ЛАБОРАТОРНАЯ РАБОТА № 3
РАБОТА С МНОГОПОТОКОВЫМИ ПРИЛОЖЕНИЯМИ В JAVA

1 Цель занятия

Сформировать практические навыки по созданию и управлению многопоточными приложениями с использованием языка программирования Java.

2 Общие теоретические сведения

2.1 Процессы, потоки и приоритеты

Обычно в многозадачной операционной системе (ОС) выделяют такие объекты, как процессы и потоки.

Процесс (process) - это объект, который создается ОС при запуске приложения. Процессу выделяется отдельное адресное пространство, это пространство физически недоступно для других процессов. Процесс может работать с файлами или с каналами связи локальной или глобальной сети.

Для каждого процесса ОС создает один главный поток (thread), который является потоком выполняющихся по очереди команд центрального процессора. При необходимости главный поток может создавать другие потоки, пользуясь для этого программным интерфейсом ОС. Все потоки, созданные процессом, выполняются в адресном пространстве этого процесса и имеют доступ к ресурсам процесса.

Если процесс создал несколько потоков, то все они выполняются параллельно, причем время центрального процессора (или нескольких центральных процессоров в многопроцессорных системах) распределяется между этими потоками.

Распределением времени центрального процессора занимается специальный модуль операционной системы - планировщик. Планировщик по очереди передает управление отдельным потокам, так что даже в однопроцессорной системе создается полная иллюзия параллельной работы запущенных потоков.

Распределение времени выполняется для потоков, а не для процессов. Потоки, созданные разными процессами, конкурируют между собой за получение процессорного времени. Каждому потоку задается приоритет его выполнения, уровень которого определяет очередность выполнения того или иного потока.

2.2 Реализация многозадачности в Java

Для создания многозадачных приложений Java необходимо воспользоваться классом java.lang.Thread. В этом классе определены все методы, необходимые для создания потоков, управления их состоянием и синхронизации.

Есть две возможности использования класса Thread.

Во-первых, можно создать собственный класс на базе класса Thread и переопределить метод run(). Новая реализация этого метода будет работать в рамках отдельного потока.

Во-вторых, создаваемый класс, может реализовать интерфейс Runnable и реализовать метод run(), который будет работать как отдельный поток.

2.2.1. Создание подкласса Thread

При использовании этого способа для потоков определяется отдельный класс, например:

```
class myThread extends Thread  {
  public void run()   {
```

```
      // здесь можно добавить код, который будет
      // выполняться в рамках отдельного потока
    }
   // здесь можно добавить специализированный для класса код
}
```

Метод run() должен быть всегда переопределен в классе, наследованном от Thread. Именно он определяет действия, выполняемые в рамках отдельного потока. Если поток используется для выполнения циклической работы, этот метод содержит внутри себя бесконечный цикл.

Метод run() получает управление при запуске потока методом start() класса Thread. В случае апплетов создание и запуск потоков обычно осуществляется в методе start() апплета.

Остановка работающего потока раньше выполнялась методом stop() класса Thread. Обычно остановка всех работающих потоков, созданных апплетом, выполняется в методе stop() апплета. Сейчас не рекомендуется использование этого метода. Завершение работы потока желательно проводить так, чтобы происходило естественное завершение метода run. Для этого используется управляющая переменная в потоке.

Пример 1. Многопоточное приложение с использованием наследников класса Thread

```
// Поток для расчета координат прямоугольника
class ComputeRects extends Thread  {
   boolean going = true;
// конструктор получает ссылку на создателя объекта - апплет
   public ComputeRects(MainApplet parentObj)     {
      parent = parentObj;
   }
   public void run()     {
      while(going)    {
         int w = parent.size().width-1, h = parent.size().height-1;
         parent.RectCoordinates
         ((int)(Math.random()*w),(int)(Math.random()*h));
      }
   }
   MainApplet parent;              // ссылка на создателя объекта
}
// Поток для расчета координат овала
class ComputeOvals extends Thread   {
   boolean going = true;
    public ComputeOvals(MainApplet parentObj)    {
      parent = parentObj;
   }
   public void run()     {
      while(going)   {
         int w = parent.size().width-1, h = parent.size().height-1;
         parent.OvalCoordinates
         ((int)(Math.random()*w),(int)(Math.random()*h));
      }
   }
   MainApplet parent; // ссылка на создателя объекта
}

public class MainApplet extends JApplet   {
   ComputeRects m_rects = null;
   ComputeOvals m_ovals = null;
   int m_rectX = 0;    int m_rectY = 0;
   int m_ovalX = 0;   int m_ovalY = 0;
   // Синхронный метод для установки координат
   // прямоугольника из другого потока
   public synchronized void RectCoordinates(int x, int y)    {
      m_rectX = x; m_rectY = y;
      this.repaint();
   }
   // Синхронный метод для установки координат овала
   // из другого потока
   public synchronized void OvalCoordinates(int x, int y)    {
```

```
        m_ovalX = x; m_ovalY = y;
        this.repaint();
    }
    @Override
    public void start()    {
        super.start();
        // Запускаем потоки
        if (m_rects == null) {
              m_rects = new ComputeRects(this);   m_rects.start();
        }
        if (m_ovals == null) {
             m_ovals = new ComputeOvals(this);   m_ovals.start();
        }
    }
    @Override
    public void stop()    {
        super.stop();
        // Останавливаем потоки
        if(m_rects != null) m_rects.going = false;
        if(m_ovals != null) m_ovals.going = false;
    }
   public void paint(Graphics g)    {
      int w = this.getWidth(), h = this.getHeight();
      g.clearRect(0, 0, w, h);
      g.setColor(Color.red);
      g.fillRect(m_rectX, m_rectY, 20, 20);
      g.setColor(Color.blue);
      g.fillOval(m_ovalX, m_ovalY, 20, 20);
   }
    public static void main(String[] args)    {   }
}
```

Обратите внимание на запуск и остановку потоков в методах start и stop соответственно, а также на объявление синхронных методов и использованием ключевого слова synchronized. Синхронизация крайне важна в многопоточных приложениях, так как потоки, работающие над одними и теми же данными одновременно, могут испортить эти данные.

2.2.2. Реализация интерфейса Runnable.

Если нет возможности расширять класс Thread, то можно применить второй способ реализации многозадачности. Допустим, уже существует класс MyClass, функциональные возможности которого удовлетворяют разработчика. Необходимо, чтобы он выполнялся как отдельный поток.

```
class MyClass implements Runnable   {
  // код класса - объявление его элементов и методов
  // этот метод получает управление при запуске потока
  public void run()   {
    // здесь можно добавить код, который будет
    // выполняться в рамках отдельного потока
  }
}
```

Добавим новый поток в пример 1, реализованный через интерфейс Runnable.

Пример 2. Доработанное многопоточное приложение

```
// Поток для расчета координат линии
class ComputeLines implements Runnable  {
   boolean going = true;
   public ComputeLines(MainApplet parentObj)    {
      parent = parentObj;
   }
   public void compute()    {
```

```
        int w = parent.size().width-1, h = parent.size().height-1;
        parent.LineCoordinates
              ((int)(Math.random()*w),(int)(Math.random()*h),
              (int)(Math.random()*w), (int)(Math.random()*h));
    }
    MainApplet parent; // ссылка на создателя объекта
    public void run()    {
      while(going) { compute(); }
    }
}

public class MainApplet extends JApplet {
  . . .             // скопируйте из предыдущего примера
    ComputeLines m_lines = null;
    int m_lineX1 = 0, m_lineX2 = 0, m_lineY1 = 0, m_lineY2 = 0;
    // Синхронный метод для установки координат
    // прямоугольника из другого потока
    public synchronized void RectCoordinates(int x, int y)    {
      m_rectX = x; m_rectY = y;
      this.repaint();
    }
    // Синхронный метод для установки координат овала
    // из другого потока
    public synchronized void OvalCoordinates(int x, int y)    {
      m_ovalX = x; m_ovalY = y;
      this.repaint();
    }
    // Синхронный метод для установки координат линии
    // из другого потока
 public synchronized void LineCoordinates(int x1, int y1, int x2, int y2)  {
      m_lineX1 = x1; m_lineX2 = x2; m_lineY1 = y1; m_lineY2 = y2;
      this.repaint();
    }
    @Override
    public void start()    {
      super.start();
      // Запускаем потоки
      . . .        // скопируйте из предыдущего примера
      if (m_lines == null) {
          m_lines = new ComputeLines(this);
          new Thread(m_lines).start();
      }
    }
    @Override
    public void stop()    {
      super.stop();
      // Останавливаем потоки
      . . .        // скопируйте из предыдущего примера
      if(m_lines != null) m_lines.going = false;
    }
   public void paint(Graphics g)    {
     . . .        // скопируйте из предыдущего примера
     g.setColor(Color.green);
     g.drawLine(m_lineX1, m_lineX2, m_lineY1, m_lineY2);
   }
    public static void main(String[] args)    {   }
}
```

2.3 Применение анимации для мультизадачности

Одним из наиболее распространенных применений апплетов является создание анимационных эффектов типа бегущей строки, мерцающих огней и других эффектов, привлекающих внимание пользователя. Для достижения таких эффектов необходим механизм, позволяющий выполнять перерисовку всего окна апплета или его части периодически с заданным интервалом. Перерисовка окна апплета выполняется методом paint(), который вызывается

виртуальной машиной Java асинхронно по отношению к выполнению другого кода апплета, если содержимое окна было перекрыто другими окнами. Для периодической перерисовки окна апплета необходимо создание потока (или нескольких потоков), которые будут выполнять рисование в окне апплета асинхронно по отношению к коду апплета. Например, можно создать поток, который периодически обновляет окно апплета, вызывая для этого метод repaint(), или рисовать из потока непосредственно в окне апплета. Также можно использовать класс таймера (этот механизм уже использовался в предыдущих работах). Таймер, по сути, тоже является потоком, выполняющимся параллельно с основным.

Рассмотрим пример апплета 3, который умеет сам себя перерисовывать при помощи дополнительного потока.

Пример 3. Самоперерисовывающийся апплет

```
public class MainApplet extends JApplet implements Runnable   {
   boolean m_isGoing = false;
   @Override
   public void run()    {
      while (m_isGoing)       {
         repaint();
         try  {    Thread.sleep(500); }
         catch(InterruptedException e) { stop(); }
      }
   }
   @Override
   public void start()    {
      super.start();
      // Запускаем поток
      m_isGoing = true;
      new Thread(this).start();
   }
   @Override
   public void stop()    {
      super.stop();
      // Останавливаем потоки
      m_isGoing = false;
      // Дадим потоку время завершиться
      try  {    Thread.sleep(500); }
      catch(InterruptedException e) {}
   }
  public void paint(Graphics g)    {
     int w = this.getWidth(), h = this.getHeight();
     g.setColor(new Color((int)(Math.random() * 255),
     (int)(Math.random() * 255), (int)(Math.random() * 255)));
     g.fillRect(0, 0, w, h);
  }
   public static void main(String[] args)    {    }
}
```

Класс Thread содержит несколько конструкторов и большое количество методов для управления потоков.

2.3.1. Некоторые методы класса Thread:

currentThread() − возвращает ссылку на выполняемый в настоящий момент объект класса Thread;

sleep() − переводит выполняемый в данное время поток в режим ожидания в течение указанного промежутка времени ;

start() − начинает выполнение потока. Это метод приводит к вызову соответствующего метода run();

run() − фактическое тело потока. Этот метод вызывает после запуска потока;

stop() − останавливает поток (устаревший метод);

isAlive() − определяет, является ли поток активным (запущенным и не остановленным);

suspend() − приостанавливает выполнение потока (устаревший метод);

resume() − возобновляет выполнение потока (устаревший метод). Этот метод работает только после вызова метода suspend();

setPriority() − устанавливает приоритет потока (принимает значение от MIN_PRIORITY до MAX_PRIORITY);

getPriority() − возвращает приоритет потока;

wait() − переводит поток в состояние ожидания выполнения условия, определяемого переменной условия;

join() − ожидает, пока данный поток не завершит своего существования бесконечно долго или в течении некоторого времени;

setDaemon() − отмечает данный поток как поток-демон или пользовательский поток. Когда в системе останутся только потоки-демоны, программа на языке Java завершит свою работу;

isDaemon() − возвращает признак потока-демона.

2.3.2. Состояние потока

Во время своего существования поток может переходить во многие состояния, находясь в одном из нижеперечисленных состояний:

Новый поток

Выполняемый поток

Невыполняемый поток

Завершенный поток

Новый поток. При создании экземпляра потока этот поток приобретает состояние “Новый поток”:

Thread myThread=new Thread();

В этот момент для данного потока распределяются системные ресурсы; это всего лишь пустой объект. В результате все, что с ним можно делать - это запустить: myThread.start();

Любой другой метод потока в таком состоянии вызвать нельзя, это приведет к возникновению исключительной ситуации.

Выполняемый поток. Когда поток получает метод start(), он переходит в состояние “Выполняемый поток”. Процессор разделяет время между всеми выполняемыми потоками согласно их приоритетам.

Невыполняемый поток. Если поток не находится в состоянии “Выполняемый поток”, то он может оказаться в состоянии “Невыполняемый поток”. Это состояние наступает тогда, когда выполняется одно из четырех условий:

Поток был приостановлен. Это условие является результатом вызова метода suspend(). После вызова этого метода поток не находится в состоянии готовности к выполнению; его сначала нужно “разбудить” с помощью метода resume(). Это полезно в том случае, когда необходимо приостановить выполнение потока, не удаляя его. Поскольку метод suspend не рекомендуется к использованию, приостановка потока должна выполняться через управляющую переменную.

Поток ожидает. Это условие является результатом вызова метода sleep(). После вызова этого метода поток переходит в состояние ожидания в течении некоторого определенного промежутка времени и не может выполняться до истечения этого промежутка. Даже если ожидающий поток имеет доступ к процессору, он его не получит. Когда указанный промежуток времени пройдет, поток переходит в состояние “Выполняемый поток”. Метод resume() не может повлиять на процесс ожидания потока, этот метод применяется только для приостановленных потоков.

Поток ожидает извещения. Это условие является результатом вызова метода wait(). С помощью этого метода потоку можно указать перейти в состояние ожидания выполнения условия, определяемого переменной условия, вынуждая его тем самым приостановить свое выполнение до

тех пор, пока данное условие удовлетворяется. Какой бы объект не управлял ожидаемым условием, изменение состояния ожидающих потоков должно осуществляться посредством одного из двух методов этого потока - notify() или notifyAll(). Если поток ожидает наступление какого-либо события, он может продолжить свое выполнение только в случае вызова для него этих методов.

Поток заблокирован другим потоком. Это условие является результатом блокировки операцией ввода-вывода или другим потоком. В этом случае у потока нет другого выбора, как ожидать до тех пор, пока не завершится команда ввода-вывода или действия другого потока. В этом случае поток считается невыполняемым, даже если он полностью готов к выполнению.

Завершенный поток. Когда метод run() завершается, поток переходит в состояние “Завершенный поток”.

2.3.3. Приоритеты потоков

В языке Java каждый поток обладает приоритетом, который оказывает влияние на порядок его выполнения. Потоки с высоким приоритетом выполняются чаще потоков с низким приоритетом. Поток наследует свой приоритет от потока, его создавшего. Если потоку не присвоен новый приоритет, он будет сохранять данный приоритет до своего завершения. Приоритет потока можно установить с помощью метода setPriority(), присваивая ему значение от MIN_PRIORITY до MAX_PRIORITY (константы класса Thread). По умолчанию потоку присваивается приоритет Thread.NORM_PRIORITY.

Потоки в языке Java планируются с использованием алгоритма планирования с фиксированными приоритетами. Этот алгоритм, по существу, управляет потоками на основе их взаимных приоритетов, кратко его можно изложить в виде следующего правила: в любой момент времени будет выполняться “Выполняемый поток” с наивысшим приоритетом. Как выполняются потоки одного и того же приоритета, в спецификации Java не описано.

2.3.4. Группы потоков

Все потоки в языке Java должны входить в состав группы потоков. В классе Thread имеется три конструктора, которые дают возможность указывать, в состав какой группы должен входить данный создаваемый поток.

Группы потоков особенно полезны, поскольку внутри их можно запустить или приостановить все потоки, а это значит, что при этом не потребуется иметь дело с каждым потоком отдельно. Группы потоков предоставляют общий способ одновременной работы с рядом потоков, что позволяет значительно сэкономить время и усилия, затрачиваемые на работу с каждым потоком в отдельности.

В приведенном ниже фрагменте программы создается группа потоков под названием genericGroup (родовая группа). Когда группа создана, создаются несколько потоков, входящих в ее состав:

```
ThreadGroup genericGroup=new ThreadGroup("My generic group");
Thread t1=new Thread(genericGroup,this);
Thread t2=new Thread(genericGroup,this);
```

Если при создании нового потока не указать, к какой конкретной группе он принадлежит, этот поток войдет в состав группы потоков main (главная группа). Иногда ее еще называют текущей группой потоков. В случае апплета main может и не быть главной группой. Право присвоения имени принадлежит Web-браузеру. Для того чтобы определить имя группы потоков, можно воспользоваться методом getName() класса ThreadGroup.

Для того, чтобы определить к какой группе принадлежит данный поток, используется метод getThreadGroup(), определенный в классе Thread. Этот метод возвращает имя группы потоков, в которую можно послать множество методов, которые будут применяться к каждому члену этой группы.

2.4 Программирование движения объекта

Для того чтобы запрограммировать движение объекта из одной точки в другую, вспомним равномерное движение.

Пусть (x1;y1) – начальная координата объекта, а (x2;y2) – конечная координата.

Направление движения задается вектором (dx;dy) = Нормализация(x2-x1; y2-y1), где

Нормализация(a,b) = (a / sqrt(a^2 + b^2); b / sqrt(a^2 + b^2)).

Таким образом, координаты объекта в момент времени T от начала движения, движущегося со скоростью V по прямой, равны: (xc;yc) = (x1; y1) + (dx;dy) * V * T;

Скорость V измеряется в пиксель/сек, т.е. количество пикселей пройденных объектом на 1 сек.

Когда (xc;yc) станет равно (x2;y2), тогда объект дошел до конечной точки. На практике сравнивать текущие координаты с концом отрезка не корректно, так как промах на 1 пиксель из-за округления, например, отправит наш объект в бесконечное путешествие по прямой. Можно, например, прикинуть время, которое потребуется объекту, чтобы дойти до конца отрезка, поделив длину отрезка на скорость объекта. Условием продолжения движения будет время от начала движения T меньшее времени, которое требуется, чтобы пройти отрезок.

Движение по окружности программируется иначе.

Пусть (x0;y0) – центр окружности движения. Тогда координаты объекта в момент времени T от начала движения, движущегося со скоростью V по окружности, равны:

(xc;yc) = (x0; y0) + R * (cos(V*T); sin(V*T)),

где R – радиус окружности.

В этом случае в отличие от предыдущего V – угловая скорость, т.е. измеряется в радиан/сек.

3 Задачи для самостоятельного решения студентами

3.1 Получить у преподавателя один из следующих вариантов

Вариант 1

1. Объект – муравей. Бывают 2 видов: рабочий и воин. Рабочие рождаются каждые N1 секунд с вероятностью P1. Воины рождаются каждые N2 секунд с вероятностью P2.
2. Муравьи-рабочие двигаются в один из углов области их обитания (например, [0;0]) по прямой со скоростью V, а затем возвращаться обратно в точку своего рождения с той же скоростью.
3. Муравьи-воины двигаются по окружности с радиусом R со скоростью V.
4. Муравей рабочий имеет размер X, а воин 1,5X.
5. Столкновением является ситуация, когда более 60% муравья перекрывается другим муравьём.

Вариант 2

1. Объект – пчела. Бывают 2 видов: трутень и рабочий. Трутни рождаются каждые N_1 секунд, если их количество менее K% от общего числа пчел, в противном случае – не рождаются вовсе. Рабочие рождаются каждые N_2 секунд с вероятностью P.
2. Пчелы-рабочие двигаются в один из углов области их обитания (например, [0;0]) по прямой со скоростью V, а затем возвращаться обратно в точку своего рождения с той же скоростью.
3. Трутни двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд.
4. Пчелы-рабочие и трутни рабочий имеет размер X.

5. Столкновением является ситуация, когда более 30% пчелы перекрывается другой пчелой.

Вариант 3

1. Объект – аквариумная рыбка. Бывают 2 видов: золотая и гуппи. Золотые рыбки рождаются каждые N_1 секунд с вероятностью P_1. Гуппи рождаются каждые N_2 секунд с вероятностью P_2.

2. Золотые рыбки двигаются по оси X от одного края области обитания до другого со скоростью V.

3. Гуппи двигаются по оси Y от одного края области обитания до другого со скоростью V.

4. Гуппи имеет размер X, а золотая рыбка 5X.

5. Столкновением является ситуация, когда гуппи перекрывает более 10% золотой рыбки.

Вариант 4

1. Объект – кролик. Бывают 2 видов: обыкновенный и альбинос. Обыкновенные кролики рождаются каждые N_1 секунд с вероятностью P_1. Альбиносы рождаются каждые N_2 секунд, при условии, что их количество менее K% от общего числа кроликов, в противном случае – не рождаются вовсе.

2. Обыкновенные кролики двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд.

3. Альбиносы двигаются по оси X от одного края области обитания до другого со скоростью V.

4. Все кролики имеют размер X.

5. Столкновением является ситуация, когда более 50% кролика перекрывается другим кроликом.

Вариант 5

1. Список объектов продажи на автомобильном рынке состоит из 2-х видов машин: грузовые и легковые. Грузовые машины генерируются каждые N1 секунд с вероятностью P1. Легковые генерируются каждые N2 секунд с вероятностью P2.

2. Грузовые машины двигаются в левую верхнюю четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке 0;0, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если машина сгенерировалась сразу в этой области, то она никуда не движется. По прибытии в конечную точку машина больше не движется.

3. Легковые машины двигаются в нижнюю правую четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке w/2;h/2, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если машина сгенерировалась сразу в этой области, то она никуда не движется. По прибытии в конечную точку машина больше не движется.

4. Легковые машины имеют размер X, а грузовые 2X.

5. Столкновением является ситуация, когда происходит любое перекрывание одного автомобиля другим.

Вариант 6

1. Рабочий коллектив компании состоит из разработчиков и менеджеров. Разработчики генерируются каждые N_1 секунд с вероятностью P_1. Менеджеры генерируются каждые N_2 секунд при условии, что их количество менее K% от общего числа разработчиков, в противном случае – не генерируются.

2. Разработчики двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд.

3. Менеджеры двигаются по окружности с радиусом R со скоростью V.

4. Разработчики и Менеджеры имеют размер X.

5. Столкновением является ситуация, когда работники перекрываются на 20%.

Вариант 7

1. Список жилых домов города состоит из двух типов: капитальный, деревянный. Капитальные дома генерируются каждые N_1 секунд с вероятностью P_1. Деревянные дома генерируются каждые N_2 секунд с вероятностью P_2.

2. Капитальные дома двигаются (в городах будущего и не такое возможно) в левую верхнюю четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке 0;0, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если дом

сгенерировался сразу в этой области, то он никуда не движется. По прибытии в конечную точку дом больше не движется.

3. Деревянные дома после генерации начинают двигаться в нижнюю правую четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке w/2;h/2, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если дом сгенерировался сразу в этой области, то он никуда не движется. По прибытии в конечную точку дом больше не движется.

4. Деревянные дома имеют размер X, а капитальные 6X.

5. Столкновением является ситуация, когда происходит любое перекрывание на 5% одного дома другим.

Вариант 8

1. Список транспортных средств на дороге состоит из двух категорий: автомобили и мотоциклы. Автомобили генерируются каждые N_1 секунд с вероятностью P_1. Мотоциклы генерируются каждые N_2 секунд с вероятностью P_2.

2. Автомобили двигаются по оси X от одного края области симуляции до другого со скоростью V.

3. Мотоциклы двигаются по оси Y от одного края области симуляции до другого со скоростью V.

4. Мотоциклы дома имеют размер X, а автомобили 2,5X.

5. Столкновением является ситуация, когда происходит любое перекрывание на 10% одного транспортного средства другим.

Вариант 9

1. Объект обучающийся. Бывает 2 видов: студент (муж. пола) и студентка (жен. пола). Студенты генерируются каждые N_1 секунд с вероятностью P_1. Студентки генерируются каждые N_2 секунд с вероятностью P_2.

2. Студенты двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд.

3. Студентки двигаются по окружности с радиусом R со скоростью V.

4. Студентки размер X, а студенты 1,2X.

5. Столкновением является ситуация, когда происходит перекрывание на 80% одного обучающегося другим.

Вариант 10

1. Картотека налоговой инспекции состоит из записей двух типов: физические и юридические лица. Физические лица генерируются каждые N_1 секунд с вероятностью P_1. Юридические лица генерируются каждые N_2 секунд с вероятностью P_2.

2. Юр. лица двигаются в левую верхнюю четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке 0;0, шириной/длиной = (w/2*h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если запись сгенерировалась сразу в этой области, то она никуда не движется. По прибытии в конечную точку запись больше не движется.

3. Физ. лица двигаются в нижнюю правую четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке w/2*h/2, шириной/длиной = (w/2*h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если запись сгенерировалась сразу в этой области, то она никуда не движется. По прибытии в конечную точку запись больше не движется.

4. Физические и юридические лица имеют размер X.

5. Столкновением является ситуация, когда происходит любое перекрывание между юридическими лицами, либо между физическими, перекрывание разных типов не рассматривается

3.2 Порядок выполнение лабораторной работы

1 Изучить особенности реализации и работы потоков в Java, описанные в настоящем документе.

2. Разработать программу упрощенной имитации поведения объектов. Объекты реализуются через наследование: абстрактный класс, интерфейс → наследники.

Рабочий цикл программы:

- запускается процесс симуляции, генерируются объекты классов согласно заданию;
- симуляция завершается, выводится статистическая информация.

Для решения задачи:

- создать интерфейс IBehaviour, задающий поведение объекта;
- создать иерархию классов, определяющие объекты по варианту и реализующие интерфейс IBehaviour.
- создать класс Habitat, определяющий размер рабочей области и хранящий список 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.

Рабочее окно программы – область визуализации среды обитания объектов.

Размер объекта не менее 8пт.

3. Симуляция должна запускаться по клавише S и останавливаться по клавише E. При остановке симуляции список уничтожается. Время симуляции и текущее число столкновений должно отображаться текстом в области визуализации и скрываться/показываться по клавише T.

4. По завершению симуляции в поле визуализации должна выводиться информация о количестве и типе сгенерированных объектов, числе столкновений, а также время симуляции. Текст должен быть форматирован, т.е. выводиться с использованием разных шрифтов и цветов.

5. Параметры симуляции задаются в классе Habitat.

6. Для создания «интеллекта» в программе

- создать абстрактный класс BaseAI, описывающий «интеллектуальное поведение» объектов по варианту. Класс должен быть выполнен в виде отдельного потока;
- «интеллектуальное поведение» включает реакцию на столкновение в виде уточнения траектории движения объектов (если они не равны, то траектории более мелкого, если равны, то обоих) на минимальную величину, но достаточную, чтобы на следующем шаге симуляции перекрытия между столкнувшимися объектами не было; смены направления движения быть не должно;
- реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
- синхронизовать работу потоков расчета интеллекта объектов с их рисованием. Рисование должно остаться в основном потоке. Синхронизация осуществляется через передачу данных в основной поток;
- добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через засыпание/пробуждение потоков.

4 Контрольные вопросы

1 Что такое процесс и поток (нить)?

2 Чем определяется порядок передачи управления потокам?

3 Какие есть способы реализации многозадачности в Java?

4 Что необходимо сделать для создания подкласса потоков (подкласса Thread)?

5 Когда запускается на выполнение метод run() подкласса Thread?

6 Какими методами класса Thread необходимо запускать поток на выполнение и останавливать его?

7 Что необходимо сделать для реализации классом интерфейса Runnable?

8 В каких состояниях может находиться поток?

9 Какой поток считается новым, выполняемым и завершенным?

10 В каких ситуациях поток является невыполняемым?

11 Когда возникают исключительные ситуации при работе с потоками?

12 Что такое группы потоков и чем они полезны?

13 Что такое родовая группа потоков и главная группа потоков?