Министерство образования и науки Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра высокопроизводительных вычислений и
параллельного программирования

Технологии обработки информации на языках высокого уровня

Учебно-методическое пособие

Ижевск
2018

УДК 004.382.2-027.1(07)+519.6(07)
ББК 32.973.202я7+22.193я7
Т384

Рекомендовано к изданию
учебно-методическим советом УдГУ

Составитель: М.А. Клочков
Рецензент: д.ф.-м.н., профессор А.П. Бельтюков

Технологии обработки информации на языках высокого уровня: учеб.-метод. пособие / Т384 Сост. М.А. Клочков. Ижевск: Изд-во «Удмуртский университет», 2018. – 40 с.

ISBN 978-5-4312-0572-9

Учебно-методическое пособие содержит примеры и задания в виде программ, написанных с использованием высокоуровневых языков Java и Prolog. Решая типовые задачи, исправляя ошибки в исходном коде, студент получает повышенный уровень компетенций, связанных с технологиями обработки знаний и данных.

Пособие предназначено для использования в рамках дисциплин, связанных с изучением информационных технологий и программирования, такими, как: «Технологии обработки информации», «Практикум по программированию», «Высокоуровневые языки программирования».

УДК 004.382.2-027.1(07)+519.6(07)
ББК 32.973.202я7+22.193я7

ISBN 978-5-4312-0572-9

ПРЕДИСЛОВИЕ

В предлагаемом пособии систематизирован опыт преподавания учебных дисциплин, в которых изучается программирование на языках высокого уровня, информационные технологии обработки данных, интеллектуальные информационные системы и способы и формы представления знаний и данных.

Для написания исходного кода выбраны два языка программирования – это Java и Prolog, первый принадлежит семейству объектно-ориентированных языков, второй принадлежит семейству языков логического типа. Java сочетает в себе удобство и простоту изучения с мощностью и гибкостью используемой иерархии классов, а также кроссплатформенность, включающую возможность создавать мобильные приложения для гаджетов. Prolog позволяет искать решение задачи, основанное на имеющихся фактах и правилах, избегая излишней привязки к структурам данных и способам их представления. Таким образом, правильно поставив задачу, имеется возможность почти сразу получить ее решение.

В методическом аспекте преподавания программирования необходимо отметить непревзойденную гибкость администрирования и эффективность организации многопользовательского режима работы, предоставляемых операционной системой Linux Debian. В дистрибутив входят установочные пакеты OpenJDK и prolog, которые содержат основные программные инструменты для работы с выбранными языками программирования. Преподаватель имеет незаменимый инструмент для организации проведения практических занятий с любым количеством обучающихся, которым для минимальной работы необходим лишь текстовый терминал с поддержкой удаленного доступа по протоколу SSH.

Одним из эффективных способов представления информации является декларативное представление, то есть определение отдельных понятий, их состояния в конкретные моменты времени и связей между ними. В языке Prolog («ПРОграммирование в терминах ЛОГики») основной является декларативная компо-

нента, так что он предназначен не столько для обработки данных, как для обработки фактов и декларативных правил. Факты представляют собой логические формулы. База знаний задается совокупностью таких формул. Логические методы обеспечивают получение новых фактов из фактов, представленных в базе знаний.

Prolog может использоваться при разработке экспертных систем, а также для следующих задач:

– доказательства теорем и вывода решений в задачах;

– создания пакетов символьной обработки при решении уравнений, дифференцировании, интегрировании и т. д.;

– разработки упрощенных версий систем ИИ;

– создания естественно-языковых интерфейсов для существующих программ;

– перевода текстов с одного языка на другой, в том числе – с одного языка программирования на другой.

Структура данного пособия имеет традиционную форму подачи учебного материала. Имеется несколько типовых разделов, а именно «Основы объектно-ориентированной модели программирования», «Работа с комплексными числами», «Вложенные и абстрактные классы, использование интерфейсов», «Работа с датой и временем», «Основы логического программирования», «Выполнение стандартных операций». В каждом из них размещен теоретический и практический учебный материал с примерами решения типовых задач.

1. Начальная настройка и авторизация

Предполагается, что авторизация осуществляется с внутренних машин Удмуртского государственного университета. Основное программно-техническое обеспечение располагается на сервере, имеющем ip-адрес **192.168.42.218.** До начала работы на сервере необходимо зарегистрировать учетную запись обучающегося. Таким образом, используется стандартная схема авторизации типа «логин/пароль».

Обычно, учебный процесс построен таким образом, что в большинстве компьютерных классов на клиентские узлы работают под управлением операционных систем семейства MS Windows. Поэтому существует проблема подключения к удаленному серверу под управлением ОС Linux Debian с клиентских узлов. Она решается путем использования, например двух программ – PuTTY, Xmanager Enterprise. Для выполнения типовых задач базового уровня освоения компетенций достаточно возможностей первой программы.

Авторизация с клиентских узлов

PuTTY – свободно распространяемый клиент для различных протоколов удалённого доступа, включая SSH, позволяет подключиться и управлять удаленным узлом (например, сервером). В **PuTTY** реализована только клиентская сторона соединения – сторона отображения, в то время как сама работа выполняется на другой стороне.

Чтобы начать работу с приложением, наведите курсор мыши на ярлык на рабочем столе и дважды щелкните левой клавишей:

Либо данное приложение также можно запустить из командной строки – **putty.exe**, либо воспользоваться инструментом «Поиск файлов и папок», либо скачать дистрибутив по ссылке https://putty.org.ru/download.html.

После того, как приложение начнет работать, откроется окно настройки, в поле **Host Name (or IP address)** напишите **192.168.42.218** и в поле **Saved Session** введите символьное обозначение сервера, например цифры **218**.

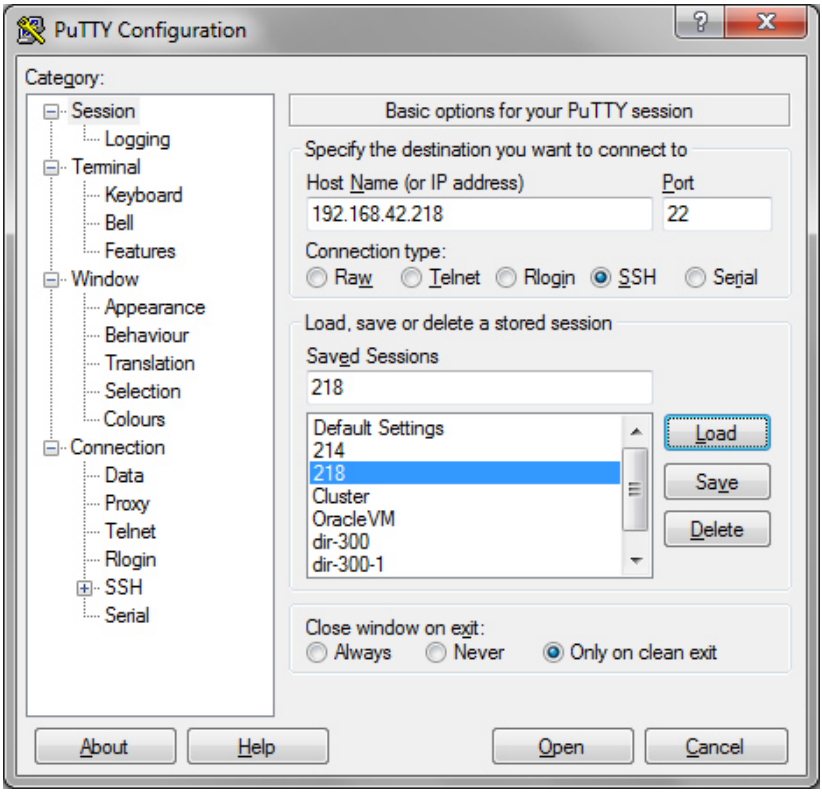

Рис. 1. Настройки соединения **Session.**

В левой части данного окна организован иерархический список настроек. Для соединения с узлом Кластера используются настройки по умолчанию, поэтому имеет смысл указать только некоторые из них, наиболее часто изменяемые пользователями: **Keyboard** и **Translation**.

Часто соединение не устанавливается правильно, если настройки соединения указаны неверно. Внимательно проверяйте ip-адрес сервера.

Для правильного отображения кириллицы на экране терминала необходимо установить в PuTTY тип кодировки UTF-8.

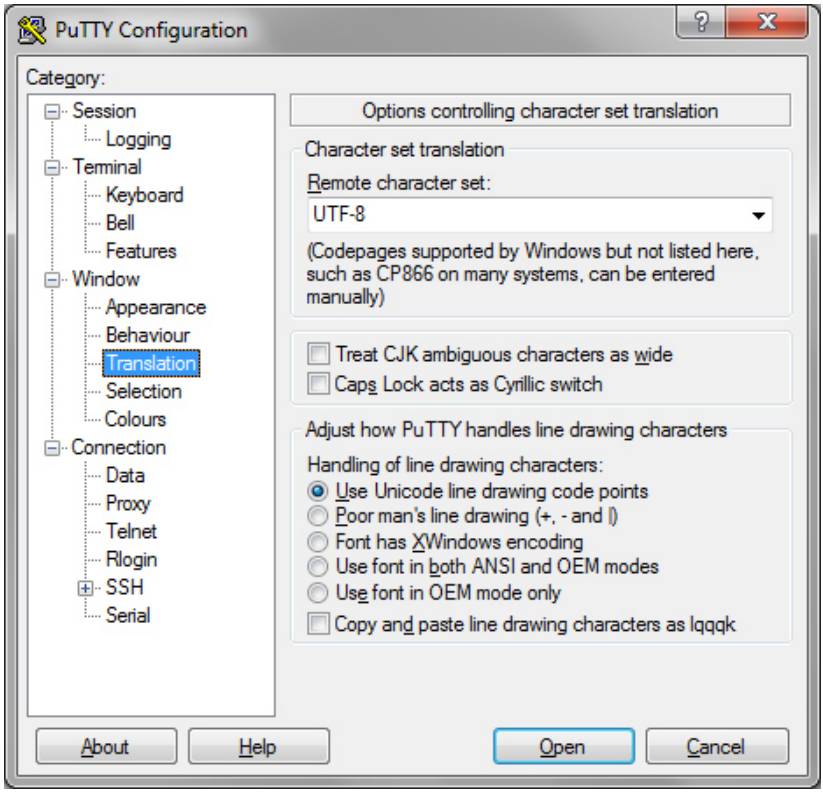

Рис. 2. Настройки соединения **Translation.**

Чтобы сохранить настройки соединения, нажмите кнопку **Save** в окне настроек **Session** (см. Рис. 1). В следующий раз, запуская приложение **PuTTY** для соединения с узлом Кластера, можно будет загрузить настройки соединения. Для этого выберите в списке сохраненных настроек (располагается ниже поля **Saved Sessions) 218** и нажмите кнопку **Load**.

Далее, нажмите кнопку **Open** в левой нижней части окна настроек соединения, и откроется окно для работы на узле как с обычной Linux-системой.

При подключении возможна выдача ошибки, связанной с указанием выбора алгоритма обмена ключами шифрования в протоколе SSH. Для исправления ситуации выберете вкладку как показано на следующем рисунке

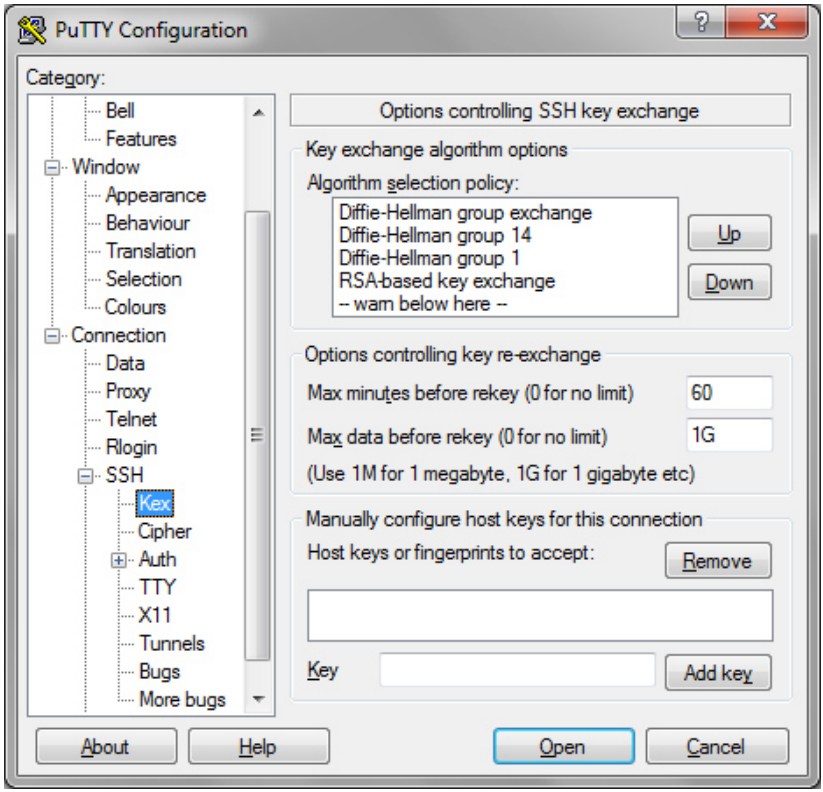

Рис. 3. Настройка выбора алгоритма обмена ключами.

Далее, в окне «Algorithm selection policy» нажимая кнопки «Up» и «Down» необходимо поднять на верхнюю позицию «Diffie-Hellman group 14», и сохранить эти изменения текущей сессии.

Если при подключении возникла ошибка, то проблема скорее всего в сетевых настройках клиентского узла. Необходимо проверить сетевое подключение.

2. Работа в среде Linux Debian

Для дальнейшей работы обучающимся необходима следующая информация по основным командам, используемым в процессе создания программ на языке ассемблер.

Команда *ls* (или *ls -al*) выводит текущий список файлов и каталогов. Данная команда необходима, чтобы найти нужную программу или исходный код, посмотреть содержимое текущего каталога. После входа в систему по умолчанию это каталог «*/home/учетная запись пользователя*» или содержимое переменной $HOME.

Команда «*cd Имя каталога*» служит для перехода в заданный каталог, «*cd ..*» – переход на один уровень вверх по дереву каталогов. Просто команда *cd* без параметров – возврат в домашний каталог.

Команда «*mkdir Имя каталога*» необходима для создания нового каталога.

Команда «*mv Источник Приемник*» в данном контексте необходима для переименования файла, например «*Источник*» – старое имя файла, «*Приемник*» – новое имя файла.

Команда *nano* используется для запуска простейшего текстового редактора подобного программе «Блокнот» в ОС MS Windows. В данном редакторе «*nano*» предполагается на начальном этапе набирать исходный код.

После его запуска в нижней части окна располагается список доступных команд, обозначенных например «*^G*» Помощь, «*^X*» Выход, где символ «^» обозначает нажатие клавиши «*Ctrl*», то есть для выхода из редактора необходимо нажать комбинацию двух кнопок на клавиатуре «*Ctrl*» + «*X*».

3. Основы работы с Java в Linux.

Покажем на примере создания простого консольного приложения как правильно компилировать и запускать программу.

Итак, на начальном этапе предлагается использовать утилиту javac, которая преобразует исходный код программы в байт-код. Предлагается написать следующий простой код.

```
----------------------------------------
class Hello{
public static void main(String[] a){
     System.out.println("Привет, мир!!!")
}}
----------------------------------------
```

Далее, сохранив его в файле с именем «*Hello.java*», скомпилируем его, выполнив команду

javac Hello.java

Если все прошло успешно, ошибок нет, то никакого дополнительного вывода текстовой информации в консоль не будет. В результате будет создан новый файл с именем «*Hello.class*», который будет готов к запуску путём исполнения команды

javac Hello

Замечание. После слова «Hello» не указывается расширение файла «.class». Его использование будет вызывать ошибку, что рекомендуется проверить самостоятельно.

Дополнительное задание: Попробуйте сохранить исходный код в файле с другим именем, отличающимся от имени главного класса. Что произойдет при исполнении этого файла?

4. Основы объектно-ориентированной модели

Основу ООМ составляют: инкапсуляция, наследование и полиморфизм. Инкапсуляция, или утаивание информации (*information hiding*) – это возможность скрыть внутреннее устройство объекта от его пользователей, предоставив через интерфейс доступ только к тем членам объекта, с которыми пользователю разрешается работать напрямую. При программировании на объектно-ориентированных языках скрываются элементы, не связанные напрямую с решением задачи, позволяя полностью сосредоточиться на самой задаче и решить ее более эффективно.

Наследованием называют возможность при описании класса указывать на его происхождение (*kind-of relationship*) от другого класса. Наследование позволяет создать новый класс, в основу которого положен существующий. В полученный таким образом класс можно внести свои изменения, а затем создать новые объекты данного типа. Этот механизм лежит в основе создания так называемой иерархии классов. Производным (*derived class*) называется создаваемый класс, производный от базового класса (*base class*). Производный класс наследует все члены базового класса. Какие именно члены базового класса наследуются производными классами, указывается через модификаторы доступа.

Полиморфизм – это функциональная возможность, позволяющая старому коду вызвать новый код. Это свойство наиболее ценно, поскольку дает возможность расширять и совершенствовать свою систему, не затрагивая существующий код.

Следующий пример содержит типичный объектный способ описания предметной области, в частности, описывается класс домашних животных.

```
  ----------------------------------------
  abstract class Pets{
    String name ;
    int age ;
    void action(){}
    void voice(){}
  }
  class Cats extends Pets{
    String breed ;
    void action(){
      System.out.println("Все кошки бегают и прыга-
ют!!!") ;
    }
    void voice(){
      System.out.println("Мяу-мяу-муррр!") ;
    }
    Cats( String breed ){
      this.breed = breed ;
    }
    Cats( int age1 ){
      this.age = age1 ;
    }
  }

  class lab1{
  public static void main(String a[]){
    Cats mycat1 = new Cats( "Мейн-кун" ) ;
    Cats mycat2 = new Cats( 99 ) ;
    mycat1.action() ; mycat1.voice() ;
    System.out.println("Cat breed = " + mycat1.breed )
;
  }}
  ----------------------------------------
```

Дополнительное задание: добавьте в данный исходный код описание дополнительного класса «Домашние собаки» с дополнительным конструктором, позволяющим определять имя хозяина собаки.

Пример. Имеется исходный код на языке программирования Java, иллюстрирующий выполнение операций с комплексными числами в объектно-ориентированном стиле. В данном тексте присутствуют опечатки, необходимо их исправить и запустить программу.

```
  ----------------------------------------
  class Complex{
     private static final double EPS = le-12;
  // Точность вычислений
     private double re, im;
  // Действительная и мнимая часть

  // Четыре конструктора
     Complex(double re, double im) {
        this, re = re; this.im = im;
     }
     Complex(double re){this(re, 0.0); }
     Complex(){this(0.0, 0.0); }
     Complex(Complex z){this(z.re, z.im) ; }
        // Методы доступа
     public double getRe(){return re;}
     public double getlmf){return im;}
     public Complex getZ(){return new Complex(re,
im);}
     public void setRe(double re){this.re = re;}
     public void setlm(double im){this.im = im;}
     public void setZ(Complex z){re = z.re; im = z.im;}
        // Модуль и аргумент комплексного числа
     public double mod(){return Math.sqrt(re * re + im
* im);}
     public double arg()(return Math.atan2(re, im);}
        // Проверка: действительное число?
     public boolean isReal(){return Math.abs(im) <
EPS;}
     public void pr(){    // Вывод на экран
        System.out.println(re + (im < 0.0 ? "" : '"+")
+ im + "i");
     }
```

```
        // Переопределение методов класса Object
      public boolean equals(Complex z){
         return Math.abs(re -'z.re) < EPS &&
                Math.abs(im - z.im) < EPS;
      }
      public String toString(){
         return "Complex: " + re + " " + im;
      }

   // Методы, реализующие операции +=, -=, *=, /=
      public void add(Complex z){re += z.re; im +=
z.im;}
      public void sub(Complex z){re -= z.re; im —=
z.im;}
      public void mul(Complex z){
         double t = re * z.re — im * z. im;
                im = re * z.im + im * z.re;
                re = t;
      }
      public void div(Complex z){
         double m = z.mod();
         double t = re * z.re — im * z.im;
         im = (im * z.re — re * z.im) / m;
         re = t / m;
      }

   // Методы, реализующие операции +, -, *, /
      public Complex plus(Complex z){
         return new Complex(re + z.re, im + z im);
      }
      public Complex minus(Complex z){
         return new Complex(re - z.re, im - z.im);
      }
      public Complex asterisk(Complex z){
         return new Complex(
            re * z.re - im * z.im, re * z.im + im * z
re);
      }
   public Complex slash(Complex z){
      double m = z.mod();
      return new Complex(
         (re * z.re - im * z.im) / m, (im * z.re - re *
z.im) / m);
      }
```

```
   }
      // Проверим работу класса Complex
   public class ComplexTest{
      public static void main(Stringf] args){
         Complex z1 = new Complex(),
                 z2 = new Complex(1.5),
                 z3 = new Complex(3.6, -2.2),
                 z4 = new Complex(z3);
         System.out.printlnf);       // Оставляем пустую
строку
         System.out.print("z1 = "); z1.pr();
         System.out.print("z2 = "); z2.pr();
         System.out.print("z3 = "); z3.pr();
         System.out.print ("z4 = "}; z4.pr();
         System.out.println(z4);
   // Работает метод toString()
         z2.add(z3);
         System.out.print("z2 + z3 = "}; z2.pr();
         z2.div(z3);
         System.out.print("z2 / z3 = "); z2.pr();
         z2 = z2.plus(z2);
         System.out.print("z2 + z2 = "); z2.pr();
         z3 = z2.slash(z1);
         System.out.print("z2 / z1 = "); z3.pr();
      }
   }
   ----------------------------------------
```

Замечание. Программа выдает результаты выполнения арифметических операций с комплексными числами. В одном из действий происходит деление на ноль, необходимо исправить данную ошибку.

5. Вложенные и абстрактные классы, использование интерфейсов

В описание класса можно включать описание другого класса, в нем следующего и т.д. Все вложенные классы можно разделить на вложенные классы-члены класса (*member classes*), описанные вне методов, и вложенные локальные классы (*local classes*), описанные внутри методов и/или блоков. Локальные классы, как и все локальные переменные, не являются членами

класса. Предлагается следующий пример исходного кода с описанием возможностей использования вложенных классов.

```
  ----------------------------------------
  class Nested{
  static private int pr;   // Переменная pr объявлена
статической
  // чтобы к ней был доступ из статических классов A и
AB
  String s = "Member of Nested";
  // Вкладываем статический класс.
  static class .A{      // Полное имя этого класса —
Nested.A
    private int a=pr;
    String s = "Member of A";
      // Во вложенньм класс A вкладываем еще один ста-
тический класс
    static class AB{    // Полное имя класса —
Nested.A.AB
      private int ab=pr;
      String s = "Member of AB";
    }
  }
  //В класс Nested вкладываем нестатический класс
  class B{     // Полное имя этого класса — Nested.B
    private int b=pr;
    String s = "Member of B";
    // В класс B вкладываем еще один класс
    class BC{  // Полное имя класса — Nested.B.BC
      private int bc=pr;
      String s = "Member of BC";
    }
    void f(final int i){    // Без слова final перемен-
ные i и j
      final int j = 99;     // нельзя использовать в
локальном классе D
      class D{              // Локальный класс D извес-
тен только внутри f()
        private int d=pr;
        String s = "Member of D";
        void pr(){
                // Обратите внимание на то, как разли-
чаются
```

```
                // переменные с одним и тем же именем
"s"
          System.out.println(s + (i+j));   // "s" экви-
валентно "this.s"
          System.out.println(B.this.s);
          System.out.println(Nested.this.s);
  //      System.out.println(AB.this.s);    // Нет дос-
тупа
  //      System.out.println(A.this.s);     // Нет дос-
тупа
        }
      }
      D d = new D();    // Объект определяется тут же,
в методе f()
      d.pr();           // Объект известен только в ме-
тоде f()
    }
  }
  void m(){
    new Object(){
  // Создается объект безымянного класса,
  // указывается конструктор его суперкласса
      private int e = pr;
      void g(){
        System.out.println("From g()) ;
      }
    }.g();   // Тут же выполняется метод только что
созданного объекта
    }
  }
  public class NestedClasses{
    public static void main(String[] args){
      Nested nest = new Nested();
  // Последовательно раскрываются
  // три матрешки
      Nested.A theA = nest.new A(); // Полное имя клас-
са и уточненная
                                    // операция new. Но
конструктор только вложенного класса
      Nested.A.AB theAB = theA.new AB();
  // Те же правила. Операция
  // new уточняется только одним именем
      Nested.B theB = nest.new B(); // Еще одна матреш-
ка
```

```
      Nested.B.BC theBC = theB.new BC();
      theB.f(999);  // Методы вызываются обычным обра-
зом
      nest.m();
    }
  }
  }
  ----------------------------------------
```

Класс, содержащий абстрактный метод, является абстрактным классом, кроме того, абстрактным классом может быть класс, не содержащий абстрактных методов, но имеющий модификатор *abstract*.

Абстрактный класс обладает следующими свойствами:

- не может иметь экземпляров;
- подклассы абстрактного класса могут иметь экземпляры, если в них переопределяются все абстрактные методы, и для каждого метода определяется реализация;
- если подкласс абстрактного класса не реализует все методы, унаследованные от абстрактного суперкласса, то он сам становится абстрактным классом.

Следует обратить внимание на два важных момента:

1) Ссылки на экземпляры подклассов абстрактного класса могут быть присвоены элементам массива абстрактного суперкласса;

2) Методы подклассов можно вызывать для ссылок из массива абстрактного суперкласса, даже если эти методы определены в суперклассе как абстрактные.

Предлагается реализовать следующий пример исходного кода с описанием абстрактных классов для решения геометрических задач.

```
  ----------------------------------------
  abstract class Shape{
       public abstract double area();
       public abstract void circumference();
  }
  class Circle extends Shape{
       protected double r;
       static final double pi = 3.1416;
```

```
        public Circle( double r){ this.r = r; }
        public double area(){ Sytem.out.println("Circle
area="+ pi*r*r);
                return pi*r*r; }
        public void circumference(){ Sys-
tem.out.println("Circle circumference = "+
                                              2*pi*r  );
   }}
   class Rectangle extends Shape{
        protected double w, h;
        public Rectangle( double w, double h){ this.w =
w; this.h = h; }
        public double area(){
         System.out.println("Rectangle area = "+ w*h );
           return w*h;}
        public void circumference(){ Sys-
tem.out.println("Rectangle circumference = "+ 2*(w+h));
}
   }
   public class ACDemo{
        public static void main(String args[]){
        Shape[] shp = new Shape[4];
        shp[0] = new Circle(2.0);
        shp[1] = new Rectangle(2.0, 2.0);
        shp[2] = new Circle(3.0);
        shp[3] = new Rectangle(4.0, 4.0);
        double total_area = 0;
        for( int i = 0; i < shp.length; i++)
                total_area += shp[i].area();
                System.out.println("total_area = "+ to-
tal_area );
        }
   }
   ----------------------------------------
```

Интерфейс – это коллекция описаний методов (без их реализаций) и констант.

Интерфейсы используются в следующих ситуациях:

- описания сходства между классами, не связанными отношением наследования;
- описания методов, которые один или большее число классов могут реализовать;

- демонстрации программного интерфейса объектов без показа их классов (такие объекты называются анонимными и могут быть полезны при передаче пакета классов другим разработчикам).

Интерфейс является ссылочным типом и может быть использован всюду, где допускается применение ссылочного типа, например, в качестве аргумента метода или описания переменной.

```
abstract class Shape{
public abstract double area();
public abstract void circumference();
}
class Circle extends Shape{
protected double r;
static final double pi = 3.1416;
public Circle( double r){ this.r = r; }
public double area(){ System.out.println("Circle
                                    area="+ pi*r*r);
return pi*r*r; }
public        void        circumference(){        Sys-
tem.out.println("Circle circumference = "+
                                       2*pi*r ; }
}
class Rectangle extends Shape{
protected double w, h;
public Rectangle( double w, double h){ this.w = w;
this.h = h; }
public double area(){   Sytem.out.println("Rectangle
area = "+ w*h );
return w*h;}
public        void        circumference(){        Sys-
tem.out.println("Rectangle circumference = "+
                                        2*(w+h)); }
}
class NamedRectangle extends Rectangle implements
NamedShape{
String st;
NamedRectangle( double w, double h, String s ){ su-
per( w, h ); st = s;}
public void put(){ System.out.println("Rectangle
                             name is  "+ st);}
}
```

```
class NamedCircle extends Circle implements Named-
Shape{
String st;
NamedCircle( double r, String s ){ super( r );
                                    st = s;}
public void put(){ System.out.println("Circle name is
"+ st);}
      }
interface NamedShape{ public void put(); }

public class lab3{
public static void main(String args[]){
Shape[] shp = new Shape[4];
NamedShape[] ns = new NamedShape[4];
NamedCircle c1 = new NamedCircle(2.0, "fig1");
NamedRectangle r1 = new NamedRectangle(2.0, 2.0,
                                          "fig2");
NamedCircle c2 = new NamedCircle(3.0, "fig3");
NamedRectangle r2 = new NamedRectangle(4.0, 4.0,
"fig4");
shp[0] = c1;      ns[0] = c1;
shp[1] = r1;      ns[1] = r1;
shp[2] = c2;      ns[2] = c2;
shp[3] = r2;      ns[3] = r2;
double total_area = 0;
for( int i = 0; i < shp.length; i++){
ns[i].put();
shp[i].circumference();
total_area += shp[i].area();}
System.out.println("total_area = "+ total_area );
}}
```

5.1. Практические задания к разделу

Используя указанные ниже примеры предметных областей, написать программу с использованием интерфейсов, абстрактных и вложенных классов.

1. Описать предметную область «Успеваемость студента»;
2. Описать предметную область «Средняя школа»;
3. Описать предметную область «Институт УдГУ»;

4. Описать предметную область «Магазин продовольственных товаров»;
5. Описать предметную область «it-компания»;
6. Описать предметную область «Склад запасных частей»;
7. Описать предметную область «Салон сотовой связи»;
8. Описать предметную область «Транспортная компания»;
9. Описать предметную область «Аптека»;
10. Описать предметную область «Заведение общепита»;
11. Описать предметную область «Банк»;
12. Описать предметную область «Дошкольное учреждение»;
13. Описать предметную область «Почта»;
14. Описать предметную область «Страховая компания»;
15. Описать предметную область «Кинотеатр».

6. Работа с датой и временем

Методы работы с датами и показаниями времени собраны в два класса: Calendar и Date из пакета java.util.

Объект класса Date хранит число миллисекунд, прошедших с 1 января 1970 года 00:00:00 по Гринвичу. Это «день рождения» UNIX, он называется «Epoch».

Класс Date удобно использовать для отсчета промежутков времени в миллисекундах.

Получить текущее число миллисекунд, прошедших с момента Epoch на той машине, где выполняется программа, можно статическим методом System.currentTimeMillis().

В классе Date два конструктора. Конструктор Date() заносит в создаваемый объект текущее время машины, на которой выполняется программа, по системным часам, а конструктор Date(long millisec) — указанное число.

Получить значение, хранящееся в объекте, можно методом long getTime(),

установить новое значение — методом setTime(long newTime).

Три логических метода сравнивают отсчеты времени:

boolean after (long when) — возвращает true, если время when больше данного;

boolean before (long when) — возвращает true, если время when меньше данного;

boolean after (Object when) — возвращает true, если времена when — объекта класса Date и данное совпадают.

Еще два метода, сравнивая отсчеты времени, возвращают отрицательное число типа int, если данное время меньше аргумента when; нуль, если времена совпадают; положительное число, если данное время больше аргумента when:

int compareTo(Date when);

int compareTo(Object when) — если when не относится к объектам класса Date, создается исключительная ситуация.

Преобразование миллисекунд, хранящихся в объектах класса Date, в текущее время и дату производится методами класса Calendar.

Класс Calendar — абстрактный, в нем собраны общие свойства календарей: юлианского, григорианского, лунного. В Java API пока есть только одна его реализация — подкласс GregorianCalendar.

Поскольку Calendar — абстрактный класс, его экземпляры создаются четырьмя статическими методами по заданной локали и/или часовому поясу:

Calendar getlnstance()

Calendar getlnstance(Locale loc)

Calendar getlnstance(TimeZone tz)

Calendar getlnstance(TimeZone tz, Locale loc)

Для работы с месяцами определены целочисленные константы от JANUARY до DECEMBER, для работы с днями недели — константы MONDAY до SUNDAY.

Первый день недели можно узнать методом int getFirstDayOfWeek(), а установить — методом setFirstDayOfWeek(int day), например:

setFirstDayOfWeek(Calendar.MONDAY)

Остальные методы позволяют просмотреть время и часовой пояс или установить их.

Различные способы представления дат и показаний времени можно осуществить методами, собранными в абстрактный класс DateFormat и его подкласс SimpleDateFormat из пакета java.text.

Класс DateFormat предлагает четыре стиля представления даты и времени:

стиль SHORT представляет дату и время в коротком числовом виде: 27.04.01 17:32; в локали США: 4/27/01 5:32 PM;

стиль MEDIUM задает год четырьмя цифрами и показывает секунды: 27.04.2001 17:32:45; в локали США месяц представляется тремя буквами;

стиль LONG представляет месяц словом и добавляет часовой пояс: 27 апрель 2001 г. 17:32:45 GMT+03.-00;

стиль FULL в русской локали таков же, как и стиль LONG; в локали США добавляется еще день недели.

Есть еще стиль DEFAULT, совпадающий со стилем MEDIUM.

При создании объекта класса SimpieDateFormat можно задать в конструкторе шаблон, определяющий какой-либо другой формат, например:

SimpieDateFormat sdf = new SimpieDateFormat("dd-MM-yyyy hh.mm"); System.out.println(sdf.format(new Date()));

Получим вывод в таком виде: 13-01-2017 17.32.

Далее, приводится пример исходного кода для работы с календарем. Предлагается исправить его, чтобы вывод на печать календаря дат на текущий месяц был корректным.

```
_____________________________________
import java.util.*;
public class CaltndarTest {
      public static void main(String[] args){
      GregorianCalendar d = new GregorianCalendar ();
      int today = d.get(Calendar.DAY_OF_MONTH);
      int month = d.get(Calendar.MONTH);
      // Устанавливаем объект d на первую дату месяца
      d.set(Calendar.DAY_OF_MONTH, 1);
      // Создаем объект d с текущей датой
      int weekday = d.get(Calendar.DAY_OF_WEEK);
      // Выводим на печать заголовок таблицы
      System.out.println("Вс Пн Вт Ср Чт Пт Сб");
```

```
        // Вставляем первую строку календаря
        for (int i = Calendar.SUNDAY; i< weekday; i++)
                System.out.print("");
        do
        {
                // Выводим на печать день
                int day = d.get(Calendar.DAY_OF_MONTH);
                if(day < 10) System.out.print("");
                System.out.print(day);
                // Отметить текущий день звездочкой
                if (day == today)System.out.print("*");
                else
                        System.out.print("  ");
                // После каждой субботы начинаем новую
строку
                if(weekday == Calendar.SATURDAY)
                        System.out.println();
                //| Передвинуть объект d на новый день
                d.add(Calendar.DAY_OF_MONTH, 1);
                weekday = d.get(Calendar.DAY_OF_WEEK);

        }
        while (d.get(Calendar.MONTH) == month);
        // Цикл завершается, когда объект d установлен
        // на первый день следующего месяца
        //Выводим на экран последнюю строку
        if(weekday != Calendar.SUNDAY)
                System.out.println();
        }
  }
  ----------------------------------------
```

7. Работа со строками

Текстовые строки в языке Java являются объектами. Они представляются экземплярами класса String или класса StringBuffer.

В объектах класса String хранятся строки-константы неизменной длины и содержания. Это значительно ускоряет обработку строк и позволяет экономить память, разделяя строку между объектами, использующими ее.

Длину строк, хранящихся в объектах класса StringBuffer, можно менять, вставляя и добавляя строки и символы, удаляя подстроки или сцепляя несколько строк в одну строку. Во многих случаях, когда надо изменить длину строки типа String, компилятор Java неявно преобразует ее к типу StringBuffer, меняет длину, потом преобразует обратно в тип string. Например, следующее действие

String s = "Это" + " одна " + "строка";

компилятор выполнит так:

String s = new StringBuffer().append("Это").append(" одна ").append("строка").toString();

Будет создан объект класса StringBuffer, в него последовательно добавлены строки "Это", " одна ", "строка", и получившийся объект класса StringBuffer будет приведен к типу string методом toString().

Напомним, что символы в строках хранятся в кодировке Unicode, в которой каждый символ занимает два байта. Тип каждого символа char.

В следующем примере исходного кода приводятся типичные примеры использования методов для выполнения операций над строками.

```
  ----------------------------------------
  class TestString {
  public static void main(String args[]) {
        String str = "Now is the winter of our discon-
tent";
        System.out.println("The string is: " + str);
        System.out.println("Length of this string: "+
str.length());
        System.out.println("The character at position 5:
"+ str.charAt(5));
        System.out.println("The substring from 11 to 17:
"+ str.substring(11, 17));
        System.out.println("The index of the character
d: "+ str.indexOf('d'));
        System.out.print("The index of the beginning of
the ");
        System.out.println("substring \"winter\":"+
str.indexOf("winter"));
```

```
        System.out.println("The string in upper case: "+
str.toUpperCase());
   }}
   ----------------------------------------
```

Замечание. Класс StringTokenizer из пакета java.util небольшой, в нем три конструктора и шесть методов.

Первый конструктор StringTokenizer (string str) создает объект, готовый разбить строку str на слова, разделенные пробелами, символами табуляций '\t', перевода строки '\n' и возврата каретки '\r'. Разделители не включаются в число слов.

Второй конструктор StringTokenizer (string str. string delimeters) задает разделители вторым параметром deiimeters, например:

StringTokenizer("Казнить,нельзя:пробелов-нет", " \t\n\r,:-");

Здесь первый разделитель — пробел. Потом идут символ табуляции, символ перевода строки, символ возврата каретки, запятая, двоеточие, дефис. Порядок расположения разделителей в строке deiimeters не имеет значения. Разделители не включаются в число слов.

Третий конструктор позволяет включить разделители в число слов:

StringTokenizer(string str, string deiimeters, boolean flag);

Если параметр flag равен true, то разделители включаются в число слов, если false — нет. Например:

StringTokenizer("a - (b + c) / b * c", " \t\n\r+*-/(), true);

В разборе строки на слова активно участвуют два метода:

метод nextToken () возвращает в виде строки следующее слово;

логический метод hasMoreTokens () возвращает true, если в строке еще есть слова, и false, если слов больше нет.

Третий метод countTokens () возвращает число оставшихся слов.

Четвертый метод nextToken(string newDeiimeters) позволяет «на ходу» менять разделители. Следующее слово будет выделено по новым разделителям newDelimeters; новые разделители

действуют далее вместо старых разделителей, определенных в конструкторе или предыдущем методе nextToken ().

Оставшиеся два метода nextElement() и hasMoreElements() реализуют интерфейс Enumeration. Они просто обращаются к методам nextToken() и hasMoreTokens().

Приведем следующий пример.

```
_____________________________________
  String s = "Строка, которую мы хотим разобрать на
слова";
  StringTokenizer st = new StringTokenizer(s, "
\t\n\r,.");
  while(st.hasMoreTokens()){
        // Получаем слово и что-нибудь делаем с ним, на-
пример,
        // просто выводим на экран
        System.out.println(st.nextToken()) ;
  }
_____________________________________
```

7.1. Практические задания к разделу

Написать программу, выводящую на экран тестовую информацию в соответствии со следующими правилами.

1. Определяет количество слов в тексте, оканчивающихся на гласную букву;
2. Поочередно выводит каждое слово в последовательности 2, 1, 3;
3. Определяет количество слов в строке, у которых первый и последний символы совпадают;
4. Определяет количество слов в строке, начинающихся на гласную букву;
5. Выводит каждое слово строки, содержащее максимальное количество символов;
6. Выводит каждое слово строки, содержащее минимальное количество символов;
7. Поочередно выводит все символы строки, отличные от букв и пробела;
8. Определяет количество букв 'a' в последнем слове текста;

9. Определяет самую длинную последовательность цифр в строке (считать, что любое количество пробелов между двумя цифрами не прерывает последовательности цифр);

10. Определяет порядковый номер заданного слова в каждом предложении текста (заданное слово вводится с клавиатуры);

11. Выводит строку на экран дисплея еще раз, убирая из него заданное слово и удаляя лишние пробелы;

12. Выводит строку на экран дисплея еще раз, меняя в нем местами заданные слова и удаляя лишние пробелы;

13. Выводит текст на экран дисплея еще раз, убирая лишние пробелы между словами и начиная каждое предложение с новой строки;

14. Определяет наибольшее количество подряд идущих пробелов в строке;

15. Выводит строку на экран дисплея еще раз, заменяя в заданном слове определённые строчные буквы прописными.

8. Основы логического программирования

Конечной целью составления компьютерной программы является создание инструмента, предназначенного для решения задачи из некоторой прикладной области. Прикладная область - это некоторая абстрактная часть мира или область знаний. Структура прикладной области состоит из значимых для этой области сущностей, функций и отношений.

Процесс составления программы на Prolog в основном сходен с процессом построения теории в логике предикатов.

1) Программист анализирует значимые сущности, функции и отношения из прикладной области и выбирает обозначения для них.

2) Программист семантически определяет каждую значимую функцию и каждое значимое отношение. Для отношений указывается, какие конкретные их реализации дают истину, а какие - ложь.

3) Программист аксиоматически определяет каждое отношение при помощи фраз Prolog. Аксиоматическое определение отношения будет удачным, если оно отразит смысл семантическо-

го определения. Множеством аксиоматических определений всех значимых для заданной предметной области отношений является программа, моделирующая структуру этой области.

4) Для выполнения запросов к множеству фраз программист или пользователь применяет интерпретатор языка Prolog. Совокупность, состоящую из запроса, множества фраз программы и интерпретатора языка можно рассматривать как алгоритм решения задач из прикладной области. При этом запрос и фразы программы представляют собой начальные формулы алгоритма, а интерпретатор содержит правила преобразования этих формул. Интерпретатор играет роль активной силы, которая выполняет выводы из фраз программы и тем самым реализует или развертывает отношения определенными фразами программы.

Язык Prolog основан на концепциях логического программирования, предложенных профессором Лондонского университета Робертом Ковальским, и представляет собой среду, ориентированную на рассуждения или дедукцию.

Р. Ковальский описывает сущность логического программирования фразой «Алгоритм = Логика + Управление».

В языке Prolog сочетается использование нескольких важных концепций, к числу которых относятся:

1) применение фраз Хорна для представления знаний;

2) дескриптивный стиль программирования;

3) как декларативная, так и процедурная семантика;

4) возможность чередовать программный текст метауровня с текстом объектного уровня.

Логическая или, точнее, хорновская логическая программа состоит из набора хорновских фраз, которые в языке Пролог называются фактами и правилами.

Структура этих правил и фактов такова, что, с одной стороны, с их помощью довольно естественно описываются многие задачи, а с другой стороны, они допускают простую процедурную интерпретацию. Два этих обстоятельства и позволяют использовать язык фактов и правил в качестве языка программирования.

Фактом называется формула вида $P(t_1,\ldots,t_n)$, где P – предикатный символ, $t_1,\ldots,t_n$ – термы, построенные из переменных, констант и функциональных символов. С точки зрения математической логики, факты – это атомарные формулы.

Правилом называется формула вида $A_1 \,\&\ldots\&\, A_n \Rightarrow A_0$, где все A_i – это атомарные формулы. В Prolog данная формула запишется в виде: $A_0 :- A_1, A_2, \ldots, A_n.$, таким образом, операция конъюнкции обозначается запятой, в конце правила ставится точка.

Запросом к логической программе называется формула вида $?- A_1, A_2, \ldots, A_n.$, здесь вопросительный знак служит для обозначения приглашения ввода запроса, дополнительно вводить его не надо. Данный запрос, если он не содержит переменных, читается так: «Верно ли, что выполняется A_1 и A_2 и … и A_n»? Если же в атомарных формулах запроса содержатся переменные (обозначаются с прописной буквы) $X_1, X_2, \ldots, X_m$, то его следует читать: «Для каких объектов $X_1, X_2, \ldots, X_m$ верно A_1 и … и A_n»?

8. Основы работы с GNU Prolog в Linux

В работе необходимо учитывать некоторые особенности интерфейса и синтаксиса Prolog. Для запуска интерпретатора выполняем команду prolog в командной строке, например

```
mike@mkdeb214:~$ prolog
```

в результате появится строка-приглашение к дальнейшему вводу

```
mike@mkdeb214:~$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?-
```

Интерпретатор ждет дальнейших действий пользователя. Например, необходимо исполнить следующий исходный код, записанный в файл «lab1.pro»

```
----------------------------------------
man(ivan).
womam(ann).
parent(ivan,ann).
father(X,Y):-parent(X,Y),man(X).
----------------------------------------
```

Этот небольшой код содержит три факта и одно правило – «являться отцом». Для того, чтобы загрузить данный код, необходимо в строке-приглашении интерпретатора prolog ввести команду

```
mike@mkdeb214:~$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- ['lab1.pro'].
```

Таким образом, имя файла заключается в «одинарные» кавычки и записывается в квадратных скобках, в конце ставится точка. Если в программе нет ошибок, то будет исполнен следующий вывод

```
mike@mkdeb214:~$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- ['lab1.pro'].
compiling /home/mike/lab1.pro for byte code...
/home/mike/lab1.pro compiled, 4 lines read - 737 bytes written, 5 ms

yes
```

Это означает, что синтаксические ошибки отсутствуют, и интерпретатор ждет целеуказания от пользователя, например

```
| ?- father(X,Y).

X = ivan
Y = ann
```

```
yes
```

Здесь мы ставим цель – «определить всех отцов», получаем единственный ответ, исходя из нашей базы знаний, что «отец» X=ivan для Y=ann.

Замечание. Все переменные начинаются с заглавного «большой» символа, значения переменных записываются прописными «маленькими» символами.

Еще приведем следующий пример запроса

```
| ?- father(petr,Y).

no
```

Этот запрос означает – «определить всех Y, для которых petr является отцом». Естественно, что таких данных в нашей программе нет, поэтому выводится соответствующий ответ.

Замечание. Для выхода из программы необходимо использовать команду halt. Например

```
| ?- halt.
mike@mkdeb214:~$
```

возвращаемся в режим командного интерпретатора bash.

8.1. Практические задания к разделу

Предлагается набрать, запустить и отладить следующие примеры исходного кода

Пример 1. Имеется следующий исходный код, содержащий информацию о торговых марках и странах-производителях бытовой техники.

```
----------------------------------------
model(daewoo,korea).
model(viewsonik,usa).
model(eizo,germany).
model(epson,china).
model(hitachi,japan).
model(sony,X) :- model(viewsonik,X).
----------------------------------------
```

Необходимо получить ответы на следующие запросы:

```
| ?- model(daewoo,korea).
```

```
| ?- model(X,usa).
| ?- model(X,Y).
```

Пример 2. Имеется следующий исходный код, содержащий информацию о марках автомобилей и их странах-производителях.

```
----------------------------------------
model(daewoo,korea).
model(ford,usa).
model(mersedes,germany).
model(tabria,ukraine).
model(hitachi,japan).
model(vaz,russia).
model(gaz,X) :- model(vaz,X).
----------------------------------------
```

Также, необходимо получить ответы на следующие запросы:

```
| ?- model(ford,korea).
| ?- model(X,russia).
| ?- model(X,Y).
```

Пример 3. Составить и отладить следующую программу для организации турнира по настольному теннису среди студентов второго и третьего курса.

```
----------------------------------------
turnir(maxim,2).
turnir(yura,3).
turnir(nikolay,2).
turnir(oleg,2).
turnir(vova,3).
----------------------------------------
```

Необходимо получить ответы на следующие запросы:

```
| ?- turnir(X1,2), turnir(X2,3), X1\=X2.
| ?- turnir(X1,2), turnir(X2,3), X1=X2.
| ?- turnir(X1,2), turnir(X2,2), X1\=X2.
```

Пример 4. Составить и отладить в программу для организации турнира по волейболу среди команд факультетов университета. Отдельно предусмотрев турнир мужских и женских команд.

```
----------------------------------------
turnir(evgeniy,male,imitif).
```

```
turnir(marina,female,igz).
turnir(alex,male,igz).
turnir(lola,female,imotif).
turnir(serge,male,imotif).
```

Необходимо получить ответы на следующие запросы:

```
| ?- turnir(X1,P,Y1), turnir(X2,P,Y2),
Y1\=Y2.
```

Пример 5. Составить программу для вычисления значений (n-2)!

```
factorial(3,1).
factorial(N,Res) :- N>1, N1 is N-1,
factorial(N1,FacN1), Res is (N-2)*FacN1.
```

Пример 6. Составить программу для вычисления значения:

C_m^n, где, например, m = 5, n = 3, $C_5^3 = \frac{5!}{3!2!}$.

```
factorial(1,1).
factorial(N,Res) :- N>1, N1 is N-1,
factorial(N1,FacN1), Res is N * FacN1.
```

Соответствующий запрос выглядит так:

```
| ?- factorial(5,F5), factorial(3,F3), fac-
torial(2,F2), Result is F5/(F3*F2),
write("5/(3!*2!)="), write(Result).
```

9. Обработка списков

Список является основной структурой в Prolog. Элементы списка должны разделяться запятыми и заключаться в квадратные скобки, например: [1,2,3], [dog, cat, canary]. Символ "|" разделяет список на две части: начало списка и остаток списка; если начало списка состоит из одного элемента, то части называются голова списка и хвост списка. Если H – голова списка, а T – хвост, то список представляется термом [H|T].

Пример. Имеется список из пяти имен. Необходимо составить программу для выяснения есть ли введенное имя в списке.

```
member1(Name,[Name|_]).
member1(Name,[_|Tail]) :- member1(Name,Tail).
```

Соответствующий запрос выглядит так:

```
| ?-member1(eric,[jhon,leo,eric,frenk]).
```

Пример. К множеству целых чисел {0, 2, 4, 8, 12, 18} добавить множество чисел {-5, -2, - 8, -15, 20}.

```
append1([],List,List).
append1([X|L1],List2,[X|L3]) :-
append1(L1,List2,L3).
writelist([]).
writelist([Head|Tail]) :- write(Head),
write("   "), writelist(Tail).
goal(X,Y) :- append1(X,Y,Z), writelist(Z).
```

Соответствующий запрос выглядит так:

```
| ?-goal([0,2,4,8,12,18],[-5,-2,-8,-15,20]).
```

Дополнительное задание 1: Написать программу, способную производить «мутантов», то есть гибридов различных животных. Животные задаются их названиями в виде атомов. Два животных производят на свет мутанта, если окончание названия первого животного совпадает с началом названия второго.

Каждое животное задается с помощью факта animal(<животное>). Необходимо определить предикат mutant, вызов которого в виде цели mutant(X) сопоставит переменной X различные мутанты.

Вот результаты, полученные на множестве животных (крокодил, черепаха, карибу, лошадь, хамелеон, буйвол, волк):

крокодилошадь,
буйволк,
карибуйвол,
черепахамелеон,
волкрокодил,

волкарибу,

буйволошадь.

Дополнительное задание 2: Ученики A,B,C,D и E участвовали в одном конкурсе. Пытаясь угадать результаты соревнований, некто предполагал, что получится последовательность A,B,C,D,E. Но оказалось, что он не указал верно ни место какого-либо из участников и никакой пары следующей непосредственно друг за другом учеников. Некто другой, предполагая результат D,A,E,C,B, угадал правильно места двух учеников, а также две пары (непосредственно следующих друг за другом учеников).

Каков был на самом деле результат конкурса?

Список литературы

1. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М.: Мир, 1990.-235 с.

2. Братко И. Программирование на языке Пролог для искусственного интеллекта. - М.: Мир, 1990.-560 с.

3. Зюзьков В.М. Искусственный интеллект и экспертные системы. Логическое программирование. Часть 2: Учебное пособие. - Томск: Томский межвузовский центр дистанционного образования, 1999. - 63 с.

4. Васильев А. Н. Java. Объектно-ориентированное программирование : [учеб. пособие] для магистров и бакалавров / А. Н. Васильев. - Санкт-Петербург : Питер, 2014. - 395, [1] с. : ил. ; 70x100/16. - (Учебное пособие, Стандарт третьего поколения). - Библиогр.: с. 377. - Алф. указ.: с. 396.

5. Ноутон П. Java 2 : пер. с англ. / П. Ноутон, Г. Шилдт. - СПб. : БХВ - Петербург, 2005. - 1050 с. ; 70x100/16. - Пер. с англ. - Предм. указ. : с. 1034-1050. - Рус. яз. - ISBN 0-07-211976-4 (англ). - 5-94157-012-0 (рус.).

СОДЕРЖАНИЕ

Учебное издание

Составитель: Михаил Аркадьевич Клочков

Технологии обработки информации на языках высокого уровня

Учебно-методическое пособие

Отпечатано с оригинал-макета заказчика
Подписано в печать 1.02.18. Формат 60x84$^{1/16}$.
Печать офсетная. Усл. печ. л. Уч.-изд. л.
Заказ № Тираж 30 экз.

Издательство «Удмуртский университет»
426034, Ижевск, Университетская, д. 1, корп. 4, каб. 207
Тел./факс: + 7 (3412) 500-295. E-mail: editorial@udsu.ru