МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники

Отчет по лабораторной работе №1-2

по дисциплине «Технология программирования»

Тема: «Основы программирования на Java.

Обработка событий. Механизм делегирования событий. Разработка графического интерфейса приложения.»

Вариант №2

Выполнил:

студент группы АВТ-716

Тесленко Н.Б.

Проверил:

Михайленко Дмитрий Анатольевич

Новосибирск, 2020 г.

#### Практические задания

Лабораторная работа №1:

1. Познакомиться с особенностями технологии Java и изучить синтаксис языка Java.
2. Изучить основные понятия и термины обработки событий в модели делегирования событий на Java (учебно-методическое пособие, главы 1 и 2).
3. Разработать учебную программу. Основная ее задача – разработка упрощенной имитации поведения объектов (все последующие лабораторные работы будут расширять это задание). Объекты реализуются через наследование: абстрактный класс + интерфейс → наследники.

Рабочий цикл программы:

* запускается процесс симуляции по клавише, генерируются объекты классов согласно заданию;
* симуляция завершается по другой клавише, выводится статистическая информация.

1. Для решения задачи:

* Разработать абстрактный класс объекта, согласно варианту индивидуального задания.
* Создать интерфейс IBehaviour, задающий поведение объекта (методы: move(), getx(), gety(), sety(), setx() и другие. Далее будут реализоваться алгоритмы движения объектов в окне программы).
* Реализовать иерархию классов, определяющих объекты по варианту и реализующие интерфейс IBehaviour.
* Создать класс Habitat (среда), определяющий размер рабочей области и хранящий массив объектов, с параметрами, заданными вариантом. Предусмотреть в классе метод Update, вызывающийся по таймеру и получающий на вход время, прошедшее от начала симуляции. В данном методе должны генерироваться новые объекты и помещаться в поле визуализации в случайном месте. Визуализация объекта - использовать готовые небольшие картинки;

Рабочее окно программы – область визуализации среды обитания объектов;

1. Симуляция должна запускаться по клавише **B** и останавливаться по клавише **E**. При остановке симуляции список очищается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише **T**;
2. По завершению симуляции область отображения объектов очищается и в поле визуализации выводится информация о количестве и типе сгенерированных объектов, а также время симуляции. Текст должен быть форматирован, т.е. выводиться с использованием разных шрифтов и цветов.

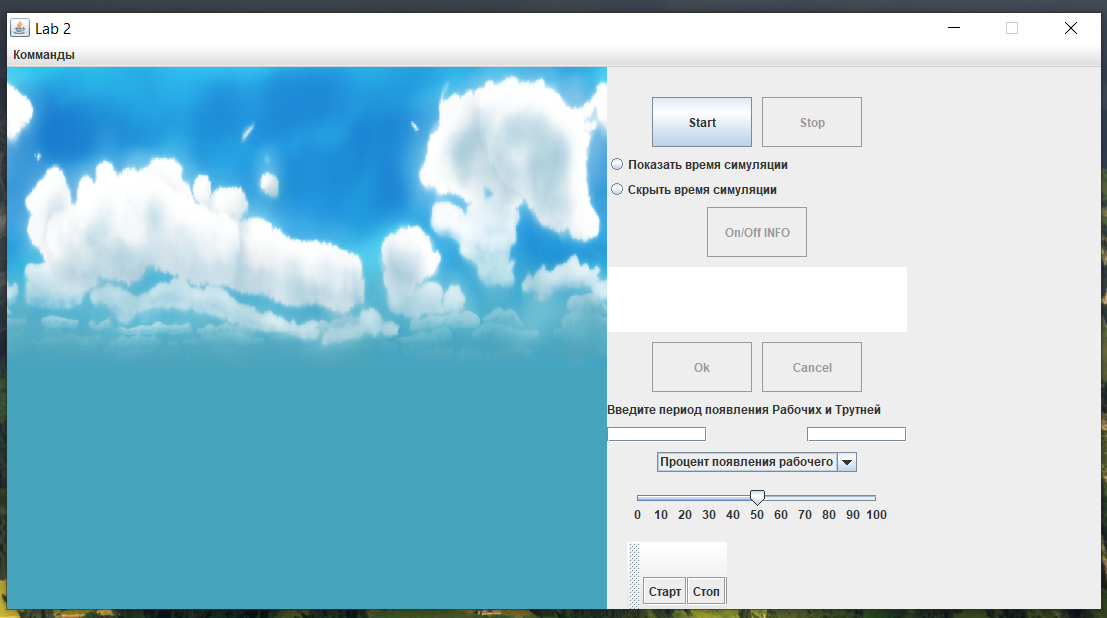
Параметры симуляции задаются в классе Habitat.

Лабораторная работа №2:

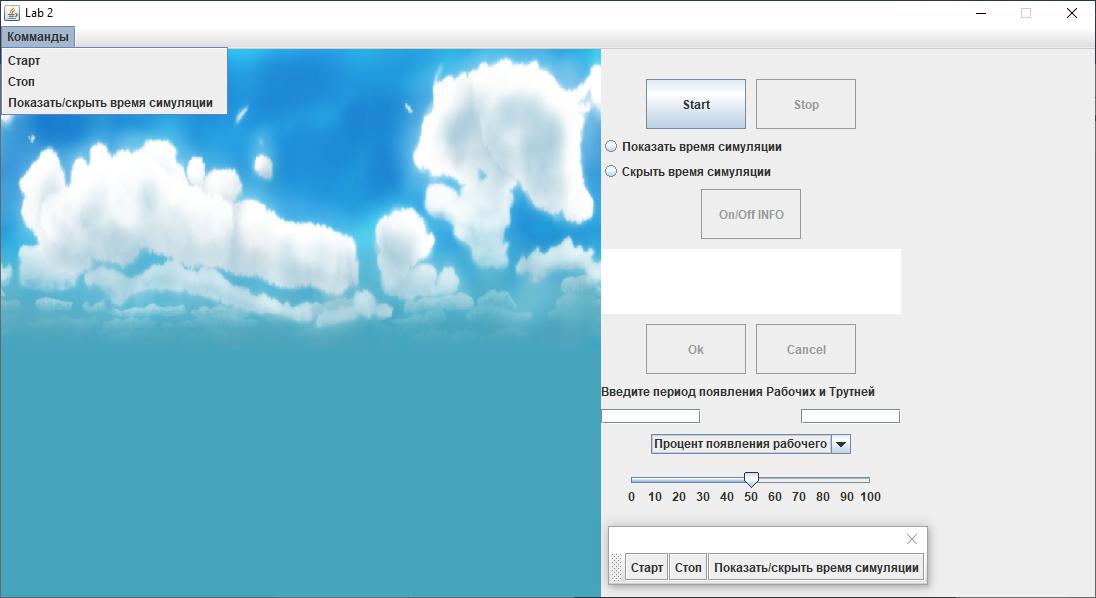
1. Познакомиться с основными графическими библиотеками Java - AWT и Swing и их основными компонентами. Изучить классы менеджеров компоновки, классы создания меню приложения, основных и диалоговых окон и т.д.
2. Для хранения генерируемых объектов использовать динамический массив объектов.
3. Доработать 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 меню и панель инструментов, в которых продублировать основные команды 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 полностью 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 - JComboBox и JSlider  (шаг значений 10);
   * Дополнить интерфейс поясняющими метками.

**Результаты работы программы**

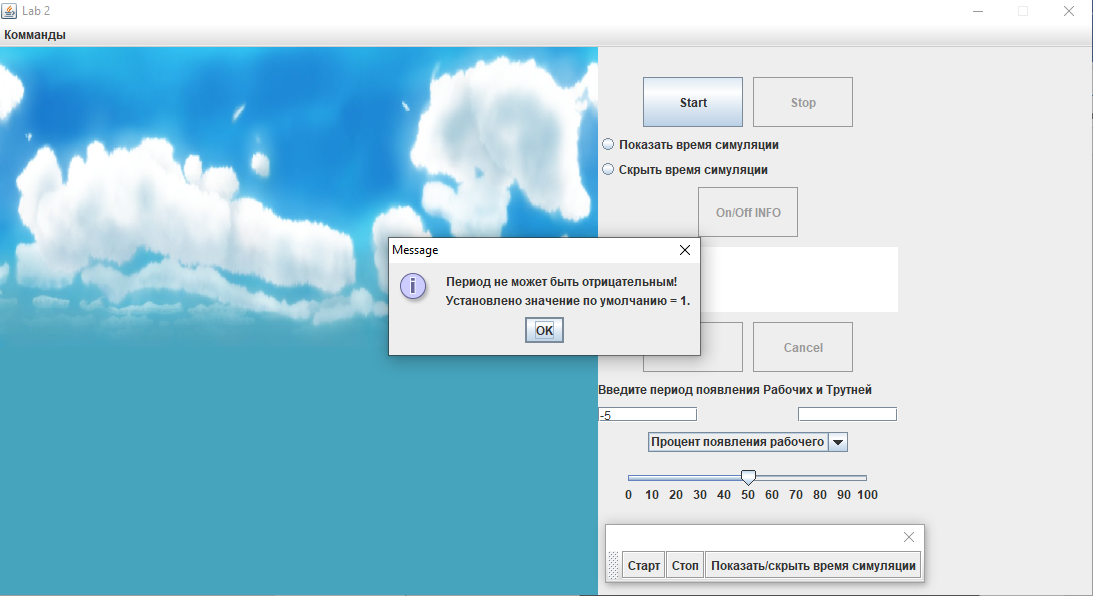
При запуске программы пользователь видит следующее окно:



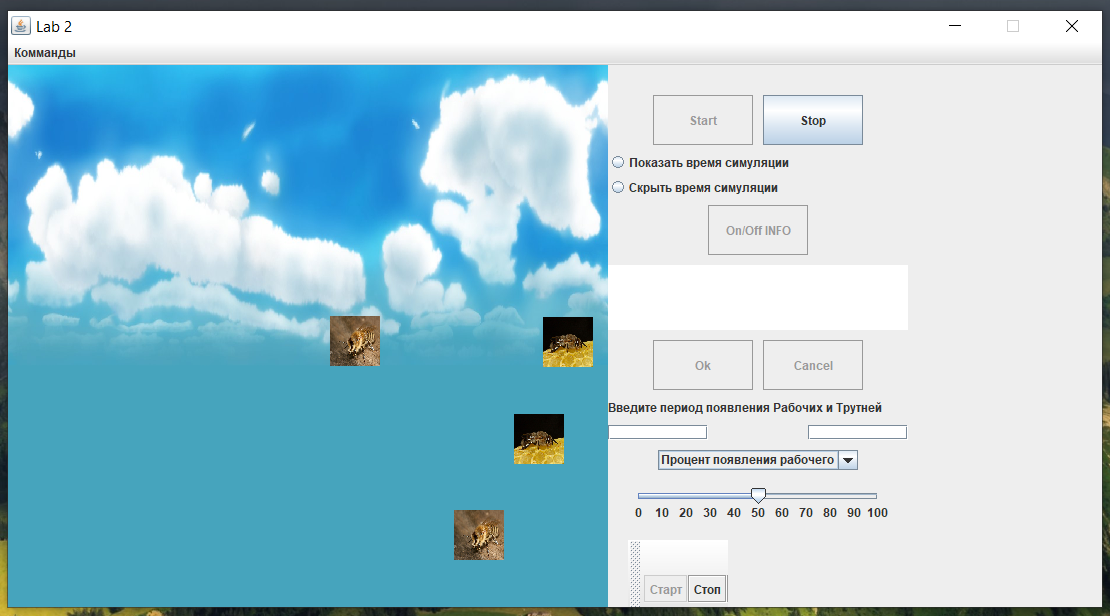
Есть возможность управлять функционалом при помощи вложенного меню в основном меню с элементами: Старт, Стоп, Показать/скрыть время. Кнопки в панели и меню сообщаются между собой.



Если ввести в TextField недопустимые значения, то пользователь увидит модальное окно, а сами значения будут изменены на значения по умолчанию.



После окончания моделирования пользователь видит статистику по выполнению программы. Если нажать «Отмена», то симуляция продолжится, иначе - закончится.



**Листинг программы**

***Main.java***

import javax.swing.\*;  
import java.awt.event.\*;  
  
  
public class Main {  
 /\*Panels\*/  
 static public JFrame *window* = new JFrame("Lab 2");  
 static public JPanel *bees* = new JPanel();  
 static public JPanel *panel* = new JPanel();  
 /\*Habitat\*/  
 static Habitat *habitat*;  
 /\*Time\*/  
 static private long *timePrev* = 0, *startTime*, *timeStop* = 0, *timeContinue* = 0;  
 static private int *time*;  
 /\*Window's size\*/  
 static public int *windowWIDTH* = 1100, *HEIGHT* = 600, *beesWIDTH* = 600, *panelWIDTH* = 500;  
 /\*Boolean variables\*/  
 static boolean *bText* = false;  
 static boolean *bSimulation* = false;  
 static boolean *slWorker* = true;  
 /\*Text fields\*/  
 static JLabel *textTimeSimulation* = new JLabel(""), *textPeriod* = new JLabel("Введите период появления " + "Рабочих и Трутней");  
 static JTextArea *textResults* = new JTextArea("");  
 static JTextField *intWorker* = new JTextField(""), *intDrone* = new JTextField("");  
 /\*Buttons\*/  
 static JButton *btStart* = new JButton("Start"), *btStop* = new JButton("Stop"),  
 *btOk* = new JButton("Ok"), *btCancel* = new JButton("Cancel");  
 static JToggleButton *btInfo* = new JToggleButton("On/Off INFO");  
 static JRadioButton *timeOn* = new JRadioButton("Показать время симуляции"),  
 *timeOff* = new JRadioButton("Скрыть время симуляции");  
 /\*Sliders\*/  
 static String *arrSliders*[] = {"Процент появления рабочего", "'Процент' появления трутня"};  
 static JComboBox<String> *comboBoxSliders* = new JComboBox<String>(*arrSliders*);  
 static JSlider *slBees* = new JSlider(0, 100, 50);  
 /\*Menu\*/  
 static JMenuBar *jMenuBar* = new JMenuBar();  
 static JMenu *jmCommands* = new JMenu("Комманды");  
 static JMenuItem *jmiStart* = new JMenuItem("Старт"), *jmiStop* = new JMenuItem("Стоп"), *jmiTime* = new JMenuItem("Показать/скрыть время симуляции");  
 static JToolBar *jToolBar* = new JToolBar();  
 static JButton *jtbStart* = new JButton("Старт"), *jtbStop* = new JButton("Стоп"), *jtbTime* = new JButton("Показать/скрыть время симуляции");  
  
  
 public static void main(String[] args){  
 *init*();  
 *event\_handling*();  
  
 *habitat* = new Habitat();  
  
 while(true){  
 *print\_time\_simulation*();  
 if(*bSimulation*) {  
 long timeCur = System.*currentTimeMillis*() - *startTime*;  
 long step = timeCur - *timePrev*;  
 if (step >= 1000) {  
 *time* = (int) (timeCur / 1000);  
 *habitat*.update(*time*);  
 *timePrev* = timeCur;  
 }  
 }  
 }  
 }  
 static void change\_T(){  
 *bText* = !*bText*;  
 }  
 static void change\_B(){  
 *startTime* = System.*currentTimeMillis*();  
 *bSimulation* = true;  
 *btStart*.setEnabled(false);  
 *btStop*.setEnabled(true);  
 *jmiStart*.setEnabled(false);  
 *jmiStop*.setEnabled(true);  
 *jtbStart*.setEnabled(false);  
 *jtbStop*.setEnabled(true);  
 *btOk*.setEnabled(false);  
 *btCancel*.setEnabled(false);  
 *btInfo*.setEnabled(false);  
 Worker.*beeWorker* = Drone.*beeDrone* = 0;  
 *habitat*.clear();  
 *timePrev* = 0;  
 }  
 static void change\_E(){  
 *bSimulation* = false;  
 *timeStop* = System.*currentTimeMillis*();  
 *btInfo*.setEnabled(true);  
 }  
 static void print\_time\_simulation(){  
 if(*bText*){  
 *textTimeSimulation*.setText(*time*+" секунд после начала симуляции");  
 }else {  
 *textTimeSimulation*.setText("");  
 }  
 *panel*.repaint();  
 }  
 static void print\_results(boolean TF){  
 if(TF) {  
 *textResults*.setText("Количество сгенерированных пчёл-рабочих: " + Worker.*beeWorker* +  
 "\nКоличество сгенерированных пчёл-трутней: " + Drone.*beeDrone* +  
 "\nОбщее количество сгенерированных пчёл: " + (Drone.*beeDrone* + Worker.*beeWorker*) +  
 "\nВремя симуляции: " + *time*);  
 }else{  
 *textResults*.setText("");  
 }  
 }  
 static void event\_handling(){  
 *window*.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyTyped(KeyEvent e) {  
 if(String.*valueOf*(e.getKeyChar()).equals("t") || String.*valueOf*(e.getKeyChar()).equals("T")){  
 *change\_T*();  
 }  
 if(String.*valueOf*(e.getKeyChar()).equals("b") || String.*valueOf*(e.getKeyChar()).equals("B")){  
 *change\_B*();  
 }  
 if(String.*valueOf*(e.getKeyChar()).equals("e") || String.*valueOf*(e.getKeyChar()).equals("E")){  
 *change\_E*();  
 }  
 }  
 });  
 *btStart*.addActionListener(e -> {  
 *change\_B*();  
 *window*.requestFocus();  
 });  
 *btStop*.addActionListener(e -> {  
 *change\_E*();  
 *window*.requestFocus();  
 });  
 *btInfo*.addActionListener(e -> {  
 *window*.requestFocus();  
 if(!*bSimulation*) {  
 if (*btInfo*.isSelected()) {  
 *print\_results*(true);  
 *btOk*.setEnabled(true);  
 *btCancel*.setEnabled(true);  
 } else {  
 *print\_results*(false);  
 *btOk*.setEnabled(false);  
 *btCancel*.setEnabled(false);  
 }  
 }  
 });  
 *timeOn*.addActionListener(e -> {  
 *bText* = true;  
 *window*.requestFocus();  
 });  
 *timeOff*.addActionListener(e -> {  
 *bText* = false;  
 *window*.requestFocus();  
 });  
 *btOk*.addActionListener(e -> {  
 *window*.requestFocus();  
 *bSimulation* = false;  
 *btStart*.setEnabled(true);  
 *btStop*.setEnabled(false);  
 *jmiStart*.setEnabled(true);  
 *jmiStop*.setEnabled(false);  
 *jtbStart*.setEnabled(true);  
 *jtbStop*.setEnabled(false);  
 *btOk*.setEnabled(false);  
 *btCancel*.setEnabled(false);  
 *btInfo*.setEnabled(false);  
 *btInfo*.setSelected(false);  
 *timeStop* = *timeContinue* = 0;  
 *habitat*.clear();  
 *print\_results*(false);  
 *time* = 0;  
 });  
 *btCancel*.addActionListener(e -> {  
 *window*.requestFocus();  
 *timeContinue* = System.*currentTimeMillis*();  
 *startTime* += *timeContinue* - *timeStop*;  
 *bSimulation* = true;  
 *btStart*.setEnabled(false);  
 *btStop*.setEnabled(true);  
 *jmiStart*.setEnabled(false);  
 *jmiStop*.setEnabled(true);  
 *jtbStart*.setEnabled(false);  
 *jtbStop*.setEnabled(true);  
 *btOk*.setEnabled(false);  
 *btCancel*.setEnabled(false);  
 *btInfo*.setEnabled(false);  
 *btInfo*.setSelected(false);  
 *print\_results*(false);  
 });  
 *intWorker*.addActionListener(e -> {  
 *window*.requestFocus();  
 try {  
 int tmp = Integer.*parseInt*(*intWorker*.getText());  
 if(tmp > 0){  
 Habitat.*N2* = tmp;  
 JOptionPane.*showMessageDialog*(*window*, "Установлен период появления\n" +  
 "рабочих, равный " + Habitat.*N2* + " секунд.");  
 }else{  
 throw new Exception();  
  
 }  
 }catch (Exception e1){  
 Habitat.*N2* = 1;  
 JOptionPane.*showMessageDialog*(*window*, "Период не может быть отрицательным!" +  
 "\nУстановлено значение по умолчанию = 1.");  
 }  
  
 });  
 *intDrone*.addActionListener(e -> {  
 *window*.requestFocus();  
 try {  
 int tmp = Integer.*parseInt*(*intDrone*.getText());  
 if (tmp > 0) {  
 Habitat.*N1* = tmp;  
 JOptionPane.*showMessageDialog*(*window*, "Установлен период появления\n" +  
 "трутней, равный " + Habitat.*N1* + " секунд.");  
 } else {  
 throw new Exception();  
 }  
 }catch (Exception e1){  
 Habitat.*N1* = 1;  
 JOptionPane.*showMessageDialog*(*window*, "Период не может быть отрицательным!" +  
 "\nУстановлено значение по умолчанию = 1.");  
 }  
 });  
 *comboBoxSliders*.addActionListener(e -> {  
 *window*.requestFocus();  
 String tmp = (String)*comboBoxSliders*.getSelectedItem();  
 if(tmp.equals("Процент появления рабочего")){  
 *slWorker* = true;  
 }else{  
 *slWorker* = false;  
 }  
 });  
 *slBees*.addChangeListener(e -> {  
 *window*.requestFocus();  
 if(*slWorker*){  
 Habitat.*P* = *slBees*.getValue();  
 }else{  
 Habitat.*K* = *slBees*.getValue();  
 }  
 });  
 *jmiStart*.addActionListener(e -> {  
 *change\_B*();  
 *window*.requestFocus();  
 });  
 *jmiStop*.addActionListener(e -> {  
 *change\_E*();  
 *window*.requestFocus();  
 });  
 *jmiTime*.addActionListener(e -> {  
 *change\_T*();  
 *window*.requestFocus();  
 });  
 *jtbStart*.addActionListener(e -> {  
 *change\_B*();  
 *window*.requestFocus();  
 });  
 *jtbStop*.addActionListener(e -> {  
 *change\_E*();  
 *window*.requestFocus();  
 });  
 *jtbTime*.addActionListener(e -> {  
 *change\_T*();  
 *window*.requestFocus();  
 });  
 }  
 static void init(){  
 *window*.setLayout(null);  
 *window*.setSize(*windowWIDTH*, *HEIGHT*);  
 *window*.setDefaultCloseOperation(*window*.*EXIT\_ON\_CLOSE*);  
 *window*.setVisible(true);  
 *window*.requestFocus();  
  
 *bees*.setLayout(null);  
 *bees*.setBounds(0,0,*beesWIDTH*, *HEIGHT*);  
  
 *panel*.setLayout(null);  
 *panel*.setBounds(*beesWIDTH*,0,*panelWIDTH*,*HEIGHT*);  
 *textTimeSimulation*.setBounds(0,0, 300, 20);  
 *panel*.add(*textTimeSimulation*);  
 *btStart*.setBounds(45, 30, 100, 50);  
 *panel*.add(*btStart*);  
 *btStop*.setBounds(155, 30, 100, 50);  
 *panel*.add(*btStop*);  
 *timeOn*.setBounds(0, 90, 300, 15);  
 *panel*.add(*timeOn*);  
 *timeOff*.setBounds(0, 115, 300, 15);  
 *panel*.add(*timeOff*);  
 ButtonGroup buttonGroup = new ButtonGroup();  
 buttonGroup.add(*timeOn*);  
 buttonGroup.add(*timeOff*);  
 *btInfo*.setBounds(100, 140, 100, 50);  
 *panel*.add(*btInfo*);  
 *textResults*.setBounds(0, 200, 300, 65);  
 *panel*.add(*textResults*);  
 *btOk*.setBounds(45, 275, 100, 50);  
 *panel*.add(*btOk*);  
 *btCancel*.setBounds(155, 275, 100, 50);  
 *panel*.add(*btCancel*);  
 *textPeriod*.setBounds(0, 335, 300, 15);  
 *panel*.add(*textPeriod*);  
 *intWorker*.setBounds(0, 360, 100, 15);  
 *panel*.add(*intWorker*);  
 *intDrone*.setBounds(200, 360, 100, 15);  
 *panel*.add(*intDrone*);  
 *comboBoxSliders*.setBounds(50, 385, 200, 20);  
 *panel*.add(*comboBoxSliders*);  
 *slBees*.setBounds(20, 415, 260, 50);  
 *slBees*.setMajorTickSpacing(10);  
 *slBees*.setPaintLabels(true);  
 *slBees*.setSnapToTicks(true);  
 *panel*.add(*slBees*);  
  
 *jmCommands*.add(*jmiStart*);  
 *jmCommands*.add(*jmiStop*);  
 *jmCommands*.add(*jmiTime*);  
 *jMenuBar*.add(*jmCommands*);  
  
 *jToolBar*.add(*jtbStart*);  
 *jToolBar*.add(*jtbStop*);  
 *jToolBar*.add(*jtbTime*);  
 *jToolBar*.setBounds(20, 475, 100, 100);  
 *panel*.add(*jToolBar*);  
  
 *textResults*.setEditable(false);  
 *btStop*.setEnabled(false);  
 *btInfo*.setEnabled(false);  
 *btOk*.setEnabled(false);  
 *btCancel*.setEnabled(false);  
  
 *window*.add(*bees*);  
 *window*.add(*panel*);  
 *window*.setJMenuBar(*jMenuBar*);  
 *window*.setResizable(false);  
 }  
}

***Habitat.java***

import javax.swing.\*;  
import java.awt.\*;  
  
public class Habitat {  
  
 private int WIDTH = Main.*beesWIDTH*, HEIGHT = Main.*HEIGHT*;  
 static public int *N1* = 3, *N2* = 2, *K* = 40, *P* = 80;  
 private int imageSize = 50;  
 private ImageIcon imageIcon = new ImageIcon(new ImageIcon(getClass().getResource(  
 "/Images/sky2.png")).getImage().getScaledInstance(WIDTH,HEIGHT, Image.*SCALE\_DEFAULT*));  
  
 Habitat(){  
 JLabel tmp = new JLabel();  
 tmp.setBounds( 0, 0, imageIcon.getIconWidth(), imageIcon.getIconHeight());  
 tmp.setIcon(imageIcon);  
 Main.*bees*.add(tmp, 0, 0);  
 Main.*bees*.repaint();  
 }  
 void update(int time){  
 if(time % *N2* == 0)  
 {  
 int random = (int)Math.*floor*(Math.*random*()\*100);  
 if(random <= *P*) {  
 int x = (int) Math.*floor*(Math.*random*() \* (WIDTH - imageSize));  
 int y = (int) Math.*floor*(Math.*random*() \* (HEIGHT - imageSize));  
 Worker worker = new Worker(x, y);  
 Worker.*beeWorker*++;  
 }  
 }  
 if(time % *N1* == 0 && Worker.*beeWorker* != 0){  
 int dronePercent = 100\*Drone.*beeDrone*/(Drone.*beeDrone* + Worker.*beeWorker*);  
 if(dronePercent < *K*){  
 int x = (int)Math.*floor*(Math.*random*()\*(WIDTH-imageSize));  
 int y = (int)Math.*floor*(Math.*random*()\*(HEIGHT-imageSize));  
 Drone drone = new Drone(x, y);  
 Drone.*beeDrone*++;  
 }  
 }  
 }  
 void clear(){  
 Main.*bees*.removeAll();  
 new Habitat();  
 }  
}

***Worker.java***

import javax.swing.\*;  
import java.awt.\*;  
  
public class Worker extends Character{  
  
 static public int *beeWorker* = 0;  
 ImageIcon imageIcon = new ImageIcon(new ImageIcon(getClass().getResource(  
 "/Images/Worker.png")).getImage().getScaledInstance(50,50,Image.*SCALE\_DEFAULT*));  
  
 public Worker(int x, int y){  
 jlb = new JLabel();  
 jlb.setBounds(x, y, imageIcon.getIconWidth(), imageIcon.getIconHeight());  
 jlb.setIcon(imageIcon);  
 Main.*bees*.add(jlb,0,0);  
 Main.*bees*.repaint();  
 setx(x);  
 sety(y);  
 }  
}

***Drone.java***

import javax.swing.\*;  
import java.awt.\*;  
  
public class Drone extends Character {  
  
 static public int *beeDrone* = 0;  
 ImageIcon imageIcon = new ImageIcon(new ImageIcon(getClass().getResource(  
 "/Images/Drone.png")).getImage().getScaledInstance(50,50,Image.*SCALE\_DEFAULT*));  
  
 public Drone(int x, int y){  
 jlb = new JLabel();  
 jlb.setBounds(x, y, imageIcon.getIconWidth(), imageIcon.getIconHeight());  
 jlb.setIcon(imageIcon);  
 Main.*bees*.add(jlb,0,0);  
 Main.*bees*.repaint();  
 setx(x);  
 sety(y);  
 }  
}

***Character.java***

import javax.swing.\*;  
  
public abstract class Character implements iBehaviour {  
  
 int x, y;  
 JLabel jlb;  
  
 @Override  
 public void movexy(int x, int y) {}  
 @Override  
 public void setx(int x) {this.x = x;}  
 @Override  
 public void sety(int y) {this.y = y;}  
 @Override  
 public int getx() {return this.x;}  
 @Override  
 public int gety() {return this.y;}  
}

***iBehaviour.java***

public interface iBehaviour {  
 void movexy(int x, int y);  
 void setx(int x);  
 void sety(int y);  
 int getx();  
 int gety();  
}