;  
 }  
  
 statistics.setText("<html>Время симуляции: " + timeSum + " с <br>Итого пчел: " +  
 (Habitat.*numberOfDrones* + Habitat.*numberOfWorkers*) +  
 "<br>Трутней: " + Habitat.*numberOfDrones* + "<br>Рабочих: " + Habitat.*numberOfWorkers*);  
 statistics.setVisible(true);  
 repaint(); // очистка фрейма перерисовкой  
 isStatisticsVisible = true;  
 }  
 break;  
 case 't':  
 if (!isStatisticsVisible) {  
 if (!isTimeVisible) {  
 timeModule.setVisible(true);  
 isTimeVisible = true;  
 } else {  
 timeModule.setVisible(false);  
 isTimeVisible = false;  
 }  
 }  
 break;  
 default:  
 break;  
 }  
 }  
 });  
  
 // кнопки старт и стоп  
 Dimension buttonSize = new Dimension(150, 40);  
 JButton startButton = new JButton("Старт");  
 startButton.setPreferredSize(buttonSize);  
 startButton.addActionListener(new ActionListener() {  
 SimulationTask updating = null;  
  
 public void actionPerformed(ActionEvent e) {  
 if (!isTimerRunning) { // без условия таймер бы ускорялся  
 if (isStatisticsVisible) {  
 statistics.setVisible(false);  
 isStatisticsVisible = false;  
 }  
 start();  
 updating = new SimulationTask(habitat, time);  
 timer.schedule(updating, 0, 1000);  
 }  
 requestFocus();  
 }  
 });  
 control.add(startButton);  
  
 JButton stopButton = new JButton("Стоп");  
 stopButton.setPreferredSize(buttonSize);  
 stopButton.addActionListener(new ActionListener() {  
 SimulationTask updating = null;  
  
 public void actionPerformed(ActionEvent e) {  
 if (isTimerRunning) {  
 timer.cancel();  
 isTimerRunning = false;  
 }  
 if (isShowModalDialog) {  
 int response = stop();  
 switch (response) {  
 case JOptionPane.*YES\_OPTION*:  
 System.*out*.println("JOptionPane.YES\_Option");  
 Habitat.*array*.clear(); // очистка коллекции  
 repaint();  
 break;  
 case JOptionPane.*NO\_OPTION*:  
 System.*out*.println("JOptionPane.NO\_Option");  
 \_continue();  
 updating = new SimulationTask(habitat, time);  
 timer.schedule(updating, 0, 1000);  
 }  
 } else {  
 Habitat.*array*.clear(); // очистка коллекции  
 repaint();  
 }  
 requestFocus();  
 }  
 });  
 control.add(stopButton);  
  
 // флажок показывать информацию. объединена в группу кнопок с ее дублированным вариантом в меню  
 JCheckBox statShow = new JCheckBox("Показывать информацию");  
 statShow.setBackground(control.getBackground());  
 statShow.addActionListener(e -> {  
 System.*out*.println("Information is shown/hidden.");  
 isShowModalDialog = !isShowModalDialog; // меняем состояние булеаны  
 requestFocus();  
 });  
 control.add(statShow);  
  
 // панель переключателей отображения времени симуляции  
 JPanel timeRadioButtonPanel = new JPanel(new GridLayout(2, 1));  
 timeRadioButtonPanel.setBackground(control.getBackground());  
 control.add(timeRadioButtonPanel);  
  
 // переключатели отображения времени симуляции  
 JRadioButton showSimTime = new JRadioButton("Показывать время симуляции");  
 showSimTime.addActionListener(e -> {  
 System.*out*.println("Simulation time is shown.");  
 timeModule.setVisible(true);  
 isTimeVisible = true;  
 requestFocus();  
 });  
 JRadioButton hideSimTime = new JRadioButton("Скрывать время симуляции");  
 hideSimTime.addActionListener(e -> {  
 System.*out*.println("Simulation time is hidden.");  
 timeModule.setVisible(false);  
 isTimeVisible = false;  
 requestFocus();  
 });  
 showSimTime.setBackground(control.getBackground());  
 hideSimTime.setBackground(control.getBackground());  
 timeRadioButtonPanel.add(showSimTime);  
 timeRadioButtonPanel.add(hideSimTime);  
  
 // связываем переключатели отображения времени симуляции в группу  
 ButtonGroup timeGroup = new ButtonGroup();  
 timeGroup.add(showSimTime);  
 timeGroup.add(hideSimTime);  
 timeGroup.setSelected(hideSimTime.getModel(), true); // при запуске программы выбрано "Скрывать время симуляции"  
  
 // панель периода рождения объектов  
 JPanel periodPanel = new JPanel();  
 periodPanel.setLayout(new GridLayout(3, 1));  
 periodPanel.setBackground(control.getBackground());  
 control.add(periodPanel);  
  
 JLabel periodHint = new JLabel("Периоды рождения:");  
 periodPanel.add(periodHint);  
  
 // панель для размещения текстовых полей  
 JPanel periodEditPanel = new JPanel();  
 periodEditPanel.setLayout(new BoxLayout(periodEditPanel, BoxLayout.*X\_AXIS*));  
 periodEditPanel.setBackground(control.getBackground());  
 periodPanel.add(periodEditPanel);  
  
 JPanel dronePeriodPanel = new JPanel(new GridLayout(2, 1));  
 dronePeriodPanel.setBackground(control.getBackground());  
 periodEditPanel.add(dronePeriodPanel);  
  
 JLabel droneHint = new JLabel("Трутень:");  
 dronePeriodPanel.add(droneHint);  
  
 // поля ввода пользователем периода рождения  
 JTextField periodDrone = new JTextField(Integer.*toString*(Habitat.*N1*));  
 periodDrone.addActionListener(e -> {  
 try {  
 Habitat.*N1* = Integer.*valueOf*(periodDrone.getText());  
 System.*out*.println("New period of drones: " + Habitat.*N1*);  
 if (Habitat.*N1* < 0)  
 throw new Exception();  
 } catch (Exception e1) {  
 Habitat.*N1* = 4;  
 periodDrone.setText(Integer.*toString*(Habitat.*N1*));  
 JOptionPane.*showMessageDialog*(null,  
 "Введено неверное значение. Введите корректный период рождения.",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 });  
 dronePeriodPanel.add(periodDrone);  
  
 JPanel WorkerPeriodPanel = new JPanel(new GridLayout(2, 1));  
 WorkerPeriodPanel.setBackground(control.getBackground());  
 periodEditPanel.add(WorkerPeriodPanel);  
  
 JLabel WorkerHint = new JLabel("Рабочий:");  
 WorkerPeriodPanel.add(WorkerHint);  
  
 JTextField periodWorker = new JTextField(Integer.*toString*(Habitat.*N2*));  
 periodWorker.addActionListener(e -> {  
 try {  
 Habitat.*N2* = Integer.*valueOf*(periodWorker.getText());  
 System.*out*.println("New period of Workers: " + Habitat.*N2*);  
 if (Habitat.*N2* < 0)  
 throw new Exception();  
 } catch (Exception e1) {  
 Habitat.*N2* = 3;  
 periodWorker.setText(Integer.*toString*(Habitat.*N2*));  
 JOptionPane.*showMessageDialog*(null,  
 "Введено неверное значение. Введите корректный период рождения.",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 });  
 WorkerPeriodPanel.add(periodWorker);  
  
 // панель с компонентами, связанными с вероятностью рождения  
 JPanel probabilityPanel = new JPanel(new GridLayout(4, 1));  
 probabilityPanel.setBackground(control.getBackground());  
 control.add(probabilityPanel);  
  
 // варианты в JComboBox  
 String[] items = {  
 "0",  
 "10",  
 "20",  
 "30",  
 "40",  
 "50",  
 "60",  
 "70",  
 "80",  
 "90",  
 "100"  
 };  
  
 // вероятность трутня регулируется с помощью JComboBox  
 probabilityPanel.add(new JLabel("Процент трутней:"));  
 JComboBox<String> comboBox = new JComboBox<>(items);  
 comboBox.setSelectedItem(Integer.*toString*(Habitat.*K*));  
 comboBox.addActionListener(e -> {  
 Habitat.*K* = Integer.*valueOf*(items[comboBox.getSelectedIndex()]);  
 System.*out*.println("New probability of drones: " + Habitat.*K*);  
 requestFocus();  
 });  
 probabilityPanel.add(comboBox);  
  
 // вероятность рабочих регулируется с помощью JSlider  
 probabilityPanel.add(new JLabel("Вероятность рождения рабочего:"));  
 JSlider slider = new JSlider(0, 100, Habitat.*P*);  
 slider.setBackground(control.getBackground());  
 slider.setMajorTickSpacing(10); // шаг 10 единиц  
 slider.setPaintLabels(true); // отображение числовых меток (шаг 10 ед.)  
 slider.setPaintTicks(true); // отображение засечек  
 slider.addChangeListener(e -> {  
 Habitat.*P* = slider.getValue();  
 System.*out*.println("New probability of workers: " + Habitat.*P*);  
 requestFocus();  
 });  
 probabilityPanel.add(slider);  
  
 // панель времени жизни  
 JPanel lifePanel = new JPanel();  
 lifePanel.setLayout(new GridLayout(3, 1));  
 lifePanel.setBackground(control.getBackground());  
 control.add(lifePanel);  
  
 JLabel lifeHint = new JLabel("Время жизни пчел:");  
 lifePanel.add(lifeHint);  
  
 // панель для размещения текстовых полей  
 JPanel lifeEditPanel = new JPanel();  
 lifeEditPanel.setLayout(new BoxLayout(lifeEditPanel, BoxLayout.*X\_AXIS*));  
 lifeEditPanel.setBackground(control.getBackground());  
 lifePanel.add(lifeEditPanel);  
  
 JPanel DroneLifePanel = new JPanel(new GridLayout(2, 1));  
 DroneLifePanel.setBackground(control.getBackground());  
 lifeEditPanel.add(DroneLifePanel);  
  
 JLabel droneLifeHint = new JLabel("Трутень:");  
 DroneLifePanel.add(droneLifeHint);  
  
 // поля ввода пользователем периода рождения  
 JTextField lifeDrone = new JTextField(Integer.*toString*(Habitat.*droneTimeOfLife*));  
 lifeDrone.addActionListener(e -> {  
 try {  
 Habitat.*droneTimeOfLife* = Integer.*valueOf*(lifeDrone.getText());  
 System.*out*.println("New lifetime of drones: " + Habitat.*droneTimeOfLife*);  
 if (Habitat.*droneTimeOfLife* < 0)  
 throw new Exception();  
 } catch (Exception e1) {  
 Habitat.*droneTimeOfLife* = 4;  
 lifeDrone.setText(Integer.*toString*(Habitat.*droneTimeOfLife*));  
 JOptionPane.*showMessageDialog*(null,  
 "Введено неверное значение. Введите корректное время жизни.",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 });  
 DroneLifePanel.add(lifeDrone);  
  
 JPanel WorkerLifePanel = new JPanel(new GridLayout(2, 1));  
 WorkerLifePanel.setBackground(control.getBackground());  
 lifeEditPanel.add(WorkerLifePanel);  
  
 JLabel WorkerLifeHint = new JLabel("Рабочий:");  
 WorkerLifePanel.add(WorkerLifeHint);  
  
 JTextField lifeWorker = new JTextField(Integer.*toString*(Habitat.*WorkerTimeOfLife*));  
 lifeWorker.addActionListener(e -> {  
 try {  
 Habitat.*WorkerTimeOfLife* = Integer.*valueOf*(lifeWorker.getText());  
 System.*out*.println("New lifetime of workers: " + Habitat.*WorkerTimeOfLife*);  
 if (Habitat.*WorkerTimeOfLife* < 0)  
 throw new Exception();  
 } catch (Exception e1) {  
 Habitat.*WorkerTimeOfLife* = 5;  
 lifeWorker.setText(Integer.*toString*(Habitat.*WorkerTimeOfLife*));  
 JOptionPane.*showMessageDialog*(null,  
 "Введено неверное значение. Введите корректное время жизни.",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 });  
 WorkerLifePanel.add(lifeWorker);  
  
 // кнопка текущие объекты  
 JButton currentObjectsButton = new JButton("Список объектов");  
 currentObjectsButton.setPreferredSize(buttonSize);  
 currentObjectsButton.addActionListener(e -> {  
 String currentObjectsText = "";  
 for (int i = 0; i < Habitat.*array*.size(); i++) {  
 String type;  
 int number = i + 1;  
 if (Habitat.*array*.get(i) instanceof Drone) {  
 type = "трутень";  
 } else type = "рабочий";  
 long lifetime = Habitat.*birthdays*.get(Habitat.*array*.get(i).id);  
 int id = Habitat.*array*.get(i).id;  
 currentObjectsText += number + ".\nТип: " + type + "\nВремя рождения: " +  
 lifetime + "\nИдентификатор: " + id + "\n";  
 }  
 if (Habitat.*array*.isEmpty()) currentObjectsText = "Объекты не сгенерированы.";  
 TextArea currentObjects = new TextArea(currentObjectsText);  
 currentObjects.setEditable(false);  
 JOptionPane.*showMessageDialog*(this,  
 currentObjects,  
 "Текущие объекты",  
 JOptionPane.*INFORMATION\_MESSAGE*);  
 requestFocus();  
 });  
 control.add(currentObjectsButton);  
  
  
 /\*  
 \*  
 \* МЕНЮ  
 \*  
 \* \*/  
  
 JMenuBar menuBar = new JMenuBar();  
 {  
 JMenu menuSimulation = new JMenu("Управление");  
 menuBar.add(menuSimulation);  
  
 JMenuItem startMenuItem = new JMenuItem("Старт");  
 startMenuItem.addActionListener(new ActionListener() {  
 SimulationTask updating = null;  
  
 public void actionPerformed(ActionEvent e) {  
 if (!isTimerRunning) { // без условия таймер бы ускорялся  
 if (isStatisticsVisible) {  
 statistics.setVisible(false);  
 isStatisticsVisible = false;  
 }  
 start();  
 updating = new SimulationTask(habitat, time);  
 timer.schedule(updating, 0, 1000);  
 }  
 requestFocus();  
 }  
 });  
 menuSimulation.add(startMenuItem);  
  
 JMenuItem stopMenuItem = new JMenuItem("Стоп");  
 stopMenuItem.addActionListener(new ActionListener() {  
 SimulationTask updating = null;  
  
 public void actionPerformed(ActionEvent e) {  
 if (isTimerRunning) {  
 timer.cancel();  
 isTimerRunning = false;  
 }  
 if (isShowModalDialog) {  
 int response = stop();  
 switch (response) {  
 case JOptionPane.*YES\_OPTION*:  
 System.*out*.println("JOptionPane.YES\_Option");  
 Habitat.*array*.clear(); // очистка фрейма  
 repaint(); // перерисовка очищенного фрейма  
 break;  
 case JOptionPane.*NO\_OPTION*:  
 System.*out*.println("JOptionPane.NO\_Option");  
 \_continue();  
 updating = new SimulationTask(habitat, time);  
 timer.schedule(updating, 0, 1000);  
 }  
 } else {  
 Habitat.*array*.clear(); // очистка коллекции  
 repaint();  
 }  
 requestFocus();  
 }  
 });  
 menuSimulation.add(stopMenuItem);  
  
 JMenu menuInfo = new JMenu("Информация");  
 menuBar.add(menuInfo);  
  
 JCheckBoxMenuItem statItem = new JCheckBoxMenuItem("Показывать информацию");  
 statItem.addActionListener(e -> {  
 System.*out*.println("Information is shown/hidden.");  
 isShowModalDialog = !isShowModalDialog; // меняем состояние булеаны  
 });  
 menuInfo.add(statItem);  
  
 statItem.addActionListener(e -> { // синхронизация продублированных кнопок "показать информацию"  
 if (statItem.isSelected() && !statShow.isSelected()) {  
 statShow.setSelected(true);  
 }  
 if (!statItem.isSelected() && statShow.isSelected()) {  
 statShow.setSelected(false);  
 }  
 });  
 statShow.addActionListener(e -> {  
 if (statShow.isSelected() && !statItem.isSelected()) {  
 statItem.setSelected(true);  
 }  
 if (!statShow.isSelected() && statItem.isSelected()) {  
 statItem.setSelected(false);  
 }  
 });  
  
 menuInfo.addSeparator();  
  
 JRadioButtonMenuItem showSimTimeItem = new JRadioButtonMenuItem("Показывать время симуляции", false);  
 showSimTimeItem.addActionListener(e -> {  
 System.*out*.println("Simulation time is shown.");  
 timeModule.setVisible(true);  
 isTimeVisible = true;  
 requestFocus();  
 });  
 menuInfo.add(showSimTimeItem);  
  
 JRadioButtonMenuItem hideSimTimeItem = new JRadioButtonMenuItem("Скрывать время симуляции", true);  
 hideSimTimeItem.addActionListener(e -> {  
 System.*out*.println("Simulation time is hidden.");  
 timeModule.setVisible(false);  
 isTimeVisible = false;  
 requestFocus();  
 });  
 menuInfo.add(hideSimTimeItem);  
  
 ButtonGroup timeItemGroup = new ButtonGroup();  
 timeItemGroup.add(showSimTimeItem);  
 timeItemGroup.add(hideSimTimeItem);  
  
 }  
 setJMenuBar(menuBar);  
  
 this.setVisible(true);  
 }  
}

import javax.swing.\*;  
import java.awt.\*;  
import java.util.\*;  
  
class Habitat {  
 static ArrayList<Bee> *array*; // массив объектов  
 static HashSet<Integer> *identifiers*; // массив идентификаторов объектов  
 static TreeMap<Integer, Long> *birthdays*; // массив времени рождения объектов  
 static int *numberOfDrones*; // переменные для записи статистики о родившихся животных  
 static int *numberOfWorkers*;  
 static int *N1* = 4;  
 static int *N2* = 3; // время генерации  
 static int *K* = 30;  
 static int *P* = 80; // вероятность генерации в %  
 static int *iK* = 0;  
 static int *droneTimeOfLife* = 8;  
 static int *WorkerTimeOfLife* = 6;  
 private Image droneImg = new ImageIcon("drone.png").getImage();  
 private Image workerImg = new ImageIcon("worker.png").getImage();  
 private int areaSizeX, areaSizeY; // размер области генерации  
  
 Habitat(int x, int y) {  
 areaSizeX = x;  
 areaSizeY = y;  
 *array* = new ArrayList<>();  
 *identifiers* = new HashSet<>();  
 *birthdays* = new TreeMap<>();  
 }  
  
 void update(long time) {  
 int x, y, id;  
 System.*out*.println("time: " + time);  
 for (int i = 0; i < Habitat.*array*.size(); i++) {  
 if (Habitat.*array*.get(i) instanceof Drone && (time - Habitat.*array*.get(i).timeOfBirth) >= *droneTimeOfLife*) {  
 *array*.remove(i);  
 *birthdays*.remove(i);  
 *identifiers*.remove(i);  
 --i;  
 --*iK*;  
 System.*out*.println("Drone has deleted.");  
 } else if (Habitat.*array*.get(i) instanceof Worker && (time - Habitat.*array*.get(i).timeOfBirth) >= *WorkerTimeOfLife*) {  
 *array*.remove(i);  
 *birthdays*.remove(i);  
 *identifiers*.remove(i);  
 --i;  
 System.*out*.println("Worker has deleted.");  
 }  
 }  
 if (*iK* < (Habitat.*array*.size() \* *K* / 100) && time % *N1* == 0 && time != 0) { // вероятность выполнения условия  
 x = new Random().nextInt(areaSizeX - 175);  
 y = new Random().nextInt(areaSizeY - 175);  
 id = new Random().nextInt();  
 Drone drone = new Drone(droneImg, x, y, id, time);  
 *array*.add(drone);  
 *identifiers*.add(id);  
 *birthdays*.put(id, drone.timeOfBirth);  
 System.*out*.println("drone has generated. time of birth: " + drone.timeOfBirth + ", id: " + id);  
 *numberOfDrones*++;  
 *iK*++;  
 }  
 if (new Random().nextInt(100) < *P* && time % *N2* == 0 && time != 0) { //

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

x = new Random().nextInt(areaSizeX - 150);  
 y = new Random().nextInt(areaSizeY - 150);  
 id = new Random().nextInt();  
 Worker puppy = new Worker(workerImg, x, y, id, time);  
 *array*.add(puppy);  
 *identifiers*.add(id);  
 *birthdays*.put(id, puppy.timeOfBirth);  
 System.*out*.println("Worker has generated. time of birth: " + puppy.timeOfBirth + ", id: " + id);  
 *numberOfWorkers*++;  
 }  
 }  
}

import javax.swing.\*;  
import java.awt.\*;  
  
class VPanel extends JPanel {  
 VPanel(int x, int y){  
 setPreferredSize(new Dimension(x, y));  
 setLayout(new FlowLayout(FlowLayout.*CENTER*));  
 }  
 @Override  
 protected void paintComponent(Graphics g){  
 super.paintComponent(g);  
 for (int i = 0; i < Habitat.*array*.size(); i++) {  
 g.drawImage(Habitat.*array*.get(i).img, Habitat.*array*.get(i).getX(),  
 Habitat.*array*.get(i).getY(), null);  
 }  
 }  
}

import java.awt.\*;  
  
abstract class Bee implements IBehaviour{  
 int x, y;  
 Image img;  
 int id;  
 long timeOfBirth;  
 public void move(){}  
 public void setX(int x){ this.x = x;}  
 public void setY(int y){ this.y = y;}  
 @Override  
 public int getX(){ return x; }  
 @Override  
 public int getY(){ return y; }  
}

import java.awt.\*;  
  
class Drone extends Bee{  
 Drone(Image img, int x, int y, int id, long timeOfBirth){  
 this.img = img;  
 this.x = x;  
 this.y = y;  
 this.id = id;  
 this.timeOfBirth = timeOfBirth;  
 }  
}

import java.awt.\*;  
  
class Worker extends Bee{  
 Worker(Image img, int x, int y, int id, long timeOfBirth){  
 this.img = img;  
 this.x = x;  
 this.y = y;  
 this.id = id;  
 this.timeOfBirth = timeOfBirth;  
 }  
}