

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Гомельский государственный технический университет имени П.О. Сухого»
Кафедра «Информационные технологии»

Лабораторный практикум
старшего преподавателя Стефановского И.Л.
по дисциплине

«Визуальные средства разработки программных приложений»

для специальности 1–40 01 02 – "Информационные системы и технологии
(по направлениям)»,
направления специальности 1 40 01 02– 01 – «Информационные системы и
технологии (в проектировании и производстве)»

Содержание

ВВЕДЕНИЕ	3
1. Лабораторная работа №1 «Создание собственных классов и интерфейсов в Java »	4
2. Лабораторная работа №2 «Наследование в Java»	11
3. Лабораторная работа №3 «Работа со строками»	15
4. Лабораторная работа №4 «Коллекции»	20
5. Лабораторная работа №5 «Разработка приложений с графическим интерфейсом в Java»	26
6. Лабораторная работа №6 «Потоки ввода-вывода»	32
7. Лабораторная работа №7 «Использование внутренних классов»	40
8. Лабораторная работа №8 «Построение модели программной системы»	45
9. Лабораторная работа №9 «Разработка многопоточных приложений»	52
10. Лабораторная работа №10 «Разработка веб-служб»	60
11. Лабораторная работа №11 «Создание библиотек в Java»	70
12. Лабораторная работа №12 «Обработка данных с использованием XML»	72
Список использованных источников	78

ВВЕДЕНИЕ

Целью дисциплины «Визуальные средства разработки программных приложений» является формирование у студентов теоретических знаний о современных языках программирования высокого уровня, приемах, методах и технологиях конструирования программ различного уровня сложности, обучение студентов основным принципам программирования сложных систем и создания серверных приложений; формирование практических навыков конструирования современных программных комплексов; формирование у студентов теоретических знаний и практических навыков в области объектно-ориентированного программирования.

Основными задачами дисциплины являются следующие:

- усвоение основных понятий объектно-ориентированного программирования;
- усвоение языковых средств, используемых для создания программных комплексов.
- знакомство с функциональными возможностями современных языков программирования в создании серверных приложений;
- овладение навыками работы с основными инструментальными средствами конструирования и создания прикладных программных продуктов различной сложности, используя различные технологии;
- приобретение студентами практических навыков решения задач с использованием современных методов программирования;
- обучение студентов самостоятельной работе и хорошей ориентации в области технологий и программных комплексов.

Лабораторный практикум предназначен для выполнения лабораторных работ по курсу «Компьютерная графика» для студентов специальностей 1-40 01 02 «Информационные системы и технологии» направления специальности 1-40 01 02-01 – «Информационные системы и технологии (в проектировании и производстве)».

Цель данного практикума:

- усвоение основных понятий объектно-ориентированного программирования в языке программирования JAVA;
- усвоение языковых средств JAVA, используемых для создания программных комплексов.
- знакомство с функциональными возможностями современных языков программирования в создании серверных приложений;
- овладение навыками работы с основными инструментальными средствами конструирования и создания прикладных программных продуктов различной сложности, используя различные технологии;
- приобретение студентами практических навыков решения задач с использованием современных методов программирования;
- обучение студентов самостоятельной работе и хорошей ориентации в области технологий и программных комплексов.

Лабораторная работа №1 «Создание собственных классов и интерфейсов в Java »

Цель работы: изучить способы создания собственных классов и интерфейсов в Java

Теоретические сведения

Любой класс, содержащий методы `abstract`, также должен быть объявлен, как `abstract`. Поскольку у таких классов отсутствует полная реализация, их представителей нельзя создавать с помощью оператора `new`. Кроме того, нельзя объявлять абстрактными конструкторы и статические методы. Любой подкласс абстрактного класса либо обязан предоставить реализацию всех абстрактных методов своего суперкласса, либо сам должен быть объявлен абстрактным.

```
abstract class A {
    abstract void callme();
    void metoo() {
        System.out.println("Inside A's metoo method");
    }
}
class B extends A {
    void callme() {
        System.out.println("Inside B's callme method");
    }
}
class Abstract {
    public static void main(String args[]) {
        A a = new B();
        a.callme();
        a.metoo();
    }
}
```

В нашем примере для вызова реализованного в подклассе класса А метода `callme` и реализованного в классе А метода `metoo` используется динамическое назначение методов, которое мы обсуждали раньше.

```
C:\> Java Abstract
```

```
Inside B's callme method
Inside A's metoo method
```

Пакет (`package`) — это некий контейнер, который используется для того, чтобы изолировать имена классов. Например, вы можете создать класс `List`, заключить его в пакет и не думать после этого о возможных конфликтах, которые могли бы возникнуть если бы кто-нибудь еще создал класс с именем `List`.

Интерфейс — это явно указанная спецификация набора методов, которые должны быть представлены в классе, который реализует эту спецификацию. Реализация же этих методов в интерфейсе отсутствует. Подобно абстрактным классам интерфейсы обладают замечательным дополнительным свойством — их можно многократно наследовать. Конкретный класс может быть наследником лишь одного суперкласса, но зато в нем может быть реализовано неограниченное число интерфейсов.

Все идентификаторы, которые мы до сих пор использовали в наших примерах, располагались в одном и том же пространстве имен (name space). Это означает, что нам во избежание конфликтных ситуаций приходилось заботиться о том, чтобы у каждого класса было свое уникальное имя. Пакеты — это механизм, который служит как для работы с пространством имен, так и для ограничения видимости. У каждого файла .java есть 4 одинаковых внутренних части, из которых мы до сих пор в наших примерах использовали только одну. Ниже приведена общая форма исходного файла Java.

одинокый оператор package (необязателен)
любое количество операторов import (необязательны)
одинокое объявление открытого (public) класса
любое количество закрытых (private) классов пакета (необязательны)

Оператор package

Первое, что может появиться в исходном файле Java — это оператор package, который сообщает транслятору, в каком пакете должны определяться содержащиеся в данном файле классы. Пакеты задают набор отдельных пространств имен, в которых хранятся имена классов. Если оператор package не указан, классы попадают в безымянное пространство имен, используемое по умолчанию. Если вы объявляете класс, как принадлежащий определенному пакету, например,

package java.awt.image;
то и исходный код этого класса должен храниться в каталоге java/awt/image.

Трансляция классов в пакетах

При попытке поместить класс в пакет, вы сразу натолкнетесь на жесткое требование точного совпадения иерархии каталогов с иерархией пакетов. Вы не можете переименовать пакет, не переименовав каталог, в котором хранятся его классы. Эта трудность видна сразу, но есть и менее очевидная проблема.

Оператор import

После оператора package, но до любого определения классов в исходном Java-файле, может присутствовать список операторов import. Пакеты являются хорошим механизмом для отделения классов друг от друга,

поэтому все встроенные в Java классы хранятся в пакетах. Общая форма оператора `import` такова:

```
import пакет1 [.пакет2].(имякласса/*);
```

Здесь *пакет1* — имя пакета верхнего уровня, *пакет2* — это необязательное имя пакета, вложенного в первый пакет и отделенное точкой. И, наконец, после указания пути в иерархии пакетов, указывается либо имя класса, либо метасимвол звездочка. Звездочка означает, что, если Java-транслятору потребуется какой-либо класс, для которого пакет не указан явно, он должен просмотреть все содержимое пакета со звездочкой вместо имени класса. В приведенном ниже фрагменте кода показаны обе формы использования оператора `import` :

```
import java.util.Date  
import java.io.*;
```

Ограничение доступа

Java предоставляет несколько уровней защиты, обеспечивающих возможность тонкой настройки области видимости данных и методов. Из-за наличия пакетов Java должна уметь работать еще с четырьмя категориями видимости между элементами классов :

- Подклассы в том же пакете.
- Не подклассы в том же пакете.
- Подклассы в различных пакетах.
- Классы, которые не являются подклассами и не входят в тот же пакет.

В языке Java имеется три уровня доступа, определяемых ключевыми словами: `private` (закрытый), `public` (открытый) и `protected` (защищенный), которые употребляются в различных комбинациях. Содержимое ячеек таблицы определяет доступность переменной с данной комбинацией модификаторов (столбец) из указанного места (строка).

На первый взгляд все это может показаться чрезмерно сложным, но есть несколько правил, которые помогут вам разобраться. Элемент, объявленный `public`, доступен из любого места. Все, что объявлено `private`, доступно только внутри класса, и нигде больше. Если у элемента вообще не указан модификатор уровня доступа, то такой элемент будет виден из подклассов и классов того же пакета. Именно такой уровень доступа используется в языке Java по умолчанию. Если же вы хотите, чтобы элемент был доступен извне пакета, но только подклассам того класса, которому он принадлежит, вам нужно объявить такой элемент `protected`. И наконец, если вы хотите, чтобы элемент был доступен только подклассам, причем независимо от того, находятся ли они в данном пакете или нет — используйте комбинацию `private protected`.

Интерфейсы

Интерфейсы Java созданы для поддержки динамического выбора (resolution) методов во время выполнения программы. Интерфейсы похожи на классы, но в отличие от последних у интерфейсов нет переменных

представителей, а в объявлениях методов отсутствует реализация. Класс может иметь любое количество интерфейсов. Все, что нужно сделать — это реализовать в классе полный набор методов всех интерфейсов. Сигнатуры таких методов класса должны точно совпадать с сигнатурами методов реализуемого в этом классе интерфейса. Интерфейсы обладают своей собственной иерархией, не пересекающейся с классовой иерархией наследования. Это дает возможность реализовать один и тот же интерфейс в различных классах, никак не связанных по линии иерархии классового наследования. Именно в этом и проявляется главная сила интерфейсов. Интерфейсы являются аналогом механизма множественного наследования в C++, но использовать их намного легче.

Оператор interface

Определение интерфейса сходно с определением класса, отличие состоит в том, что в интерфейсе отсутствуют объявления данных и конструкторов. Общая форма интерфейса приведена ниже:

```
interface имя {  
    тип_результата имя_метода1(список параметров);  
    тип имя_final-переменной = значение;  
}
```

Обратите внимание — у объявляемых в интерфейсе методов отсутствуют операторы тела. Объявление методов завершается символом ; (точка с запятой). В интерфейсе можно объявлять и переменные, при этом они неявно объявляются final - переменными. Это означает, что класс реализации не может изменять их значения. Кроме того, при объявлении переменных в интерфейсе их обязательно нужно инициализировать константными значениями. Ниже приведен пример определения интерфейса, содержащего единственный метод с именем callback и одним параметром типа int.

```
interface Callback {  
    void callback(int param);  
}
```

Задание на лабораторную работу

Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы **setТип()**, **getТип()**, **toString()**. Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль.

Варианты:

1. **Student:** id, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Создать массив объектов. Вывести:

- а) список студентов заданного факультета;
- б) списки студентов для каждого факультета и курса;
- с) список студентов, родившихся после заданного года;

d) список учебной группы.

2. **Customer:** id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета.

Создать массив объектов. Вывести:

- a) список покупателей в алфавитном порядке;
- b) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. **Patient:** id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.

Создать массив объектов. Вывести:

- a) список пациентов, имеющих данный диагноз;
- b) список пациентов, номер медицинской карты у которых находится в заданном интервале.

4. **Abiturient:** id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

- a) список абитуриентов, имеющих неудовлетворительные оценки;
- b) список абитуриентов, средний балл у которых выше заданного;
- c) выбрать заданное число n абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

5. **Book:** id, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Переплет.

Создать массив объектов. Вывести:

- a) список книг заданного автора;
- b) список книг, выпущенных заданным издательством;
- c) список книг, выпущенных после заданного года.

6. **House:** id, Номер квартиры, Площадь, Этаж, Количество комнат, Улица, Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

- a) список квартир, имеющих заданное число комнат;
- b) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;
- c) список квартир, имеющих площадь, превосходящую заданную.

7. **Phone:** id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Дебет, Кредит, Время городских и междугородных разговоров.

Создать массив объектов. Вывести:

- a) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;
- b) сведения об абонентах, которые пользовались междугородной

связью;

с) сведения об абонентах в алфавитном порядке.

8. **Car:** id, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.

Создать массив объектов. Вывести:

а) список автомобилей заданной марки;

б) список автомобилей заданной модели, которые эксплуатируются больше n лет;

с) список автомобилей заданного года выпуска, цена которых больше указанной.

9. **Product:** id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество.

Создать массив объектов. Вывести:

а) список товаров для заданного наименования;

б) список товаров для заданного наименования, цена которых не превосходит заданную;

с) список товаров, срок хранения которых больше заданного.

10. **Train:** Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс).

Создать массив объектов. Вывести:

а) список поездов, следующих до заданного пункта назначения;

б) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

с) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

11. **Bus:** Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

а) список автобусов для заданного номера маршрута;

б) список автобусов, которые эксплуатируются больше 10 лет;

с) список автобусов, пробег у которых больше 100000 км.

12. **Airlines:** Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.

Создать массив объектов. Вывести:

а) список рейсов для заданного пункта назначения;

б) список рейсов для заданного дня недели;

с) список рейсов для заданного дня недели, время вылета для которых больше заданного.

13. **Abiturient:** id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

- а) список абитуриентов, имеющих неудовлетворительные оценки;
- б) список абитуриентов, средний балл у которых выше заданного;
- с) выбрать заданное число n абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

14. **Book:** id, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Переплет.

Создать массив объектов. Вывести:

- а) список книг заданного автора;
- б) список книг, выпущенных заданным издательством;
- с) список книг, выпущенных после заданного года.

15. **House:** id, Номер квартиры, Площадь, Этаж, Количество комнат, Улица, Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

- а) список квартир, имеющих заданное число комнат;
- б) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;
- с) список квартир, имеющих площадь, превосходящую заданную.

Контрольные вопросы

1. Правильен ли оператор определения переменной: `double myValue = 1f;`
2. Чем оператор `break` в языке Java отличается от оператора `break` в C/C++?
3. Как виртуальная машина Java (интерпретатор) вычисляет значения выражений?
4. Что такое отрицательная бесконечность?
5. Чем различаются строковые литералы и переменные типов `String`, `StringBuffer` и `StringBuilder`?
6. Какие исключения могут быть возбуждены при арифметической обработке данных?
7. Каким ограничениям должно удовлетворять выражение в предложении `case` переключателя?
8. Охарактеризуйте назначение составных частей оператора перехвата исключений.
9. Где ошибка в этом операторе:
`switch (индексСимвола)`
`{ default : ... break; case 5L: case 7: ... break; ... }`
10. Какие виды литералов существуют в языке Java?
11. Правильно ли объявление: `boolean flag = 0; ?`

Лабораторная работа №2 «Наследование в Java»

Цель работы: изучить механизм наследование в Java

Теоретические сведения

Вторым фундаментальным свойством объектно-ориентированного подхода является наследование (первый – инкапсуляция). Классы-потомки имеют возможность не только создавать свои собственные переменные и методы, но и наследовать переменные и методы классов-предков. Классы-потомки принято называть подклассами. Непосредственного предка данного класса называют его суперклассом. В очередном примере показано, как расширить класс `Point` таким образом, чтобы включить в него третью координату `z`.

```
class Point3D extends Point { int z;  
Point3D(int x, int y, int z) {  
this.x = x;  
this.y = y;  
this.z = z; }  
Point3D() {  
this(-1,-1,-1);  
} }
```

В этом примере ключевое слово `extends` используется для того, чтобы сообщить транслятору о намерении создать подкласс класса `Point`. Как видите, в этом классе не понадобилось объявлять переменные `x` и `y`, поскольку `Point3D` унаследовал их от своего суперкласса `Point`.

super

В примере с классом Point3D частично повторялся код, уже имевшийся в суперклассе. Вспомните, как во втором конструкторе мы использовали `this` для вызова первого конструктора того же класса. Аналогичным образом ключевое слово `super` позволяет обратиться непосредственно к конструктору суперкласса (в Delphi / C++ для этого используется ключевое слово `inherited`).

```
class Point3D extends Point { int z;
Point3D(int x, int y, int z) {
super(x, y);    // Здесь мы вызываем конструктор суперкласса this.z=z;
public static void main(String args[]) {
Point3D p = new Point3D(10, 20, 30);
System.out.println( " x = " + p.x + " y = " + p.y +
                    " z = " + p.z);
} }
```

Вот результат работы этой программы:

```
C:\> java Point3D
```

```
x = 10 y = 20 z = 30
```

Замещение методов

Новый подкласс Point3D класса Point наследует реализацию метода distance своего суперкласса (пример PointDist.java). Проблема заключается в том, что в классе Point уже определена версия метода distance(mt x, int y), которая возвращает обычное расстояние между точками на плоскости. Мы должны *заместить* (override) это определение метода новым, пригодным для случая трехмерного пространства. В следующем примере проиллюстрировано и *совмещение* (overloading), и *замещение* (overriding) метода distance.

```
class Point { int x, y;
Point(int x, int y) {
this.x = x;
this.y = y;
}
double distance(int x, int y) {
int dx = this.x - x;
int dy = this.y - y;
return Math.sqrt(dx*dx + dy*dy);
}
double distance(Point p) {
return distance(p.x, p.y);
}
}
class Point3D extends Point { int z;
Point3D(int x, int y, int z) {
super(x, y);
this.z = z;
(
double distance(int x, int y, int z) {
int dx = this.x - x;
int dy = this.y - y;
int dz = this.z - z;
return Math.sqrt(dx*dx + dy*dy + dz*dz);
}
double distance(Point3D other) {
return distance(other.x, other.y, other.z);
}
double distance(int x, int y) {
double dx = (this.x / z) - x;
double dy = (this.y / z) - y;
```

```

return Math.sqrt(dx*dx + dy*dy);
}
}
class Point3DDist {
public static void main(String args[]) {
Point3D p1 = new Point3D(30, 40, 10);
Point3D p2 = new Point3D(0, 0, 0);
Point p = new Point(4, 6);
System.out.println("p1 = " + p1.x + " ", " + p1.y + " ", " + p1.z);
System.out.println("p2 = " + p2.x + " ", " + p2.y + " ", " + p2.z);
System.out.println("p = " + p.x + " ", " + p.y);
System.out.println("p1.distance(p2) = " + p1.distance(p2));
System.out.println("p1.distance(4, 6) = " + p1.distance(4, 6));
System.out.println("p1.distance(p) = " + p1.distance(p));
} }

```

Ниже приводится результат работы этой программы:

```

C:\> Java Point3DDist
p1 = 30, 40, 10
p2 = 0, 0, 0
p = 4, 6
p1.distance(p2) = 50.9902
p1.distance(4, 6) = 2.23607
p1.distance(p) = 2.23607

```

Обратите внимание — мы получили ожидаемое расстояние между трехмерными точками и между парой двумерных точек. В примере используется механизм, который называется *динамическим назначением методов* (dynamic method dispatch).

Динамическое назначение методов

Давайте в качестве примера рассмотрим два класса, у которых имеют простое родство подкласс / суперкласс, причем единственный метод суперкласса замещен в подклассе.

```

class A { void callme() {
System.out.println("Inside A's callrne method");
class B extends A { void callme() {
System.out.println("Inside B's callme method");
} }
class Dispatch {
public static void main(String args[]) {
A a = new B();
a.callme();
} }

```

Обратите внимание — внутри метода `main` мы объявили переменную `a` класса `A`, а проинициализировали ее ссылкой на объект класса `B`. В следующей строке мы вызвали метод `callme`. При этом транслятор проверил наличие метода `callme` у класса `A`, а исполняющая система, увидев, что на самом деле в переменной хранится представитель класса `B`, вызвала не метод класса `A`, а `callme` класса `B`. Ниже приведен результат работы этой программы:

```
C:\> Java Dispatch  
Inside B's calime method
```

Задание на лабораторную работу

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы `equals()`, `hashCode()`, `toString()`.

Варианты:

1. Создать объект класса **Текст**, используя класс **Абзац**. Методы: дополнить текст, вывести на консоль текст, заголовок текста.
2. Создать объект класса **Автомобиль**, используя класс **Колесо**. Методы: ехать, заправляться, менять колесо, вывести на консоль марку автомобиля.
3. Создать объект класса **Самолет**, используя класс **Крыло**. Методы: летать, задавать маршрут, вывести на консоль маршрут.
4. Создать объект класса **Беларусь**, используя класс **Область**. Методы: вывести на консоль столицу, количество областей, площадь, областные центры.
5. Создать объект класса **Планета**, используя класс **Материк**. Методы: вывести на консоль название материка, планеты, количество материков.
6. Создать объект класса **Звездная система**, используя классы **Планета**, **Звезда**, **Луна**. Методы: вывести на консоль количество планет в звездной системе, название звезды, добавление планеты в систему.
7. Создать объект класса **Компьютер**, используя классы **Винчестер**, **Дисковод**, **ОЗУ**. Методы: включить, выключить, проверить на вирусы, вывести на консоль размер винчестера.
8. Создать объект класса **Квадрат**, используя классы **Точка**, **Отрезок**. Методы: задание размеров, растяжение, сжатие, поворот, изменение цвета.
9. Создать объект класса **Круг**, используя классы **Точка**, **Окружность**. Методы: задание размеров, изменение радиуса, определение принадлежности точки данному кругу.
10. Создать объект класса **Котёнок**, используя классы **Животное**, **Кошка**. Методы: вывести на консоль имя, подать голос, рожать потомство (создавать себе подобных).

11. Создать объект класса **Наседка**, используя классы **Птица**, **Кукушка**. Методы: летать, петь, нести яйца, высиживать птенцов.
12. Создать объект класса **Текстовый файл**, используя класс **Файл**. Методы: создать, переименовать, вывести на консоль содержимое, дополнить, удалить.
13. Создать объект класса **Одномерный массив**, используя класс **Массив**. Методы: создать, вывести на консоль, выполнить операции (сложить, вычесть, перемножить).
14. Создать объект класса **Простая дробь**, используя класс **Число**. Методы: вывод на экран, сложение, вычитание, умножение, деление.
15. Создать объект класса **Дом**, используя классы **Окно**, **Дверь**. Методы: закрыть на ключ, вывести на консоль количество окон, дверей.

Контрольные вопросы

1. Что такое сигнатура метода? А контракт метода?
2. Что такое интерфейс?
3. Что можно сделать при помощи переменной `super`?
4. В чем отличие результата применения модификатора `final` к классу и к переменной?
5. Что такое абстрактный метод?
6. Будет ли ошибка выдана компилятором, если класс реализует два интерфейса, имеющих методы с одинаковой сигнатурой?
7. Можно ли объявить двумерный массив так:
`int[] twoDimArray[] = new int[10][];`?
8. Чем абстрактный класс отличается от интерфейса?
9. Что произойдет с переменной `nameOfChild` в результате выполнения операторов:
`String nameOfChild = "Bill";`
`nameOfChild += " Gates";`
10. Что делает конструктор класса? Должен ли он обязательно явно присутствовать в объявлении класса?
11. Какие существуют виды ссылочных типов? Как реализуются ссылочные переменные?

Лабораторная работа №3 «Работа со строками»

Цель работы: изучить классы, используемые в Java, для работы со строками

Теоретические сведения

В языках C и C++ отсутствует встроенная поддержка такого объекта, как строка. В них при необходимости передается адрес последовательности байтов, содержимое которых трактуется как символы до тех пор, пока не

будет встречен нулевой байт, отмечающий конец строки. В пакет `java.lang` встроен класс, инкапсулирующий структуру данных, соответствующую строке. Этот класс, называемый `String`, не что иное, как объектное представление неизменяемого символьного массива. В этом классе есть методы, которые позволяют сравнивать строки, осуществлять в них поиск и извлекать определенные символы и подстроки. Класс `StringBuffer` используется тогда, когда строку после создания требуется изменять.

Конструкторы

Как и в случае любого другого класса, вы можете создавать объекты типа `String` с помощью оператора `new`. Для создания пустой строки используется конструктор без параметров:

```
String s = new String();
```

Приведенный ниже фрагмент кода создает объект `s` типа `String` инициализируя его строкой из трех символов, переданных конструктору в качестве параметра в символьном массиве.

```
char chars[] = { 'a', 'b', 'c' };  
String s = new String(chars);  
System.out.println(s);
```

Этот фрагмент кода выводит строку «abc». Итак, у этого конструктора — 3 параметра:

```
String(char chars[], int начальныйИндекс, int числоСимволов);
```

Используем такой способ инициализации в нашем очередном примере:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };  
String s = new String(chars,2,3);  
System.out.println(s);
```

Этот фрагмент выведет «cde».

Специальный синтаксис для работы со строками

В Java включено несколько приятных синтаксических дополнений, цель которых — помочь программистам в выполнении операций со строками. В числе таких операций создание объектов типа `String` слияние нескольких строк и преобразование других типов данных в символьное представление.

Создание строк

Java включает в себя стандартное сокращение для этой операции — запись в виде литерала, в которой содержимое строки заключается в пару двойных кавычек. Приводимый ниже фрагмент кода эквивалентен одному из предыдущих, в котором строка инициализировалась массивом типа `char`.

```
String s = "abc";  
System.out.println(s);
```

Один из общих методов, используемых с объектами `String` — метод `length`, возвращающий число символов в строке. Очередной фрагмент выводит число 3, поскольку в используемой в нем строке — 3 символа.

```
String s = "abc";
```



```
System.out.println(s.length);
```

В Java интересно то, что для каждой строки-литерала создается свой представитель класса String, так что вы можете вызывать методы этого класса непосредственно со строками-литералами, а не только со ссылочными переменными. Очередной пример также выводит число 3.

```
System.out.println("abc".Length());
```

Слияние строк

Строку

```
String s = «He is » + age + " years old.";
```

в которой с помощью оператора + три строки объединяются в одну, прочесть и понять безусловно легче, чем ее эквивалент, записанный с явными вызовами тех самых методов, которые неявно были использованы в первом примере:

```
String s = new StringBuffer("He is ").append(age);  
s.append(" years old.").toString();
```

По определению каждый объект класса String не может изменяться. Нельзя ни вставить новые символы в уже существующую строку, ни поменять в ней одни символы на другие. И добавить одну строку в конец другой тоже нельзя. Поэтому транслятор Java преобразует операции, выглядящие, как модификация объектов String, в операции с родственным классом StringBuffer.

Последовательность выполнения операторов

Давайте еще раз обратимся к нашему последнему примеру:

```
String s = "He is " + age + " years old.";
```

В том случае, когда age — не String, а переменная, скажем, типа int, в этой строке кода заключено еще больше магии транслятора. Целое значение переменной int передается совмещенному методу append класса StringBuffer, который преобразует его в текстовый вид и добавляет в конец содержащейся в объекте строки. Вам нужно быть внимательным при совместном использовании целых выражений и слияния строк, в противном случае результат может получиться совсем не тот, который вы ждали. Взгляните на следующую строку:

```
String s = "four: " + 2 + 2;
```

Быть может, вы надеетесь, что в s будет записана строка «four: 4»? Не угадали — с вами сыграла злую шутку последовательность выполнения операторов. Так что в результате получается "four: 22".

Для того, чтобы первым выполнилось сложение целых чисел, нужно использовать скобки :

```
String s = "four: " + (2 + 2);
```

Преобразование строк

В каждом классе `String` есть метод `toString` — либо своя собственная реализация, либо вариант по умолчанию, наследуемый от класса `Object`. Класс в нашем очередном примере замещает наследуемый метод `toString` своим собственным, что позволяет ему выводить значения переменных объекта.

```
class Point {  
    int x, y;  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    public String toString() {  
        return "Point[" + x + ", " + y + "]";  
    }  
}  
class toStringDemo {  
    public static void main(String args[]) {  
        Point p = new Point(10, 20);  
        System.out.println("p = " + p);  
    }  
}
```

Ниже приведен результат, полученный при запуске этого примера.

```
C:\> Java toStringDemo  
p = Point[10, 20]
```

Задание на лабораторную работу

Создать приложение, удовлетворяющее требованиям, приведенным в задании.

1. В тексте нет слов, начинающихся одинаковыми буквами. Напечатать слова текста в таком порядке, чтобы последняя буква каждого слова совпадала с первой буквой последующего слова. Если все слова нельзя напечатать в таком порядке, найти такую цепочку, состоящую из наибольшего количества слов.
2. Найти наибольшее количество предложений текста, в которых есть одинаковые слова.
3. Найти такое слово в первом предложении, которого нет ни в одном из остальных предложений.
4. Во всех вопросительных предложениях текста найти и напечатать без повторений слова заданной длины.
5. В каждом предложении текста поменять местами первое слово с последним, не изменяя длины предложения.

6. В предложении из n слов первое слово поставить на место второго, второе – на место третьего, и т.д., $(n-1)$ -е слово – на место n -го, n -е слово поставить на место первого. В исходном и преобразованном предложениях между словами должны быть или один пробел, или знак препинания и один пробел.
7. Текст шифруется по следующему правилу: из исходного текста выбирается 1, 4, 7, 10-й и т.д. (до конца текста) символы, затем 2, 5, 8, 11-й и т.д. (до конца текста) символы, затем 3, 6, 9, 12-й и т.д. Зашифровать заданный текст.
8. На основании правила кодирования, описанного в предыдущем примере, расшифровать заданный набор символов.
9. Напечатать слова русского текста в алфавитном порядке по первой букве. Слова, начинающиеся с новой буквы, печатать с красной строки.
10. Рассортировать слова русского текста по возрастанию доли гласных букв (отношение количества гласных к общему количеству букв в слове).
11. Слова английского текста, начинающиеся с гласных букв, рассортировать в алфавитном порядке по первой согласной букве слова.
12. Все слова английского текста рассортировать по возрастанию количества заданной буквы в слове. Слова с одинаковым количеством расположить в алфавитном порядке.
13. Найти такое слово в первом предложении, которого нет ни в одном из остальных предложений.
14. Во всех вопросительных предложениях текста найти и напечатать без повторений слова заданной длины.
15. В каждом предложении текста поменять местами первое слово с последним, не изменяя длины предложения.

Контрольные вопросы

1. В чем особенности строковых переменных?
2. Как переменные различных видов передаются в качестве параметров методам?
3. Что можно сделать при помощи переменной `this`?
4. Что такое элементы класса и элементы экземпляра класса, чем они отличаются друг от друга? Как нужно указывать, что переменная или метод является элементом класса, а не экземпляра?
5. Для чего используются модификаторы доступа? Какие существуют модификаторы доступа, как они ограничивают доступ к элементам?
6. Что позволяет делать процесс наследования?
7. Что такое суперкласс и подкласс?
8. Как ведут себя строковые переменные при передаче их в качестве параметров?
9. Что такое повторное использование кода?
10. Какие заранее определенные переменные содержит каждый класс Java?

11. Что такое скрытие переменной, затемнение переменной и замещение метода?

Лабораторная работа №4 «Коллекции»

Цель работы: изучить работу с коллекциями в Java

Теоретические сведения

Библиотека классов языка включает в себя набор вспомогательных классов, широко используемых в других встроенных пакетах Java. Эти классы расположены в пакетах `java.lang` и `java.util`. Они используются для работы с наборами объектов, взаимодействия с системными функциями низкого уровня, для работы с математическими функциями, генерации случайных чисел и манипуляций с датами и временем.

Простые оболочки для типов

Как вы уже знаете, Java использует встроенные примитивные типы данных, например, `int` и `char` ради обеспечения высокой производительности. Эти типы данных не принадлежат к классовой иерархии Java. Они передаются методам по значению, передать их по ссылке невозможно. По этой причине для каждого примитивного типа в Java реализован специальный класс.

Number

Абстрактный класс `Number` представляет собой интерфейс для работы со всеми стандартными скалярными типами: — `long`, `int`, `float` и `double`.

У этого класса есть методы доступа к содержимому объекта, которые возвращают (возможно округленное) значение объекта в виде значения каждого из примитивных типов:

- `doubleValue()` возвращает содержимое объекта в виде значения типа `double`.
- `floatValue()` возвращает значение типа `float`.
- `intValue()` возвращает значение типа `int`.
- `longValue()` возвращает значение типа `long`.

Double и Float

`Double` и `Float` — подклассы класса `Number`. В дополнение к четырем методам доступа, объявленным в суперклассе, эти классы содержат несколько сервисных функций, которые облегчают работу со значениями `double` и `float`. У каждого из классов есть конструкторы, позволяющие инициализировать объекты значениями типов `double` и `float`, кроме того, для удобства пользователя, эти объекты можно инициализировать и объектом `String`, содержащим текстовое представление вещественного числа. Приведенный ниже пример иллюстрирует создание представителей класса `Double` с помощью обоих конструкторов.

```
class DoubleDemo {
```

```
public static void main(String args[]) {  
    Double d1 = new Double(3.14159);  
    Double d2 = new Double("3.14159E-5");  
    System.out.println(d1 + " = " + d2 + " -> " + d1.equals(d2));  
}
```

Как вы можете видеть из результата работы этой программы, метод `equals` возвращает значение `true`, а это означает, что оба использованных в примере конструктора создают идентичные объекты класса `Double`.

```
C:\> java DoubleDemo  
3.14159 = 3.14159 -> true
```

Integer и Long

Класс `Integer` — класс-оболочка для чисел типов `int`, `short` и `byte`, а класс `Long` — соответственно для типа `long`. Помимо наследуемых методов своего суперкласса `Number`, классы `Integer` и `Long` содержат методы для разбора текстового представления чисел, и наоборот, для представления чисел в виде текстовых строк. Различные варианты этих методов позволяют указывать основание (систему счисления), используемую при преобразовании. Обычно используются двоичная, восьмеричная, десятичная и шестнадцатеричная системы счисления.

Character

`Character` — простой класс-оболочка типа `char`. У него есть несколько полезных статических методов, с помощью которых можно выполнять над символом различные проверки и преобразования.

Boolean

Класс `Boolean` — это очень тонкая оболочка вокруг логических значений, она бывает полезна лишь в тех случаях, когда тип `boolean` требуется передавать по ссылке, а не по значению.

Перечисления

В Java для хранения групп однородных данных имеются массивы. Они очень полезны при использовании простых моделей доступа к данным. Перечисления же предлагают более совершенный объектно-ориентированный путь для хранения наборов данных сходных типов. Перечисления используют свой собственный механизм резервирования памяти, и их размер может увеличиваться динамически. У них есть интерфейсные методы для выполнения итераций и для просмотра. Их можно индексировать чем-нибудь более полезным, нежели простыми целыми значениями.

Интерфейс Enumeration

`Enumeration` — простой интерфейс, позволяющий вам обрабатывать элементы любой коллекции объектов. В нем задается два метода. Первый из них — метод `hasMoreElements`, возвращающий значение типа `boolean`. Он возвращает значение `true`, если в перечислении еще остались элементы, и `false`, если у данного элемента нет следующего. Второй метод — `nextElement` — возвращает обобщенную ссылку на объект класса `Object`, которую, прежде

чем использовать, нужно преобразовать к реальному типу содержащихся в коллекции объектов.

Ниже приведен пример, в котором используется класс Enum, реализующий перечисление объектов класса Integer, и класс EnumerateDemo, создающий объект типа Enum, выводящий все значения перечисления. Обратите внимание на то, что в объекте Enum не содержится реальных данных, он просто возвращает последовательность создаваемых им объектов Integer.

```
import java.util.Enumeration;
class Enum implements Enumeration {
    private int count = 0;
    private boolean more = true;
    public boolean hasMoreElements() {
        return more;
    }
    public Object nextElement() {
        count++;
        if (count > 4) more = false;
        return new Integer(count);
    }
}
class EnumerateDemo {
    public static void main(String args[]) {
        Enumeration enum = new Enum();
        while (enum.hasMoreElements()) {
            System.out.println(enum.nextElement());
        }
    }
}
Vector
```

Vector — это способный увеличивать число своих элементов массив ссылок на объекты. Внутри себя Vector реализует стратегию динамического расширения, позволяющую минимизировать неиспользуемую память и количество операций по выделению памяти. Объекты можно либо записывать в конец объекта Vector с помощью метода addElement, либо вставлять в указанную индексом позицию методом insertElementAt. Вы можете также записать в Vector массив объектов, для этого нужно воспользоваться методом copyInto. После того, как в Vector записана коллекция объектов, можно найти в ней индивидуальные элементы с помощью методов Contains, indexOf и lastIndexOf. Кроме того методы elementAt, firstElement и lastElement позволяют извлекать объекты из нужного положения в объекте Vector.

Stack

Stack — подкласс класса Vector, который реализует простой механизм типа “первым вошел — первым вышел” (FIFO). В дополнение к стандартным методам своего родительского класса, Stack предлагает метод push для помещения элемента в вершину стека и pop для извлечения из него верхнего элемента. С помощью метода peek вы можете получить верхний элемент, не

удаляя его из стека. Метод `empty` служит для проверки стека на наличие элементов — он возвращает `true`, если стек пуст. Метод `search` ищет заданный элемент в стеке, возвращая количество операций `pop`, которые требуются для того чтобы перевести искомым элемент в вершину стека. Если заданный элемент в стеке отсутствует, этот метод возвращает `-1`.

Ниже приведен пример программы, которая создает стек, заносит в него несколько объектов типа `Integer`, а затем извлекает их.

```
import java.util.Stack;
import java.util.EmptyStackException;
class StackDemo {
    static void showpush(Stack st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack st = new Stack();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        }
        catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}
```

Ниже приведен результат, полученный при запуске этой программы. Обратите внимание на то, что обработчик исключений реагирует на попытку извлечь данные из пустого стека. Благодаря этому мы можем аккуратно обрабатывать ошибки такого рода.

Dictionary

`Dictionary` (словарь) — абстрактный класс, представляющий собой хранилище информации типа “ключ-значение”. Ключ — это имя, по которому осуществляется доступ к значению. Имея ключ и значение, вы

можете записать их в словарь методом `put(key, value)`. Для получения значения по заданному ключу служит метод `get(key)`. И ключи, и значения можно получить в форме перечисления (объект `Enumeration`) методами `keys` и `elements`. Метод `size` возвращает количество пар “ключ-значение”, записанных в словаре, метод `isEmpty` возвращает `true`, если словарь пуст. Для удаления ключа и связанного с ним значения предусмотрен метод `remove(key)`.

HashTable

`HashTable` — это подкласс `Dictionary`, являющийся конкретной реализацией словаря. Представителя класса `HashTable` можно использовать для хранения произвольных объектов, причем для индексации в этой коллекции также годятся любые объекты. Наиболее часто `HashTable` используется для хранения значений объектов, ключами которых служат строки (то есть объекты типа `String`).

Задание на лабораторную работу

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода

Варианты:

1. В кругу стоят N человек, пронумерованных от 1 до N . При ведении счета по кругу вычеркивается каждый второй человек, пока не останется один. Составить две программы, моделирующие процесс. Одна из программ должна использовать класс **`ArrayList`**, а вторая – **`LinkedList`**. Какая из двух программ работает быстрее? Почему?
2. Задан список целых чисел и число X . Не используя вспомогательных объектов и не изменяя размера списка, переставить элементы списка так, чтобы сначала шли числа, не превосходящие X , а затем числа, большие X .
3. Написать программу, осуществляющую сжатие английского текста. Построить для каждого слова в тексте оптимальный префиксный код по алгоритму Хаффмена. Использовать класс **`PriorityQueue`**.
4. Реализовать класс `Graph`, представляющий собой неориентированный граф. В конструкторе класса передается количество вершин в графе. Методы должны поддерживать быстрое добавление и удаление ребер.
5. На базе коллекций реализовать структуру хранения чисел с поддержкой следующих операций:
 - добавление/удаление числа;
 - поиск числа, наиболее близкого к заданному (т.е. модуль разницы минимален).
6. Реализовать класс, моделирующий работу N -местной автостоянки. Машина подъезжает к определенному месту и едет вправо, пока не

встретится свободное место. Класс должен поддерживать методы, обслуживающие приезд и отъезд машины.

7. Во входном файле хранятся две разреженные матрицы A и B. Построить циклически связанные списки SA и SB, содержащие ненулевые элементы соответственно матриц A и B. Просматривая списки, вычислить: а) сумму $S = A + B$; б) произведение $P = A * B$.

8. Во входном файле хранятся наименования некоторых объектов. Построить список C1, элементы которого содержат наименования и шифры данных объектов, причем элементы списка должны быть упорядочены по возрастанию шифров. Затем “сжать” список C1, удаляя дублирующие наименования объектов.

9. Во входном файле расположены два набора положительных чисел; между наборами стоит отрицательное число. Построить два списка C1 и C2, элементы которых содержат соответственно числа 1-го и 2-го набора таким образом, чтобы внутри одного списка числа были упорядочены по возрастанию. Затем объединить списки C1 и C2 в один упорядоченный список, изменяя только значения полей ссылочного типа.

10. Во входном файле хранится информация о системе главных автодорог, связывающих г. Минск с другими городами Беларуси. Используя эту информацию, постройте дерево, отображающее систему дорог республики, а затем, продвигаясь по дереву, определить минимальный по длине путь из г. Минска в другой заданный город. Предусмотреть возможность для последующего сохранения дерева в виртуальной памяти.

11. Один из способов шифрования данных, называемый «двойным шифрованием», заключается в том, что исходные данные при помощи некоторого преобразования последовательно шифруются на некоторые два ключа K1 и K2. Разработать и реализовать эффективный алгоритм, позволяющий находить ключи K1 и K2 по исходной строке и ее зашифрованному варианту. Проверить, оказался ли разработанный способ действительно эффективным, протестировав программу для случая, когда оба ключа K1 и K2 являются 20-битными (время ее работы не должно превосходить одной минуты).

12. На плоскости задано N точек. Вывести в файл описания всех прямых, которые проходят более чем через одну точку из заданных. Для каждой прямой указать, через сколько точек она проходит. Использовать класс **HashMap**.

Контрольные вопросы

1. Для чего используются классы-коллекции?
2. Что такое класс Vector? Для чего он используется? Каковы его достоинства и недостатки? Какая структура данных используется в классе Vector?
3. Назовите особенности организации класса Stack.
4. Что такое wildcards в технологии generic?
5. Что такое регулярное выражение и для чего оно используется?

6. Для чего применяется класс Hashtable? Какая структура данных используется в классе Hashtable?
7. Что такое методы generic? Зачем они применяются? Приведите примеры.
8. Что такое коэффициент загрузки?
9. Назначение и особенности применения класса Properties.
10. Каково назначение интерфейса Collection?
11. Опишите возможности применения интерфейсов Map, Set и List.

Лабораторная работа №5 «Разработка приложений с графическим интерфейсом в Java»

Цель работы: изучить создание приложений с графическим интерфейсом в Java

Теоретические сведения

Построение графического интерфейса GUI в Java кажется достаточно простым. Но простота оказывается не такой простой, если учесть, что имеются две отдельных, но тесно связанных между собой библиотеки графических классов: Abstract Windows Toolkit (AWT) и Java Foundation Classes (JFC), которая известна как Swing. AWT построен на основе графической системы каждой платформы, на которой выполняется JVM. Например, когда создается кнопка на Windows, AWT создает контрольный элемент из графической библиотеки Windows. Чтобы избавиться от такой зависимости, была разработана библиотека на самой Java. Именно она называется Java Foundation Classes (Swing). AWT осталась для совместимости. Разработчики Java быстро поняли преимущества классов Swing и стали широко их использовать. Разработчики апплетов, однако, еще долго использовали AWT из-за того, что браузеры не поддерживали Swing. Наконец, есть еще одна проблема, связанная с тем, что пользователи отключают в браузерах поддержку Java. Обычно это объясняется проблемами безопасности и рекомендациями, что Java должна быть отключена. Другая группа программистов требует, чтобы была установлена Java 2 с поддержкой Java. Обычно такой подход можно использовать только для корпоративных приложений, когда администраторы могут контролировать пользовательские компьютеры. Но так как примерно 90% Web-приложений используются неконтролируемыми пользователями, приходится до сих пор применять AWT. Контейнер – это объект, который может содержать другие графические объекты. В этой лекции рассматриваются вопросы использования AWT контейнеров. Контейнеры могут помещаться в другие контейнеры и т. д. Основные классы контейнеров:

- **Applet**— это контейнер для использования в браузере.
- **Frame**— это окно верхнего уровня с заголовком и границей.

- **Panel**— это прямоугольник, в который можно помещать другие компоненты, в том числе и другие объекты Panel.
- **ScrollPane**— это панель, в которой автоматически реализован вертикальный и горизонтальный скроллинг.
- **Dialog**— это диалоговое окно.

Рассмотрим эти контейнеры, а затем компоненты, которые могут размещаться в них.

Апплеты (Applet)

На самом деле при создании апплета используется базовый класс `java.applet.Applet`, который обеспечивает всю функциональность, необходимую для связи с браузером и его виртуальной машиной (JVM), поэтому написание апплета выполняется очень просто: необходимо переопределить некоторые методы, чтобы апплет соответствовал предъявляемым требованиям.

Класс `java.applet.Applet` есть в JVM, которая является частью браузера. Когда браузер получает файл HTML с Web-сервера, он находит тэг `applet` и выгружает требуемый файл `.class` для указанного апплета. Апплет должен наследовать класс `java.applet.Applet` содержать переопределенные методы. JVM будет обращаться к этим методам и, соответственно, вызывать переопределенные.

Запуск апплетов

Напишем простой апплет в несколько строк:

```
import java.awt.*;
import java.applet.Applet;

public class HelloApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Hello, Applet", 10, 10);
    }
}
import java.applet.Applet;
```

В этом простом примере апплет просто выводит на странице строку "Hello, Applet". Для того, чтобы использовать апплет, в HTML-файл вставляется тег – ссылка на апплет.

```
This is the Hello Applet
<applet
code=HelloApplet
```

```
width=200  
height=200>  
</applet>
```

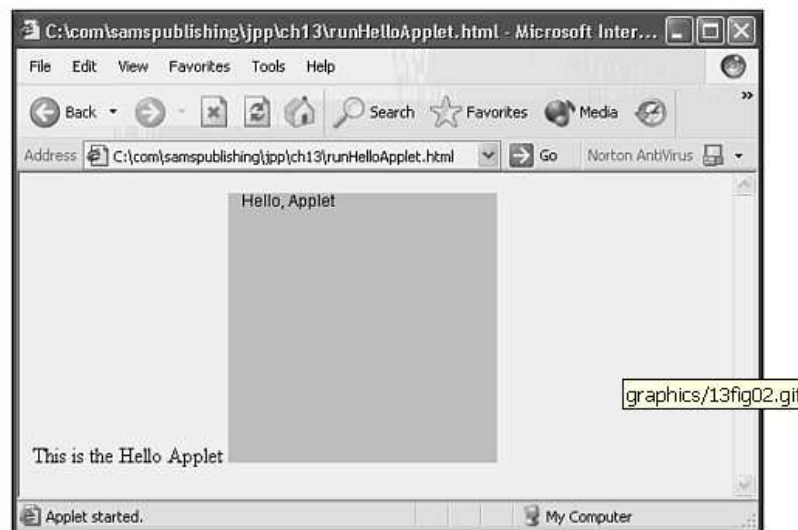
Тег `<applet>` указывает, что апплет с именем `HelloApplet.class` должен быть выгружен с того же сервера, откуда и сам HTML-файл. Размер области, занятой апплетом, указывается в параметрах `width` и `height`. Для запуска HTML-файла с апплетом можно воспользоваться разными способами. Во-первых, можно просто запустить браузер и указать в строке адреса полный путь к файлу:

`C:\runHelloApplet.html`

Если на компьютере установлен Web-сервер, то нужно поместить и HTML-файл и класс апплета в директорию Web-сервера и указать в строке адреса браузера:

`http://127.0.0.1/runHelloApplet.html`

В любом случае запустится файл `runHelloApplet.html` в браузере:



Серая часть страницы – область апплета. Фраза "This is the Hello Applet" помещена кодом HTML. А фраза "Hello, Applet" выводится в окно браузера кодом апплета. Хотя апплеты и не самое важное в Java сейчас, с их помощью можно создавать интересные Web-страницы.

Фреймы (Frame)

Класс `Frame` – контейнер, который чаще всего используется для создания приложений на Java. Даже при создании `Panels` и `ScrollPanels` их обычно помещают в объект `Frame`. Создать объект типа `Frame` можно двумя

способами. Первый – создание производного от Frame класса. Второй – объявить объект Frame в методе main(). В листинге показан первый способ:

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

public class FrameExtender extends Frame
{

    /** Конструктор */
    public FrameExtender()
    {
        addWindowListener(new WinCloser());
        setTitle("Just a Frame");
        setBounds( 100, 100, 200, 200);
        setVisible(true);
    }
    public static void main(String args[])
    {
        FrameExtender fe = new FrameExtender();
    }
}

class WinCloser extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```

В этом примере расширяется класс Frame. При этом вся функциональность класса наследуется.

```
public class FrameExtender extends Frame
```

В отличие от апплета, приложения не имеют метода `init()`. В конструкторе выполняются все действия по инициализации приложения.

```
public FrameExtender()
```

Для того, чтобы окна закрывались корректно, с очисткой всех необходимых ресурсов, необходимо добавить класс WindowListener. Swing обеспечивает более элегантный способ, но об этом будет речь позже.

```
addWindowListener(new WinCloser());
```

Заголовок окна добавляется методом:

```
setTitle("Just a Frame");
```

Размер начального окна устанавливается следующим методом.

```
setBounds( 100, 100, 200, 200);
```

Затем окно делается видимым:

```
setVisible(true);
```

В метод `main()` создается объект типа `FrameExtender`, при этом выполняется конструктор.

```
public static void main(String args[])
{
    FrameExtender fe = new FrameExtender();
}
}
```

Класс `WinCloser` наследует класс `WindowAdapter`. Класс `WindowAdapter` – это абстрактный класс, который обеспечивает набор методов, выполняющих действия по умолчанию. Программист задает собственную реализацию переопределенных методов. Мы должны обеспечить корректное закрытие окна. Более подробно об обработке событий рассказывается в лекции 14. А сейчас просто выполняется закрытие окна.

```
class WinCloser extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
```

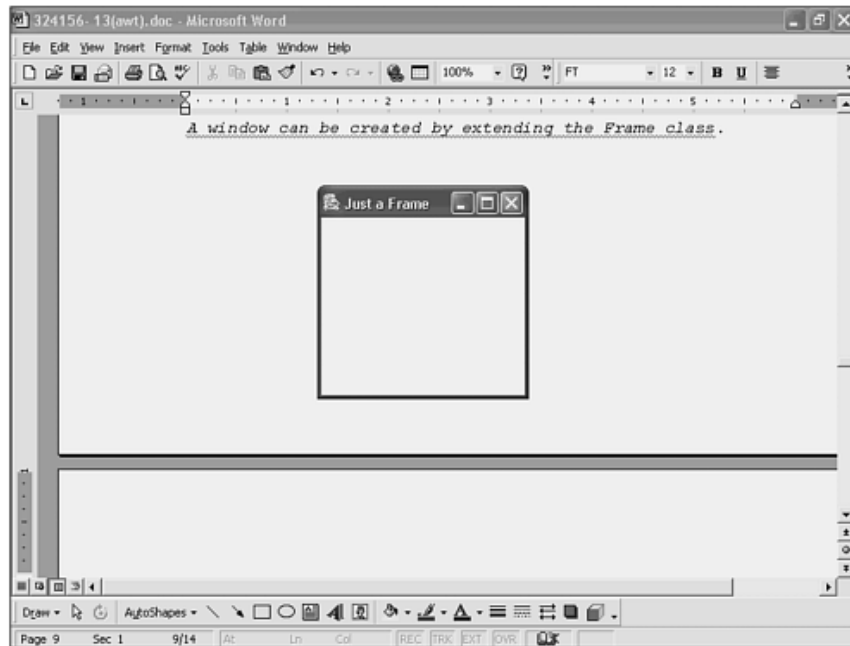
Метод `exit()` класса `System` приводит к завершению задачи и освобождению ресурсов приложения.

```
System.exit(0);
```

Чтобы запустить пример, откомпилируйте его, а затем наберите командную строку:

```
C:\>java FrameExtender
```

Вы увидите на экране примерно то, что показано на рисунке:



Здесь объект типа `Frame` создается в методе `main()` создаваемого класса `FrameInstantiator`.

Задание на лабораторную работу

В следующих заданиях выполнить рисунок в окне апплета или фрейма.

1. Создать классы **Point** и **Line**. Объявить массив из n объектов класса **Point**. Для объекта класса **Line** определить, какие из объектов **Point** лежат на одной стороне от прямой линии и какие на другой. Реализовать ввод данных для объекта **Line** и случайное задание данных для объекта **Point**.
2. Создать классы **Point** и **Line**. Объявить массив из n объектов класса **Point** и определить в методе, какая из точек находится дальше всех от прямой линии.
3. Создать класс **Triangle** и класс **Point**. Объявить массив из n объектов класса **Point**, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.
4. Определить класс **Rectangle** и класс **Point**. Объявить массив из n объектов класса **Point**. Написать функцию, определяющую, какая из точек лежит снаружи, а какая – внутри прямоугольника.
5. Реализовать полиморфизм на основе абстрактного класса **AnyFigure** и его методов. Вывести координаты точки, треугольника, тетраэдра.
6. Определить класс **Line** для прямых линий, проходящих через точки $A(x_1, y_1)$ и $B(x_2, y_2)$. Создать массив объектов класса **Line**. Определить, используя функции, какие из прямых пересекаются, а какие совпадают. Нарисовать все пересекающиеся прямые.
7. Определить классы **Triangle** и **NAngle**. Определить, какой из m -введенных n -угольников имеет наибольшую площадь.

8. Задать движение по экрану строк (одна за другой) из массива строк. Направление движения по апплету и значение каждой строки выбираются случайным образом.
9. Задать составление строки из символов, появляющихся из разных углов апплета и выстраивающихся друг за другом. Процесс должен циклически повторяться.
10. Задать движение окружности по апплету так, чтобы при касании границы окружность отражалась от нее с эффектом упругого сжатия.
11. Изобразить в апплете приближающийся издали шар, удаляющийся шар. Шар должен двигаться с постоянной скоростью.
12. Изобразить в окне приложения (апплета) отрезок, вращающийся в плоскости экрана вокруг одной из своих концевых точек. Цвет прямой должен изменяться при переходе от одного положения к другому.
13. Создать класс **Triangle** и класс **Point**. Объявить массив из n объектов класса **Point**, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.
14. Определить класс **Rectangle** и класс **Point**. Объявить массив из n объектов класса **Point**. Написать функцию, определяющую, какая из точек лежит снаружи, а какая – внутри прямоугольника.
15. Реализовать полиморфизм на основе абстрактного класса **AnyFigure** и его методов. Вывести координаты точки, треугольника, тетраэдра.

Контрольные вопросы

1. Какие классы входят в состав пакета java.applet, их назначение.
2. Какие классы входят в состав пакета java.awt, их назначение.
3. Какие классы входят в состав пакета java.awt.datatransfer, их назначение.
4. Какие классы входят в состав пакета java.awt.event, их назначение.
5. Какие классы входят в состав пакета java.awt.peer, их назначение.
6. Каким образом происходит инициализация апплета ?
7. Какие библиотеки для создания GUI в Java вы знаете ?

Лабораторная работа №6 «Потоки ввода-вывода»

Цель работы: Изучить применение классов пакета java.io для организации ввода-вывода данных в приложениях на языке Java.

Теоретические сведения

Обобщенное понятие источника ввода относится к различным способам получения информации: к чтению дискового файла, символов с клавиатуры, либо получению данных из сети. Аналогично, под обобщенным понятием вывода также могут пониматься дисковые файлы, сетевое соединение и т.п. Эти абстракции дают удобную возможность для работы с вводом-выводом (I/O), не требуя при этом, чтобы каждая часть вашего кода понимала разницу между, скажем, клавиатурой и сетью. В Java эта абстракция называется потоком (stream) и реализована в нескольких классах пакета java.io. Ввод инкапсулирован в классе InputStream, вывод — в OutputStream. В Java есть несколько специализаций этих абстрактных классов, учитывающих различия при работе с дисковыми файлами, сетевыми соединениями и даже с буферами в памяти.

File

File — единственный объект в java.io, который работает непосредственно с дисковыми файлами. Хотя на использование файлов в апплетах наложены жесткие ограничения, файлы по-прежнему остаются основными ресурсами для постоянного хранения и совместного использования информации. Каталог в Java трактуется как обычный файл, но с дополнительным свойством — списком имен файлов, который можно просмотреть с помощью метода list.

Для определения стандартных свойств объекта в классе File есть много разных методов. Однако, класс File несимметричен. Есть много методов, позволяющих узнать свойства объекта, но соответствующие функции для изменения этих свойств отсутствуют. В очередном примере используются различные методы, позволяющие получить характеристики файла:

```
import java.io.File;
class FileTest {
    static void p(String s) {
        System.out.println(s);
    }
    public static void main(String args[]) {
        File f1 = new File("/java/COPYRIGHT");
        p("File Name:" + f1.getName());
        p("Path:" + f1.getPath());
        p("Abs Path:" + f1.getAbsolutePath());
        p("Parent:" + f1.getParent());
        p(f1.exists() ? "exists" : "does not exist");
        p(f1.canWrite() ? "is writeable" : "is not writeable");
        p(f1.canRead() ? "is readable" : "is not readable");
        p("is " + (f1.isDirectory() ? " " : "not") + " a directory");
        p(f1.isFile() ? "is normal file" : "might be a named pipe");
        p(f1.isAbsolute() ? "is absolute" : "is not absolute");
        p("File last modified:" + f1.lastModified());
        p("File size:" + f1.length() + " Bytes");
    }
}
```

Существует также несколько сервисных методов, использование которых ограничено обычными файлами (их нельзя применять к каталогам). Метод `renameTo(File dest)` переименовывает файл (нельзя переместить файл в другой каталог). Метод `delete` уничтожает дисковый файл. Этот метод может удалять только обычные файлы, каталог, даже пустой, с его помощью удалить не удастся.

Каталоги

Каталоги — это объекты класса `File`, в которых содержится список других файлов и каталогов. Если `File` ссылается на каталог, его метод `isDirectory` возвращает значение `true`. В этом случае вы можете вызвать метод `list` и извлечь содержащиеся в объекте имена файлов и каталогов. В очередном примере показано, как с помощью метода `list` можно просмотреть содержимое каталога.

```
import java.io.File;
class DirList {
    public static void main(String args[]) {
        String dirname = "/java"; // имя каталога
        File f1 = new File(dirname);
        if (f1.isDirectory()) { // является ли f1 каталогом
            System.out.println("Directory of ' + dirname);
            String s[] = f1.list();
            for ( int i=0; i < s.length; i++) {
                File f = new File(dirname + "/" + s[i]);
                if (f.isDirectory()) { // является ли f каталогом
                    System.out.println(s[i] + " is a
directory");
                } else {
                    System.out.println(s[i] + " is a file");
                }
            }
            System.out.println(dirname + " is not a directory");
        }
    }
}
```

В процессе работы эта программа вывела содержимое каталога `/java` моего персонального компьютера в следующем виде:

```
C:\> java DirList
Directory of /java
bin is a directory
COPYRIGHT is a file
README is a file
InputStream
```

`InputStream` — абстрактный класс, задающий используемую в Java модель входных потоков. Все методы этого класса при возникновении ошибки возбуждают исключение `IOException`. Ниже приведен краткий обзор методов класса `InputStream`.

- `read()` возвращает представление очередного доступного символа во входном потоке в виде целого.

- `read(byte b[])` пытается прочесть максимум `b.length` байтов из входного потока в массив `b`. Возвращает количество байтов, в действительности прочитанных из потока.
- `read(byte b[], int off, int len)` пытается прочесть максимум `len` байтов, расположив их в массиве `b`, начиная с элемента `off`. Возвращает количество реально прочитанных байтов.
- `skip(long n)` пытается пропустить во входном потоке `n` байтов. Возвращает количество пропущенных байтов.
- `available()` возвращает количество байтов, доступных для чтения в настоящий момент.
- `close()` закрывает источник ввода. Последующие попытки чтения из этого потока приводят к возбуждению `IOException`.
- `mark(int readlimit)` ставит метку в текущей позиции входного потока, которую можно будет использовать до тех пор, пока из потока не будет прочитано `readlimit` байтов.
- `reset()` возвращает указатель потока на установленную ранее метку.
- `markSupported()` возвращает `true`, если данный поток поддерживает операции `mark/reset`.

OutputStream

Как и `InputStream`, `OutputStream` — абстрактный класс. Он задает модель выходных потоков Java. Все методы этого класса имеют тип `void` и возбуждают исключение `IOException` в случае ошибки. Ниже приведен список методов этого класса:

- `write(int b)` записывает один байт в выходной поток. Обратите внимание — аргумент этого метода имеет тип `int`, что позволяет вызывать `write`, передавая ему выражение, при этом не нужно выполнять приведение его типа к `byte`.
- `write(byte b[])` записывает в выходной поток весь указанный массив байтов.
- `write(byte b[], int off, int len)` записывает в поток часть массива — `len` байтов, начиная с элемента `b[off]`.
- `flush()` очищает любые выходные буферы, завершая операцию вывода.
- `close()` закрывает выходной поток. Последующие попытки записи в этот поток будут возбуждать `IOException`.

Файловые потоки

FileInputStream

Класс `FileInputStream` используется для ввода данных из файлов. В приведенном ниже примере создается два объекта этого класса, использующие один и тот же дисковый файл.

```
InputStream f0 = new FileInputStream("/autoexec.bat");
File f = new File("/autoexec.bat");
InputStream f1 = new FileInputStream(f);
```

Когда создается объект класса `FileInputStream`, он одновременно с этим открывается для чтения. `FileInputStream` замещает шесть методов абстрактного класса `InputStream`. Попытки применить к объекту этого класса

методы `mark` и `reset` приводят к возбуждению исключения `IOException`. В приведенном ниже примере показано, как можно читать одиночные байты, массив байтов и поддиапазон массива байтов. В этом примере также показано, как методом `available` можно узнать, сколько еще осталось непрочитанных байтов, и как с помощью метода `skip` можно пропустить те байты, которые вы не хотите читать.

```
import java.io.*;
import java.util.*;
class FileInputStreamS {
public static void main(String args[]) throws Exception {
int size;
InputStream f1 = new FileInputStream("/wwwroot/default.htm");
size = f1.available();
System.out.println("Total Available Bytes: " + size);
System.out.println("First 1/4 of the file: read()");
for (int i=0; i < size/4; i++) {
System.out.print((char) f1.read());
}
System.out.println("Total Still Available: " + f1.available());
System.out.println("Reading the next 1/8: read(b[])");
byte b[] = new byte[size/8];
if (f1.read(b) != b.length) {
System.err.println("Something bad happened");
}
String tmpstr = new String(b, 0, 0, b.length);
System.out.println(tmpstr);
System.out.println("Still Available: " + f1.available());
System.out.println("Skipping another 1/4: skip()");
f1.skip(size/4);
System.out.println("Still Available: " + f1.available());
System.out.println("Reading 1/16 into the end of array");
if (f1.read(b, b.length-size/16, size/16) != size/16) {
System.err.println("Something bad happened");
}
System.out.println("Still Available: " + f1.available());
f1.close();
}
}
FileOutputStream
```

У класса `FileOutputStream` — два таких же конструктора, что и у `FileInputStream`. Однако, создавать объекты этого класса можно независимо от того, существует файл или нет. При создании нового объекта класс `FileOutputStream` перед тем, как открыть файл для вывода, сначала создает его.

В очередном нашем примере символы, введенные с клавиатуры, считываются из потока `System.in` - по одному символу за вызов, до тех пор, пока не заполнится 12-байтовый буфер. После этого создаются три файла. В первый из них, `file1.txt`, записываются символы из буфера, но не все, а через один — нулевой, второй и так далее. Во второй, `file2.txt`, записывается весь ввод, попавший в буфер. И наконец в третий файл записывается половина буфера, расположенная в середине, а первая и последняя четверти буфера не выводятся.

```
import java.io.*;
class FileOutputSteamS {
    public static byte getInput()[] throws Exception {
        byte buffer[] = new byte[12];
        for (int i=0; i<12; i++) {
            buffer[i] = (byte) System.in.read();
        }
        return buffer;
    }
    public static void main(String args[]) throws Exception {
        byte buf[] = getInput();
        OutputStream f0 = new FileOutputStream("file1.txt");
        OutputStream f1 = new FileOutputStream("file2.txt");
        OutputStream f2 = new FileOutputStream("file3.txt");
        for (int i=0; i < 12; i += 2) {
            f0.write(buf[i]);
        }
        f0.close();
        f1.write(buf);
        f1.close();
        f2.write(buf, 12/4, 12/2);
        f2.close();
    } }
StringBufferInputStream
```

`StringBufferInputStream` идентичен классу `ByteArrayInputStream` с тем исключением, что внутренним буфером объекта этого класса является экземпляр `String`, а не байтовый массив. Кроме того, в Java нет соответствующего ему класса `StringBufferedOutputStream`. У этого класса есть единственный конструктор:

```
StringBufferInputStream( String s)
```

Задание на лабораторную работу

Выполнить задания, сохраняя объекты приложения в одном или нескольких файлах с применением механизма сериализации. Объекты могут содержать поля, помеченные как **static**, а также **transient**.

1. Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве. Студентов, Преподавателей и Курсов при обучении может быть несколько.
2. Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может за-блокировать КК за превышение кредита.
3. Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или при иных обстоятельствах.
4. Система Вступительные экзамены. Абитуриент регистрируется на Факультет, сдает Экзамены. Преподаватель выставляет Оценку. Система подсчитывает средний балл и определяет Абитуриентов, зачисленных в учебное заведение.
5. Система Библиотека. Читатель оформляет Заказ на Книгу. Система осуществляет поиск в Каталоге. Библиотекарь выдает Читателю Книгу на абонемент или в читальный зал. При невозвращении Книги Читателем он может быть занесен Администратором в «черный список».
6. Система Конструкторское бюро. Заказчик представляет Техническое Задание (ТЗ) на проектирование многоэтажного Дома. Конструктор регистрирует ТЗ, определяет стоимость проектирования и строительства, выставляет Заказчику Счет за проектирование и создает Бригаду Конструкторов для выполнения Проекта.
7. Система Телефонная станция. Абонент оплачивает Счет за разговоры и Услуги, может попросить Администратора сменить номер и отказаться от услуг. Администратор изменяет номер, Услуги и временно отключает Абонента за неуплату.
8. Система Автобаза. Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля.
9. Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».
10. Система Железнодорожная касса. Пассажир делает Заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск подходящего Поезда. Пассажир делает выбор Поезда и получает Счет на оплату. Администратор вводит номера Поездов, промежуточные и конечные станции, цены.
11. Система Городской транспорт. На Маршрут назначаются Автобус, Троллейбус или Трамвай. Транспортные средства должны двигаться с определенным для каждого Маршрута интервалом. При поломке на Маршрут

должен выходить резервный транспорт или увеличиваться интервал движения.

12. Система Аэрофлот. Администратор формирует летную Бригаду (пилоты, штурман, радист, стюардессы) на Рейс. Каждый Рейс выполняется Самолетом с определенной вместимостью и дальностью полета. Рейс может быть отменен из-за погодных условий в Аэропорту отлета или назначения. Аэропорт назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

13. Система Периодические издания. Читатель может сделать Заявку, предварительно выбрав периодические Издания из списка. Система подсчитывает сумму для оплаты. Читатель оплачивает заявку. Администратор добавляет Заявку в «черный список», если Клиент не оплачивает её в определённый срок.

14. Система Заказ гостиницы. Клиент оставляет Заявку на Номер, указав количество мест в номере, класс апартаментов и время пребывания. Администратор рассматривает Заявку, подтверждает или отклоняет её. Результат просматривает Клиент. В случае подтверждения Заявки Клиент оплачивает услуги.

15. Система Жилищно-коммунальные услуги. Квартиросъемщик отправляет Заявку, в которой указывает род работ, масштаб и желаемое время выполнения. Диспетчер формирует соответствующую Бригаду и регистрирует её в Плане работ. Диспетчер может отклонить Заявку в случае занятости всех Бригад.

16. Система Прокат автомобилей. Клиент выбирает Автомобиль из списка доступных, заполняет форму Заказа, указывая паспортные данные, срок аренды. Администратор может отклонить Заявку, указав причины отказа. При подтверждении Заявки Клиент оплачивает Заказ. Система выписывает сумму. В случае повреждения Автомобиля Клиентом Администратор вносит соответствующие пометки.

Контрольные вопросы

1. Что такое потоки ввода-вывода и для чего они нужны?
2. Какие классы Java являются базовыми для работы с потоками?
3. В чем разница между байтовыми и символьными потоками?
4. Как получить свойства файла? Какие свойства файла можно узнать?
5. Какие стандартные потоки ввода-вывода существуют в Java, каково их назначение? На базе каких классов создаются стандартные потоки?
6. Чем являются потоки System.in, System.out, System.err?
7. Как создать файловый поток для чтения и записи данных?
8. В чем заключается особенность создания потока, связанного с локальным файлом?
9. Как создать поток для форматированного обмена данными, связанного с локальным файлом?

10. Как добавить буферизацию для потока форматированного обмена данными, связанного с локальным файлом?

Лабораторная работа №7 «Использование внутренних классов»

Цель работы: научиться работать с внутренними классами

Теоретические сведения

Классы, определенные внутри других классов, называются внутренними. Обычно внутренние классы не отличаются от любых других, за исключением того, что они определены в теле другого класса. Например:

```
public class Outer {  
    public class Inner {  
        private int i;  
        public void myMethod(){ ... }  
    }  
}
```

Область существования

Внутренний класс, определенный вне методов класса, принадлежит классу и может применяться внутри класса (как поля класса). Объекты внутреннего класса могут быть созданы внутри любого метода класса. Внутренние классы могут быть определены внутри метода класса. Тогда они имеют область применения только внутри того метода, где они определены (как автоматические переменные). В этом методе можно создавать объекты данного класса, но не в других методах внешнего класса. Внутренние классы, определенные внутри метода, имеют ряд ограничений:

У них не может быть модификатора доступа

Они не могут быть объявлены как static

Внутренние классы могут использовать только те переменные и параметры метода, которые объявлены final

Внутренние классы полностью «видят» все поля и методы их внешних классов

```
public class Outer  
{  
    private int a = 5;  
    public class Inner {  
        private int i=1;  
        public void myMethod() {  
            System.out.println( "a=" + a + ", i=" + i );  
        }  
    }  
}
```



```

public static void main( String[] args ) {
    Outer.Inner innerClass = new Outer().new Inner();
    innerClass.myMethod();
}
}

```

Внутренний класс `Inner` имеет доступ к собственной переменной `i` так же, как и к полю класса `Outer` переменной `a`, хотя она и имеет доступ `private`.

Модификаторы доступа

Из-за того, что внутренние классы, определенные вне методов, похожи на поля, можно применять к ним модификатор доступа. Модификатор доступа определяет, можно ли использовать объекты внутреннего класса вне внешнего класса. Если Вы объявляете внутренний класс `public`, то можно его использовать, указав имя внешнего класса, а затем, через точку, имя внутреннего класса:

```

Outer outer = new Outer();
Outer.Inner inner = o.new Inner()

```

Или:

```

Outer.Inner inner = new Outer().new Inner();

```

Статические внутренние классы

Внутренние классы могут иметь атрибут `static`. Так как они статические, то они не связаны с объектами внешнего класса. Это значит, что можно создавать объекты внутренних классов, не создавая объекта внешнего класса. Внутренние статические классы имеют некоторые ограничения по доступу к членам внешнего класса:

Методы статического внутреннего класса не имеют доступа к полям и методам внешнего класса, если они не статические.

Методы статического внутреннего класса имеют доступ только к статическим полям и методам внешнего класса

Листинг демонстрирует эти ограничения:

```

public class OuterTest
{
    public static int outerInt = 5;

    public static class StaticInner {
        public static int doubleVal( int n ) {

```

```

        System.out.println( "outerInt=" + outerInt );
        return 2*n;
    }
}

public void testInner() {
    int a = 5;
    System.out.println( "a=" + a + ", doubleVal=" +
        StaticInner.doubleVal( a ) );
}

public static void main( String[] args ) {
    int n = 7;
    System.out.println( "n=" + n + ", doubleVal=" +
        OuterTest.StaticInner.doubleVal( n ) );
    OuterTest out = new OuterTest();
    out.testInner();
}
}

```

Внутренний класс StaticInner использует статическое поле outerInt класса OuterTest в методе doubleVal. Если бы outerInt не были бы static, класс StaticInner не мог бы использовать ее. Метод main показывает, как применять внутренний статический класс, не создавая объект внешнего класса:

```
OuterTest.StaticInner.doubleVal( n );
```

Анонимные внутренние классы

1. Они обладают следующими особенностями:
2. Создаются без имени
3. Определяются внутри метода
4. Не имеют конструктора
5. Объявляются и определяются в одном операторе
6. Чаще всего используются для обработки событий

О ссылках на объекты

При создании переменной типа класса создается ссылка на объект этого класса. Ссылка нужна, чтобы обратиться к объекту класса, но она не жестко связана с конкретным объектом. Если Вы создаете два объекта, а потом выполняете присваивание, то обе ссылки указывают на один и тот же объект, а второй объект оказывается неиспользуемым и будет уничтожен при сборке мусора:

```

public class Number {
    private int number;

```

```

public Number( int number ) {
    this.number = number;
}

public int getNumber() {
    return this.number;
}

public void setNumber( int number ) {
    this.number = number;
}

public static void main( String[] args ) {
    Number one = new Number( 1 );
    Number two = new Number( 2 );
    System.out.println( "Beginning: " );
    System.out.println( "One = " + one.getNumber() );
    System.out.println( "Two = " + two.getNumber() );

    // Assign two to one
    two = one;
    System.out.println( "\nAfter assigning two to one: " );
    System.out.println( "One = " + one.getNumber() );
    System.out.println( "Two = " + two.getNumber() );

    // Change the value of two
    two.setNumber( 3 );
    System.out.println( "\nAfter modifying two: " );
    System.out.println( "One = " + one.getNumber() );
    System.out.println( "Two = " + two.getNumber() );
}
}

```

Задание на лабораторную работу

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода

1. Создать класс Notepad (записная книжка) с внутренним классом или классами, с помощью объектов которого могут храниться несколько записей на одну дату.
2. Создать класс Payment (покупка) с внутренним классом, с помощью объектов которого можно сформировать покупку из нескольких товаров.

3. Создать класс Account (счет) с внутренним классом, с помощью объектов которого можно хранить информацию обо всех операциях со счетом (снятие, платежи, поступления).
4. Создать класс Зачетная Книжка с внутренним классом, с помощью объектов которого можно хранить информацию о сессиях, зачетах, экзаменах.
5. Создать класс Department (отдел фирмы) с внутренним классом, с помощью объектов которого можно хранить информацию обо всех должностях отдела и обо всех сотрудниках, когда-либо занимавших конкретную должность.
6. Создать класс Catalog (каталог) с внутренним классом, с помощью объектов которого можно хранить информацию об истории выдач книги читателям.
7. Создать класс СССР с внутренним классом, с помощью объектов которого можно хранить информацию об истории изменения территориального деления на области и республики.
8. Создать класс City (город) с внутренним классом, с помощью объектов которого можно хранить информацию о проспектах, улицах, площадях.
9. Создать класс CD (mp3-диск) с внутренним классом, с помощью объектов которого можно хранить информацию о каталогах, подкаталогах и записях.
10. Создать класс Mobile с внутренним классом, с помощью объектов которого можно хранить информацию о моделях телефонов и их свойствах.
11. Создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.
12. Создать класс Календарь с внутренним классом, с помощью объектов которого можно хранить информацию о выходных и праздничных днях.
13. Создать класс Shop (магазин) с внутренним классом, с помощью объектов которого можно хранить информацию об отделах, товарах и услуг.
14. Создать класс Справочная Служба Общественного Транспорта с внутренним классом, с помощью объектов которого можно хранить информацию о времени, линиях маршрутов и стоимости проезда.
15. Создать класс Computer (компьютер) с внутренним классом, с помощью объектов которого можно хранить информацию об операционной системе, процессоре и оперативной памяти.

Контрольные вопросы

1. Какие типы внутренних классов вы знаете ?
2. В чем отличие анимированных классов ?
3. Что такое статические внутренние классы ?
4. Какие из фрагментов кода скомпилируются без ошибки?
 - 1)
import java.util.*;
package First;

```

class My{/* тело класса*/}
2)
package mypack;
import java.util.*;
public class First{/* тело класса*/}
3)
/*комментарий */
package first;
import java.util.*;
class First{/* тело класса*/}

```

5. Какие классы чаще всего используются для обработки событий ?
6. Каков механизм использования внутренних классов ?
7. Средства Java для работы с внутренними классами ?
8. Эффективное использование внутренних классов.
9. Необходимость использования внутренних классов.

Лабораторная работа №8 «Построение модели программной системы»

Цель работы: научиться создавать модели программных систем и реализовывать их с использованием языка программирования Java

Теоретические сведения

Моделирование данных является важнейшим процессом при проектировании программного обеспечения (ПО). По этой причине, разработчики CASE-средств в своих продуктах вынуждены уделять моделированию данных повышенное внимание. Являясь признанным лидером в области объектных методологий, фирма Rational Software Corporation, тем не менее, до недавнего времени такого средства не имела. Основной причиной этого, по-видимому, является ориентация на язык Unified Modeling Language (UML), как универсальный инструмент моделирования. UML полностью покрывает потребности моделирования данных. Сложившаяся на протяжении десятилетий технология моделирования данных, традиции, система понятий и колоссальный опыт разработчиков не могли далее игнорироваться. Немаловажную роль здесь сыграла и необходимость формального контроля моделей данных, что является абсолютно необходимым при проектировании мало-мальски больших схем баз данных и что UML не обеспечивает в достаточной степени. И, наконец, последней причиной, побудившей специалистов Rational Software Corporation к созданию собственного средства моделирования данных, является требование построения эффективных физических моделей, прежде всего для конкретных СУБД - лидеров рынка.

В начале 2000 года фирма Rational Software Corporation анонсировала появление собственного средства моделирования данных – Data Modeler, и в настоящее время оно доступно специалистам, например, использующим в своей работе Rational Rose 2000.

Целью данной лабораторной работы является знакомство с основными возможностями этого нового средства.

Авторы Data Modeler, прежде всего, ориентировались на создание инструмента проектирования физической модели данных. При этом не произошло отказа от UML как от средства моделирования данных, а некоторым образом были смещены акценты: теперь UML предполагается использовать для построения логической модели. По сути, логическая модель - это та же объектная модель, состоящая из объектов - сущностей. Переход от логической модели к физической и, наоборот, в части моделирования данных обеспечивается Rational Rose автоматически. Для этого введено соответствие элементов моделей (табл. 8.1).

Таблица 8.1. Соответствие элементов логической и физической модели

Логическая модель	Физическая модель
Class (Класс)	Table (Таблица)
Operation (Операция)	Constraint (Ограничение)
Attribute (Атрибут)	Column (Колонка)
Package (Пакет)	Scheme (Схема)
Component (Компонент)	Database (База данных)
Association (Ассоциация)	Relationship (Связь)
Нет	Trigger (Триггер)
Нет	Index (Индекс)

Rose Data Modeler

После установки Rational Rose в специальной редакции (Rational Rose Professional Data Modeler Edition) в разделе главного меню Tools появляется новый раздел Data Modeler (рис. 8.1).

В разделе Data Modeler имеются два пункта: “Add Schema” и “Reverse Engineer...”. Пункт “Add Schema” используется для создания новых схем БД, а пункт “Reverse Engineer” - для построения модели на основе существующей схемы БД.

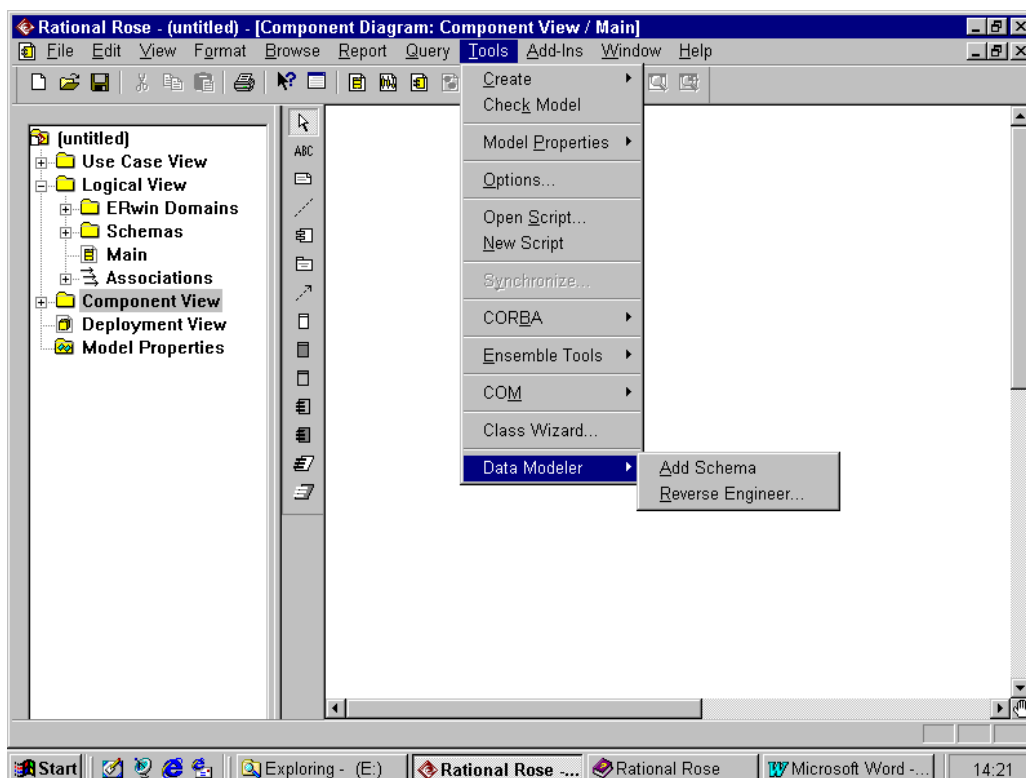


Рисунок 8.1- Отображение компоненты Data Modeler в меню Rational Rose

Создание диаграммы модели данных

После создания схемы (рис. 8.1) в ней можно сформировать диаграмму модели данных. Эта диаграмма позволит добавить, изменить или просмотреть таблицы и другие элементы модели данных, поскольку играет ту же роль, что и диаграмма классов в объектной модели. Хотя можно добавлять элементы моделирования данных непосредственно в браузере, диаграмма модели данных позволяет показать в графическом виде, как сами элементы, так и отношения между ними. Для любой схемы можно создать любое необходимое число. Создание диаграммы модели данных осуществляется следующим образом:

1. В браузере щелкните правой кнопкой мыши на схеме.
2. Выберите Data Modeler > New > Data Model Diagram.
3. Введите имя новой диаграммы.
4. Дважды щелкните на диаграмме для ее открытия.

Как и другие диаграммы в среде Rose, диаграмма модели данных имеет специализированную панель инструментов для добавления таблиц, отношений и других элементов моделирования. Кнопки этой панели перечислены в таблице 8.2.

Таблица 8.2 – Значки панели инструментов для диаграммы модели данных

Значки	Назначение
--------	------------










	Курсор принимает форму стрелки для выделения элемента
	Добавляет в диаграмму текстовое поле
	Добавляет к элементу диаграммы примечание
	Соединяет примечание с элементом диаграммы
	Добавляет в диаграмму таблицу
	Рисует неидентифицируемое отношение между двумя таблицами
	Рисует идентифицируемое отношение между двумя таблицами
	Добавляет в диаграмму представление
	Рисует зависимость между двумя таблицами

Диаграмма Data Model предоставляет следующие возможности:

- создание и редактирование таблиц и их элементов (колонок, ограничений, индексов, триггеров и т. п.);
- создание и редактирование идентифицирующих связей между таблицами;
- создание и редактирование неидентифицирующих связей.

Основные возможности по работе с таблицей доступны, если войти в контекстном меню в пункт “Open Specification”. Появляющееся на экране окно включает следующую информацию (рис. 8.2).

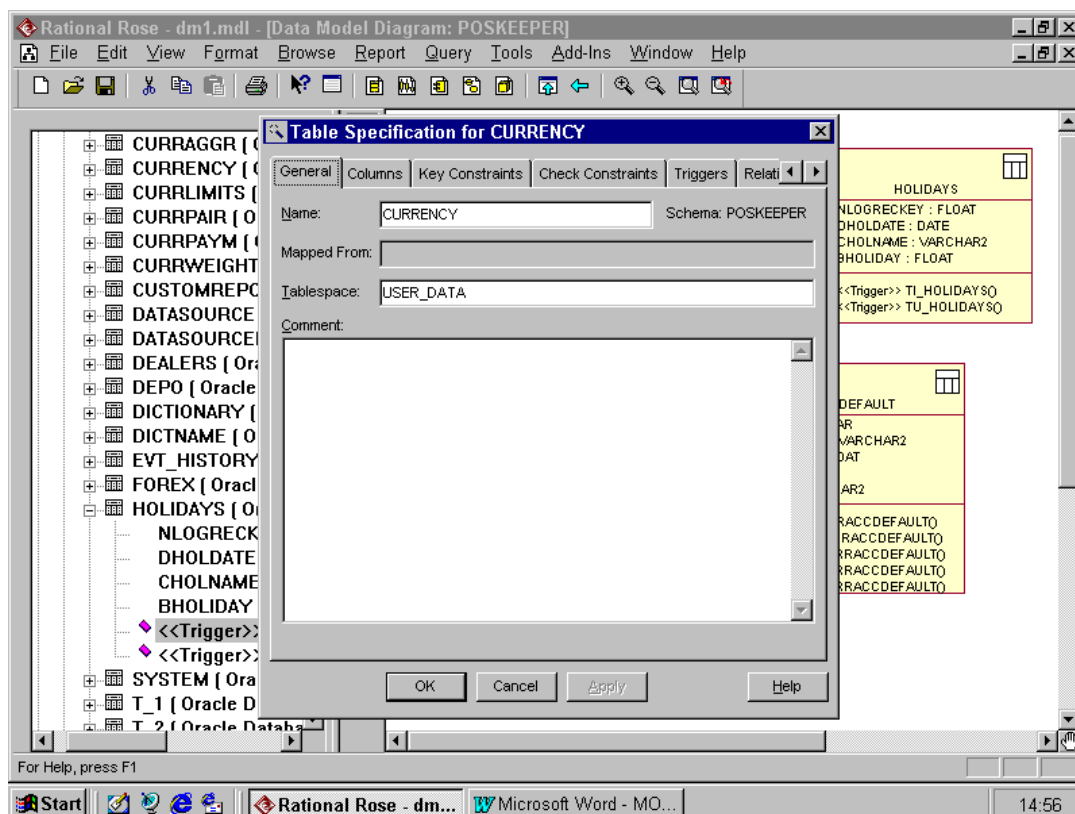


Рисунок 8.2 - Окно спецификации таблицы

При редактировании спецификации таблицы обеспечиваются следующие возможности (табл. 13.3).

Таблица 8.3 - Спецификация таблицы БД

Закладка	Описание
General	Вводится общая информация о таблице.
Columns	Задается описание колонок. Здесь можно добавить или отредактировать свойства колонок, задать тип, длину, обязательность (NULL, NOT NULL), а также пометить, что колонка входит в состав первичного ключа. Типы колонок соответствуют типам конкретной выбранной СУБД.
Key Constraints	Задаются ограничения на колонки таблицы. Здесь можно задать ограничение на уникальность первичного ключа, ограничение на уникальность альтернативных ключей, а также просто определить индекс.
Check Constraints	Задаются выражения – инварианты, которые должны выполняться для всех строк таблицы.
Triggers	Содержит список триггеров, который можно отредактировать, в том числе добавив новый триггер.
Relationships	При наличии связей между таблицами, закладка содержит полный список связей.

Добавление отношений

Отношения в модели данных подобны отношениям в объектной модели. В объектной модели отношение связывает два класса, а в модели данных — две таблицы. В Rose поддерживаются два основных типа отношений: идентифицируемые отношения (identifying relationship) и неидентифицируемые отношения (non-identifying relationship).

В обоих случаях для поддержки отношений в дочернюю таблицу добавляется внешний ключ. При идентифицируемом отношении внешний ключ становится частью первичного ключа в дочерней таблице. В этом случае дочерняя таблица не может содержать запись, не связанную с записью в родительской таблице. Идентифицируемые отношения моделируются составными агрегациями.

Неидентифицируемые отношения тоже создают внешний ключ в дочерней таблице, но он не становится частью первичного ключа в дочерней таблице. При неидентифицируемом отношении мощность (множественность) определяет то, будет ли запись в дочерней таблице существовать без связи с записью в родительской таблице. Если мощность равна 1, должна присутствовать родительская запись. Если мощность равна 0..1, присутствие родительской записи необязательно. Неидентифицируемые отношения моделируются ассоциациями.

Для редактирования свойств связи требуется войти в пункт контекстного меню “Open specification”.

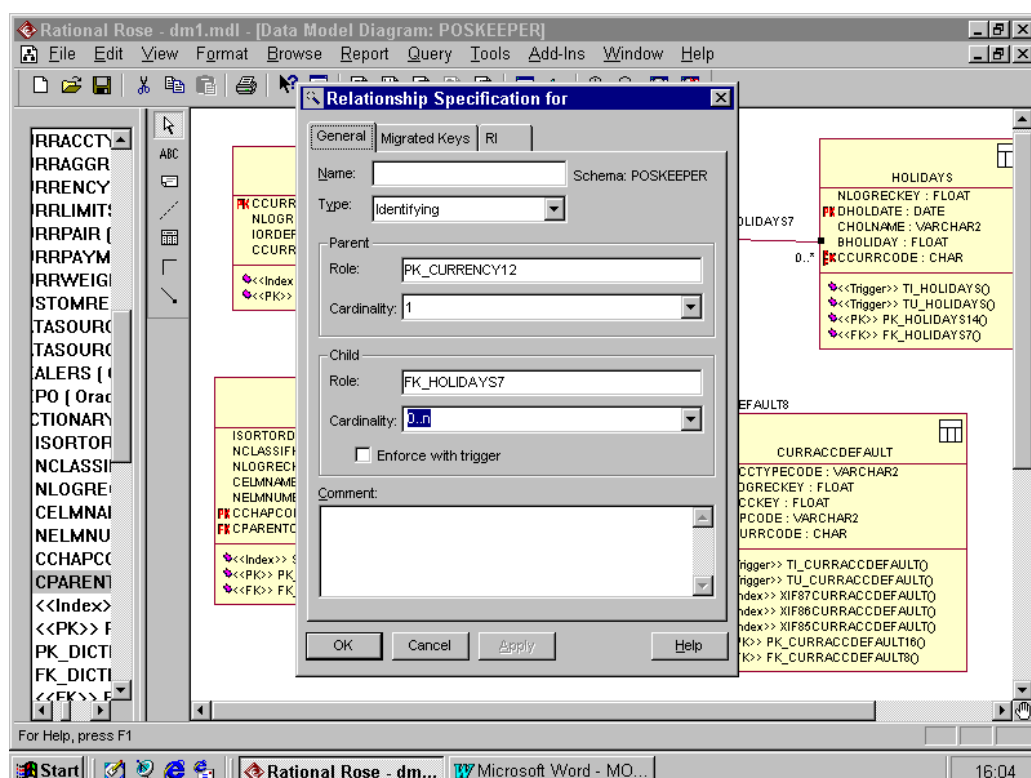


Рисунок 8.3 - Окно спецификации связи

Задание на лабораторную работу

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода

Порядок выполнения работы.

1. Определить иерархию классов (в соответствии с вариантом).
2. Определить в классе статическую компоненту - ссылку на начало связанного списка объектов и статическую функцию для просмотра списка.
3. Реализовать классы.
4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.
5. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе.

Методические указания.

1. Для определения иерархии классов связать отношением наследования классы, приведенные в приложении (для заданного варианта). Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.
2. Определить в классах все необходимые конструкторы.
3. Компонентные данные класса специфицировать как **protected**.
4. Для добавления объекта в список предусмотреть метод класса, т.е. объект сам добавляет себя в список. Например, `a.Add()` **a** добавляет себя в список.
5. Включение объекта в список можно выполнять при создании объекта, т.е. поместить операторы включения в конструктор. В случае иерархии классов, включение объекта в список должен выполнять **только** конструктор базового класса. Вы должны продемонстрировать оба этих способа.
6. Список просматривать путем вызова метода **Show** каждого объекта.
7. Статический метод просмотра списка вызывать не через объект, а через класс.

Варианты заданий.

Перечень классов:

- 1) студент, преподаватель, персона, завкафедрой;
- 2) служащий, персона, рабочий, инженер;
- 3) рабочий, кадры, инженер, администрация;
- 4) деталь, механизм, изделие, узел;
- 5) организация, страховая компания, судостроительная компания, завод;
- 6) журнал, книга, печатное издание, учебник;
- 7) тест, экзамен, выпускной экзамен, испытание;
- 8) место, область, город, мегаполис;

- 9) игрушка, продукт, товар, молочный продукт;
- 10) квитанция, накладная, документ, чек;
- 11) автомобиль, поезд, транспортное средство, экспресс;
- 12) двигатель, двигатель внутреннего сгорания, дизель, турбореактивный двигатель;
- 13) республика, монархия, королевство, государство;
- 14) млекопитающие, парнокопытные, птицы, животное;
- 15) корабль, пароход, парусник, корвет.

По согласованию с преподавателем можно реализовать модель программной системы (включающей все необходимые классы и диаграмму наследования) для курсовой работы.

Контрольные вопросы

1. Каким образом строятся модели программных систем ?
2. Необходимость проектирования.
3. Разбиение системы на компоненты.
4. Какие уровни сложности системы вы знаете ?
5. Насколько эффективно использование Java для проектирования программных систем ?
6. Преимущества Java для проектирования программных систем ?
7. Что такое UML ?
8. Связь модели классов и диаграмм классов.

Лабораторная работа №9 «Разработка многопоточных приложений»

Цель работы: изучить создание многопоточных приложений с использованием языка программирования JAVA.

Теоретические сведения

Класс Thread инкапсулирует все средства, которые могут вам потребоваться при работе с подпроцессами. При запуске Java-программы в ней уже есть один выполняющийся подпроцесс. Вы всегда можете выяснить, какой именно подпроцесс выполняется в данный момент, с помощью вызова статического метода Thread.currentThread(). После того, как вы получите дескриптор подпроцесса, вы можете выполнять над этим подпроцессом различные операции даже в том случае, когда параллельные подпроцессы отсутствуют. В очередном примере показано, как можно управлять выполняющимся в данный момент подпроцессом:

```
class CurrentThreadDemo {
```

```

public static void main(String args[]) {
    Thread t = Thread.currentThread();
    t.setName("My Thread");
    System.out.println("current thread: " + t);
    try {
        for (int n = 5; n > 0; n--) {
            System.out.println(" " + n);
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {
            System.out.println("interrupted");
        }
    }
}

```

В этом примере текущий подпроцесс хранится в локальной переменной `t`. Затем мы используем эту переменную для вызова метода `setName`, который изменяет внутреннее имя подпроцесса на “My Thread”, с тем, чтобы вывод программы был удобочитаемым. На следующем шаге мы входим в цикл, в котором ведется обратный отсчет от 5, причем на каждой итерации с помощью вызова метода `Thread.sleep()` делается пауза длительностью в 1 секунду. Аргументом для этого метода является значение временного интервала в миллисекундах, хотя системные часы на многих платформах не позволяют точно выдерживать интервалы короче 10 миллисекунд. Обратите внимание — цикл заключен в `try/catch` блок. Дело в том, что метод `Thread.sleep()` может возбуждать исключение `InterruptedException`. Это исключение возбуждается в том случае, если какому-либо другому подпроцессу понадобится прервать данный подпроцесс. В данном примере мы в такой ситуации просто выводим сообщение о перехвате исключения.

Обратите внимание на то, что в текстовом представлении объекта `Thread` содержится заданное нами имя легковесного процесса — `My Thread`. Число 5 — это приоритет подпроцесса, оно соответствует приоритету по умолчанию, “main” — имя группы подпроцессов, к которой принадлежит данный подпроцесс.

Runnable

Не очень интересно работать только с одним подпроцессом, а как можно создать еще один? Для этого нам понадобится другой экземпляр класса `Thread`. При создании нового объекта `Thread` ему нужно указать, какой программный код он должен выполнять. Вы можете запустить подпроцесс с помощью любого объекта, реализующего интерфейс `Runnable`. Для того, чтобы реализовать этот интерфейс, класс должен предоставить определение метода `run`. Ниже приведен пример, в котором создается новый подпроцесс.

```

class ThreadDemo implements Runnable {
    ThreadDemo() {
        Thread ct = Thread.currentThread();
        System.out.println("currentThread: " + ct);
        Thread t = new Thread(this, "Demo Thread");
    }
}

```

```

System.out.println("Thread created: " + t);
t.start();
try {
    Thread.sleep(3000);
}
catch (InterruptedException e) {
    System.out.println("interrupted");
}
System.out.println("exiting main thread");
}
public void run() {
    try {
        for (int i = 5; i > 0; i--) {
            System.out.println(i);
            Thread.sleep(1000);
        }
    }
    catch (InterruptedException e) {
        System.out.println("child interrupted");
    }
    System.out.println("exiting child thread");
}
public static void main(String args[]) {
    new ThreadDemo();
}
}

```

Обратите внимание на то, что цикл внутри метода run выглядит точно так же, как и в предыдущем примере, только на этот раз он выполняется в другом подпроцессе. Подпроцесс main с помощью оператора new Thread(this, "Demo Thread") создает новый объект класса Thread, причем первый параметр конструктора — this — указывает, что нам хочется вызвать метод run текущего объекта. Затем мы вызываем метод start, который запускает подпроцесс, выполняющий метод run. После этого основной подпроцесс (main) переводится в состояние ожидания на три секунды, затем выводит сообщение и завершает работу. Второй подпроцесс — “Demo Thread” — при этом по-прежнему выполняет итерации в цикле метода run до тех пор пока значение счетчика цикла не уменьшится до нуля.

Приоритеты подпроцессов

Если вы хотите добиться от Java предсказуемого независимого от платформы поведения, вам следует проектировать свои подпроцессы таким образом, чтобы они по своей воле освобождали процессор. Ниже приведен пример с двумя подпроцессами с различными приоритетами, которые не ведут себя одинаково на различных платформах. Приоритет одного из подпроцессов с помощью вызова setPriority устанавливается на два уровня выше Thread. NORM_PRIORITY, то есть, умалчиваемого приоритета. У другого подпроцесса приоритет, наоборот, на два уровня ниже. Оба этих подпроцесса запускаются и работают в течение 10 секунд. Каждый из них

выполняет цикл, в котором увеличивается значение переменной-счетчика. Через десять секунд после их запуска основной подпроцесс останавливает их работу, присваивая условию завершения цикла while значение true и выводит значения счетчиков, показывающих, сколько итераций цикла успел выполнить каждый из подпроцессов.

```
class Clicker implements Runnable {  
int click = 0;  
private Thread t;  
private boolean running = true;  
public clicker(int p) {  
t = new Thread(this);  
t.setPriority(p);  
}  
public void run() {  
while (running) {  
click++;  
} }  
public void stop() {  
running = false; }  
public void start() {  
t.start();  
} }  
class HiLoPri {  
public static void main(String args[]) {  
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);  
clicker hi = new clicker(Thread.NORM_PRIORITY + 2);  
clicker lo = new clicker(Thread.NORM_PRIORITY - 2);  
lo.start();  
hi.start();  
try Thread.sleep(-10000) {  
}  
catch (Exception e) {  
}  
lo.stop();  
hi.stop();  
System.out.println(lo.click + " vs. " + hi.click);  
} }
```

По значениям, фигурирующим в распечатке, можно заключить, что подпроцессу с низким приоритетом достается меньше на 25 процентов времени процессора:

```
C:\>java HiLoPri  
304300 vs. 4066666
```

Синхронизация

Когда двум или более подпроцессам требуется параллельный доступ к одним и тем же данным (иначе говоря, к совместно используемому ресурсу),

нужно позаботиться о том, чтобы в каждый конкретный момент времени доступ к этим данным предоставлялся только одному из подпроцессов. Java для такой синхронизации предоставляет уникальную, встроенную в язык программирования поддержку. В других системах с параллельными подпроцессами существует понятие *монитора*. Монитор — это объект, используемый как защелка. Только один из подпроцессов может в данный момент времени владеть монитором. Когда под-процесс получает эту защелку, говорят, что он *вошел* в монитор. Все остальные подпроцессы, пытающиеся войти в тот же монитор, будут заморожены до тех пор пока подпроцесс-владелец не выйдет из монитора.

Вы можете видеть из приведенного ниже результата работы программы, что `sleep` в методе `call` приводит к переключению контекста между подпроцессами, так что вывод наших 3 строк-сообщений перемешивается:

```
[Hello.  
[Synchronized  
]  
[World  
]  
]
```

Это происходит потому, что в нашем примере нет ничего, способного помешать разным подпроцессам вызывать одновременно один и тот же метод одного и того же объекта. Для такой ситуации есть даже специальный термин — *race condition* (состояние гонки), означающий, что различные подпроцессы пытаются опередить друг друга, чтобы завершить выполнение одного и того же метода. В этом примере для того, чтобы это состояние было очевидным и повторяемым, использован вызов `sleep`. В реальных же ситуациях это состояние, как правило, трудноуловимо, поскольку непонятно, где именно происходит переключение контекста, и этот эффект менее заметен и не всегда воспроизводится от запуска к запуску программы. Так что если у вас есть метод (или целая группа методов), который манипулирует внутренним состоянием объекта, используемого в программе с параллельными подпроцессами, во избежание состояния гонки вам следует использовать в его заголовке ключевое слово `synchronized`.

Взаимодействие подпроцессов

В Java имеется элегантный механизм общения между подпроцессами, основанный на методах `wait`, `notify` и `notifyAll`. Эти методы реализованы, как `final`-методы класса `Object`, так что они имеются в любом Java-классе. Все эти методы должны вызываться только из синхронизированных методов. Правила использования этих методов очень просты:

- `wait` — приводит к тому, что текущий подпроцесс отдает управление и переходит в режим ожидания — до тех пор пока другой под-процесс не вызовет метод `notify` с тем же объектом.
- `notify` — выводит из состояния ожидания первый из подпроцессов, вызвавших `wait` с данным объектом.

- `notifyAll` — выводит из состояния ожидания все подпроцессы, вызвавшие `wait` с данным объектом.

Ниже приведен пример программы с наивной реализацией проблемы поставщик-потребитель. Эта программа состоит из четырех простых классов: класса `Q`, представляющего собой нашу реализацию очереди, доступ к которой мы пытаемся синхронизовать; поставщика (класс `Producer`), выполняющегося в отдельном подпроцессе и помещающего данные в очередь; потребителя (класс `Consumer`), тоже представляющего собой подпроцесс и извлекающего данные из очереди; и, наконец, крохотного класса `PC`, который создает по одному объекту каждого из перечисленных классов.

```
class Q {
    int n;
    synchronized int get() {
        System.out.println("Got: " + n);
        return n;
    }
    synchronized void put(int n) {
        this.n = n;
        System.out.println("Put: " + n);
    }
}
class Producer implements Runnable {
    Q q;
    Producer(Q q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int i = 0;
        while (true) {
            q.put(i++);
        }
    }
}
class Consumer implements Runnable {
    Q q;
    Consumer(Q q) {
        this.q = q;
        new Thread(this, "Consumer").start();
    }
    public void run() {
        while (true) {
            q.get();
        }
    }
}
class PC {
    public static void main(String args[]) {
```

```

Q q = new Q();
new Producer(q);
new Consumer(q);
} }

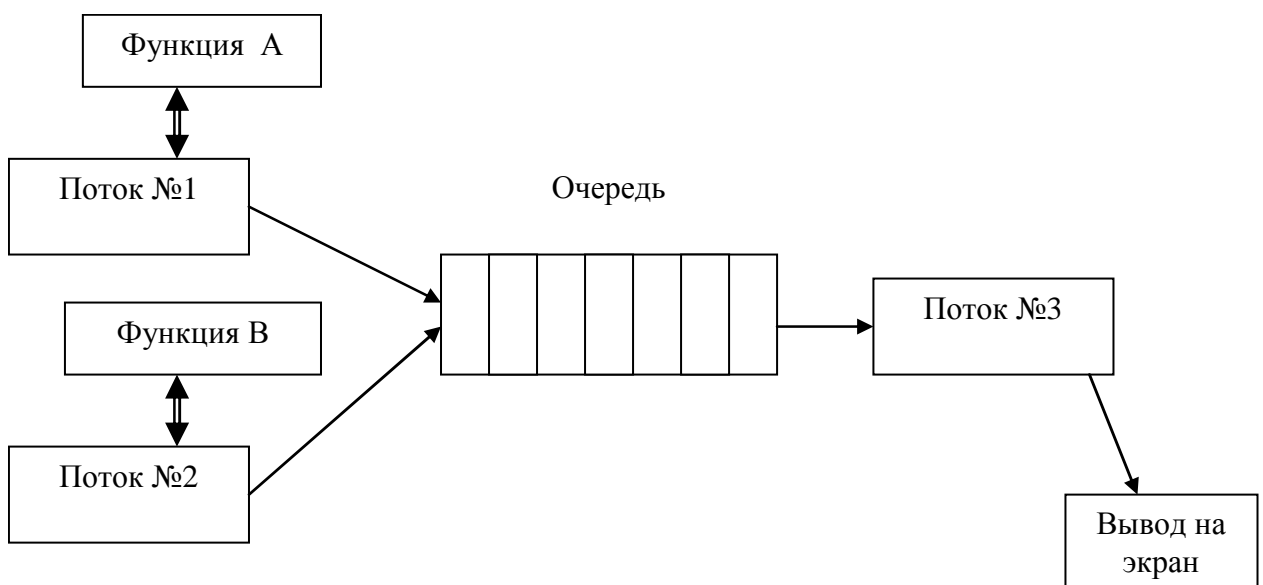
```

Хотя методы put и get класса Q синхронизованы, в нашем примере нет ничего, что бы могло помешать поставщику переписывать данные по того, как их получит потребитель, и наоборот, потребителю ничего не мешает многократно считывать одни и те же данные.

Как видите, после того, как поставщик помещает в переменную n значение 1, потребитель начинает работать и извлекает это значение 5 раз подряд. Положение можно исправить, если поставщик будет при занесении нового значения устанавливать флаг, например, заносить в логическую переменную значение true, после чего будет в цикле проверять ее значение до тех пор пока поставщик не обработает данные и не сбросит флаг в false.

Задание на лабораторную работу

Разработать приложение, в котором выполняется следующий алгоритм: два параллельных потока циклически выполняют вызов функций (согласно варианта). Каждый поток использует свою функцию. Результаты работы каждый поток помещает в общую очередь конечной длины. Третий поток забирает из очереди результаты работы функций и отображает их на экране в произвольной форме. Схема взаимодействия потоков приведена на рисунке:



Очередь должна быть реализована с использованием объектов синхронизации.

Потоки должны корректно завершаться при завершении приложения.

Для реализации выбрать любые две функции из таблицы согласно варианта.

№	Приложение	Функции
1	Целочисленный калькулятор	Операции: сложение, вычитание, умножение, целочисленное деление, НОД (наибольший общий делитель)
2	Калькулятор чисел с плавающей точкой	Операции: сложение, вычитание, умножение, деление квадратный корень \sqrt{x}
3	Математический калькулятор №1	Операции: $\sin x$, $\cos x$, e^x , $\log_{10} x$, $\log_2 x$
4	Математический калькулятор №2	Операции: $\sin x$, $\cos x$, $\operatorname{tg} x$, $\sqrt[y]{x}$, x^y ,
5	Редактор строки	Операции: сложение, замена одной подстроки другой, замена строчных букв прописными (в т.ч. для русских букв)
6	Редактор строки	Операции: «переворот» текста, перевод в транслитерацию, замена прописных букв строчными (в т.ч. для русских букв)
7	Работа с комплексными числами	Операции: сложение, вычитание, умножение
8	Восьмеричный калькулятор	Операции: Сложение, вычитание, умножение, остаток от деления
9	Троичный калькулятор	Операции: Сложение, вычитание, умножение, остаток от деления
10	Конвертер целых чисел №1	Преобразование в системы счисления: 16-тиричную, 8-ричную, 3-ичную, 2-ичную
11	Конвертер целых чисел №2	Преобразование из систем счисления: 16-тиричной, 8-ричной, 3-ичной, 2-ичной
12	Перевод числа от 0 до 99 в текстовый вид	Перевод на языки: русский, белорусский, английский
13	Конвертер единиц длины	Перевод длины из метров в: километры, мили, ярды, футы, дюймы
14	Проверка делимости чисел	Проверка делимости целых чисел на: 2, 3, 5, 10
15	Преобразование текста №1	Замена всех: цифр на символ «*», пробелов на символ «_», русских букв на символ «#»

№	Приложение	Функции
16	Преобразование текста №2	Преобразование текста: каждое новое слово с большой буквы, а остальной текст – маленькими (и для русских букв), каждое новое предложение с большой буквы, а остальной текст – маленькими (и для русских букв).
17	Преобразование текста №3	Замена в тексте: все гласные буквы становятся большими, все согласные буквы становятся большими, все знаки препинания дублируются (удваиваются).
18	Конвертер единиц объёма	Перевод объёма из литров в: галлоны, кубические футы, кубические дюймы
19	Перевод единиц времени	Перевод отрезка времени, заданного в виде числа суток в: число часов, число минут, число секунд
20	Логические операции	Операции с логическими операндами (Истина, Ложь): «И», «ИЛИ», «Исключающее ИЛИ».

Контрольные вопросы

1. Что такое процесс и поток? В чем разница между ними?
2. Чем определяется порядок передачи управления потокам?
3. Что такое приоритет потока, каков диапазон приоритетов?
4. Может ли быть изменен приоритет выполняющегося потока?
5. Будет ли низкоприоритетный поток исполняться, если существуют высокоприоритетные потоки, постоянно нуждающиеся в процессорном времени?
6. Какие есть способы реализации многозадачности в Java?
7. Что необходимо сделать для создания подкласса потоков (подкласса Thread)?
8. Когда запускается на выполнение метод run() подкласса Thread?
9. Что произойдет при запуске задачи, если метод run определить, например, так:
10. `public void run(int argument) { ... } ?`

Цель работы: изучить способы создания веб-служб в Java

Теоретические сведения

Web-сервисы - новое слово в технологии распределенных систем. Спецификация Open Net Environment (ONE) корпорации Sun Microsystems и инициатива .Net корпорации Microsoft обеспечивают инфраструктуры для написания и развертывания Web-сервисов. В настоящий момент имеется несколько определений Web-сервиса. Web-сервисом может быть любое приложение, имеющее доступ к Web, например, Web-страница с динамическим содержанием. В более узком смысле Web-сервис - это приложение, которое предоставляет открытый интерфейс, пригодный для использования другими приложениями в Web.

Спецификация ONE Sun требует, чтобы Web-сервисы были доступны через HTTP и другие Web-протоколы, чтобы дать возможность обмениваться информацией посредством XML-сообщений и чтобы их можно было найти через специальные сервисы - сервисы поиска. Для доступа к Web-сервисам разработан специальный протокол - Simple Object Access Protocol (SOAP), который представляет средства взаимодействия на базе XML для многих Web-сервисов. Web-сервисы особенно привлекательны тем, что могут обеспечить высокую степень совместимости между различными системами. Огромный потенциал Web-сервисов определяется не технологией, примененной для их создания. HTTP, XML и другие протоколы, используемые Web-сервисами, не новы. Функциональная совместимость и масштабируемость Web-сервисов подразумевает, что разработчики могут быстро создавать большие приложения и более крупные Web-сервисы из меньших Web-сервисов. Спецификация Sun Open Net Environment описывает архитектуру для создания интеллектуальных Web-сервисов. Интеллектуальные Web-сервисы задействуют общее операционное окружение. Совместно используя контекст, интеллектуальные Web-сервисы могут выполнять стандартную аутентификацию для финансовых транзакций, предоставлять рекомендации и указания в зависимости от географического местоположения компаний, участвующих в электронном бизнесе. Согласно определению W3C, "WSDL - формат XML для описания сетевых сервисов как набора конечных операций, работающих при помощи сообщений, содержащих документно-ориентированную или процедурно-ориентированную информацию". Документ WSDL полностью описывает интерфейс Web-сервиса с внешним миром. Он предоставляет информацию об услугах, которые можно получить, воспользовавшись методами сервиса, и способах обращения к этим методам. Таким образом, в случае если сигнатура метода Web-сервиса точно не известна (например, она изменилась со временем), у целевого Web-сервиса может быть запрошено WSDL-описание - файл, в котором эта информация будет содержаться.

Следующим слоем технологии является сервис Universal Description, Discovery and Integration (UDDI). Эта технология предполагает ведение

реестра Web -сервисов. Подключившись к этому реестру, потребитель сможет найти Web -сервисы, которые наилучшим образом подходят для решения его задач. Технология UDDI дает возможность поиска и публикации нужного сервиса, причем эти операции могут быть выполнены как человеком, так и другим Web -сервисом или специальной программой-клиентом. UDDI, в свою очередь, также представляет собой Web -сервис.

Таким образом, Web -сервисы являются еще одной реализацией системного программного обеспечения промежуточного слоя. Отличительной чертой этой технологии является ее независимость от используемого программного и аппаратного обеспечения, а также использование широко применяемых открытых стандартов (таких как XML) и стандартных коммуникационных протоколов.

В настоящее время Web -сервисы являются очень активно продвигаемой технологией и позиционируются как средство решения целого ряда задач.

Следует отметить, что с их применением могут строиться и так называемые "стандартные" приложения, где в качестве Web -сервиса оформляется серверная часть.

Простой протокол доступа к объектам (SOAP)

Базовым протоколом, обеспечивающим взаимодействие в среде Web -сервисов, является протокол SOAP.

Протокол SOAP разработали корпорации IBM, Lotus Development Corporation, Microsoft, Develop-Mentor и Userland Software. Этот протокол основан на HTTP-XML. Он позволяет приложениям взаимодействовать между собой через Internet, используя для этого XML -документы, называемые сообщениями SOAP. Протокол SOAP совместим с любой объектной моделью, поскольку он включает только те функции и методы, которые абсолютно необходимы для формирования коммуникационной инфраструктуры. Таким образом, SOAP является независимым от платформы и конкретных приложений, а для его реализации может применяться любой язык программирования. SOAP поддерживает практически любой транспортный протокол. SOAP также поддерживает любые методы кодирования данных, которые позволяют приложениям, основанным на SOAP, посылать в сообщениях SOAP информацию практически любого типа (например, изображения, объекты, документы и т.д.).

Сообщение SOAP содержит конверт, который описывает содержимое, предполагаемого получателя сообщения и требования к обработке сообщения.

Необязательный элемент **header** (заголовок) сообщения SOAP содержит инструкции по обработке для приложений, которые принимают сообщение. Заголовок также может содержать информацию о маршрутизации. С помощью заголовка **header** поверх SOAP могут надстраиваться более сложные протоколы. Записи в заголовке могут модульно расширять сообщение для таких задач, как аутентификация, управление транзакциями и проведение платежей. Тело SOAP -сообщения содержит специфичные для

приложения данные, предназначенные для предполагаемого получателя сообщения.

Исходный код Web-сервиса

После того, как указанные настройки будут закончены, можно приступить непосредственно к примеру.

Файл с исходным кодом Web -сервиса располагается в директории `src`, называется **Hello.java** и имеет следующий вид:

```
1 // Hello.java
2 package helloservice.endpoint;
3
4 import javax.jws.WebMethod;
5 import javax.jws.WebService;
6
7 @WebService()
8 public class Hello {
9     private String message = new String("Hello, ");
10
11     @WebMethod()
12     public String sayHello(String name) {
13         return message + name + ".";
14     }
15 }
```

Первое, что обращает на себя внимание, - удивительная краткость написанного кода. Но не стоит обольщаться: дело в том, что технология разработки Web -сервисов, которую мы будем использовать, просто скрывает от разработчика большую часть работы по реализации Web -сервиса. Фактически, все, что должен сделать разработчик, - реализовать код самих вызываемых методов; абсолютно всю работу по реализации механизмов, позволяющих вызывать эти методы удаленно, берет на себя используемая нами технология.

Рассматриваемый класс **Hello** удовлетворяет всем указанным ограничениям. Кроме того, он объявляет единственный метод, аннотированный как **WebMethod** (строка 11), который принимает параметр типа **String** и возвращает его же с присоединенной в начале константной строкой.

Собственно, на этом разработка Web -сервиса заканчивается. Следующее, что необходимо сделать, - откомпилировать его, пропустить через утилиту **wsgen** для генерации вспомогательных классов, создать **war-file**, содержащий в себе откомпилированное приложение и необходимые ресурсы, и затем разместить и зарегистрировать его на сервере приложений.

В комплекте с примерами, поставляемыми в пакете The Java Web Services Tutorial, поставляются также скрипты для их компиляции. Эти скрипты предназначены для специального инструментального средства

компиляции, которое называется ant (исполняющая часть **ant** устанавливается вместе с Sun Java System Application Server).

Разработчики примеров для пакета The Java Web Services Tutorial постарались на славу, и теперь для компиляции и установки приложения необходимо выполнить лишь несколько простых команд. Мы воспользуемся этим обстоятельством, а затем подробно рассмотрим, что стоит за каждой из этих простых команд и какие действия при этом выполняются.

Компиляция и инсталляция на сервере приложений

Итак, первое, что предстоит сделать, - откомпилировать приложение. Для компиляции в настройках сборки определена специальная цель (target) - build.

Набрав в командной строке команду **asant build** (**asant** - вызов командного файла, запускающего **ant**, **build** - имя цели, которую он должен выполнить), получим следующий вывод:

Buildfile: build.xml

javaee-home-test:

init:

prepare:

[echo] Creating the required directories....

[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\
jaxws\helloservice\build

compile-service:

[echo] Compiling the server-side source code ...

[javac] Compiling 1 source file to H:\Java\
jwstutorial20_new\examples\jaxws\helloservice\build

[wsген] command line: wsimport -classpath

H:\Java\AppServer\lib\activation.jar;

H:\Java\AppServer\lib\admin-cli.jar;

H:\Java\AppServer\lib\appserv-admin.jar;

H:\Java\AppServer\lib\appserv-cmp.jar;

H:\Java\AppServer\lib\appserv-deployment-client.jar;

H:\Java\AppServer\lib\appserv-ext.jar;

H:\Java\AppServer\lib\appserv-jstl.jar;

H:\Java\AppServer\lib\appserv-jwsacc.jar;

H:\Java\AppServer\lib\appserv-launch.jar;

H:\Java\AppServer\lib\appserv-rt.jar;

H:\Java\AppServer\lib\appserv-tags.jar;

H:\Java\AppServer\lib\appserv-upgrade.jar;

H:\Java\AppServer\lib\appserv-ws.jar;

H:\Java\AppServer\lib\com-sun-commons-launcher.jar;

H:\Java\AppServer\lib\com-sun-commons-logging.jar;
H:\Java\AppServer\lib\dbschema.jar;
H:\Java\AppServer\lib\j2ee-svc.jar;
H:\Java\AppServer\lib\j2ee.jar;
H:\Java\AppServer\lib\javaee.jar;
H:\Java\AppServer\lib\jhall.jar;
H:\Java\AppServer\lib\jmxremote_optional.jar;
H:\Java\AppServer\lib\jsf-impl.jar;
H:\Java\AppServer\lib\mail.jar;
H:\Java\AppServer\lib\sun-appserv-ant.jar;
H:\Java\AppServer\lib\toplink-essentials-agent.jar;
H:\Java\AppServer\lib\toplink-essentials.jar;
H:\Java\AppServer\jdk\lib\tools.jar;
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\build -d
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\ build -keep -s
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\ build
-verbose helloservice.endpoint.Hello [wsgen] Note: ap round: 1
[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]
[wsgen] [should process method: sayHello hasWebMethods: true]
[wsgen] [endpointReferencesInterface: false]
[wsgen] [declaring class has WebSevice: true]
[wsgen] [returning: true]
[wsgen] [WrapperGen - method: sayHello(java.lang.String)]
[wsgen] [method.getDeclaringType():
helloservice.endpoint.Hello]
[wsgen] [requestWrapper: helloservice.endpoint.jaxws.SayHello]
[wsgen] [ProcessedMethods Class: java.lang.Object]
[wsgen] helloservice\endpoint\jaxws\SayHello.java
[wsgen] helloservice\endpoint\jaxws\SayHelloResponse.java
[wsgen] Note: ap round: 2
[wsgen] [completing model for endpoint:
helloservice.endpoint.Hello]
[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]
[wsgen] [should process method: sayHello hasWebMethods: true]
[wsgen] [endpointReferencesInterface: false]
[wsgen] [declaring class has WebSevice: true]
[wsgen] [returning: true]
[wsgen] [WebServiceReferenceCollector - method:
sayHello(java.lang.String)]
[wsgen] [ProcessedMethods Class: java.lang.Object]
build-service:
build:
BUILD SUCCESSFUL
Total time: 9 seconds

Поскольку мы намеренно включили опцию вывода отладочной информации для **ant**, вывод получился довольно обширный.

Первое, что делается для компиляции программы, - создается специальная директория **build**, в которую будут помещены откомпилированные модули. Она создается в текущей директории. Затем вызывается компилятор **javac**, который компилирует наш класс **Hello.java**, а результат компиляции кладет в директорию **build**. Поскольку класс **Hello** определен в пакете **helloservice.endpoint**, в директории **build** будет создана соответствующая система каталогов и файл **Hello.class** будет помещен в каталог **./build/helloservice/endpoint**.

Следующим шагом вызывается утилита **wsgen**, которая формирует вспомогательные классы. По умолчанию исходные коды этих классов после компиляции уничтожаются, однако, выставив опцию **keep=true** (эта и другие опции могут быть установлены в файле **build.properties**), исходные коды можно сохранить. Помещаются они в пакет **jaxws** того же пакета, которому принадлежит и класс. Соответственно, для нашего примера исходные файлы (а затем и откомпилированные классы) будут располагаться в директории **./build/helloservice/endpoint/jaxws**. После того как утилита **wsgen** отработала, мы имеем откомпилированный пакет **helloservice.endpoint.jaxws**, содержащий необходимые вспомогательные классы. На этом шаге компиляция нашего Web-сервиса закончена. Следующим этапом необходимо подготовить модуль развертывания. В нашем случае это делается с помощью команды:

```
asant create-war
```

Вывод получаем следующий:

```
Buildfile: build.xml
```

```
prepare-assemble:
```

```
[echo] Creating the assemble directory....
```

```
[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\  
jaxws\helloservice\assemble
```

```
[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\  
jaxws\helloservice\assemble\war
```

```
create-war:
```

```
[echo] Creating the WAR ...
```

```
[war] Building war: H:\Java\jwstutorial20_new\examples\  
jaxws\helloservice\assemble\war\hello-jaxws.war
```

```
BUILD SUCCESSFUL
```

```
Total time: 3 seconds
```

Создается отдельный каталог **assemble**, в нем создается каталог **war**, в котором формируется файл **hello-jaxws.war**. Этот файл представляет собой архив, в который помещены откомпилированные файлы нашего приложения и некоторые вспомогательные файлы. Теперь у нас полностью готов модуль

развертывания, который мы можем установить в сервере приложений. Установка может быть выполнена командой:

```
asant deploy
```

Результат выполнения команды следующий:

```
Buildfile: build.xml deploy:
```

```
admin_command_common:
```

```
[echo] Doing admin task deploy assemble/war/hello-jaxws.war
```

```
[sun-appserv-admin] Executing: deploy --port 4848 --host
```

```
localhost --passwordfile "H:\Java\jwstutorial20_new\examples\
```

```
common\admin-password.txt" --user admin assemble/war/
```

```
hello-jaxws.war
```

```
[sun-appserv-admin] Command deploy executed successfully.
```

BUILD SUCCESSFUL

Total time: 43 seconds

В процессе установки, кроме прочего, был сгенерирован WSDL -файл, описывающий установленный Web -сервис. Как уже говорилось, этот файл содержит полное описание Web -сервиса, включая названия его методов, а также количество и типы передаваемых и возвращаемых параметров. Этот файл является важной составляющей частью технологии, поскольку он позволяет строить приложения, осуществляющие динамические вызовы методов Web -сервисов. Кроме того, этот файл может быть использован для автоматической генерации вспомогательных классов (классов-прокси) для обращения к Web -сервису.

Тестирование Web-сервиса

Итак, наш Web -сервис успешно установлен. Осталось только убедиться в том, что он действительно работает. Чуть позже мы напишем специальное приложение-клиент, которое будет обращаться к нашему Web -сервису, а пока воспользуемся средствами, предоставляемыми нам Sun Java System Application Server. Дело в том, что этот сервер приложений способен самостоятельно динамически выстроить среду для вызова методов установленных в нем Web -сервисов. Всей необходимой информацией, а именно: имена публикуемых методов, количество и тип принимаемых и возвращаемых методами параметров - он обладает.

Для того чтобы воспользоваться указанной возможностью, нужно выбрать нужный нам сервис в списке сервисов (в правой части окна браузера) и нажать кнопку "Test" .

Откроется новое окно браузера, в котором отобразится динамически построенная сервером страница. На этой странице перечислены все опубликованные методы Web -сервиса (в нашем случае - один метод **sayHello**) и реализован интерфейс для их вызова. Если ввести в соответствующее поле строку и нажать кнопку - вызовется метод Web -сервиса и введенное значение будет передано ему в качестве параметра. Кроме всего прочего, на результирующей странице отобразятся SOAP -

сообщения, соответственно, отправленные Web -сервису и пришедшие от него в качестве ответа.

Результирующая страница будет иметь следующий вид.

На странице видны значения и типы переданных параметров, ответ, который возвратил метод Web -сервиса, - как и ожидалось, ответ представляет собой строку "Hello, Web-service test", - а также отправленный и полученный пакеты.

Таким образом, разработанный нами Web -сервис успешно установлен в сервере приложений и может обрабатывать запросы клиентов, в чем мы убедились, используя тестовое окружение, предоставляемое сервером приложений.

Задание на лабораторную работу

Создать приложение, реализующее веб-сервис. Веб-сервис должен выполнять операции в соответствии с вариантом:

№	Приложение	Функции веб-сервиса
1	Целочисленный калькулятор	Операции: сложение, вычитание, умножение, целочисленное деление, НОД (наибольший общий делитель)
2	Калькулятор чисел с плавающей точкой	Операции: сложение, вычитание, умножение, деление квадратный корень \sqrt{x}
3	Математический калькулятор №1	Операции: $\sin x$, $\cos x$, e^x , $\log_{10} x$, $\log_2 x$
4	Математический калькулятор №2	Операции: $\sin x$, $\cos x$, $\operatorname{tg} x$, $\sqrt[n]{x}$, x^y ,
5	Редактор строки	Операции: сложение, замена одной подстроки другой, замена строчных букв прописными (в т.ч. для русских букв)
6	Редактор строки	Операции: «переворот» текста, перевод в транслитерацию, замена прописных букв строчными (в т.ч. для русских букв)
7	Работа с комплексными числами	Операции: сложение, вычитание, умножение
8	Восьмеричный калькулятор	Операции: Сложение, вычитание, умножение, остаток от деления

№	Приложение	Функции веб-сервиса
9	Троичный калькулятор	Операции: Сложение, вычитание, умножение, остаток от деления
10	Конвертер целых чисел №1	Преобразование в системы счисления: 16-тиричную, 8-ричную, 3-ичную, 2-ичную
11	Конвертер целых чисел №2	Преобразование из систем счисления: 16-тиричной, 8-ричной, 3-ичной, 2-ичной
12	Перевод числа от 0 до 99 в текстовый вид	Перевод на языки: русский, белорусский, английский
13	Конвертер единиц длины	Перевод длины из метров в: километры, мили, ярды, футы, дюймы
14	Проверка делимости чисел	Проверка делимости целых чисел на: 2, 3, 5, 10
15	Преобразование текста №1	Замена всех: цифр на символ «*», пробелов на символ «_», русских букв на символ «#»
16	Преобразование текста №2	Преобразование текста: каждое новое слово с большой буквы, а остальной текст – маленькими (и для русских букв), каждое новое предложение с большой буквы, а остальной текст – маленькими (и для русских букв).
17	Преобразование текста №3	Замена в тексте: все гласные буквы становятся большими, все согласные буквы становятся большими, все знаки препинания дублируются (удваиваются).
18	Конвертер единиц объёма	Перевод объёма из литров в: галлоны, кубические футы, кубические дюймы
19	Перевод единиц времени	Перевод отрезка времени, заданного в виде числа суток в: число часов, число минут, число секунд
20	Логические операции	Операции с логическими операндами (Истина, Ложь): «И», «ИЛИ», «Исключающее ИЛИ».

Контрольные вопросы

1. Назовите преимущества веб-сервисов.
2. Порядок развертывания веб-сервиса.
3. Автоматическое и ручное развертывание веб-сервиса.
4. Использование аннотация при написании веб-сервиса.
5. Что такое System Application Server ?

Лабораторная работа №11 «Создание библиотек в Java»

Цель работы: изучить способы создания собственных библиотек в Java

Теоретические сведения

Библиотеки создаются с использованием программы *jar*.

Синтаксис вызова

jar c[t|x[f][m][v] [jar-файл] [файл описания] [файлы]

Описание

Программа *jar* используется для создания архивных файлов Java (JAR) и работы с ними. JAR-файл представляет собой сжатый ZIP-файл с дополнительным файлом описания. Синтаксис команды *jar* напоминает синтаксис команды *tar* (tape archive — архив на магнитной ленте) ОС UNIX.

Параметры командной строки *jar* задаются в виде блока записанных слитно букв, которые передаются в виде одного аргумента, а не через отдельные аргументы командной строки. Первая буква такого аргумента задает необходимое действие, которое должна выполнить программа *jar*. Остальные буквы в этом аргументе являются необязательными. Различные аргументы файлов зависят от того, какие буквы параметров заданы.

Параметры

Первым аргументом командной строки *jar* является набор символов, задающих операцию, которая должна быть выполнена. Первый символ определяет основную операцию и является обязательным. Возможны следующие варианты:

c Создать новый JAR-архив. В качестве последних аргументов командной строки *jar* необходимо указать список файлов и/или каталогов.

t Вывести список файлов, содержащихся в JAR-архиве. Если задано имя JAR-файла с помощью параметра *f*, то список файлов выводится для него. В противном случае имя JAR-файла читается со стандартного устройства ввода.

x Извлечь содержимое JAR-архива. Если задано имя JAR-файла с помощью параметра *f*, то извлекается содержимое этого файла. В противном случае имя JAR-файла читается со стандартного устройства ввода. Когда командная строка завершается списком файлов и/или каталогов, из JAR-

архива извлекаются только файлы и каталоги, перечисленные в этом списке. В противном случае из архива извлекаются все файлы.

Вслед за идентификатором, определяющим выполняемое действие, могут следовать необязательные параметры:

f Указывает на то, что имя JAR-файла, который необходимо создать, из которого нужно извлечь файлы или получить список содержащихся файлов, задается в командной строке. Если **f** используется вместе с **c**, **t** или **x**, имя JAR-файла должно задаваться в качестве второго аргумента командной строки вызова `jar` (т.е. оно должно располагаться непосредственно за блоком параметров). Когда этот параметр не задан, `jar` записывает создаваемый JAR-файл в стандартное устройство вывода или читает его со стандартного устройства ввода.

m Используется только в сочетании с параметром **c** и указывает на то, что `jar` должна читать файл описания, указанный в командной строке и использовать его в качестве основы для создания описания, которое включается в JAR-файл. Когда этот параметр задается после параметра **f**, имя файла описания должно указываться после имени создаваемого архива. Если **m** стоит перед параметром **f**, то имя файла описания должно предшествовать имени файла создаваемого архива.

v Описание. Если этот параметр задается вместе с параметром **c**, `jar` выводит имя каждого добавляемого в архив файла со статистикой его сжатия. Когда параметр используется в сочетании с **t**, `jar` выводит список файлов, в котором кроме имени файла содержится его объем и дата последнего изменения. Если **v** указывается одновременно с **x**, то `jar` выводит имя каждого извлекаемого из архива файла.

Примеры

Создание простого JAR-архива:

```
% jar cvf my.jar *.java images
```

Получение списка содержимого архива:

```
% jar tvf your.jar
```

Извлечение файла описания из JAR-файла:

```
% jar xf the.jar META-INF/MANIFEST.MF
```

Создание JAR-файла с заданным описанием:

```
% jar cfmv YesNoDialog.jar manifest.stub oreilly/beans/yesno
```

Смотри также

javakey

Java — интерпретатор Java

Задание на лабораторную работу

Программную систему, разработанную в лабораторной работе № 8, разбить на пакеты и реализовать в виде `jar`-библиотеки. Аргументировать принадлежность каждого класса каждому пакету.

Контрольные вопросы

1. Что такое библиотека ?
2. Преимущества использования библиотек.
3. Недостатки использования библиотек.
4. Этапы создания библиотек в Java.
5. Использование утилиты jar для создания библиотек в Java.

Лабораторная работа №12 «Обработка данных с использованием XML»

Цель работы: изучить методы обработки данных с использованием XML в Java

Теоретические сведения

Sun обеспечивает с помощью Java API для XML Parsing (JAXP) обеспечивает два способа чтения XML документов:

- Модель событий,
- Модель дерева.

JDOM – открытый проект, который обеспечивает механизм работы с XML документами с помощью классов коллекций Java. Входящий в Simple API для XML (SAX) парсер – управляется событиями: программа регистрирует прослушиватель для парсера, а парсер получает данные из потока, определяя XML элементы. Модель Document Object Model (DOM) строит дерево представления XML документа и обеспечивает Application Programmers Interface (API) для доступа к данным в дереве.

SAX и DOM используются с разными целями и имеют свои достоинства и недостатки. Парсер SAX использует малый объем памяти, потому что он не сохраняет информацию, а просто просматривает документ и сообщает о тех элементах, которые обнаружены. Он достаточно быстр, но при его использовании надо программисту самому строить структуры в памяти для сохранения данных из документа. Модель DOM медленная и требует много памяти, но она обеспечивает сохранение данных в памяти, дает возможность использовать существующие классы и методы для доступа к данным.

Если Вы используете XML, чтобы загружать данные в Ваши структуры, то лучше применять SAX. Если же Вам нужно найти какое-то значение, содержащееся в документе XML или вычислить что-то на основе данных в XML (например, подсчитать количество книг), следует использовать SAX.

С другой стороны, если Вам нужен доступ ко всему документу в памяти, следует применять DOM. Если Вы изменяете содержимое документа, а затем выводите модифицированные данные (например, сохраняете в файл), следует применять DOM.

Выбор обусловлен тем, как Вы собираетесь использовать данные, полученные из документа XML.

Все классы, необходимые для работы с XML, включены в JSDK, начиная с версии 1.5.

Разбор XML документов с помощью Simple API для XML (SAX)

Использование SAX-разбора для XML документа требует наличия SAX-парсера и обработчика той информации, которая будет получена от парсера.

JAXP обеспечивает класс SAX -парсер: `javax.xml.parsers.SAXParser`; причем SAX парсер доступен через класс `javax.xml.parsers.SAXParserFactory`. Класс `SAXParserFactory` – это набор методов для получения SAX-парсеров. Вызов метода `newSAXParser()` позволяет получить ссылку на сконфигурированный SAX-парсер. Оба эти класса – абстрактные, поэтому Вы должны обеспечить реализацию некоторых методов классов.

Ваша задача – создать обработчик документа. Он определяется классом `org.xml.sax.helpers.DefaultHandler`. Класс реализует несколько интерфейсов, но самый интересный - интерфейс `org.xml.sax.ContentHandler`.

Эти методы вызывает парсер документа. Когда начинается разбор документа, вызывается метод `startDocument()`, когда найден элемент `<books>`, вызывается метод `startElement()`.

Класс `DefaultHandler` реализует интерфейсы `org.xml.sax.DTDHandler`, `org.xml.sax.EntityHandler` и `org.xml.sax.ErrorHandler`. Этот класс предназначен для обработки ошибок при разборе. Для реализации обработчика лучше, чтобы он наследовал от класса `DefaultHandler`.

Обрабатываем сообщения SAX-парсера в методах нашего класса `DefaultHandler`.

```
001: public class SAXBook {
002:     private String title;
003:     private String author;
004:     private String category;
005:     private float price;
006:
007:     public SAXBook() {
008:     }
009:
010:     public SAXBook( String title,
011:                    String author,
012:                    String category,
013:                    float price ) {
014:         this.title = title;
015:         this.author = author;
016:         this.category = category;
```

```

017:    this.price = price;
018: }
019:
020: public String getTitle() {
021:     return this.title;
022: }
023:
024: public void setTitle( String title ) {
025:     this.title = title;
026: }
027:
028: public String getAuthor() {
029:     return this.author;
030: }
031:
032: public void setAuthor( String author ) {
033:     this.author = author;
034: }
035:
036: public String getCategory() {
037:     return this.category;
038: }
039:
040: public void setCategory( String category ) {
041:     this.category = category;
042: }
043:
044: public float getPrice() {
045:     return this.price;
046: }
047:
048: public void setPrice( float price ) {
049:     this.price = price;
050: }
051:
052: public String toString() {
053:     return "Book: " + title + ", " + category + ", " +
        author + ", " + price;
054: }
055:}

```

Листинг – это простой JavaBean для полей класса: title, author, category и price. Определен также метод toString() для получения значений всех полей.

```

001:import java.util.ArrayList;

```

```

002:
003:public class SAXBooks {
004:    private ArrayList bookList = new ArrayList();
005:
006:    public SAXBooks() {
007:    }
008:
009:    public void addBook( SAXBook book ) {
010:        this.bookList.add( book );
011:    }
012:
013:    public SAXBook getBook( int index ) {
014:        if( index >= bookList.size() ) {
015:            return null;
016:        }
017:        return( SAXBook )bookList.get( index );
018:    }
019:
020:    public SAXBook getLastBook() {
021:        return this.getBook( this.getBookSize() - 1 );
022:    }
023:
024:    public int getBookSize() {
025:        return bookList.size();
026:    }
027:}

```

Листинг показывает класс `SAXBooks`, поддерживающий коллекцию объектов `SAXBook` в массиве `java.util.ArrayList` и обеспечивающий методы для вставки книги, получения данных о книге, получения общего числа книг. В этом классе есть еще один метод `getLastBook()`, который возвращает последнюю книгу в массиве `ArrayList`. Этот метод используется в файле в листинге:

```

001:import java.io.*;
002:import org.xml.sax.*;
003:import org.xml.sax.helpers.DefaultHandler;
004:import javax.xml.parsers.SAXParserFactory;
005:import javax.xml.parsers.ParserConfigurationException;
006:import javax.xml.parsers.SAXParser;
007:
008:public class SAXSample {
009:    public static void main( String[] args ) {
010:        try {
011:            File file = new File( "book.xml" );
012:            if( !file.exists() ) {

```

```

013:         System.out.println( "Couldn't find file..." );
014:         return;
015:     }
016:
017:     // Use the default (non-validating) parser
018:     SAXParserFactory factory = SAXParserFactory.newInstance();
019:
020:     // Create an instance of our handler
021:     MyHandler handler = new MyHandler();
022:
023:     // Parse the file
024:     SAXParser saxParser = factory.newSAXParser();
025:     saxParser.parse( file, handler );
026:     SAXBooks books = handler.getBooks();
027:
028:     for( int i=0; i<books.getBookSize(); i++ ) {
029:         SAXBook book = books.getBook( i );
030:         System.out.println( book );
031:     }
032:
033: }
034: catch( Throwable t ) {
035:     t.printStackTrace();
036: }
037: }
038: }

```

Задание на лабораторную работу

Выполнить задания, сохраняя объекты приложения в одном или нескольких файлах XML. Четные варианты используют SAX-парсер, нечетные DOM-парсер.

1. Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве. Студентов, Преподавателей и Курсов при обучении может быть несколько.
2. Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может за-блокировать КК за превышение кредита.
3. Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из

Больницы по окончании лечения, при нарушении режима или при иных обстоятельствах.

4. Система Вступительные экзамены. Абитуриент регистрируется на Факультет, сдает Экзамены. Преподаватель выставляет Оценку. Система подсчитывает средний балл и определяет Абитуриентов, зачисленных в учебное заведение.

5. Система Библиотека. Читатель оформляет Заказ на Книгу. Система осуществляет поиск в Каталоге. Библиотекарь выдает Читателю Книгу на абонемент или в читальный зал. При невозвращении Книги Читателем он может быть занесен Администратором в «черный список».

6. Система Конструкторское бюро. Заказчик представляет Техническое Задание (ТЗ) на проектирование многоэтажного Дома. Конструктор регистрирует ТЗ, определяет стоимость проектирования и строительства, выставляет Заказчику Счет за проектирование и создает Бригаду Конструкторов для выполнения Проекта.

7. Система Телефонная станция. Абонент оплачивает Счет за разговоры и Услуги, может попросить Администратора сменить номер и отказаться от услуг. Администратор изменяет номер, Услуги и временно отключает Абонента за неуплату.

8. Система Автобаза. Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля.

9. Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».

10. Система Железнодорожная касса. Пассажир делает Заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск подходящего Поезда. Пассажир делает выбор Поезда и получает Счет на оплату. Администратор вводит номера Поездов, промежуточные и конечные станции, цены.

11. Система Городской транспорт. На Маршрут назначаются Автобус, Троллейбус или Трамвай. Транспортные средства должны двигаться с определенным для каждого Маршрута интервалом. При поломке на Маршрут должен выходить резервный транспорт или увеличиваться интервал движения.

12. Система Аэрофлот. Администратор формирует летную Бригаду (пи-лоты, штурман, радист, стюардессы) на Рейс. Каждый Рейс выполняется Самолетом с определенной вместимостью и дальностью полета. Рейс может быть отменен из-за погодных условий в Аэропорту отлета или назначения. Аэропорт назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

13. Система Периодические издания. Читатель может сделать Заявку, предварительно выбрав периодические Издания из списка. Система подсчитывает сумму для оплаты. Читатель оплачивает заявку.

Администратор добавляет Заявку в «черный список», если Клиент не оплачивает её в определённый срок.

14. Система Заказ гостиницы. Клиент оставляет Заявку на Номер, указав количество мест в номере, класс апартаментов и время пребывания. Администратор рассматривает Заявку, подтверждает или отклоняет её. Результат просматривает Клиент. В случае подтверждения Заявки Клиент оплачивает услуги.

15. Система Жилищно-коммунальные услуги. Квартиросъемщик отправляет Заявку, в которой указывает род работ, масштаб и желаемое время выполнения. Диспетчер формирует соответствующую Бригаду и регистрирует её в Плате работ. Диспетчер может отклонить Заявку в случае занятости всех Бригад.

Контрольные вопросы

1. Типы парсеров в Java.
2. Этапы работы SAX- парсера.
3. Этапы работы DOM- парсера.
4. Преимущества SAX- парсера.
5. Преимущества DOM- парсера.
6. Недостатки SAX- парсера.
7. Недостатки DOM- парсера.

Список использованных источников

1. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 1. Графика, JavaBeans, интерфейс пользователя. - М.: Бином, 2003. ISBN: 5-9518-0017-X.
2. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 2. Распределенные приложения. - М.: Бином, 2003. ISBN: 5-9518-0051-X.
3. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 3. Корпоративные системы, сервлеты, JSP, web-сервисы. - М.: Бином, 2003. ISBN: 5-9518-0034-X.
4. Дейтел Х. М., Дейтел П. Д. Как программировать на Java. Книга 1. Основы программирования. - М.: Бином, 2003. ISBN: 5-9518-0015-3.
5. Дейтел Х. М., Дейтел П. Д. Как программировать на Java. Книга 2. Файлы, сети, базы данных. - М.: Бином, 2006. ISBN: 5-9518-0127-3.
6. Ноутон П., Шилдт Г. Java2. Наиболее полное руководство. - СПб.: BHV, 2001. ISBN: 0-07-211976-4, 5-94157-012-0.
7. Биллиг В.А. Основы программирования на Java/ - М.: Изд-во «Интернет-университет информационных технологий – ИНТУИТ.ру», 2006. – 488с.
8. Шилдт Г.С. Java: Учебный курс. – СПб.: Питер, 2002. – 512с.
9. Шилдт Г.С. Полный справочник по Java. – М.: Издательский дом «Вильямс», 2004. – 752с.