МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Лабораторная работа №1 по дисциплине «Технология программирования»*

*Вариант 2*

Группа: АВТ-610

Студенты: Князев Р.И., Дунаев Н.Ю.

Преподаватель: Васюткина И. А.

Новосибирск 2017

**Темы:**

*Основы программирования на Java. Обработка событий. Механизм делегирования событий. Порождающие паттерны проектирования классов/объектов.*

**Задание:**

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.
3. Параметры симуляции задаются в классе Habitat.

***Вариант 2***

Объект – пчела. Бывают 2 видов: трутень и рабочий. Трутни рождаются каждые N1 секунд, если их количество менее K% от общего числа пчел, в противном случае – не рождаются вовсе. Рабочие рождаются каждые N2 секунд с вероятностью P.

***Описание структуры классов:***

Для реализации программы были созданы следующие классы:

* IBehaviour
* Bee
* BeeWorker
* BeeTruten
* Habitat
* Window
* Main

**IBehaviour** *–* интерфейс, содержащий следующие методы:

**void** setX(**int** n) – Принимает на вход координату Х – расположение пчелы на оси абсцисс.  
**void** setY(**int** n) - Принимает на вход координату Y – расположение пчелы на оси ординат.  
**int** getX() – возвращает координату Х  
**int** getY() – возвращает координату Y  
**void** move();

**Bee** – абстрактный класс, наследуемый от интерфейса IBehaviour, содержащий следующие методы:

**public** Bee() – конструктор без параметров

**public** Bee(**int** x,**int** y) – конструктор с параметрами – координатами

**public void** setX(**int** n)

**public void** setY(**int** n)

**public int** getX()

**public int** getY()

**BeeWorker** – класс – наследник от Bee, предназначен для создания объекта – рабочей пчелы, имеющий конструктор с параметрами, вызывающий конструктор базового класса через super(x,y).

**public** BeeWorker(**int** x,**int** y)

**BeeTruten** - класс – наследник от Bee, предназначен для создания объекта – ленивой пчелы, имеющий конструктор с параметрами, вызывающий конструктор базового класса через super(x,y) и кол-во пчел-трутней.

**public** BeeTruten(**int** x,**int** y)

**Habitat** – класс, реализующий появление пчел, содержащий таймер, параметры генерации пчел (время и вероятность), а так же класс Updater – наследник класса TimerTask, запускающий секундомер, содержащий следующие методы:

**public** Habitat(Window myWind)

**public void** update(**double** elapsed)

**Window** – класс-наследник класса JFrame. Отвечает за генерацию окна и отрисовку в нем пчел, а также вывода статистики и времени симуляции, содержащий следующие методы:

Window()

**public void** setMas(Ant[] mas)

**public void** setTextTime(**double** sec)

**public void** changeTimeState()

**public void** paint(Graphics gr)

**private void** paintStatictics(Graphics gr)

**private void** paintSimulation(Graphics gr)

**public void** setSimulationState(**boolean** st)

**Main** – главный класс программы, содержащий обязательный метод main.

**public static void** main(String[] args)

***Описание функционала программы:***

За генерацию объектов отвечает класс **Habitat**. Внутри него реализован класс **Updater**, наследуемый от **TimerTask**. Метод *run* создает таймер, вызывая в конце метод *update*, который проверяет, выполняются ли все условия для создания объекта. Если условия выполнены, в массив объектов добавляем объект пчелы-рабочего, или пчелы-трутня.

**public** Habitat(Window myWind) {  
 **habit** = **this**;  
 **wind** = myWind;  
 **wind**.setTextTime(0);  
 **gr** = **wind**.getGraphics();  
 **tim**.schedule(**new** Updater(**habit**), 1000, 100);  
 **wind**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyPressed(KeyEvent e) {

// Старт при нажимании на клавишу «B»  
 **if** (e.getKeyCode() == KeyCode.***B***.getCode())  
 {// если симуляция не запущена  
 **if** (!**isStarted**) {  
 *// выделяем память и обнуляем счетчики пчел* **mas** = **new** Bee[1000];  
 Bee.*n* = 0;  
 BeeTruten.*trNumber* = 0;  
 **wind**.setMas(**mas**);

//флаг на то, что симуляция запущена и состояние симуляции true   
 **isStarted** = **true**;  
 **wind**.setSimulationState(**isStarted**);  
 }  
 }

//конец, если нажали «E»  
 **if** (e.getKeyCode() == KeyCode.***E***.getCode())  
 {

//если симуляция запущена  
 **if** (**isStarted**) {  
 *//флаг на то, что симуляция остановлена* **isStarted** = **false**;  
  
 **wind**.setSimulationState(**isStarted**);  
 **wind**.paint(**gr**);  
  
 **firstRun** = **true**;  
 **time** = 0;  
 **secForTrut** = 0;  
 **secForWorker** = 0;  
 }  
 }

//показать время, если нажали на «T»  
 **if** (e.getKeyCode() == KeyCode.***T***.getCode())  
 {  
 **wind**.changeTimeState();  
 }  
 }  
 });  
  
 }  
  
 **private class** Updater **extends** TimerTask  
 {  
 Habitat **obj**;  
 **public** Updater(Habitat hab)  
 {  
 **obj** = hab;  
 }  
  
 @Override

//запуск таймера  
 **public void** run() {  
 **if** (**firstRun**) //если первый запуск таймера  
 {  
 **startTime** = System.*currentTimeMillis*();  
 **lastTime** = **startTime**;  
 **firstRun** = **false**;  
 }

// узнаем текущее время, чтобы отнять от него время с последнего обновления и

//узнать, сколько прошло времени с последнего обновления(elapsed)  
 **long** curTime = System.*currentTimeMillis*();  
 **double** elapsed = (curTime - **lastTime**) / 1000.0;  
 **obj**.update(elapsed);  
 **lastTime** = System.*currentTimeMillis*();  
 }  
 }  
//проверка условий создания объектов  
 **public void** update(**double** elapsed)  
 {  
 **if** (**isStarted**)  
 {  
 **secForTrut** += elapsed;  
 **secForWorker** += elapsed;  
 **time** += elapsed;

//согласно постановке задачи, если время для появления пчелы пришло, и пчел-трутней меньше в заданное кол-во процентов  
 **if** (**secForTrut** > **timeSpawnTruten**) {  
 **if** ((**float**) BeeTruten.*trNumber* / Bee.*n* < **trutenPersent**) {

//в массив mas добавляем объект пчелы ( поля объекта – координаты отрисовки)  
 **mas**[Bee.*n*] = **new** BeeTruten((**int**)(Math.*random*() \* 720), 40 + (**int**)(Math.*random*() \* 380));  
 }// обнуляем время для трутней  
 **secForTrut** -= **timeSpawnTruten**;  
 }

//если нужное кол-во времени прошло, с вероятностью создаем пчелу  
 **if** (**secForWorker** > **timeSpawnWorker**) {  
 **if** (Math.*random*() >= (1 - **workerCreateChance**)) {  
 **mas**[Bee.*n*] = **new** BeeWorker((**int**) (Math.*random*() \* 720),40 + (**int**)(Math.*random*() \* 380));  
 }  
 **secForWorker** -= **timeSpawnWorker**;  
 }  
 **wind**.setTextTime(**time**);  
 }  
 **wind**.paint(**gr**);  
 }  
}

Класс **Window**

**public class** Window **extends** JFrame {  
 Bee[] **mas**;  
 Image **worker**;  
 Image **trut**;  
 String **time** = **""**;  
 **boolean timeState** = **true**;  
 **boolean simulationState** = **true**;  
  
 Window()  
 {

// Параметры окна  
 setBounds(200,100,800,500);  
 **this**.setResizable(**false**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 *setDefaultLookAndFeelDecorated*(**true**);

// загружаем изображения пчел из папки с проектом  
 **worker** = **new** ImageIcon(**"Roman.png"**).getImage();  
 **trut** = **new** ImageIcon(**"Dun.png"**).getImage();  
 }  
  
 **public void** setMas(Bee[] mas)  
 {  
 **this**.**mas** = mas;  
 }  
  
 **public void** setTextTime(**double** sec)//установка строки со временем  
 {  
 DecimalFormat format = **new** DecimalFormat(**"#0.00"**);  
 **time** = format.format(sec);  
 }  
  
 **public void** changeTimeState()  
 {  
 **timeState** = !**timeState**;  
 }  
  
 @Override  
 **public void** paint(Graphics gr)//отрисовка кадра  
 {   
 **super**.paint(gr);

//Если состояние симуляции пчел, отрисовываем пчел  
 **if** (**simulationState**)  
 paintSimulation(gr);  
 **else** {

//иначе отрисовываем статистику  
 paintStatictics(gr);  
 }  
 }  
  
 **private void** paintStatictics(Graphics gr)//отрисовка статистики  
 {  
 gr.setFont(**new** Font(**"Verdana"**,Font.***BOLD***,12));  
 gr.drawString(**"Всего пчел: "** + Bee.*n*,350,160);  
 gr.setFont(**new** Font(**"Times New Roman"**,Font.***PLAIN***,14));  
 gr.setColor(Color.***RED***);  
 gr.drawString(**"Рабочих: "** +(Bee.*n* - BeeTruten.*trNumber*),350,180);  
 gr.setFont(**new** Font(**"Verdana"**,Font.***BOLD***,12));  
 gr.drawString(**"Трутней: "** + BeeTruten.*trNumber*, 350,200);  
 gr.setColor(Color.***GREEN***);  
 gr.drawString(**"Время симуляции "** + **time**,350,220 );  
 gr.setColor(Color.***BLACK***);  
  
 }  
  
 **private void** paintSimulation(Graphics gr)  
 {  
 gr.setFont(**new** Font(**"Arial"**,Font.***ITALIC***,13));

//Проверяем массив объектов-пчел  
 **for**(**int** i =0; i < Bee.*n* ; i++)  
 {  
//Если объект принадлежит классу Пчел-рабочих, рисуем пчел-рабочих, получая координаты из полей соответствующего класса  
 **if** (**mas**[i] **instanceof** BeeWorker)  
 {  
 gr.drawImage(**worker**,**mas**[i].getX(),**mas**[i].getY(),**this**);  
  
 }  
 **else** {

//иначе аналогично рисуем пчел-трутней  
 gr.drawImage(**trut**, **mas**[i].getX(), **mas**[i].getY(), **this**);  
 }  
 }  
 **if** (**timeState**)  
 gr.drawString(**"Прошло времени: "** + **time** ,50,90);  
 }  
  
 **public void** setSimulationState(**boolean** st)  
 {  
 **simulationState** = st;  
 }  
}

**Вывод:**

В ходе выполнения лабораторной работы мы познакомились с особенностями технологии Java и изучили синтаксис языка. Разработали программу, симулирующую появление пчел с заданными условиями, при выполнении которой изучили такие понятия, как интерфейс, класс. Познакомились с библиотеками Timer, KeyAdapter, EventObject.