Практические занятия в компьютерном класса

Занятие №5 «Наследование»

Разработка программ, использующий отношение наследования между классами. Применение интегрированной среды программирования **Eclipse** для разработки своих классов, использующих отношение наследования.

Задание №1

Разработать класс, который моделирует точку на плоскости. С помощью методов класса пользователь должен иметь возможность размещать точку в заданном месте на плоскости, сдвигать ее на заданное расстояние по осям, вычислять расстояние между начальным и текущим расположением точки и выполнять отладочную печать.

После того, как класс будет разработан и протестирован создать класс-наследник, моделирующий точку в пространстве. Протестировать его работу.

С помощью разработанных классов создать несколько точек (как в пространстве, так и на плоскости), после чего каждая точка выполнит несколько случайных смещений. Найти точку, которая удалилась на самое большое расстояние от начального положения.

Решение

Как обычно, загружаем интегрированную среду программирования **Eclipse**, закрываем свои старые проекты и создаем новый *Java* проект с именем Theme5. В данном проекте создаем пакет classWork5, а в нем будем размещать свои классы.

Стандартным образом создадим класс Point. При создании класса отключим автоматическое создание функции main(). Будет автоматически создан и добавлен в пакет файл Point.java, содержащий класс Point:

```
package classWork5;

public class Point {

}
```

Для реализации нужной функциональности добавим поля, предназначенные для хранения начальных и текущих координат точки:

```
public class Point {
   private int x, y;          // текущие координаты точки
   private final int x0, y0; // начальное положение точки
}
```

Затем к классу следует добавить конструкторы, которые нужны для корректного создания объектов данного класса. Для создания конструкторов, как и на прошлом занятии, нужно вызвать команду меню *Source | Generate Constructor using Fields …*. На экран будет выведено диалоговое окно *Generate Constructor using Fields*, с помощью которого можно указать, какой конструктор будет создавать система. Для нашего класса потребуется два конструктора: конструктор с двумя параметрами и конструктор по умолчанию. Система сгенерирует такой код:

```
public Point(int x, int y) {
   this.x = x;
```

```
    this.y = y;
}

public Point() {
}
```

Немного исправим автоматически сгенерированные конструкторы. Они примут такой вид:

```
public Point(int x, int y) {
    this.x = x0 = x;
    this.y = y0 = y;
}

public Point() {
    this(0, 0);
}
```

Далее к классу добавим методы, реализующие размещение точки в заданном месте на плоскости (метод setPosition), сдвиг ее на заданное расстояние по осям (метод shift), вычисление расстояние между начальным и текущим расположением точки (метод distance) и выполнение отладочной печати (метод toString).

```
public void setPosition(int x, int y) {
    this.x = x;
    this.y = y;
}

public void shift(int x, int y) {
    this.x += x;
    this.y += y;
}

public String toString() {
    return "(" + x + ", " + y + ")";
}

public double distance() {
    return Math.sqrt((x - x0)*(x - x0) +
            (y - y0)*(y - y0));
}
```

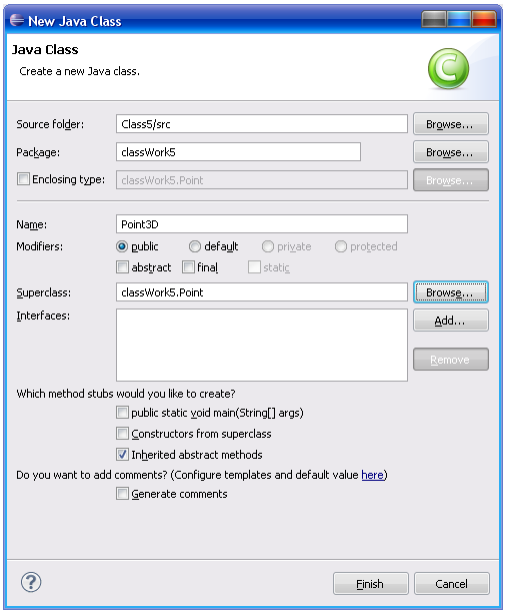
Самостоятельно добавить в класс функцию main(), в которой организовать проверку работы методов класса. Создать точки с помощью разных конструкторов, проверить работу всех методов класса.

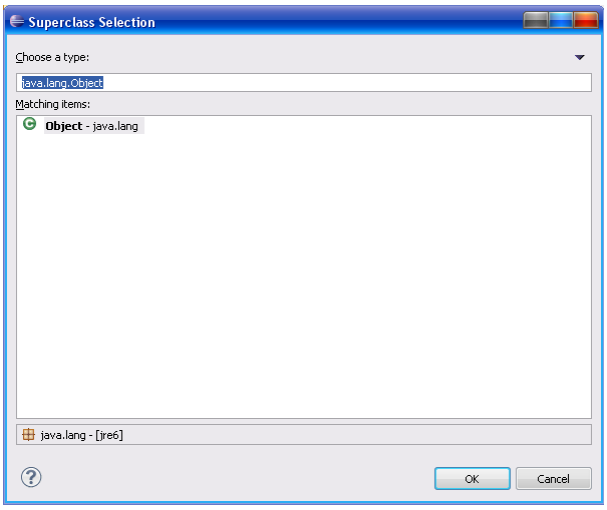
Для моделирования точки в трехмерном пространстве создадим класс-наседник. Назовем его Point3D. При создании этого класса с помощью **Eclipse** нужно будет для него указать базовый класс. Для этого, в поле *name* (см. Рис.) укажем имя создаваемого класса (Point3D), а в поле *Superclass* следует указать его базовый класс. Для этого нажимает кнопку *Browse*, расположенную сразу справа от этого поля. На экран будет выведено диалоговое окно *Superclass Selection*. В текстовом поле *Choose a type* начинаем записывать название базового класса. По мере записи, в списке *Matching items*, расположенном сразу под текстовым полем ,отображаются все похожие по названию классы, из которых можно выбрать базовый. Выбрав в этом списке нужный базовый класс можно либо два раза щелкнуть на нем мышью, либо нажать кнопку *OK*. Выбранный базовый класс будет отображаться в текстовом поле *Superclass* диалогового окна создания класса (см. Рис.). Значения остальных управляющих элементов также показано на рисунке. После нажатия

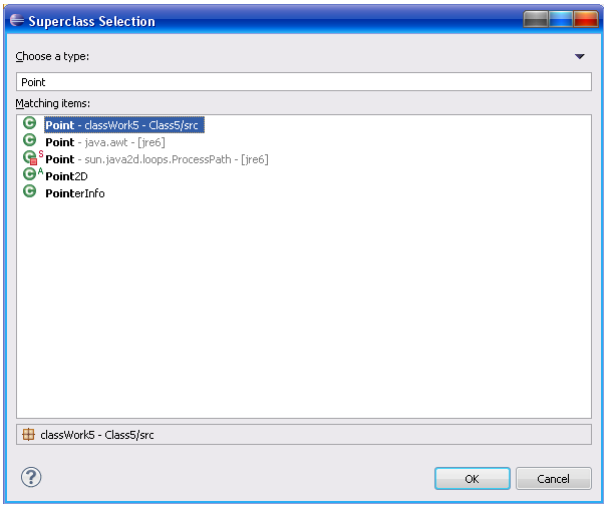
кнопки *Finish* система создаст требуемый класс-наследник.

```
package classWork5;

public class Point3D extends Point {
}
```





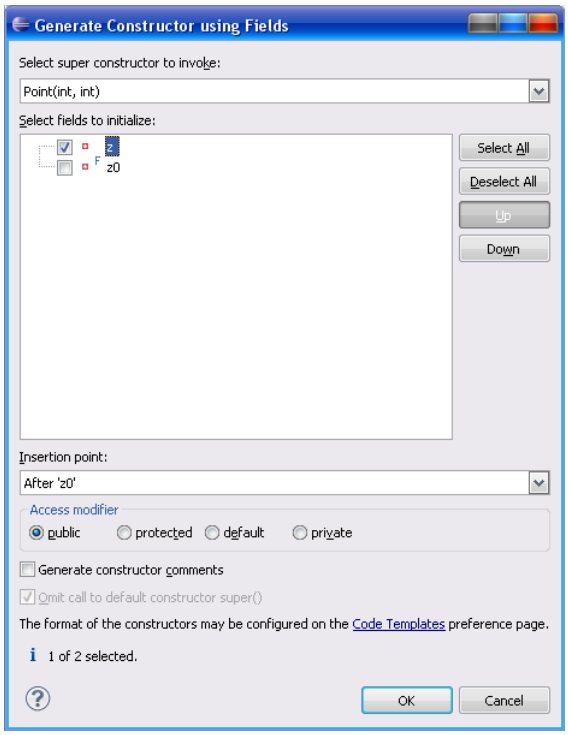


Добавим в класс требуемые поля, которых нет базовом классе.

```
package classWork5;

public class Point3D extends Point {
    private int z;
    private final int z0;
}
```

Для корректной инициализации экземпляров этого класса создадим два конструктора: конструктор с тремя параметрами и конструктор по умолчанию. Для этого выберем команду меню *Source | Generate Constructor using Fields …*. На экран будет выведено диалоговое окно *Generate Constructor using Fields*, с помощью которого можно указать, какой конструктор будет создавать система. Сначала создадим конструктор с тремя параметрами. Для этого, в раскрывающемся списке *Select super constructor to invoke* выберем требуемый конструктор базового класса, который будет автоматически вызываться в начале работы создаваемого конструктора. В нашем случае это конструктор Point(int, int) (см. Рис.). Затем в списке *Select fields to invoke* нужно указать те поля, которые нужно инициализировать (поле z).



В результате будет создан конструктор такого вида:

```
public Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
}
```

Исправим его, добавив инициализацию константы z0.

```
public Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
    this.z = z0 =z;
}
```

Аналогично создаем конструктор без параметров, который в результате, примет такой вид:

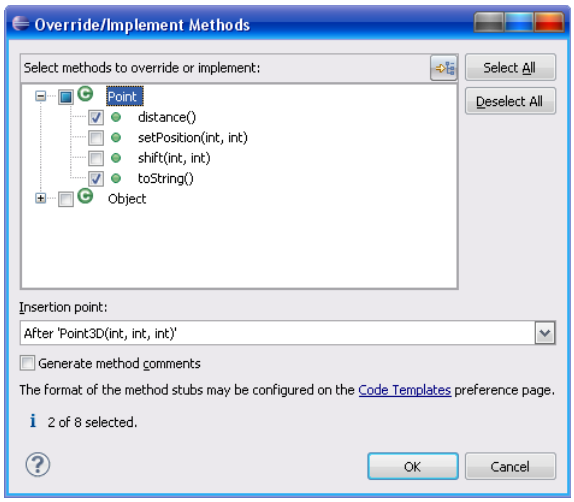
```
public Point3D() {
   super();
   this.z = z0 =0;
}
```

Затем добавим перегрузим методы, полученные в наследство от класса Point для того, чтобы они правильно работали с объектами класса Point3D.

```
public void setPosition(int x, int y, int z) {
   super.setPosition(x, y);
   this.z = z;
}

public void shift(int x, int y, int z) {
   super.shift(x, y);
   this.z += z;
}
```

Затем переопределим методы класса Point для того, чтобы они правильно работали с объектами класса Point3D. Для того, чтобы среда **Eclispe** помогла в разработке этих методов нужно выбрать команду меню *Source* | *Iverride/Implement Methods*... . На экран будет выведено диалоговое окно с указанием базовых классов и методов, которые можно переопределить. Выберем нужные методы для переопределения и укажем, где в классе они будут располагаться (см. Рис).



После того, как будет нажата кнопка *OK*, среда разработки автоматически создаст заготовки методов, выбранных для переопределения:

```
@Override
public String toString() {
   // TODO Auto-generated method stub
   return super.toString();
}

@Override
public double distance() {
   // TODO Auto-generated method stub
   return super.distance();
}
```

Для корректной работы немного изменим автоматически созданные методы:

```
@Override
public String toString() {
   // TODO Auto-generated method stub
   String str = super.toString();
   return str.substring(0, str.length()-1) +
           ", " + z + ")";
}

@Override
public double distance() {
   // TODO Auto-generated method stub
   double tmp = super.distance();
   return Math.sqrt(tmp*tmp +
      (z - z0)*(z - z0));
}
```

Самостоятельно добавить в класс функцию main(), в которой организовать проверку работы методов класса. Создать точки с помощью разных конструкторов, проверить работу всех методов класса.

Для дальнейшей работы **самостоятельно** создайте класс PointGame. Данный класс будет содержать только метод main(). В этом методе пользователь должен сообщить, сколько точек должно быть в массиве. Создать массив требуемого размера, создать точки – элементы этого массива, сместить их на заданное количество случайных шагов, вывести на экран состояние массива и определить ту точку, которая удалилась на самое большое расстояние от начального положения.

Возможный вариант класса может выглядеть так:

```
package classWork5;

public class PointGame {

/**
 * @param args
 */
public static void main(String[] args) {
   // TODO Auto-generated method stub
   // TODO Auto-generated method stub
   final int a = -5, b= 5;
   java.util.Scanner in = new java.util.Scanner(System.in);
   System.out.print("Сколько точек: ");
   int num = in.nextInt();

   Point pts[] = new Point[num];
   int x, y, z;

   for (int i = 0; i < pts.length; i++) {
      System.out.print("Точка " + (i+1)+ "  ");
      if (i%2 == 0) {
         System.out.print("x, y: ");
         x = in.nextInt();
         y = in.nextInt();
         pts[i] = new Point(x, y);
      } else {
```

```
            System.out.print("x, y, z: ");
            x = in.nextInt();
            y = in.nextInt();
            z = in.nextInt();
            pts[i] = new Point3D(x, y, z);
        }

    }

    System.out.println("Начальное положение");
    for (Point pt : pts)
        System.out.println("" + pt);

    for (int j = 1; j < 10; j++)
        for (Point pt : pts) {
            if (pt instanceof Point) {
                x = a + (int) ((b-a)*Math.random());
                y = a + (int) ((b-a)*Math.random());
                pt.shift(x, y);
            } else if (pt instanceof Point3D) {
                x = a + (int) ((b-a)*Math.random());
                y = a + (int) ((b-a)*Math.random());
                z = a + (int) ((b-a)*Math.random());
                ((Point3D)pt).shift(x, y, z);
            }
        }

    System.out.println("Финальное положение");
    for (Point pt : pts)
        System.out.println("" + pt);

    int n = 0;
    System.out.println("Точка: " + n + "  " + pts[n] +
            "  расстояние " + pts[n].distance());
    for (int i = 1; i < pts.length; i++) {
        System.out.println("Точка: " + i + "  " + pts[i] +
                "  расстояние " + pts[i].distance());
        if (pts[i].distance() > pts[n].distance())
            n = i;
    }

    System.out.println("На максимальное расстояние отошла точка с индексом " + n);
    System.out.println(pts[n]);
}

}
```

Задание №2

С заданной точностью найти численные решения уравнения вида:

$$f(x)=0$$

методом дихотомии.

Необходимо написать «универсальную» программу, которая могла бы находить решения уравнения для разных функциональных зависимостей $f(x)$, определяющих левую часть уравнения.

Для тестирования задачи найти корни уравнения:

$$x^2+2\cos\left(x+\frac{\pi}{6}\right)=3x-2$$

на интервалах:

- ✔ $[0.5, 1.5]$ (примерное значение 1.0318);
- ✔ $[2.5, 3.5]$ (примерное значение 2.96071).

Математический метод

Предположим, что проведено отделение корней уравнения, т. е. найдены такие интервалы $[a, b]$ для переменной x, что на каждом отрезке расположен только один корень, который необходимо уточнить с заданной погрешностью ε. Метод дихотомии, или метод половинного деления, заключается в следующем: определяем середину отрезка $[a, b]$

$$\bar{x}=a+\frac{(b-a)}{2}$$

и вычисляем функцию $f(\bar{x})$. Далее делаем выбор, какую из двух частей отрезка взять для дальнейшего уточнения корня. Если левая часть уравнения $f(x)$ является непрерывной функция аргумента x, то корень будет находиться в той половине отрезка, на концах которой $f(x)$ имеет разные знаки. Итерационный (повторяющийся) процесс будем продолжать до тех пор, пока интервал $[a, b]$ не станет меньше заданной погрешности ε.

Решение

Попробуем написать программу, которая может решать с помощью метода дихотомии различные уравнения. Попробуем создать программу, которая основана на использовании абстрактных классов. Создадим первый класс AbstrcatEquation, который содержит абстрактную функцию, представляющую левую часть уравнения и функцию решения уравнения.

Абстрактный класс создается стандартным образом, только диалоговом окне создания класса нужно установить флажок abstract (см. Рис.).

Получится класс такого вида:

```
package classWork5;

public abstract class AbstractEquation {
}
```

Напишем требуемые методы. Класс примет такой вид :

```
package classWork5;
```

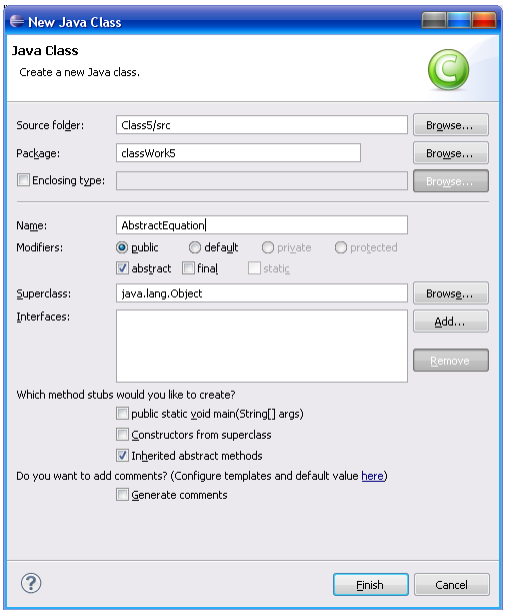
```
public abstract class AbstractEquation {

   abstract double f(double x);

   double solve(double a, double b, double eps) {
      if (f(a)*f(b) > 0) {
         System.out.println("Неверно задан интервал поиска корня");
         System.exit(-1);
      }
      double x = a + 0.5 * (b - a);

      while (Math.abs(b - a) > eps) {
         if (f(a)*f(x) > 0)
            a = x;
         else
            b = x;
         x = a + 0.5 * (b - a);
      }

      return x;
   }
}
```



Теперь можно создать класс-наследник с конкретной функцией f(). Создадим его стандартным образом, включив создание функции main(). Для этого, как и раньше, в поле *Superclass* нужно указать базовый класс (AbstractEquation) (как сделать — см. предыдущее задание). Для удобства работы установим флажок *Inherited abstract methods*. Будет создан класс такого вида:

```
package classWork5;

public class MyEquation extends AbstractEquation {

   @Override
   double f(double x) {
```

```
        // TODO Auto-generated method stub
        return 0;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Напишем текст этих методов. Класс может быть таким:

```
package classWork5;

public class MyEquation extends AbstractEquation {

    @Override
    double f(double x) {
        // TODO Auto-generated method stub
        return x*x + 2 * Math.cos(x + Math.PI/6) - 3 * x + 2;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        AbstractEquation me = new MyEquation();

        java.util.Scanner in = new java.util.Scanner(System.in);
        System.out.println("Интервал локализации корня [a, b]");
        System.out.print("a: ");
        double a = in.nextDouble();
        System.out.print("b: ");
        double b = in.nextDouble();
        double eps;
        do {
            System.out.println("Точность расчетов (0 < eps <= 0.1): ");
            eps = in.nextDouble();
        } while ((eps <= 0) || (eps > 0.1));

        double res = me.solve(a, b, eps);

        System.out.println("Корень x: " + res);
        System.out.println("Abs(f(x): " + Math.abs(me.f(res)));
    }

}
```

Проверить работу программы на тестовых примерах.

Задание №3

Для $t \in [0,5]$ найти численное решение системы обыкновенных дифференциальных уравнений:

$$\begin{cases} x' &= A\,x - B\,x\,y, \\ y' &= C\,x\,y - D\,y. \end{cases}$$

с коэффициентами:

$$A = 2, \qquad B = 0.02, \qquad C = 0.0002, \qquad D = 0.8,$$

удовлетворяющее таким начальным условиям:

1) $x(0) = 3000$, $y(0) = 120$,
2) $x(0) = 5000$, $y(0) = 100$.

Результаты расчетов (значения функций x и y для различных моментов времени t) сохранить в файле. С помощью какой-либо программы построения графиков нарисовать графики решений.

Математические методы

Если задачу об отыскании всех решений дифференциального уравнения удается свести к конечному числу алгебраических операций, операций интегрирования и дифференцирования известных функций, то говорят, что уравнение интегрируется в квадратурах. В реальных приложениях крайне редко встречаются уравнения, интегрируемые в квадратурах. Поэтому для исследования дифференциальных уравнений широко используются приближенные, численные методы их решения.

Пусть дана система обыкновенных дифференциальных уравнений:

$$\begin{cases} \dfrac{\mathrm{d}\,y_1}{\mathrm{d}\,t} &= f_1(t, y_{1,}\ldots, y_i, \ldots, y_N), \\ \ldots \quad \ldots & \qquad \ldots \\ \dfrac{\mathrm{d}\,y_i}{\mathrm{d}\,t} &= f_i(t, y_{1,}\ldots, y_i, \ldots, y_N), \\ \ldots \quad \ldots & \qquad \ldots \\ \dfrac{\mathrm{d}\,y_N}{\mathrm{d}\,t} &= f_N(t, y_{1,}\ldots, y_i, \ldots, y_N). \end{cases}$$

, с начальными условиями

$$\begin{cases} y_1(t_0) &= y_{10}, \\ \ldots \quad \ldots & \quad \ldots \\ y_i(t_0) &= y_{i0}, \\ \ldots \quad \ldots & \quad \ldots \\ y_N(t_0) &= y_{N0}. \end{cases}$$

Численное решение на отрезке $[a, b]$ задачи Коши состоит в построении таблицы приближенных значений функций y_i решения $y(t)$ в узлах сетки $a = t_0 < t_1 < \ldots < t_j < \ldots < t_N = b$.

Простейшим методом построения численного решения для систем обыкновенных дифференциальных уравнений является метод *Эйлера*. Он реализуется такой реккурентной формулой:

$$y_{i(j+1)} = y_{i(j)} + h\,f_i(t_j, y_{1(j)}, \ldots, y_{i(j)}, \ldots, y_{N(j)}),$$

здесь $i = 1, \ldots, N$, h — шаг интегрирования. Этот метод прост, но обладает достаточно большой погрешностью, и имеет систематическое накопление ошибок.

Пожалуй, метод *Рунге – Кутта четвертого порядка* является наиболее распространенным методом решения систем обыкновенных дифференциальных уравнений при постоянном шаге интегрирования h. Его достоинством является высокая точность и меньшая склонность к возникновению неустойчивого решения. Алгоритм реализации метода

Рунге – Кутта четвертого порядка заключается в циклических вычислениях $y_{i(j+1)}$ на каждом $(j+1)$ шаге по формулам:

$$k_{1i}=h f_i(t_j, y_{1(j)},\ldots, y_{i(j)},\ldots, y_{N(j)})$$

$$k_{2i}=h f_i(t_j+h/2, y_{1(j)}+k_{11}/2,\ldots, y_{i(j)}+k_{1i}/2,\ldots, y_{N(j)}+k_{1N}/2)$$

$$k_{3i}=h f_i(t_j+h/2, y_{1(j)}+k_{21}/2,\ldots, y_{i(j)}+k_{2i}/2,\ldots, y_{N(j)}+k_{2N}/2)$$

$$k_{4i}=h f_i(t_j+h, y_{1(j)}+k_{11},\ldots, y_{i(j)}+k_{1i},\ldots, y_{N(j)}+k_{1N})$$

$$y_{i(j+1)}=y_{ij}+(k_{1i}+2k_{2i}+2k_{3i}+k_{4i})/6$$

При переходе от одной формулы к другой задаются или вычисляются соответствующие значения t и y_i и находятся значения функций $f_i(t_j, y_{1(j)},\ldots, y_{i(j)},\ldots, y_{N(j)})$.

Решение

Данную систему обыкновенных дифференциальных уравнений можно решить каким либо численным методом за счет того, что численный алгоритм внедрить в тот класс, который описывает систему. Но это не очень удобный подход, так как либо изменение алгоритма решения уравнения, либо изменение характеристик самой системы дифференциальных уравнений потребует полного или частичного переписывания класса.

Система дифференциальных уравнений (физика) и алгоритм численного решения дифференциальных уравнений — это совсем разные концепции, которые должны рассматриваться как разные классы. Создадим два абстрактных класса, один будет представлять систему обыкновенных дифференциальных уравнений, а другой — «решатель» системы обыкновенных дифференциальных уравнений, в которых будут находится самые общие функции, реализующие функциональность соответствующих объектов.

В созданном для данного занятия про *Java* проекте с именем Theme создаем пакет ode, в котором создадим абстрактные классы классы.

Стандартным образом создадим абстрактный класс ODE, представляющий систему обыкновенных дифференциальных уравнений. (Подробности про создание — предыдущие примеры).

```
package ode;

public abstract class ODE {
   // Получить состояние системы
   abstract public double[] getState();
   // По состоянию системы вычислить производные
   abstract public void getRate(double t, double[] state, double[] rate);
}
```

Аналогично создаем второй абстрактный класс ODESolver, представляющий «решатель» системы обыкновенных дифференциальных уравнений.

```
package ode;

public abstract class ODESolver {
   abstract public void initialize(double stepSize);
   abstract public double step(double t);
   abstract public void setStepSize(double stepSize);
   abstract public double getStepSize();
}
```

В данном пакете создадим класс-наследник класса ODESolver, представляющий «решатель» системы обыкновенных дифференциальных уравнений методом Эйлера. Стандартным образом создаем этот класс и создаем нужные переопределенные методы.

```
package ode;

public class Euler extends ODESolver {

   private double stepSize = 0.1;
   private int numberOfEquations = 0;
   private double[] rate = null;
   private ODE ode;

   public Euler(ODE ode) {
      // TODO Auto-generated constructor stub
      this.ode = ode;
      numberOfEquations = ode.getState().length;
      rate = new double[numberOfEquations];
   }

   @Override
   public void initialize(double stepSize) {
      // TODO Auto-generated method stub
      setStepSize(stepSize);
   }

   @Override
   public double step(double t) {
      // TODO Auto-generated method stub
      double[] state = ode.getState();
      ode.getRate(t, state, rate);
      for (int i = 0; i < numberOfEquations; i++) {
         state[i] = state[i] + stepSize*rate[i];
      }
      return stepSize;
   }

   @Override
   public void setStepSize(double stepSize) {
      // TODO Auto-generated method stub
      this.stepSize = stepSize;
   }

   @Override
   public double getStepSize() {
      // TODO Auto-generated method stub
      return stepSize;
   }

}
```

В созданном для данного занятия про *Java* проекте с именем Theme создаем пакет odeSystem, в котором создадим на основе абстрактного класса ODE класс-наследник, представляющий решаемую систему уравнений. Возможный вид этого класса:

```
package odeSystem;

import ode.ODE;

public class OdeSystem extends ODE {

   private double a, b, c, d;
   private double[] state = null; //state[0] => x[t], state[1] => y[t]

   public OdeSystem(double[] params, double[] state) {
      super();
      this.a = params[0];
      this.b = params[1];
      this.c = params[2];
      this.d = params[3];
      this.state = state;
   }

   public OdeSystem() {
      super();
   }

   public void setParams(double[] params) {
      this.a = params[0];
      this.b = params[1];
      this.c = params[2];
      this.d = params[3];
   }

   public void setState(double[] state) {
      this.state = state;
   }

   @Override
   public double[] getState() {
      // TODO Auto-generated method stub
      return state;
   }

   @Override
   public void getRate(double t, double[] state, double[] rate) {
      // TODO Auto-generated method stub
      rate[0] = a*state[0] - b*state[0]*state[1]; // x'[t] = a*x[t] - b*x[t]*y[t]
      rate[1] = c*state[0]*state[1] - d*state[1]; // y'[t] = c*x[t]*y[t] - d*y[t]
   }

}
```

Затем в этом пакете создадим класс-приложение, т. е. запускающий класс для этого приложения.

```
package odeSystem;

import java.io.*;
import ode.*;
```

```
public class OdeSystApp {

   /**
    * @param args
    */
   public static void main(String[] args) throws IOException {
      // TODO Auto-generated method stub
      OdeSystem syst = new OdeSystem();
      syst.setParams(new double[] {2.0, 0.02, 0.0002, 0.8});
      syst.setState(new double[] {3000, 120});
      ODESolver odeSolver = new Euler(syst);
      //ODESolver odeSolver = new RK4(syst);
      PrintWriter out = new PrintWriter(new FileWriter("Result.dat"));

      double t = 0, tEnd = 5.0, tStep = 0.001;
      System.out.println("t: " + t + "  x: " + syst.getState()[0] +
            "  y: " + syst.getState()[1]);
      out.printf("%14.6e%14.6e%14.6e\n", t,
                  syst.getState()[0], syst.getState()[1]);

      odeSolver.setStepSize(tStep);
      while (t <= tEnd) {
         odeSolver.step(t);
         t += tStep;
         System.out.println("t: " + t + "  x: " + syst.getState()[0] +
               "  y: " + syst.getState()[1]);
         out.printf("%14.6e%14.6e%14.6e\n", t,
                     syst.getState()[0], syst.getState()[1]);
      }
      out.close();
   }

}
```

В данном классе нужно особо обратить внимание на выделенные жирным шрифтом строки — реализацию записи текстовой информации в файл.

С помощью этой программы следует решить систему уравнений с разной величиной шага: 0.1 , 0.01 , 0.001 , … и т. д. По полученным файлам данных построить графики и посмотреть, как меняется вид решения в зависимости от величины шага. Подобрать такой шаг, чтобы можно было получить достаточно точное решение.

В пакете ode реализовать еще один метод численного решения системы обыкновенных дифференциальных уравнений — RK4, представляющий реализацию метода Рунге – Кутта 4 порядка с постоянным шагом. С его помощью получить решение данной системы. Для реализованных методов сравнить величину шага, при котором получено «достаточно точное» решение.

Построение графиков таблично заданных функций

Для построения графиков функций можно использовать программу *GnuPlot*. Данная программа относится к фонду бесплатного программного обеспечения и реализована для разных операционных систем и типов компьютеров. Последние версии программы можно свободно получить из *Интернет* с сайта разработчика.

С программой обычно работают в одном из двух режимов:

- *интерактивном* — пользователь вводит команды построения и оформления графиков в командной строке программы (эти команды можно выбрать либо из соответствующих пунктов меню главного окна программы, либо с помощью панелей инструментов);
- *пакетном* — пользователь готовит специальный пакетный файл с командами, который передается как параметр программе в момент ее вызова. При этом программа работает в фоновом режиме, а построение, оформление и экспорт графика в нужный формат происходит автоматически.

Программа поставляется с хорошей справочной системой, где указана ВСЯ необходимая для работы информация. Кратко рассмотрим основы работы с программой.

После запуска программы в главном окне нужно указать рабочую папку и загрузить пакетный файл с командами. Для этого следует выбрать пункт *File* | *Change directory* главного окна программы или нажать кнопку *ChDir* панели инструментов. На экран будет выведено стандарное диалоговое окно выбора папки. После выбора нужной папки (в ней должны находиться файл с данными, по которым нужно построить график, и пакетный файл с командами *GnuPlot*) и подтвержения выбора в строке ввода программы будет введена и выполнена нужная команда перехода (см. **Рис**).

После этого следует загрузить командный файл *GnuPlot* (текстовый файл с нужными командами). Обычно, командные файлы *GnuPlot* имеющий расширение имени файла **`PLT`**. Для загрузки и выполнения нужного командного файла следует либо выбрать пункт меню *File* | *Open* главного окна программы, либо нажать кнопку *Open* панели задач. На экран будет выведено стандартное окно выбора файла, в котором нужно указать требуемый пакетный файл. После подтвержения выбора программа *GnuPlot* выполнит все команды, находящиеся в файле. При этом в строке ввода главного окна программы будет указана выполненная команда загрузки пакетного файла.

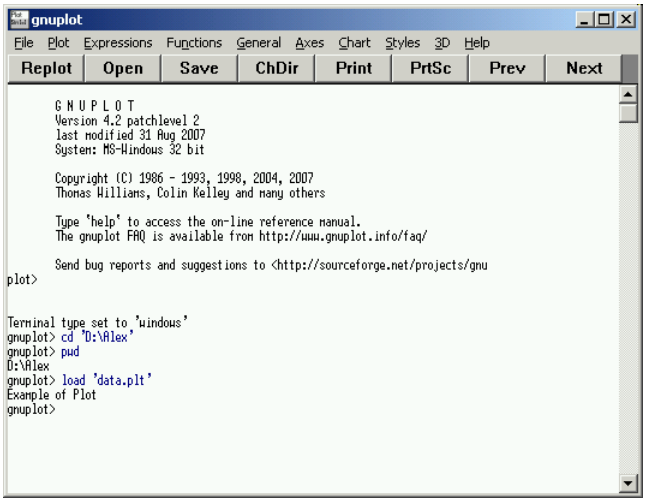


Рисунок 3.1. Главное окно программы *GnuPlot*

Пример пакетного файла для построения двумерного графика по набору точек, хранящихся в файле приведен ниже. В пакетном файле указаны краткие комментарии к

основным командам. При необходимости все о командах можно узнать в справочной системе программы.

```
print "Example of Plot" # вывод текста в окно редактора
reset
set nogrid  # без сетки на графике
set nokey   # без автоматической легенды
set xrange [0.5:5.5]  # диапазон для x на графике
set yrange [0.0:1.7]  # диапазон для y на графике
set tics out          #  штрихи у осей наружу
set xtics border nomirror norotate 0.0, 1, 5.5  # цифры у оси x
set ytics border nomirror norotate 0.0, 0.5, 1.6 # цифры у оси y
# количество разбиений большого интервала по оси x
set mxtics 10
# количество разбиений большого интервала по оси y
set mytics 5
# текстова метка на графике
# (это одна строка файла. ↵ – символ продолжения строки)
set label 1 "Title" at 0.7, 1.55 left norotate font ↵ "Arial
Italic,24"
# построение графика в окне
plot 'Data1.dat' using 1:2 with lines linetype 1 linewidth 3
pause -1 "Press OK to make emf file or Exit to stop..."
# экспорт графика в emf формат
set terminal emf color solid linewidth 5 "Times Italic" 24
set output 'Fig1.emf'
replot
# восстановление выводного устройства
set terminal windows
set output
pause -1 "Press any key to continue..."
reset
```