Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение «Томский государственный университет систем управления и радиоэлектроники»

(ТУСУР)

**УТВЕРЖДАЮ**

Заведующий кафедрой ЭМИС

_____ И.Г. Боровской

«____» _____ 2012 г.

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ И САМОСТОЯТЕЛЬНОЙ РАБОТЕ СТУДЕНТОВ

по дисциплине

# «Корпоративные информационные системы»

Составлено кафедрой «экономической математики, информатики и статистики»

Для студентов, обучающихся по направлению подготовки 230200 – Информационные системы

Форма обучения – очная

**Составитель:**

Должность, место работы _____ А.А. Матолыгин

«____» _____ 2012 г.

Томск 2012 г.

Методические указания к лабораторным работам и выполнению самостоятельной работы студентов составлены в соответствии с требованиями государственного образовательного стандарта высшего профессионального образования по направлению подготовки 230200 – Информационные системы.

Методические указания направлены на приобретение студентами необходимых навыков работы с информационными системами на предприятии, а так же применяемыми в них информационными технологиями и моделями и позволит их использовать в практической информатике и вычислительной технике. В рамках курса предусматривается изучение языка программирования Java, как одной из технологий построения распределенных вычислений в информационных системах.

**ОГЛАВЛЕНИЕ**

## Лабораторная работа 1. Структура программы на языке Java. Простейшие программы на языке Java

*Цель работы.* Изучить структуры программы на языке программирования Java, Примитивные типы данных аналогичны типам данных в языках C и C++.

Традиционно первой программой при изучении языков программирования предлагается вывод на экран сообщения. Но прежде чем эту программу записать, нужно сделать еще несколько предварительных замечаний.

Язык Java является интерпретируемым. Чтобы создать программу-приложение на Java, нужно в любом текстовом редакторе набрать текст программы, записать этот текстовый файл (при этом соблюдаются определенные правила, касающиеся имени файла, о которых будет сказано ниже). Затем этот текстовый файл компилируется: в случае отсутствия в тексте программы ошибок компилятор транслирует программу в платформенно-независимый байт-код, записываемый в один или несколько файлов с расширением *.class. В этом (этих) файле содержатся инструкции для так называемой виртуальной машины Java (JVM – Java Virtual Machine), которые будут выполняться (по крайней мере, так предполагается, и это предположение верно в подавляющем большинстве случаев) одинаково успешно на любой платформе, на которой есть JVM. Для запуска приложения необходимо запустить интерпретатор, являющийся составной частью JVM, которому нужно указать в качестве одного из параметров имя запускаемого на выполнение файла. Частности рассмотрим ниже на конкретном примере.

Разработчиком языка Java, его компилятора и интерпретатора является компания Sun (java.sun.com), и последние версии компилятора и виртуальной машины можно получить с сайта этой компании. Итак, первая программа – “Hello, Date!”

```
1  // HelloDate.java
2  import java.util.*;
3  public class HelloDate {
4      public static void main(String[] args) {
5      System.out.println("Hello, it's: ");
6      System.out.println(new Date());
7    }
8  }
```

Пример (HelloDate.java)

Двойной слэш «//» в первой строке означает комментарий, который распространяется до конца строки. В строке 2 предложение import загружает дополнительные классы из библиотеки java.util (в частности, класс Date). Автоматически (без использования предложения import) подключается только библиотека классов java.lang, в частности, классы System и String “берутся” как раз из нее. Одним из полей класса System является out, представляющий собой статический объект класса PrintStream, одним из методов которого является println().

В строке 3 стоит определение класса – ключевое слово public является спецификатором доступа, слово class является началом блока определения, далее следует имя класса. Имя файла должно совпадать с именем класса. Если в тексте файла определяется несколько классов, то имя файла должно совпадать с тем из них, который имеет спецификатор доступа public. Кроме того, нужно иметь в виду, что Java различает регистры букв. Имена классов принято писать с заглавной буквы (хотя, если вы напишите с маленькой, то ошибкой это не будет, но будет нарушением общепринятого стиля), имена методов, полей, переменных и ссылок на объекты – с маленькой.

Класс, который будет определять запуск приложения (его имя, фактически, и является именем запускаемой программы), должен обязательно содержать метод main(), который всегда описывается так, как показано в строке 4 примера, то есть имеет спецификатор доступа public, является статическим – static, не возвращает никаких значений – void, и аргументом метода является массив строк. Выполнение программы (если она не является аплетом) всегда начинается с вызова метода main() того класса, имя которого при запуске стоит сразу после имени интерпретатора (java).

*Задание*

Написать программу на языке Java запрашивающую Ваше имя и приветствующую Вас.

*Вопросы для самопроверки:*

1. Назовите целочисленные типы данных.
2. Назовите типы данных описывающие действительные числа.
3. Что запускается Java машиной на исполнение?
4. Каково должно быть имя программы?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. - М. : Академия, 2012. - 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 6-8, 20-22.

## Лабораторная работа 2. Классы и объекты в Java

*Цель работы.* Знакомство со структурой классов в Java и решение вычислительных задач.

Язык Java является «более» объектно-ориентированным по сравнению с C++. В Java все является объектом. Подчеркнем основные черты объектно-ориентированного программирования:

1. Все является объектом. Объект – своего рода переменная, которой можно посылать сообщения с запросами совершить какие-то операции над собой.
2. Программа – это группа объектов, говорящих друг другу, что делать посредством сообщений. Сообщение можно представить как вызов функции (метода), принадлежащей определенному объекту.
3. Каждый объект имеет собственную память, состоящую из других объектов. Новый объект создается с помощью «укладывания» в него новых объектов.
4. У каждого объекта есть тип. Каждый объект является экземпляром класса, здесь «класс» является аналогом слова «тип».
5. Все объекты определенного типа могут получать одинаковые сообщения. Т.е. это значит, что можно писать код для геометрических фигур, и быть уверенным, что он подойдет для всего, что можно подогнать под понятие геометрической фигуры. Например, задавшись целью определить модель поведения геометрической фигуры, в качестве действий, которые каждая фигура должна «уметь» выполнять, можно определить следующие: фигура должна уметь нарисовать себя, стереть, двигаться на экране, менять свой цвет. И тогда к любому объекту, имеющему тип «фигура», будь то маленький черный квадрат или большой зеленый треугольник, можно обратиться с запросом (сообщением) изменить свой цвет или стереть себя. Конечно, данная модель весьма условна, и служит лишь для демонстрации одного из принципов ООП.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Классы и объекты
2. Хранение данных
3. Примитивные типы
4. Области видимости
5. Создание классов. Поля и методы классов
6. Компиляция и выполнение

7. Комментарии и встроенная документация
8. Стиль определения имен

*Задание*

Написать программу, в которой определить класс и объекты данного класса согласно индивидуального задания представленного на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Чем отличается класс от объекта?
2. Что такое конструктор?
3. Сколько конструкторов может содержаться в классе?
4. Сколько классов может быть в программе?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. - М. : Академия, 2012. - 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 13-20.

## Лабораторная работа 3. Операторы и управляющие конструкции в Java

*Цель работы.* Изучить операторы языка Java и конструкций, обеспечивающих реализацию известных типов алгоритмов

Основные операторы и управляющие конструкции в Java имеют тот же вид, что и операторы и конструкции языков C и C++.

Основные теоретические вопросы изложены в [2].

Вопросы подлежащие рассмотрению:

1. Логические операторы
2. Поразрядные операторы
3. Операторы сдвига
4. Тройной оператор «если-то»
5. Операторы приведение
6. Конструкция if-else
7. Конструкция while и do-while
8. Цикл for
9. Ключевые слова break и continue

10.Конструкция выбора switch

*Задание*

Создать три отдельных приложения, реализующих алгоритм ветвления с помощью конструкций представленных в языке и циклического алгоритма. Индивидуальные задания находятся на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Сколько конструкций принятия решений в языке Java?
2. Сколько конструкций обеспечивающих циклические алгоритмы в языкеJava?
3. Могут ли быть опущены поля в цикле for? Если да, то какие?
4. Что такое приведение типа?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 28-37.

## Лабораторная работа 4. Объекты. Решение задач в объектах в Java

*Цель работы.* Изучение вопросов построения решения задачи в объектах классов.

Для инициализации объектов в Java, как и в C++, используется специальный метод, называемый *конструктором*. Имя конструктора должно в точности совпадать с именем класса вплоть до регистров букв. Подобно любому методу, у конструктора могут быть параметры, которые позволяют передавать аргументы для инициализации объекта. Конструктор, в отличие от обычных методов, не возвращает никакого значения, и этим он отличается и от методов, возвращающих значение типа void.

Так как имя конструктора предопределено именем класса, то для того, чтобы была возможность создавать объекты разной конфигурации (то есть, возможно, будет использоваться разное количество параметров конструктора с различными типами) используется механизм перегрузки методов.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Перегрузка методов
2. Создание объектов

3. Удаление объектов
4. Инициализация членов класса
5. Работа со строками
6. Форматный вывод числовых значений

*Задание*

Создать приложение, реализующих решение задачи в объектах. Индивидуальные задания находятся на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Назовите виды конструкторов.
2. Что такое перегрузка методов?
3. Каким образом удаляются объекты из памяти?
4. Можно ли принудительно освободить память от объекта?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 38-51.

## Лабораторная работа 5. Композиция и наследование классов в Java.

*Цель работы.* Изучение вопросов наследования в языке Java.

Одним из основных преимуществ объектно-ориентированного подхода в программировании, и, в частности, в Java, является так называемое многократное использование классов. Если кто-то из программистов постарался создать и отладить код для классов, то вы можете воспользоваться функциональностью этих классов без изменения их кода.

Существует два основных механизма, позволяющих воспользоваться уже существующим кодом в своих классах – композиция и наследование, и данная глава посвящена рассмотрению этих механизмов

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Композиция
2. Наследование классов
3. Инициализация классов при наследовании

4. Восходящее преобразование
5. Композиция или наследование?
6. Ключевое слово final
7. Полиморфизм
8. Абстрактные классы и методы
9. Конструкторы и полиморфизм
10. Наследование и метод finalize()

*Задание*

Создать приложение, реализующих множественное наследование согласно приведенной в индивидуальном задании схемой. Индивидуальные задания находятся на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Чем отличается наследование от композиции?
2. Что такое восходящие преобразование?
3. Что такое полиморфизм?
4. Зачем необходим метод finalize()?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 59-72.

## Лабораторная работа 6. Интерфейсы

*Цель работы.* Получение навыков решения задач в объектах.

В Java интерфейсом называется именованная коллекция объявлений (деклараций) методов без их реализаций, в которой также могут содержаться декларации констант. Интерфейс очень сильно напоминает «чистый» абстрактный класс, но при этом классом не является. Основные отличия интерфейса от абстрактного класса:

– в интерфейсе ни один из методов не может иметь реализации, в отличие от абстрактного класса;
– класс может реализовывать несколько интерфейсов, но наследовать может только от одного класса;

– интерфейс не является частью иерархии классов: один интерфейс могут реализовывать не связанные классовой иерархией классы.

Для создания интерфейса вместо ключевого слова *class* в описании используется ключевое слово interface, перед которым, как и в определении класса, может стоять идентификатор доступа *public* либо ничего не стоять (дружественный доступ), а после которого – имя интерфейса с последующим телом интерфейса.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Понятие интерфейса. Создание интерфейса.
2. Восходящее и нисходящее преобразования.
3. Понятие внутренних классов. Статистические внутренние классы.

*Задание*

Написать программу, в которой создать интерфейс, согласно индивидуального задания, представленного на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Сформулируйте алгоритм создания интерфейса. Что может находиться в теле интерфейса?
2. Какими свойствами обладают поля, помещаемые в интерфейсе?
3. К каким последствиям для классов может привести «расширение» интерфейсов?
4. По каким причинам нисходящее преобразование является не безопасным?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 73-81.

## Лабораторная работа 7. Обработка ошибок и исключения

*Цель работы.* Знакомство с понятием исключения и получение навыков их возбуждения, знакомство с ситуацией повторного возбуждения исключения.

Исключительная ситуация возникает, когда невозможно нормальное продолжение работы метода или части программы, выполняющихся в данный момент. В случае исключительной ситуации нормальный ход выполнения программы становится невозможным из-за нехватки

информации для разрешения ситуации в текущем контексте (в отличие от нормальной ошибки). В этом случае нужно покинуть текущий контекст и передать задачу на более высокий уровень, что и происходит при возбуждении исключения.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Понятие исключительной ситуации.
2. Алгоритм возбуждения исключения.
3. Конструктор стандартных исключений.
4. Создание собственного класса, представляющего исключение, отличное от стандартных представленных в Java исключений.

*Задание*

Написать программу, в которой определить собственный класс, представляющий исключение, отличное от стандартных в Java, согласно индивидуального задания представленного на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. В каких случаях можно говорить о возникновении исключительной ситуации?
2. Какие действия вызывает ключевое слово throw ?
3. В каких ситуациях используется блок повторных попыток try{} ?
4. После какого блока в языке Java следуют обработчики исключений?
5. В чем отличие System.err от System.out ?
6. Каким образом можно перехватить любое исключение?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 82-85

## Лабораторная работа 8. Ввод и вывод данных

*Цель работы.* Знакомство с понятием класса *File*, получение навыков реализации ввода и вывода данных.

Реализация ввода и вывода данных в Java полностью опирается на понятие *потоков данных*. Поток данных – это абстракция, за которой скрывается источник или приемник данных (файл, память, сетевое

соединение), который читает или записывает информацию последовательно. Общий алгоритм чтения и записи данных является стандартным:

открыть поток на чтение (запись)
пока есть данные
    прочитать (записать) очередную порцию данных
закрыть поток

Классы стандартной IO (input/output) библиотеки Java разделяются на две части: осуществляющие ввод, осуществляющие вывод. Кроме того, эти классы делятся на два семейства – одни работают с байтами (byte), другие – с символами (char). Классов этих довольно много, и на первых порах можно легко запутаться в этом многообразии, поэтому работая с этим разделом Java лучше всегда иметь под рукой справочную информацию.

Классы *Reader* и *Writer* (библиотека java.io) являются абстрактными классами, от которых наследуются все классы, работающие с символьными 16-битными (char) потоками. По названию классов нетрудно догадаться, что *Reader* предоставляет часть функциональности и является суперклассом для классов, реализующих чтение данных, *Writer* – запись данных.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Понятие системных свойств среды. Методы класса *System*.
2. Понятие потоков данных. Классы стандартной IO (*input/output*) библиотеки Java.
3. Работа с текстовыми файлами в Java. Работа с нетекстовыми файлами. Классы *FileInputStream* и *FileOutputStream*.
4. Совместное использование байтовых и символьных потоков данных.
5. Работа со стандартными потоками ввода/вывода.
6. Сериализация объектов.
7. Чтение/запись файлов с произвольным доступом.

*Задание*

Написать программу, с использованием чтения/записи файлов с произвольным доступом (класс *RandomAccessFile*) согласно индивидуального задания представленного на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. Каким образом описываются системные свойства среды?
2. Для чего используют классы-фильтры?
3. В каких случаях можно сериализировать объект?
4. Какие классы из стандартной библиотеки ввода/вывода Java позволяют архивировать данные?

5. Какой класс предоставляет в Java функциональность произвольного доступа к содержимому файла?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 81-94.

**Лабораторная работа 9.** Создание графического пользовательского интерфейса

*Цель работы*. Знакомство с понятием библиотеки графического пользовательского интерфейса, получение навыков создания графического пользовательского интерфейса.

Аплеты – это небольшие программы, работающие внутри Web-браузеров. Как правило, аплеты загружаются при загрузке Web-станицы, и из соображений безопасности возможности аплетов значительно ограничены по сравнению с обыкновенными приложениями.

Для создания аплетов, представляемых библиотекой *AWT* необходимо создать класс, наследуемый от класса java.applet.Applet, в котором переопределить некоторые из методов, унаследованных от Applet. В частности, основными методами аплета, отвечающими за его «жизненный цикл» являются:

*init()* – метод автоматически вызывается виртуальной машиной браузера при первом запуске аплета и является некоторым аналогом конструктора обыкновенного (standalone) приложения.

*start()* – автоматически вызывается браузером для информирования аплета о том, что тот должен начать свое выполнение. Вызывается сразу после вызова метода init() и каждый раз при попадании аплета в видимую часть браузера.

*stop()* – автоматически вызывается каждый раз, когда аплет покидает видимую часть браузера и непосредственно перед вызовом метода destroy() с целью сообщить аплету, что он должен приостановить свое выполнение.

*destroy()* – автоматически вызывается при закрытии web-страницы, содержащей аплет, для информировании аплета о том, что он должен освободить ресурсы.

Имена классов библиотеки *Swing*, представляющих визуальные компоненты, похожи на имена аналогичных компонент библиотеки *AWT*, но начинаются с буквы J – JButton, JLabel, JTextField и т.д. Для того чтобы создать визуальное приложение на Java, необходимо создать визуальное полотно, на котором будут размещаться визуальные компоненты. Для этого должен использоваться один из визуальных контейнеров верхнего уровня (*top-level container*). В библиотеке *Swing* три класса представляют контейнеры верхнего уровня – *JFrame, JDialog* и *JApplet*. Экземпляр класса *JFrame* реализует простое главное окно; экземпляр класса *JDialog* реализует дочернее окно, то есть окно, которое зависит от другого (родительского) окна; экземпляр класса *JApplet* реализует отображаемую площадь аплета в окне браузера. При использовании контейнеров верхнего уровня библиотеки *Swing* визуальные компоненты, за исключением меню, добавляются не непосредственно в основной контейнер, а на панель отображения (*content*

*pane*). Панель отображения представляет собой экземпляр класса *Container*, получаемый с помощью метода *getContentPane()*.

Основные теоретические вопросы изложены в [2].

Вопросы, подлежащие рассмотрению:

1. Графические библиотеки *AWT* и *Swing*. Понятие аплетов, *AWT*-аплеты.
2. Работа с библиотекой *Swing*.
3. Обработка событий в Java.
4. Менеджеры расположения (компоновки) компонентов.
5. Создание кнопок и переключателей.
6. Выбор платформенного представления визуальных компонент.
7. Текстовые компоненты. Панели прокрутки. Всплывающие подсказки. Выпадающие списки.
8. Составные окна (вкладки). Создание границ. Создание меню. Запуск аплета как обычного приложения.
9. Использование JAR-файлов.
10. Рисование.

*Задание*

Написать программу, создающую графический пользовательский интерфейс согласно индивидуального задания представленного на сервере экономического факультета (\\server\student\матолыгин\КИС\).

*Вопросы для самопроверки:*

1. В чем заключается политика безопасности Java?
2. Каков порядок создания визуального приложения на Java?
3. Что такое концепция так называемых слушателей событий.
4. Какой класс позволяет создавать иконки из данных, представляющих изображение либо в формате GIF, либо в формате JPEG?
5. Что необходимо сделать для того, чтобы установить внешний вид графического приложения в соответствии с платформой, на которой будет выполняться программа?
6. Какой метод необходимо вызвать для создания всплывающей подсказки?
7. Какой класс отвечает за функциональность составных окон?
8. Какова последовательность действий при создании меню?
9. Какие классы из библиотеки Swing используют для представления стандартных диалоговых окон?

*Литература:*

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.

2. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 95-143.

## ЛИТЕРАТУРА

1. Головин И. Г. Языки и методы программирования: учебник для вузов / И. Г. Головин, И. А. Волкова. – М. : Академия, 2012. – 304 с.
2. Калянов Г.Н.CASE-технологии: Консалтинг в автоматизации бизнес-процессов : монография / - М. : Горячая линия-Телеком, 2000. - 318[2] с. : ил.
3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем : Учебник / - М. : Финансы и статистика, 2000, 2002. - 349 с.
4. Юдахин Р. В. Основы программирования на JAVA : Учебное пособие. - Томск : ТУСУР, 2004. – С. 95-143.
5. Эккель Брюс. Философия Java. Библиотека программиста.-СПб: Питер, 2001.-880 с.
6. Хорстманн Кей, Корнелл Гари. Библиотека профессионала. Java 2. Том 2. Тонкости программирования.: Пер. с англ.-М.: Издательский дом «Вильямс», 2002.- 1120 с.