МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НГТУ

Лабораторная работа №1

по предмету «Технология программирования»

на тему «Основы программирования на Java. Обработка событий. Механизм делегирования событий»

Факультет: АВТ Преподаватель:

Группа: АВТ-809 Михайленко Д.А.

Студент: Швыдко Е.А.

Новосибирск 2020

ЦЕЛЬ И ЗАДАНИЕ

***Вариант 11***

Объекты – домашние животные. Бывают 2 видов: кошки и собаки. Кошки генерируются каждые N1 секунд с вероятностью P1. Собаки генерируются каждые N2 секунд с вероятностью P2.

Цель работы:

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.
3. Разобрать приведенные примеры **FontsList**, **LinesDraw, KeyCodes, Timer**, **DrawFig**.
4. Разработать программу. Основная 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 (далее будут реализоваться алгоритмы движения объектов в окне программы).
* Реализовать иерархию классов, определяющих объекты по варианту и реализующие интерфейс IBehaviour.
* Создать интерфейс AbstractFactory для создания объектов классов.
* Реализовать от него класс ConcreteFactory, возвращающий нужный объект, согласно варианту индивидуального задания.
* По желанию можно использовать паттерн Factory Method.
* 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 – схематично, плюсом будет, если объект будет похож на оригинал (можно 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 уничтожается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише **T**;
2. По завершению симуляции в поле визуализации должна выводиться 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.
4. При необходимости или желанию можно использовать другие паттерны.

Ход работы:

* Был реализован интерфейс IBehaviour, реализующий методы согласно заданию

**public** **interface** **IBehaviour** {

**public** void move();

**public** int getX();

**public** int getY();

**public** void setX(int X);

**public** void setY(int Y);

}

* Были разработаны классы Cat и Dog, наследники абстрактного класса Animal

**public** **class** Cat **extends** Animal {

**private** **static** BufferedImage Image;

**static** {

**try** {

Image = ImageIO.read(**new** **File**("src/com/company/assets/cat.png"));

} **catch** (**IOException** e) {

e.printStackTrace();

}

}

**public** Cat(int \_X, int \_Y, int \_ID){

**this**.X = \_X;

**this**.Y = \_Y;

**this**.id = \_ID;

}

**public** int getID() {**return** id;}

**@Override**

**public** int getX() {

**return** **super**.getX();

}

**@Override**

**public** int getY() {

**return** **super**.getY();

}

**@Override**

**public** void setX(int X) {

**this**.X = X;

}

**@Override**

**public** void setY(int Y) {

**this**.Y = Y;

}

**@Override**

**public** void move(){

}

**public** void draw(Graphics g){

Graphics2D g2d = (Graphics2D) g;

g2d.drawImage(Image, **this**.X, **this**.Y, 80, 80, **null**);

}

}

* Был реализован паттерн Factory Method, позволяющий создавать объекты двух классов Cat и Dog. Для этого был создан интерфейс «ObjectsFactory»:

, который имеет лишь один метод, возвращающий экземпляр класса Animal

**public** **interface** **ObjectFactory** {

Car createObject(int \_id);

}

Были разработаны методы для создания экземпляров заданных классов:

**public** **class** **CatFactory** **implements** ObjectFactory {

@Override

**public** Car createObject(int \_id) {

**return** **new** Cat((int) (Math.random() \* (Habitat.WIDTH+1)), (int) (Math.random() \* (Habitat.HEIGHT+1)), \_id,));

}

}

* Был разработан класс Habitat, определяющий рабочую область программы.
* Были реализованы обработчики событий типа KeyListener, прослушивающие нажатия на клавиши “B”, “E” и “T”

this.addKeyListener(new KeyAdapter() {

**@Override**

**public** void keyReleased(KeyEvent e) {

**if**(e.getKeyCode() == KeyEvent.VK\_B){

start();

}

**if**(e.getKeyCode() == KeyEvent.VK\_E){

stop();

}

**if**(e.getKeyCode() == KeyEvent.VK\_T){

**if**(timerLabel.isVisible()) timerLabel.setVisible(**false**);

**else** timerLabel.setVisible(**true**);

}

}

});

setVisible(true);

}

**public** void start(){

timer = **new** **Timer**();

timer.scheduleAtFixedRate(**new** **TimerTask**() {

**@Override**

**public** void run() {

counter++;

habitat.update(counter/10);

timerLabel.setText("�'ремя: " + counter/10 + " сек");

}

}, 0,100);

}

**public** void stop(){

timer.cancel();

finalTimer.setText("�'ремени прошло: " + counter/10 + "сек");

finalCats.setText("Кошек: " + Habitat.cats);

finalDogs.setText("Собак: " + Habitat.dogs);

}

* Был разработан класс MainFrame – класс главного окна программы, в котором помещается сама область визуализации и элементы управления симуляцией.

**public** **class** MFrame **extends** JFrame {

**private** **Timer** timer;

**private** double counter = 0;

**private** Habitat habitat;

**private** JLabel timerLabel, finalTimer, finalCats, finalDogs;

MFrame(){

setTitle("ok");

setSize(1200,800);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setResizable(**true**);

setLocationRelativeTo(**null**);

setLayout(**null**);

setFocusable(**true**);

finalTimer = **new** JLabel();

finalCats = **new** JLabel();

finalDogs = **new** JLabel();

add(finalTimer);

add(finalCats);

add(finalDogs);

finalTimer.setBounds(500,150,200,13);

finalCats.setBounds(500,165,100,13);

finalDogs.setBounds(500,180,100,13);

timerLabel = **new** JLabel("�'ремя: 0 сек");

add(timerLabel);

timerLabel.setBounds(100,120,100,13);

habitat = **new** Habitat(**this**);

add(habitat);

habitat.setBounds(0,100,1100,600);

**this**.addKeyListener(**new** KeyAdapter() {

**@Override**

**public** void keyReleased(KeyEvent e) {

**if**(e.getKeyCode() == KeyEvent.VK\_B){

start();

}

**if**(e.getKeyCode() == KeyEvent.VK\_E){

stop();

}

**if**(e.getKeyCode() == KeyEvent.VK\_T){

**if**(timerLabel.isVisible()) timerLabel.setVisible(**false**);

**else** timerLabel.setVisible(**true**);

}

}

});

setVisible(**true**);

}

**public** void start(){

timer = **new** **Timer**();

timer.scheduleAtFixedRate(**new** **TimerTask**() {

**@Override**

**public** void run() {

counter++;

habitat.update(counter/10);

timerLabel.setText("�'ремя: " + counter/10 + " сек");

}

}, 0,100);

}

**public** void stop(){

timer.cancel();

finalTimer.setText("�'ремени прошло: " + counter/10 + "сек");

finalCats.setText("Кошек: " + Habitat.cats);

finalDogs.setText("Собак: " + Habitat.dogs);

}

}

Вывод:

В результате выполнения работы были изучены базовые типы данных Java, рассмотрены и изучены обработчики событий и их применение. Был разработан базовый пользовательский интерфейс программы.