МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

Кафедра систем автоматизированного проектирования
и информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам и курсовому проекту по дисциплине «Технологии разработки клиент-серверных приложений» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения

JAVA

Воронеж 2023

Составитель: канд. техн. наук Е.Н. Королев
УДК 681.38+681.3
Методические указания к лабораторным работам по дисциплине «Технологии разработки клиент-серверных приложений» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост. канд. техн. наук Е.Н. Королев. Воронеж, 2023. – 21 с.

Методические указания содержат краткие теоретические и практические сведения об основных технологиях разработки клиент-серверных приложений на языке Java.
Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2003 и содержатся в файле МУ_ТРКСП.pdf.

Рецензент д-р техн. наук, проф. А.А. Рындин

Ответственный за выпуск зав. кафедрой
д-р техн. наук, проф. Я.Е. Львович

Рекомендовано методическим семинаром кафедры САПРИС и методической комиссией ФИТКБ Воронежского государственного технического университета в качестве методических материалов

ВВЕДЕНИЕ

Java является объектно-ориентированным языком программирования. В отличии от многих языков, в том числе и C++, на Java нельзя писать не объектно-ориентированные программы. Из этого сразу следует один вывод. Нельзя научиться программировать на Java, не овладев основами объектно-ориентированного подхода.

5 принципов объектно-ориентированного подхода

- Все является объектом. Все данные программы хранятся в объектах. Каждый объект создается, существует какое-то время, потом уничтожается.
- Программа есть группа объектов, общающихся друг с другом.Кроме того, что объект хранит какие-то данные, он умеет выполнять различные операции над своими данными и возвращать результаты этих операций. Теоретически эти операции выполняются как реакция на получение некоторого сообщения данным объектом. Практически это происходит при вызове метода данного объекта.
- Каждый объект имеет свою память, состоящую из других объектов и/или элементарных данных. Объект хранит некоторые данные. Эти данные — это другие объекты, входящие в состав данного объекта и/или данные элементарных типов, такие как целое, вещественное, символ, и т.п.
- Каждый объект имеет свой тип (класс). Т.е. в объектно-ориентированном подходе не рассматривается возможность создания произвольного объекта, состоящего из того, например, что мы укажем в момент его создания. Все объекты строго типизированы. Мы должны сначала описать (создать) тип (класс) объекта, указав в этом описании из каких элементов (полей) будут состоять объекты данного типа. После этого мы можем создавать объекты этого типа. Все они будут состоять из одних и тех же элементов (полей).
- Все объекты одного и того же типа могут получать одни и те же сообщения. Кроме описания структуры данных, входящих в объекты данного типа, описание типа содержит описание всех сообщений, которые могут получать объекты данного типа (всех методов данного класса). Более того, в описании типа мы должны задать не только перечень и сигнатуру сообщений данного типа, но и алгоритмы их обработки.

Java Platform, Enterprise Edition, сокращенно Java EE (до версии 5.0 — Java 2 Enterprise Edition или J2EE). В 2018 Eclipse Foundation переименовала Java EE в Jakarta EE — набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий.

Спецификации детализированы настолько, чтобы обеспечить переносимость программ с одной реализации платформы на другую. Основная цель спецификаций — обеспечить масштабируемость приложений и целостность данных во время работы системы. Java EE во многом ориентирована на использование её через веб, как в интернете, так и в локальных сетях. Вся спецификация создаётся и утверждается через JCP (Java Community Process) в рамках инициативы Sun Microsystems Inc.

Java EE является промышленной технологией и в основном используется в высокопроизводительных проектах, в которых необходима надежность, масштабируемость, гибкость.

Популярности Java EE также способствует то, что Sun предлагает бесплатный комплект разработки, SDK, позволяющий предприятиям разрабатывать свои системы, не тратя больших средств. В этот комплект входит сервер приложений GlassFish с лицензией для разработки.

1. РЕАЛИЗАЦИЯ ПРИНЦИПОВ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА В JAVA

В Java для манипулирования объектами в программном коде используются ссылки на объекты (handles). Ссылка хранит в себе некоторый адрес объекта в оперативной памяти. Может быть несколько ссылок на один объект. На какой-то объект может вообще не быть ссылок (тогда он для нас безвозвратно потерян). Ссылка может не ссылаться ни на какой объект — пустая (null) ссылка. Не может быть ссылки в никуда или ссылки на какую-то произвольную область памяти.

Все ссылки имеют имя. Для манипулирования самими ссылками в программном коде необходимо как-то их обозначать. Это делается при помощи имени ссылки. Все ссылки, так или иначе, описываются, при этом каждой ссылке дается имя. Имена ссылок известны программе и встречаются в программном коде там, где нужно манипулировать объектами, на которые они ссылаются.

Все ссылки строго типизированы. При описании ссылки обязательно указывается ее тип. И эта ссылка может ссылаться только на объект данного типа. Попытка присвоить ссылке адрес объекта не того класса пресекается как на этапе трансляции программы (выдаются ошибки трансляции), так и на этапе ее выполнения (возникает исключительная ситуация `ClassCastException`).

Приведем пример описания ссылки: MyType ref;

Здесь `MyType` — имя типа (как и ссылки, все типы имеют имя), `ref` — имя ссылки. После такого описания ссылке `ref` можно присвоить значение — адрес какого-то объекта типа `MyType`.

Создание объектов.

Все объекты в Java создаются только явно, для чего используется операция new:

ref = new MyType();

Здесь создается объект типа `MyType` и его адрес заносится в `ref`. Еще один пример:

MyType ref = new MyType();

Здесь описание ссылки совмещено с инициализацией.

Класс — способ описания типа.

Для описания типов в Java используется механизм классов. За исключением базовых (иначе — элементарных) типов (`int, char, float` и др.) и интерфейсов (что это такое, мы рассмотрим позже), все остальные типы — это классы.

В простейшем случае описание класса выглядит так:

class MyClass

… // тело класса

}

Здесь class — ключевое слово, `MyClass` — имя класса. Внутри фигурных скобок находится тело класса.

Внутри тела класса описываются в произвольном порядке поля и методы класса.

Поля класса и переменные программы.

Как уже отмечалось, в классе можно описать поля класса (fields or instance variable). Поля класса определяют, из каких данных будут состоять объекты этого класса. Поля могут быть ссылками на другие объекты или элементарными данными.

В методах класса могут быть описаны переменные. Их не следует путать с полями класса.

Как и поле класса, переменная может быть либо ссылкой, либо данным базового типа. Описание переменной выглядит точно так же, как и описание поля класса, за исключением того, что ряд описателей не применимы для переменных. Отличаются же

(визуально) переменные от полей местом их описания. Поля класса описываются непосредственно в теле класса, на том же уровне вложенности, что и методы класса. Переменные описываются внутри методов. Пример:

```
class SomeClass {        // Это заголовок класса
      int i = 0;          // Это элементарное данное, поле класса
      MyType ref;         // Это ссылка, тоже поле класса

         int f() {        // Это заголовок метода
         int k = 0;       // Это элементарное данное, переменная
         MyType lref;     // Это ссылка, переменная
           . . .          // Данный метод что-то делает
             }            // Это конец метода
      . . .               // В классе могут быть и другие методы
      }                   // Это конец тела класса
```

Область видимости и время жизни переменных.

В различных языках программирования существуют различные типы или классы переменных — локальные, глобальные, статические и т.п. В Java только один тип переменных — локальные переменные. Время жизни переменной в Java определяется правилом:

- Переменная создается в точке ее описания и существует до момента окончания того блока, в котором находится данное описание.

В Java блок — это то, что начинается открывающей фигурной скобкой '{' и заканчивается закрывающей фигурной скобкой '}'.

Областью видимости переменной (scope) является фрагмент программы от точки ее описания до конца текущего блока.

Область видимости — это статическое понятие, имеющее отношение к какому-то фрагменту текста программы. Время жизни, в отличие от области видимости, — это понятие динамики выполнения программы. Время жизни переменных в Java совпадает с их областью видимости с учетом отличия самих этих понятий.

Если в блоке, где описана данная переменная, вложены другие блоки, то переменная доступна в этих блоках (обычная практика языков программирования). Но, в отличие от многих других языков, в Java запрещено переопределять переменную во вложенных блоках (т.е. описывать другую переменную с тем же именем).

Иная картина наблюдается с объектами. Объекты доступны в программе только через ссылки на них. Поэтому область видимости объекта определяется областью видимости ссылок на этот объект (на один объект может быть сколько угодно ссылок).

Время жизни объекта определяется следующим правилом.

- Объект существует, пока существует хотя бы одна ссылка на этот объект.

Это правило, однако, не утверждает, что объект будет уничтожен, как только пропадет последняя ссылка на него. Просто такой объект становится недоступным и может быть уничтожен.

- В Java нет явного уничтожения объектов. Объекты уничтожаются (говорят — утилизируются) сборщиком мусора (garbage collector), который работает в фоновом режиме параллельно с самой программой на Java.

Рассмотрим следующий фрагмент.

```
{
SomeType localReference = new SomeType();
globalReference = localReference;
}
```

Здесь `SomeType` — это некоторый класс, `localReference` — локальная переменная-ссылка, `globalReference` — некоторая внешняя, по отношению к данному блоку, переменная или поле класса (из данного фрагмента нельзя сделать однозначный вывод, что это).

В этом фрагменте порождается объект класса `SomeType` и адрес этого объекта заносится в переменную `localReference`. После этого этот же адрес из `localReference` копируется в `globalReference`. По выходу из блока переменная `localReference` уничтожается, но переменная (или поле) `globalReference` продолжает существовать. Соответственно, продолжает существовать и порожденный объект.

Описание методов класса.

В первом приближении методы класса (class methods) можно рассматривать как функции.

Описание метода выглядит следующим образом

<тип> <имя_метода> (<аргументы>) {

<тело_метода>

}

Здесь <тип> — это один из базовых типов (см. таблицу выше) или пользовательский тип (т.е. некоторое имя класса). <аргументы> — это список, возможно пустой, параметров метода. <тело_метода> — собственно программный код данного метода.

Каждый аргумент или параметр метода в данном описании — это пара "<тип> <имя_аргумента>". Аргументы отделяются друг от друга запятыми.

Описания методов расположены внутри класса, на том же уровне вложенности скобок, что и описание полей класса. Не может быть описания метода вне класса или внутри другого метода или блока.

Вызов методов.

Вызов методов отличается от вызовов функций в не объектно-ориентированных языках программирования. При вызове обычного (не статического) метода класса обязательно должен быть указан объект этого класса и метод вызывается для этого объекта. Т.е. вызов метода — это вызов метода объекта.

Формальное исключение составляет вызов метода класса из другого (или того же) метода данного класса, в этом случае объект можно не указывать. Но фактически объект и в данном случае имеется, это — тот объект, для которого был вызван вызывающий метод.

Рассмотрим это на примерах. Опишем класс `SomeClass` и в нем методы `f` и `g`.

```
class SomeClass {
int f(int k) {
. . .
}

void g() {
. . .
}
}
```

Здесь описан метод `f` с одним параметром целого типа, возвращающий целое значение и метод `g` без параметров, не возвращающий никакого значения. Приведем примеры вызова этих методов из некоторого фрагмента программы.

```
a.f(x);
b.g();
v = b.f(3);
```

В приведенном фрагменте фигурируют переменные (или поля класса) `a, x, b` и `v`. Переменные `a` и `b` должны быть описаны как ссылки с типом `SomeClass`, переменные `x` и `v` должны быть целочисленными.

Данный фрагмент демонстрирует, что объект, для которого вызывается метод, должен быть указан при помощи ссылки, имя которой записывается перед именем метода через точку.

При вызове метода класса из метода того же класса объект указывать не обязательно. Это, как указано выше, не нарушает того правила, что при вызове метода всегда должен быть определен объект, для которого этот метод вызывается. Просто в данном случае этот объект уже определен при вызове "вызывающего" метода и для него же вызывается "вызываемый" метод.

Доступ к полям класса.

Поля класса не существуют сами по себе (за исключением статических). Они расположены внутри объекта класса. Поэтому при доступе к полю должен быть определен объект.

Как и в случае вызова метода, при обращении к полю класса извне класса объект должен быть указан явно (при помощи ссылки на объект) перед именем поля через точку. Например, если в классе `SomeClass` есть поля `fld1` и `fld2`, а `obj` — ссылка на объект класса `SomeClass`, то

```
obj.fld1 = 2;
x = obj.fld2;
```

являются примерами доступа к полям `fld1`, `fld2`.

Изнутри класса, т.е. из нестатических методов класса, можно обращаться к полям класса напрямую, без указания объекта, поскольку такой объект определен при вызове данного метода.

Передача параметров.

Для параметров функций, методов, процедур в программировании существует понятие тип передачи параметра. Например, существуют понятия "передача параметра по значению" и "передача параметра по ссылке". В Java существует всего один тип передачи параметров — передача по значению. Это означает, что при вызове метода ему передается текущее значение параметра. Внутри метода можно произвольно изменять параметр, но это никак не повлияет, скажем, на переменную, которая была указана в качестве параметра вызова. Дело в том, что при передаче параметра выделяется необходимая область памяти, куда копируется значение параметра, и внутри метода работа идет с этой копией. Она будет уничтожена при выходе из метода.

Это нужно хорошо себе представлять, в особенности, когда передаются ссылки на объекты.

Рассмотрим пример. Пусть `ref` — ссылка на объект, передаваемая в качестве параметра при вызове некоторого метода `h(...)`.

```
h(ref);
```

Внутри метода `h` мы можем изменить параметр метода (т.е. присвоить ему ссылку на другой объект), но это никак не повлияет на саму ссылку `ref`, т.к. при вызове создается копия `ref` и изменяется именно она. С другой стороны мы можем внутри `h` менять данные того объекта, на который ссылается `ref`, и это реально отразится на этом объекте, т.к. создается копия только ссылки, но не самого объекта.

Простейшая программа на языке Java будет представлять собой простой класс с одним методом. Это главный метод с именем main, который должен быть обязательно public и static, а также иметь в качестве параметра массив строк, в который заносятся параметры принимаемые из командной строки.

```
public class Hello
{
public static void main(String[] arg)
{
System.out.println(“Hello”);
}
}
```

Строка System.out.println(“Hello”) выводит текст на консоль.

2. ПАКЕТЫ

Пакет представляет собой набор родственных классов. В Java пакеты играют ту же роль, что и библиотеки в других языках программирования. Для помещения или определения класса к пакету необходимо написать в начале класса оператор

```
package ru.vgtu.util
```

Имя пакета отражает иерархию и соответствует структуре каталогов, т.е. файлы пакета ru.vgtu.util располагаются в каталогах \ru\vgtu\util.

Для работы с классами какого либо пакета необходимо его импортировать:

```
import java.awt.*;
```

Можно импортировать и конкретный класс import java.awt.Frame;

3. КЛЮЧЕВЫЕ СЛОВА

Ключевое слово this – используется для ссылки на текущий объект.

```
void setAge(String age) {      this.age=age;   }
```

Ключевое слово static. Когда вы объявляете что-либо как static, это означает, что данные или метод не привязаны к данному экземпляру класса.

```
Class StaticTest{
static int i=50;
…
}
```

Если вы создадите два объекта StaticTest, для элемента StaticTest.i существует единственный блок памяти.

```
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
```

Как st1.i, так и st2.i будут иметь одинаковые значения.

4. СПЕЦИФИКАТОРЫ ДОСТУПА

В Java различают спецификаторы доступа к самому классу и данным класса. Спецификаторы доступа к данным класса:

public – доступ к члену класса возможен из любого объекта программы.

protected – доступ к члену класса разрешен любому экземпляру класса, всем его классам потомкам, а также всем другим классам пакета.

private – доступ к члену класса разрешен только экземплярам данного класса.

Если спецификатор данных не задан явно, то к этому члену класса имеют доступ все классы пакета и такой тип доступа называется дружественным.

Спецификаторы доступа к самому классу:
public- делает класс открытым, то есть доступным другим классам.
Если спецификатор данных не задан явно, то к этому классу имеют доступ все классы пакета.

Работа с файлами

Пожалуй самым полезным применением выходных потоков является процедура их записи на жесткий диск:
Фрагмент программы, выполняющий эти действия:

```
OutputStream aStream = new FileOutputStream (“путь/имя файла”); // создание файла
byte[] adata={‘1’,’2’};
aStream.write(adata);  // запись в файл
```

Часто удобно работать с файлами с помощью классов RandomAccessFile и File:

```
import java.io. RandomAccessFile
…
RandomAccessFile f = new RandomAccessFile(“c:/1.txt”,”rw”);
f.seek(a);
f.writeBytes(“korolev”);  // запись в конкретное место файла

File f f= new File(“c:/1.txt”);
long l=ff.length;   // определение размера файла
```

5. СОЗДАНИЕ ДОКУМЕНТАЦИИ – УТИЛИТА JAVADOC

Для автоматической генерации документации по вашему классу необходимо в тело класса вставить специальные комментарии. Все команды javadoc обрабатываются только внутри комментариев типа:

```
 /**
…
*/
```

Такие комментарии необходимо вставить перед самим классом, перед всеми полями и методами класса.
Пример:

```
/**
   *@autor korolev
   *@version 0.1
*/
public class docTest {
/**
     *Комментарии к полю
     */
int i;



/**
```

```
    *Комментарии к методу
     *@param name – имя клиента
     *@return возраст клиента
    */

      public int  getAge(String name) {…}
…
}
```

6. ПРОГРАММИРОВАНИЕ ОТНОШЕНИЙ МЕЖДУ КЛАССАМИ

Основные преимущества объектно-ориентированного проектирования следуют из того, что на базе существующих между классами отношений можно построить их иерархию в направлении от общих классов к специализированным. Java поддерживает 4 типа отношений между классами: отношения типа ”является”, ”имеет”, ”использует” и ”создает”,

Отношение типа «является» - когда один из классов представляет собой специализированный вариант другого класса - например попугай одновременно является и птицей.

Отношение типа «имеет» - подразумевает процедуру включения одного объекта в другой. Например попугай имеет крылья.

Отношение типа «использует» - подразумевает передачу какому либо методу класса в качестве параметра экземпляр другого класса. Например попугай использует насест.

Отношение типа «создает» - подразумевает создание экземпляра одного класса в теле другого. Например попугай создает яйца.

Приведем примеры реализации этих типов отношений:

- “**является**”;

```
public class Parrot extends Bird
{
}
```

- «**имеет**»;

```
public class Bird {
   private Wing leftWing;
   private Wing rightWing;
}
```

- «**использует**»;

```
Perch aPerch = new Perch();
theParrot.land(aPerch);
```

- «**создает**».

```
Egg layEgg() {
        Egg theEgg = new Egg();
        Return theEgg;
}
```

7. ИНТЕРФЕЙСЫ

Часто для представления особенностей предметной области встречаются классы, которые логически принадлежат сразу двум различным иерархическим структурам. Например класс Parrot определяет птицу и является одновременно товаром. При разработке языка Java специалисты фирмы Sun предпочли исключить из этого языка возможность множественного наследования классов. Интерфейсы являются

альтернативой прямому множественному наследованию. Выглядит он как абстрактный класс, в котором все методы абстрактные(пустые). Нельзя создать экземпляр абстрактного класса, то есть класса в котором есть abstract метод; Интерфейс представляет собой абстрактное описание поведения объекта.

Определение интерфейсов.

```
interface Product
public interface Product
{
//Определение цены на товар
public void setPrice(float thePrice);
//Продажа одной единицы товара
public void sell();
}
```

Реализация интерфейса:

```
Public class Tparrot extends Tbird implements Product
{
..
   public void setPrice (float thePrice)
     {
          …
      }
  public void sell ()
     {
          …
      }
}
```

8. ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

Исключительные ситуации представляют собой события, которые происходят в процессе выполнения программы и нарушают нормальное следование потока выполнения команд. Некоторые операции, например такие как работа с файлами, работа с базами данных, вообще нельзя выполнить, не обработав возможные исключительные ситуации. Компилятор не пропустит такую программу.

Организация обработки исключительных ситуаций в программе:

1. Поместите нормальный поток команд в блок try.
2. Отметьте, какие ошибки могут возникнуть в процессе выполнения нормальной последовательности команд, и убедитесь, что в каждом случае генерируется исключительная ситуация.
3. Организуйте перехват и обработку всех типов исключительных ситуаций в блоках catch.

Пример обработки ошибок с использованием операторов try и trow:

```
public class FileReader {
public static main (String args[])
{
  try
```

```
 {
  //Открыть файл в режиме чтения, и считать из него данные

  …
  } catch (Exception e)
{
System.err.println(e.getMessage());
}
}
```

9. МНОГОПОТОЧНОСТЬ

В Java самый простой способ создания нового потока состоит в определении класса производного от класса Thread, создании его нового экземпляра и запуска его метода start(). При запуске потока будет вызван его метод run().

```
public class ArchivThread extends Thread
{
private int waits;

        public ArchivThread (int waitTimeout)
        {
                waits=waitTimeout;
        }
        public void run()
        {
                while (true)
{
                try
                        {
                                …
                                sleep(waits);
                                …
                        }
                }
        }
}
```

Для создания и запуска потока необходимо выполнить следующие действия:

```
ArchivThread myThread=new ArchivThread(1000);
myThread.start();
```

10. ИСПОЛЬЗОВАНИЕ ПАКЕТА JDBC ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ

Пакет JDBC позволяет подключиться к реляционной базе данных и взаимодействовать с ней используя язык SQL. Язык SQL – это язык структурированных запросов для управления базами данных.

Перечислим основные команды SQL, необходимые для выполнения лабораторных работ.

Создание базы данных:

create database student
Создание таблиц:
create table books (id char(10 not null, title char(10)).
Вставка, удаление и обновление строк:
insert into books values(“1”,”Java 2”);
delete from books where id=”2”;
update books set title=”Java 1” where id=”1”;
Создание запроса:
select * from books;
select title from books where id=”1”.

Процесс подключения к базе данных с помощью JDBC выполняется в три этапа:
1. Установка связи между Java-программой и диспетчером базы данных.
2. Передача SQL-команды в базу данных с помощью объекта Statement.
3. Чтение полученных результатов из базы данных и использование их в программе.
Пример загрузки драйвера для MySql:
Class.forName(“org.gjt.mm.mysql.Driver”);

После регистрации драйвера его можно применять для подключения к базе данных. Для создания подключения необходимо указать точное место расположения базы данных, а также учетное имя и пароль:

```
connection = DriverManager.getConnection(
“jdbc:mysql://localhost:3306/student”,
“UserName”,”Password”);
```

Объект Statement предназначен для хранения SQL-команд и создается методом createStatement() из объекта Connection.

Объект Statement лучше всего подходит для SQL-операторов, выполняемых один раз. При пересылке объекта Statement базе данных с помощью установленного подключения СУБД запустит SQL-команду и возвратит результат ее выполнения в виде объекта ResultSet:

```
statement = connection.createStatetment();
ResultSet res=statement.executeQuery(
“select * from books”);
```

Если известно, что SQL-команда возвратит целое число, то можно использовать метод executeUpdate():

```
int kolSt = statement.executeUpdate(
“update books set title=”Java 2” where id=”1”);
```

Объект ResultSet функционирует как курсор. Для перехода к следующей строке необходимо вызвать метод next(). Разработаны все методы getТип() для всех основных типов данных которые совместимы с SQL.

```
while (res.next())
{
        String s = rs.getString(1);
}
```

Пример работы с объектом PrepereStatement:

```
String updateSt = “update recycle set id =? where name= ?”;
PrepareStatment pst = connect.prepareStatement(updateSt);
pst.setString(1,”10”);
pst.setString(2,”student”);
int res=pst.executeUpdate();
```

После работы с экземпляром класса ResultSet необходимо вызвать его метод close():
res.close();

Это также касается и классов Connection и Statement:
connection.close();
statement.close();
Получить имена столбцов и типы данных таблицы можно следующим образом:

```
String  sql = “select * from tables”
ResultSet  rs = statement.executeQuery(sql);
ResultSetMetaData  rsMeta = rs.getMetaData();
// Получаем имена столбцов и их тип
String colNames ="";
        int varCol = rsMeta.getColumnCount();
        System.out.println("kol="+varCol);
        for (int col=1;col<varCol;col++)
        {
        colNames=rsMeta.getColumnName(col);
        System.out.println("name="+colNames);
        colNames=rsMeta.getColumnTypeName(col);
        System.out.println("nameType="+colNames);
        }
```

11. АРХИТЕКТУРА СЕРВЛЕТОВ

Сервлет представляет собой программу, работающую на стороне сервера. Архитектура сервлетов состоит в том, что классический сервис обеспечивается методом service(), через который сервлету посылаются все клиентские запросы, а также методами жизненного цикла init() и destroy(), которые вызываются только при загрузке и выгрузке сервлета.

Если вы взглянете на метод service( ) из интерфейса Servlet, вы увидите, что он имеет два параметра: ServletRequest и ServletResponse. У класса HttpServlet эти два объекта расширяются на HTTP: HttpServletRequest и HttpServletResponse. Вот простой пример, который показывает использование HttpServletResponse:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ServletDemo extends HttpServlet{
int  i=0;
public void service(HttpServletRequest req, HttpServletRespons res) throws IOException
        {
        PrintWriter out=res.getWriter();
   out.print("<HEAD><TITLE>");
   out.print("A server-side strategy");
   out.print("</TITLE></HEAD><BODY>");
   out.print("<h1>Servlets Rule! " + i++);
   out.print("</h1></BODY>");
   out.close();
        }
}
```

Сервлет инициализируется только однажды путем вызова своего метода init( ) при загрузке сервлета после того, как сначала загрузиться контейнер сервлетов. Когда клиент

делает запрос к URL, соответствующий представленному сервлету, контейнер сервлетов перехватывает этот запрос и делает вызов метода service( ), затем устанавливает объекты HttpServletRequest и HttpServletResponse.

Главная забота метода service( ) состоит во взаимодействии с HTTP запросом, посланным клиентом (HttpServletRequest), и построение HTTP ответа через объект HttpServletResponse, основываясь на атрибутах, содержащихся в запросе. ServletDemo манипулирует объектом ответа не зависимо от того, что мог послать клиент.

То есть для того чтобы написать сервлет необходимо создать класс наследуемый от класса HttpServlet.

В любое время, когда форма передаст сервлету какую либо информацию, объект HttpServletRequest предводительно загружает все данные формы, хранящиеся в виде пар ключ-значения. Если известны имена полей, то можно их использовать с помощью метода getParameter() для получения нужного значения. HttpServletRespons предназначен для реализации выходного потока. Метод getWriter() позволяет получить выходной поток out, в который сервлет направляет для клиента сформированный код Html.

Можно также получить Enumeration на имена полей, если неизвестны имена всех полей.

Например:

```
 while (flds.hasMoreElements())
        {       String field=(String) flds.nextElement();
                String value=req.getParameter(field);
            }
```

String a=req.getRemoteHost – возвращает Host компьютера вызывающего данный сервлет.

Для того чтобы получить с помощью метода getParameter() значение какого-либо поля, передаваемое в сервлет MyServlet.class, html файл должен иметь форму, описанную следующим образом:

```
<form action= “servlet/MyServlet”>
<input type= “text” name= “login”>
<input type= “password” name= “password”>
<input type= “submit” name= “pasred” value= “ok”>
</form>
```

Тогда значения полей можно будет получить следующим образом:

```
String login = (String)req.getParameter(“login”)
String password = (String)req.getParameter(“password”)
```

Автоматически сгенерированный сервлет средой JBuilder выглядит следующим образом:

```
package servlet_jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Korolev Evgenii
 * @version 1.0
 */
```

```
public class Servlet2 extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  //Initialize global variables
  public void init() throws ServletException {
  }
  //Process the HTTP Get request
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Servlet2</title></head>");
    out.println("<body>");
    out.println("<p>The servlet has received a GET. This is the reply.</p>");
    out.println("</body></html>");
  }

  //Clean up resources
  public void destroy() {
  }
}
```

В данном случае метод service заменен методом doGet, что в большинстве случаев является допустимым.

Приведем примеры двух сервлетов AdderServlet.java и AdderServlet1.java выполняющих соответственно запрос на ввод имени и пароля и обработку этого запроса:

Содержимое файла AdderServlet.java:

```
package ru.sobit.statshop.account.servlet;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AdderServlet extends HttpServlet {
/*----------------------------------------------------------------------*/
  public void init() throws ServletException {
  }
/*----------------------------------------------------------------------*/
  public void doGet (HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
    PrintWriter out = res.getWriter();
       try
      {
       generatePageAccount(out);
         } catch (Exception e) {
System.out.println("Error in generatePageAccount");
              e.printStackTrace();}
  }
private void generatePageAccount(PrintWriter out) {
out.println("<html>");
```

```
out.println("<body>");
out.println("<head>");
out.println("<title>Request Parameters Example</title>");
out.println("</head>");
out.println("<body>");
out.println("<P>");
out.println("<form action=\"http://localhost:8080/Servlet/AdderServlet1\">");
out.println("Name:");
out.println("<input type=text size=20 name=name>");
out.println("<br>");
out.println("Password:");
out.println("<input type=password size=20 name=password>");
out.println("<br>");
out.println("<input type=submit>");
out.println("</form>");
out.println("</body>");
out.println("</html>");
  }
}
```

Содержимое файла AdderServlet1.java:

```
package ru.sobit.statshop.account.servlet;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AdderServlet1 extends HttpServlet {
/*---------------------------------------------------------------------*/
   public void init() throws ServletException {
                    System.out.println("init begin");
   }
/*---------------------------------------------------------------------*/
   public void doGet (HttpServletRequest req,
                              HttpServletResponse res)
                  throws ServletException, IOException {
                    System.out.println("doget begin");
                  PrintWriter out = res.getWriter();
                    String inputString1=null;
                    String inputString2=null;
                              try
                              {
                    inputString1 = req.getParameter("name");
                    inputString2 = req.getParameter("password");
                              } catch (Exception e)
                              {System.out.println("No Passed req.getParameter(inputString);");}
             String str = "Hello "+inputString1;
                    try
                    {     generatePageAccount(out,str );
```

```
                    } catch (Exception e) {System.out.println("Error in
generatePageAccount");
                        e.printStackTrace();}
          }
        /*-------------------------------------------------------------------*/
        private void generatePageAccount(PrintWriter out,String str) {
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<meta HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
charset=windows-1251\">");
        out.println("</HEAD>");
        out.println("<body bgcolor=#FFFFFF BGCOLOR=\"#D3D3D3\" >");
        out.println(str);
            out.println("</BODY>");
            out.println("</HTML>");
          }
        }
```

После размещения сервлетов на сервере можно запустить и проверить их работу. Для запуска необходимо набрать следующую строку в окне браузера: http://localhost:8080//Servlet/AdderServlet

12. ПЛАН ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Вводная работа. Основы Java

✖ Создайте класс Circle. Класс должен иметь три поля: координаты центра окружности (x,y) и диаметр окружности (d). Класс должен иметь два конструктора. Первый конструктор должен быть без параметров, в момент создания объекта он должен запрашивать с клавиатуры координаты центра окружности и его диаметр. Второй конструктор должен принимать значения координат и диаметра окружности в виде параметров. Класс Circle должен иметь метод для определения площади круга. Напишите класс Main с методом main, в котором создайте объекты Circle двумя способами и выведете площади кругов. (Вам понадобится класс java.lang.Math)

✖ Напишите программу сортировки массива из n целых чисел. Число n вводится с клавиатуры. Массив целых чисел формируется случайным образом в диапазоне от 0 до 20. Элемент массива не должен заканчиваться на 0. Вывести на экран массив до сортировки и после сортировки.

✖ В городе N проезд в трамвае осуществляется по бумажным отрывным билетам. Каждую неделю трамвайное депо заказывает в местной типографии рулон билетов с номерами от 000001 до 999999. «Счастливым» считается билетик у которого сумма первых трёх цифр номера равна сумме последних трёх цифр, как, например, в билетах с номерами 003102 или 567576. Трамвайное депо решило подарить сувенир обладателю каждого счастливого билета и теперь раздумывает, как много сувениров потребуется. С помощью программы подсчитайте сколько счастливых билетов в одном рулоне?

✖ Сравнить скорость работы с классами String, StringBuffer и StringBuilder на 10000 операциях конкатенации. Использовать класс java.util.Date.

✖ Проанализируйте программу и определите в какой последовательности будут выводиться надписи на экран

```
class Insect {
  int i = 9;
  int j;
  Insect() {
    prt("i = " + i + ", j = " + j);
    j = 39;
  }
  static int x1 =
    prt("static Insect.x1 initialized");
  static int prt(String s) {
    System.out.println(s);
    return 47;
  }
}

public class Beetle extends Insect {
  int k = prt("Beetle.k initialized");
  Beetle() {
    prt("k = " + k);
    prt("j = " + j);
  }
  static int x2 =
    prt("static Beetle.x2 initialized");
  public static void main(String[] args) {
    prt("Beetle constructor");
    Beetle b = new Beetle();
  }
} ///:~
```

Задание на лабораторную работу № 1 Работа с потоками.

Напишите программу, запускающую 2 независимых потока. Первый поток выполняет запись в файл currentdata.txt строки «the first thread writes the time:» посимвольно, по одному символу в 200 миллисекунд (или реже). Затем записывает в файл время записи строки. После чего поток спит 1 секунду. Для определения даты и времени использовать класс java.util.Calendar. Второй поток каждые 15 секунд проверяет размер файла currentdata.txt и если этот размер превысил 200 байт, то сохраняет файл под уникальным именем и обнуляет файл currentdata.txt. Уникальное имя файла должно содержать дату и время его создания. Программа должна содержать 4 класса: первый класс реализует первый поток, второй класс реализует второй поток, третий класс содержит метод main, из которого запускаются оба потока, четвертый класс содержит методы, используемые 1 и 2 классами. В третьем классе должна быть реализована возможность завершения обоих потоков по

требованию пользователя. Организовать синхронизацию потоков: переименование не должно выполняться при записи в файл первым потоком.

При работе с файлами организовать обработку исключительных ситуаций. Для реализации некоторых операций с файлами используйте классы java.io.File и java.io.FileOutputStream().

Создайте документацию по проекту в формате html с помощью javaDoc.

Задание на лабораторную работу № 2 Коллекции в Java

- ✓ Ввести с клавиатуры 10 фамилий, внести их в ArrayList. Отсортировать их и вывести на экран. Проверить сколько раз в списке встречается каждая фамилия и вывести на экран. Удалить все повторяющиеся фамилии и оставить только в единственном виде.
- ✓ Проверить скорости работы ArrayList или LinkedList для операций: добавление элемента в конец списка, удаление элемента с начала списка. Для проверки используйте 1000000 операций для добавления и 1000 для удаления.
- ✓ Отсортировать коллекцию студентов по среднему балу и вывести их фамилии и средний бал в порядке возрастания среднего бала. Использовать класс TreeMap**.** При этом создать класс Student implements Comparable и в нем реализовать метод compareTo.
- ✓ Реализовать возможность сортировки студентов по фамилии, среднему балу и возрасту с помощью интерфейса Comparator.
- ✓ Создать коллекцию студентов HashSet, в которой не будет храниться двух студентов с одинаковыми фамилиями. Для этого в классе Student переопределить методы hashCode() и equals().

Задание на лабораторную работу № 3 Рефлексия + аннотации

- Написать метод выводящий площадь и имя неизвестного объекта (фигура, квартира, стол и т.д.). Данный метод получает в качестве параметра объект неизвестного типа (Object). Требуется узнать есть ли у этого объекта метод, возвращающий площадь и вызвать его если он есть. Данный объект должен быть помечен аннотацией @haveArea, а метод должен быть помечен аннотацией @returnArea. Имя объекта должно задаваться как параметр “name” аннотации @haveArea.

Задание на лабораторную работу № 4 XML-парсеры

- С помошью DOM создать xml файл (students.xml) на основе ArrayList классов Student.
- С помощью SAX прочитать файл students.xml и создать ArrayList классов Student.

Задание на лабораторную работу № 5 JAXB

Сделать задание из предыдущей лабораторной работы с помощью JAXB.

Задание на лабораторную работу № 6 Сетевое программирование

Написать чат с использованием классов Socket и ServerSocket.

Задание на лабораторную работу № 7 Работа с Сервлетами

Написать четыре сервлета. Первый запрашивает у пользователя логин и пароль. Второй проверяет логин и пароль, выполняя запрос в базу данных. В случае подтверждения правильности отправляет пользователя на страницу приветствия, реализованную третьим сервлетом с выводом из базы данных личной информации о пользователе. В случае отсутствия зарегистрированного пользователя, его отправляют на страницу регистрации, реализованную четвертым сервлетом. После удачной регистрации передать управление первому сервлету.

Задание на лабораторную работу № 8 Работа с JavaServer Pages

Написать две страницы JSP. Первая должна генерировать запрос логина и пароля для входа в систему. Вторая должна анализировать зарегистрирован ли такой пользователь и генерировать соответствующую страницу. Регистрацию пользователя проверять по базе данных, структуру базы данных согласовать с преподавателем. При этом должен вестись журнал посещений, фиксируя количество посещений, IP адрес и время захода на страницу каждым посетителем.