Практические занятия в компьютерном класса

Занятие №5 «Интерфейсы»

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

Задание

Создать набор классов для решения задачи численного дифференцирования функции одной переменной. Реализовать программу так, что бы можно было численно дифференцировать с заданной точностью функции одной переменной различного вида. Для тестирования программы вычислить на отрезке $x \in [1.5, 6.5]$ с шагом 0.05 производные нескольких функций. Для тестирования программы использовать такие функции:

- первая функция: $f(x) = \exp(-a x^2) \cdot \sin(x)$ для заданного значения a ($a = 0.5$);
- вторая функция: значения независимой переменной — это значение параметра a, принадлежащее отрезку $a \in [1.0, 7.0]$ и изменяющееся с шагом 0.1, значение функции — решение уравнения $\operatorname{sech}(x)^2 = a \cdot x$ для заданного значения a.
- третья функция: значения независимой переменной и функции сохранены в текстовом файле данных.

Результаты дифференцирования сохранить в текстовых файлах данных.

Математические методы

Вычисление производной

Для вычисления производной воспользоваться центральной конечно – разностной формулой численного дифференцирования по трем точкам:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

Здесь x_i — точка, в которой требуется найти производную, $x_{i\pm1} = x_i \pm h$, h — шаг, с которым вычислена производная.

Для того, чтобы оценить то значение h, для которого производная вычислена с заданной точностью ε, следует вычислить последовательность $D_n = f'(x_i)$ со все уменьшающимися значениями шага h $(h_n = 0.1 \cdot h_{n-1})$. Вычисление последовательности следует проводить до тех пор, пока не будет выполнено хотя бы одно неравенство:

$$|D_{n+1} - D_n| \geqslant |D_n - D_{n-1}| \text{ или } |D_n - D_{n-1}| < \varepsilon .$$

Для того, чтобы вычислить производную таблично заданной функции, построим интерполяционный полином, который продифференцируем по этому правилу.

Построение интерполяционного полинома

Для вычисления значения функции, заданной таблично в точках между узлами таблицы можно воспользоваться методами интерполяции.

Задача интерполяции часто формулируется так: пусть функция $y = f(x)$ известна в $N+1$ точке (x_0, y_0), (x_1, y_1), (x_n, y_n), где $y_k = f(x_k)$, а значения x_k принадлежат

интервалу $[a,b]$ и удовлетворяют условиям $a \leqslant x_0 < x_1 < \ldots < x_N \leqslant b$. Требуется вычислить значение функции $f(x)$ на интервале $[a,b]$ в промежутках между точками табуляции.

Если заданные точки (x_k, y_k) известны с высокой степенью точности, то можно построить интерполяционный полином $P(x)$, который проходит через них. Когда вычисляется значение интерполяционного полинома $P(x)$ в точке $x \in [x_{0,} x_N]$, то приближение $P(x)$ называется *значением интерполяции*.

Существует много способов построения интерполяционного полинома $P(x)$. Рассмотрим **интерполяционный полином Лагранжа**.

Французский математик *Жозеф Луи Лагранж* (1736 – 1813) предложил использовать для нахождения интерполяционного полинома $P(x)$ следующий метод. Полином, проходящий через $N+1$ точку $(x_{0,} y_0)$, $(x_{1,} y_1)$, … , (x_n, y_n), степени не большей, чем N может быть записан в виде:

$$P_N(x) = \sum_{k=0}^{N} y_k L_{N,k}(x),$$

где $L_{N,k}(x)$ — коэффициенты полинома Лагранжа, основанного на этих узлах. Выражения для коэффициентов $L_{N,k}(x)$ могут быть записаны в виде:

$$L_{N,k}(x) = \frac{(x-x_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_N)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_N)} = \frac{\prod_{j=0, j\neq k}^{N}(x-x_j)}{\prod_{j=0, j\neq k}^{N}(x_k-x_j)}$$

Для каждого фиксированного k коэффициенты полинома Лагранжа $L_{N,k}(x)$ обладают свойствами:

$$L_{N,k}(x) = 1, \quad \text{когда } j = k,$$
$$L_{N,k}(x) = 0, \quad \text{когда } j \neq k.$$

Ранее, задачу вычисления интерполирующих функций было принято разделять на два этапа. На первом этапе вычислялись коэффициенты интерполирующей функции. На втором этапе эти коэффициенты использовались для вычисления полинома. В настоящий момент обычно коэффициенты вычисляют на каждом шаге вычисления полинома.

Решение уравнения

Поиск корней уравнения вида:

$$f(x) = 0$$

выполняется в два этапа. На первом этапе осуществляется *локализация корня*, т.е. определяется такой отрезок $[a,b]$, на котором исследуемая функция $f(x)$ имеет строго один корень. Этот отрезок называют *отрезком локализации* корня $\bar{x}$. На втором этапе выполняется уточнение расположения корня на отрезке локализации.

Для уточнения расположения корня воспользуемся методом секущих. Метод секущих можно рассматривать как модификацию метода Ньютона, в котором производная заменена конечно – разностным приближением. Этот метод является «двухшаговым» – для вычисления нового приближения к корню $x^{(k+1)}$ нужно знать два предыдущих приближения $x^{(k)}$, $x^{(k-1)}$. В данном методе для новое приближение к корню вычисляется по формуле:

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k-1)} - x^{(k)}}{f\left(x^{(k-1)}\right) - f\left(x^{(k)}\right)} f\left(x^{(k)}\right).$$

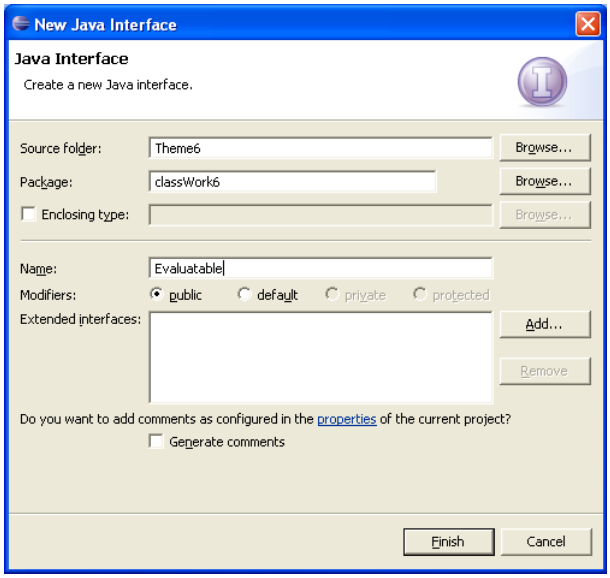
При вычислении корня уравнения будем считать, что корень найден с заданной точностью, если выполняется неравенство $|x^{(k+1)}-x^{(k)}|<\varepsilon$, где $x^{(k+1)}, x^{(k)}$ — два последовательных приближения к корню, ε — требуемая точность расчетов.

Решение

Для того, чтобы можно было дифференцировать любые функции одной переменной удобно объявить интерфейс — возможность вычислить результат, получив один аргумент. И именно переменная такого типа будет использоваться в качестве соответствующего параметра функции численного дифференцирования. Т.е. объект любой природы, реализующий указанный интерфейс может быть продифференцирован.

Как обычно, загружаем интегрированную среду **Eclipse**, закрываем свои старые проекты и создаем новый *Java* проект с именем Theme6. В данном проекте создадим пакет classWork6, в котором будем размещать классы приложения.

В начале создадим интерфейс Evaluatable, который будет гарантировать, что объект класса, который этот интерфейс реализует может вычислять значение по одному аргументу. Для того, чтобы создать такой интерфейс вызовем команду меню *File | New | Interface*. На экран будет выведено диалоговое окно создания нового интерфейса.



Данное диалоговое окно очень похоже на окно создания нового класса. В поле *Name* нужно указать название интерфейса и нажать кнопку *Finish*. Будет создан и добавлен в проект файл с именем **Evaluatable.java**, содержащий заготовку интерфейса:

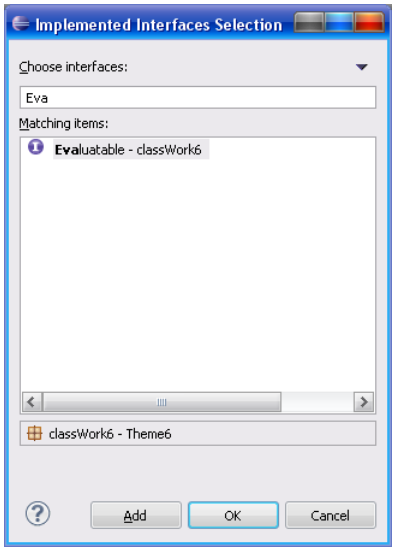
```
package classWork6;

public interface Evaluatable {
}
```

Добавим к нему метод, который нужно будет определить во всех классах, реализующих данный интерфейс. В результате интерфейс примет такой вид:

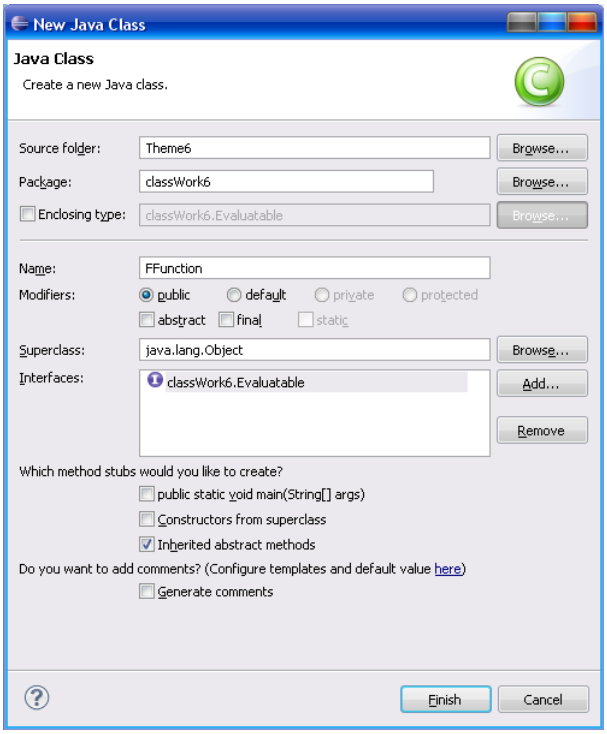
```
package classWork6;

public interface Evaluatable {
   double evalf(double x);
}
```

Далее создадим класс FFunction, который реализует данный интерфейс и

предназначен для вычисления первой функции $f(x)=\exp(-ax^2)\cdot\sin(x)$. Для этого выбираем команду меню *File | New | Class*, и в появившемся на экране диалоговом окне указываем имя класса (поле *Name*) и указываем интерфейсы, которые должны быть реализованы. Для этого нажимаем кнопку *Add*, расположенную правее списка *Interfaces*. На экран будет выведено диалоговое окно *Implemented Interfaces Selection*, предназначенное для выбора нужных интерфейсов.



В поле *Choose interfaces* начнем указывать нужный интерфейс. По мере указания в поле *Matching items* отображаются подходящие по названию интерфейсы. Найдя нужный, можно его выбрать мышью и нажать кнопку *OK*. Интерфейс будет выбран, и указан в списке *Interfaces* диалогового окна *New Java Class*.



Далее для создания заготовок функций нужно выбрать флажковое поле *Inherited abstract methods*, и запретить автоматически создавать функцию `main()`. После указания таких настроек следует нажать кнопку *Finish*.

В проект будет добавлен файл **FFunction.java**, содержащий заготовку класса,

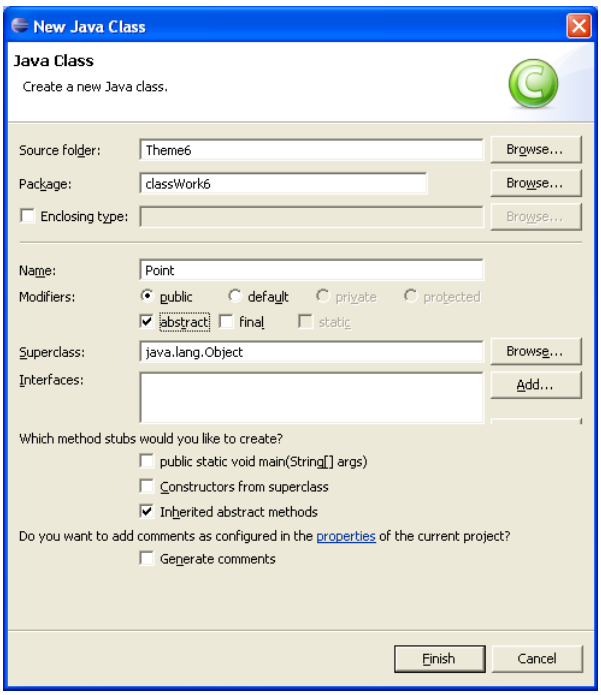
реализующего заданный интерфейс:

```
package classWork6;

public class FFunction implements Evaluatable {

   public double evalf(double x) {
      // TODO Auto-generated method stub
      return 0;
   }

}
```

Для реализации нужной функциональности добавляем закрытое поле для хранения значения параметра a, автоматически генерируем два конструктора и подправляем их код, а также реализовываем метод вычисления evalf(). В результате получится примерно такой код:

```
package classWork6;

public class FFunction implements Evaluatable {

   private double a;

   public FFunction(double a) {
      // TODO Auto-generated constructor stub
      this.a = a;
   }

   public FFunction() {
      // TODO Auto-generated constructor stub
      this(1.0);
   }

   public double evalf(double x) {
      // TODO Auto-generated method stub
      return Math.exp(-a*x*x)*Math.sin(x);
   }

}
```

Для того, чтобы реализовать вычисления оставшихся двух функций создадим два дополнительных класса. Первый класс — абстрактный класс Point. Данный класс содержит базовые возможности по представлению точки в n –мерном пространстве. В качестве полей этот класс содержит массив координат точки, имеет один конструктор и два метода доступа. Для создания класса выполняем команду меню *File | New | Class* и в появившемся на экране диалоговом окне *New Java Class* указываем такие, как на рисунке настройки.



В проект будет добавлен файл **Point.java**, содержащий заготовку создаваемого абстрактного класса:

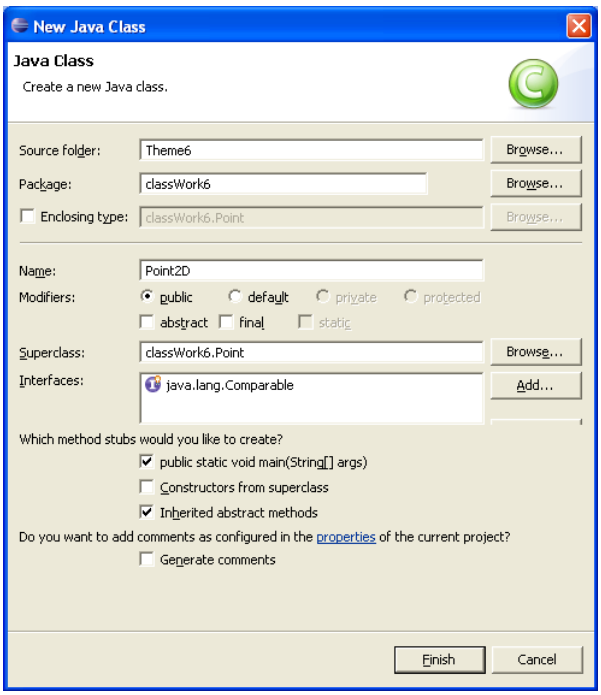
```
package classWork6;

public abstract class Point {
}
```

В данную заготовку добавляем поле — массив координат, добавляем конструктор с одним параметром — размерностью пространства и создаем методы доступа. Нужно обратить внимание на то, что в методах доступа указан не индекс, а номер точки (индекс на единицу меньше номера). Далее с помощью пункта меню *Override/Implement Methods* вызываем на экран одноименное диалоговое окно, в котором для переопределения выбираем метод toString(). В тело класса будет вставлен шаблон метода с аннотацией. В этом методе следует изменить то, что было сгенерировано по умолчанию на код, который создает такое строковое представление точки: (x, y). В результате получится класс вида:

```
package classWork6;

public abstract class Point {

   private double[] coords = null;

   public Point(int num) {
      this.coords = new double[num];
   }

   public void setCoord(int num, double x) {
      coords[num-1] = x;
   }

   public double getCoord(int num) {
      return coords[num-1];
   }
```

```
    @Override
    public String toString() {
        // TODO Auto-generated method stub
        String res = "(";
        for (double x : coords) {
            res += x + ", ";
        }
        return res.substring(0, res.length()-2) + ")";
    }

}
```

Следует обратить внимание: несмотря на то, что в классе нет абстрактных методов класс объявлен абстрактным. Так можно поступать тогда, когда мы хотим быть уверенными, что никто и никогда не сможет создать объект этого класса. Для того, чтобы было удобно работать с точкой в двумерном пространстве создадим класс Point2D — производный класс от класса Point. Данный класс будет не только производным классом от класса Point, но и будет реализовывать интерфейс Comparable, для того, чтобы можно было сортировать коллекции точек. Эти требования стандартным образом указываются в диалоговом окне создания нового класса:



Будет создан шаблон класса такого вида:

```
package classWork6;

public class Point2D extends Point implements Comparable {

    public int compareTo(Object arg0) {
        // TODO Auto-generated method stub
        return 0;
    }

    /**
```

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

    }

}
```

В созданный класс стандартным образом добавляем два конструктора (один создает объект точку из пары чисел, а второй — конструктор по умолчанию), удобные методы доступа, реализуем функцию сравнения compareTo и создаем функцию main(), в которой проверяем функциональность созданного класса. Класс Point2D может получиться примерно таким:

```
package classWork6;

public class Point2D extends Point implements Comparable <Point2D> {

    public Point2D(double x, double y) {
        super(2);
        setCoord(1, x);   setCoord(2, y);
    }

    public Point2D() {
        this(0, 0);
    }

    public double getX() {
        return getCoord(1);
    }

    public void setX(double x) {
        setCoord(1, x);
    }

    public double getY() {
        return getCoord(2);
    }

    public void setY(double y) {
        setCoord(2, y);
    }

    public int compareTo(Point2D pt) {
        // TODO Auto-generated method stub
        return Double.compare(getX(), pt.getX());
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        java.util.List<Point2D> data = new java.util.ArrayList<Point2D>();
```

```
        int num;
        double x;

        java.util.Scanner in = new java.util.Scanner(System.in);
        do {
           System.out.print("Количество точек: ");
           num = in.nextInt();
        } while (num <= 0);

        for (int i = 0; i < num; i++)
        {
           x = 1.0 + (5.0 - 1.0)*Math.random();
           data.add(new Point2D(x, Math.sin(x)));
        }

        System.out.println("Несортированные данные: ");
        for (Point2D pt : data)
           System.out.println(pt);

        System.out.println("\nОтсортированные данные: ");
        java.util.Collections.sort(data);
        for (Point2D pt : data)
           System.out.println("x = " + pt.getX() + "\ty = " + pt.getY());
     }

}
```

Далее, создадим абстрактный класс Interpolator, в котором опишем которые должна выполнять функция интерполяции данных, вне зависимости от способа их хранения и представления. Кроме того, чтобы эти объекты можно было дифференцировать, этот класс должен реализовывать интерфейс Evaluatable.

Стандартным образом создаем класс и указываем требования к нему. Получаем заготовку класса, в которую добавляем абстрактные методы, с помощью которых реализуем метод evalf() (вычисление интерполяционного полинома Лагранжа по набору заданных точек для указанного аргумента). Класс примет примерно такой вид:

```
package classWork6;

public abstract class Interpolator implements Evaluatable {

   abstract public void clear();
   abstract public int numPoints();
   abstract public void addPoint(Point2D pt);
   abstract public Point2D getPoint(int i);
   abstract public void setPoint(int i, Point2D pt);
   abstract public void removeLastPoint();
   abstract public void sort();

   public double evalf(double x) {
      // TODO Auto-generated method stub
      double res = 0.0;
      int numData = numPoints();
      double numer, denom;

      for (int k = 0; k < numData; k++) {
```

```
            numer = 1.0;
            denom = 1.0;
            for (int j = 0; j < numData; j++) {
               if (j != k) {
                  numer = numer * (x - getPoint(j).getX());
                  denom = denom * (getPoint(k).getX() - getPoint(j).getX());
               }
            }
            res = res + getPoint(k).getY()*numer/denom;
         }

         return res;
      }

}
```

Для представления второй функции для дифференцирования создадим класс ListInterpolation, наследник класса Interpolator. В этом классе точки, по которым будет строится интерполяционный полином будут храниться в списочном массиве. Стандартным образом автоматически создаем заготовку данного класса. Добавляем и исправляем автоматически сгенерированный код конструкторов. Для удобства работы создадим три конструктора — конструктор по умолчанию, а также конструкторы, которые создают объект этого класса по массиву и по списочному массиву точек. Далее реализуем все абстрактные методы базового класса и реализовываем метод main(), в котором проверяем все возможности разработанного класса. Класс может иметь такой вид:

```
package classWork6;

import java.util.*;

public class ListInterpolation extends Interpolator {

private List<Point2D> data = null;

   public ListInterpolation(List<Point2D> data) {
      // TODO Auto-generated constructor stub
      this.data = data;
   }

   public ListInterpolation() {
      // TODO Auto-generated constructor stub
      data = new ArrayList<Point2D>();
   }

   public ListInterpolation(Point2D[] data) {
      // TODO Auto-generated constructor stub
      this();
      for (Point2D pt : data)
         this.data.add(pt);
   }


   @Override
   public void clear() {
      // TODO Auto-generated method stub
```

```
    data.clear();
}

@Override
public int numPoints() {
    // TODO Auto-generated method stub
    return data.size();
}

@Override
public void addPoint(Point2D pt) {
    // TODO Auto-generated method stub
    data.add(pt);
}

@Override
public Point2D getPoint(int i) {
    // TODO Auto-generated method stub
    return data.get(i);
}

@Override
public void setPoint(int i, Point2D pt) {
    // TODO Auto-generated method stub
    data.set(i, pt);
}

@Override
public void removeLastPoint() {
    // TODO Auto-generated method stub
    data.remove(data.size()-1);
}

@Override
public void sort() {
    // TODO Auto-generated method stub
    java.util.Collections.sort(data);
}

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

    int num;
    double x;
    java.util.Scanner in = new java.util.Scanner(System.in);

    do {
        System.out.print("Количество точек: ");
        num = in.nextInt();
```

```
      } while (num <= 0);

      for (int i = 0; i < num; i++)
      {
         x = 1.0 + (5.0 - 1.0)*Math.random();
         fun.addPoint(new Point2D(x, Math.sin(x)));
      }
      System.out.println("Интерполяция по: " + fun.numPoints() + " точкам");
      System.out.println("Несортированный набор: ");
      for (int i = 0; i < fun.numPoints(); i++)
         System.out.println("Точка " + (i+1) + ": " + fun.getPoint(i));

      fun.sort();
      System.out.println("Отсортированный набор: ");
      for (int i = 0; i < fun.numPoints(); i++)
         System.out.println("Точка " + (i+1) + ": " + fun.getPoint(i));

      System.out.println("Минимальное значение x: " + fun.getPoint(0).getX());
      System.out.println("Максимальное значение x: " +
                         fun.getPoint(fun.numPoints()-1).getX());

      x = 0.5*(fun.getPoint(0).getX() + fun.getPoint(fun.numPoints()-1).getX());
      System.out.println("Значение интеролляции fun(" + x + ") = " + fun.evalf(x));
      System.out.println("Точное значение sin(" + x + ") = " + Math.sin(x));
      System.out.println("Абсолютная ошибка = " + Math.abs(fun.evalf(x)-Math.sin(x)));
   }

}
```

В метод main() следует **добавить проверку непроверенных методов**. **Проверить, как зависит ошибка интерполяции от количества используемых при интерполяции точек**.

Для представления третьей функции удобно создать класс – наследник класса ListInterpolation, дополнив его методами чтения информации из файла и записи информации в файл. В данном классе автоматически создаем конструктор по умолчанию, добавляем методы считывания информации о точках и записи такой информации в файл. Также добавляем метод main(), с помощью которого тестируем класс. Кроме того, в методе main() подготовим файл с данными для работы. Сохраним их в файле **TblFunc.dat**. В результате получится класс вида:

```
package classWork6;

import java.io.*;
import java.util.*;

public class FileListInterpolation extends ListInterpolation {

   public FileListInterpolation() {
      super();
      // TODO Auto-generated constructor stub
   }

   public void readFromFile(String fileName) throws IOException {
      BufferedReader in = new BufferedReader(new FileReader(fileName));
      String s = in.readLine();
```

```
    clear();
    while ((s = in.readLine()) != null) {
        StringTokenizer st = new StringTokenizer(s);
        double x = Double.parseDouble(st.nextToken());
        double y = Double.parseDouble(st.nextToken());
        addPoint(new Point2D(x, y));
    }
    in.close();
}

public void writeToFile(String fileName) throws IOException {
    PrintWriter out = new PrintWriter(new FileWriter(fileName));
    out.printf("%9s%25s\n", "x", "y");
    for (int i = 0; i < numPoints(); i++) {
        out.println(getPoint(i).getX() + "\t" + getPoint(i).getY());
    }
    out.close();
}

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

    int num;
    double x;
    java.util.Scanner in = new java.util.Scanner(System.in);

    do {
        System.out.print("Количество точек: ");
        num = in.nextInt();
    } while (num <= 0);

    for (int i = 0; i < num; i++) {
        x = 1.0 + (5.0 - 1.0)*Math.random();
        fun.addPoint(new Point2D(x, Math.sin(x)));
    }
    System.out.println("Интерполяция по: " + fun.numPoints() + " точкам");
    System.out.println("Несортированный набор: ");
    for (int i = 0; i < fun.numPoints(); i++)
        System.out.println("Точка " + (i+1) + ": " + fun.getPoint(i));

    fun.sort();
    System.out.println("Отсортированный набор: ");
    for (int i = 0; i < fun.numPoints(); i++)
        System.out.println("Точка " + (i+1) + ": " + fun.getPoint(i));

    System.out.println("Минимальное значение x: " + fun.getPoint(0).getX());
    System.out.println("Максимальное значение x: " +
                        fun.getPoint(fun.numPoints()-1).getX());

    System.out.println("Сохраняем в файл");
```

```
        try {
            fun.writeToFile("data.dat");
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }

        System.out.println("Считываем из файла");
        fun.clear();
        try {
            fun.readFromFile("data.dat");
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }

        System.out.println("Данные из файла: ");
        fun.sort();
        for (int i = 0; i < fun.numPoints(); i++)
            System.out.println("Точка " + (i+1) + ": " + fun.getPoint(i));

        System.out.println("Минимальное значение x: " + fun.getPoint(0).getX());
        System.out.println("Максимальное значение x: " +
                           fun.getPoint(fun.numPoints()-1).getX());
        x = 0.5*(fun.getPoint(0).getX() + fun.getPoint(fun.numPoints()-1).getX());
        System.out.println("Значение интеролляции fun(" + x + ") = " + fun.evalf(x));
        System.out.println("Точное значение sin(" + x + ") = " + Math.sin(x));
        System.out.println("Абсолютная ошибка = " + Math.abs(fun.evalf(x)-Math.sin(x)));

        System.out.println("Готовим данные для счета");
        fun.clear();
        for (x = 1.0; x <= 7.0; x += 0.1) {
            fun.addPoint(new Point2D(x, Math.sin(x)));
        }
        try {
            fun.writeToFile("TblFunc.dat");
        }
        catch (IOException ex) {
            ex.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Таким образом, третья функция для дифференцирования полностью определена. Для того, чтобы можно было полностью реализовать вторую функцию, нужно в проект добавить еще два класса. Один класс — левая часть уравнения, которое необходимо решить $\left(\operatorname{sech}(x)^2 = a \cdot x\right)$. Данный класс создается примерно так же, как и класс с первой функцией для дифференцирования. Класс должен реализовывать интерфейс Evaluatable и будет содержать одно закрытое поле (значение параметра a), два конструктора и определенный метод evalf(). Класс может быть примерно таким:

```
package classWork6;

public class LeftHand implements Evaluatable {

private double a;

   public LeftHand(double a) {
      this.a = a;
   }

   public LeftHand() {
      this(0);
   }

   public double evalf(double x) {
      // TODO Auto-generated method stub
      return 1.0/Math.pow(Math.cosh(x), 2) - a*x;
   }

}
```

Для решения задания понадобится еще и класс, в котором будут реализованы численные методы: решение нелинейного уравнения и метод дифференцирования функции одной переменной. Для удобства работы эти методы будут реализованы как статические, т. к. для их работы будет не нужно получать доступ к полям класса. Для того, что бы никто не мог создать экземпляров этого класса (для доступа к статическим методам объекты класса не нужны) создадим один закрытый конструктор по умолчанию. В класс добавим метод main(), в котором проверим работу методов на тестовых примерах. Класс может быть примерно таким:

```
package classWork6;

public class numMethods {

   private numMethods() {
      // TODO Auto-generated constructor stub
   }

   private static double meth(double x, double h, Evaluatable f) {
      return 0.5*(f.evalf(x+h) - f.evalf(x-h))/h;
   }

   public static double der(double x, double tol, Evaluatable f) {
      final int MAX = 100;
      double h = 0.1;
      double one = meth(x, h, f);
      h = 0.1*h;
      double two = meth(x, h, f);
      int i = 0;
      double tmp;
      boolean ok;
      do {
         h = 0.1*h;
         tmp = meth(x, h, f);
         ok = ( Math.abs(tmp-two) >= Math.abs(two-one) ) ||
```

```
            ( Math.abs(two-one) < tol );
        if (i > MAX) {
            System.out.print("Слишком много шагов вычислений");
            System.exit(-1);
        }
        i += 1;
        one = two;
        two = tmp;
    } while (!ok);

    return two;

}

public static double findRoot(double appr, double eps, Evaluatable f) {
    final int MAX_ITER = 100;
    int k = 0;
    double delta = 1.0e-1*appr;
    double old1 = appr, old2 = appr + delta;
    double res, error;
    do {
        k += 1;
        res = old2 - f.evalf(old2) * (old1 - old2) /
            ( f.evalf(old1) - f.evalf(old2) );
        error = Math.abs(res - old2);
        old1 = old2;
        old2 = res;
        if (k > MAX_ITER) {
            System.out.print("Слишком много шагов вычислений");
            System.exit(-1);
        }
    } while (error >= eps);

    return res;
}

/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    // Проверка метода решения уравнения
    double resEq1, resEq2;

    resEq1 = findRoot(1.0, 1.0e-7,
            new Evaluatable() {
                public double evalf(double x) {return x*x - 4;}
            }
    );

    resEq2 = findRoot(-1.0, 1.0e-7,
            new Evaluatable() {
                public double evalf(double x) {return x*x - 4;}
            }
```

```
        );

        System.out.println("Первый корень: "  + resEq1 + "\nВторой корень: " + resEq2);


        // Проверка метода дифференцирования
        ListInterpolation fun = new ListInterpolation();

        int num;
        double x = -0.5*Math.PI;
        double step = 0.1;
        java.util.Scanner in = new java.util.Scanner(System.in);


        do {
           System.out.print("Количество точек: ");
           num = in.nextInt();
        } while (num <= 0);

        for (int i = 0; i < num; i++)
        {
           x += step;
           fun.addPoint(new Point2D(x, Math.sin(x)));
        }

        x = 0.5*(fun.getPoint(0).getX() + fun.getPoint(fun.numPoints()-1).getX());
        double res = numMethods.der(x, 1.0e-5, fun);
        System.out.println("Численное  значение sin'(" + x + ") = " + res);
        System.out.println("Символьное значение sin'(" + x + ") = " + Math.cos(x));
        System.out.println("Абсолютная ошибка = " + Math.abs(res-Math.cos(x)));
     }

}
```

Нужно обратить внимание на ту часть метода main(), в которой выполняется проверка метода решения уравнения. При вызове метода findRoot использован анонимный вложенный класс. Его удобно применять, когда нужен только один объект данного класса в одном месте программы. Приведенная в вызове метода findRoot запись означает, что создается новый безымянный объект класса, реализующего интерфейс Evaluatable, в котором реализован требуемый метод evalf(). Подробнее об этом поговорим на лекции.

Для завершения программы нужно создать класс, в котором будет выполнено основное наше задание. Этот класс будет содержать единственный метод main(), который организует требуемую работу. Класс может быть примерно таким:

```
package classWork6;

import java.io.*;

public class DerivativeApplication {

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

```
        Evaluatable functs[] = new Evaluatable[3];
        functs[0] = new FFunction(0.5);
        functs[1] = new ListInterpolation();
        functs[2] = new FileListInterpolation();

        double root = 0.6;
        for (double a = 1.0; a <= 7.0; a+= 0.1) {
           root = numMethods.findRoot(root, 1.0e-6, new LeftHand(a));
           ((ListInterpolation)functs[1]).addPoint(new Point2D(a, root));
        }

        try {
           ((FileListInterpolation)functs[2]).readFromFile("TblFunc.dat");
        }
        catch (IOException ex) {
           ex.printStackTrace();
           System.exit(-1);
        }

        String fileName = "";
        for (Evaluatable f: functs) {
           System.out.println("Функция: " + f.getClass().getSimpleName());
           fileName = f.getClass().getSimpleName() + ".dat";
           PrintWriter out = new PrintWriter(new FileWriter(fileName));
           for (double x = 1.5; x <= 6.5; x += 0.05) {
              System.out.println("x: " + x + "\tf: " + f.evalf(x)
                                 + "\tf': " + numMethods.der(x, 1.0e-4, f));
              out.printf("%16.6e%16.6e%16.6e\n", x, f.evalf(x),
                         numMethods.der(x, 1.0e-4, f));
           }
           System.out.println("\n");
           out.close();
        }
     }
}
```

Протестировать работу приложения. Построить графики.