

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ECLIPSE. РАЗРАБОТКА ПРОГРАММ НА JAVA	5
1.1 Методические рекомендации.....	5
1.2 Контрольный пример выполнения лабораторной работы	8
1.3. Самостоятельное задание	15
1.4. Содержание отчета.....	16
1.5. Контрольные вопросы	16
1.6. Варианты заданий для самостоятельной работы	16
ЛАБОРАТОРНАЯ РАБОТА №2. ТИПЫ ДАННЫХ И УПРАВЛЯЮЩИЕ СТРУКТУРЫ JAVA	25
2.1 Методические рекомендации.....	25
2.2 Контрольный пример выполнения лабораторной работы	39
2.3 Содержание отчета.....	42
2.4. Контрольные вопросы	42
2.5 Варианты заданий для самостоятельной работы	42
ЛАБОРАТОРНАЯ РАБОТА №3. КЛАССЫ И ОБЪЕКТЫ В JAVA	50
3.1. Методические рекомендации.....	50
3.3 Содержание отчета.....	59
3.4 Контрольные вопросы	59
3.5. Варианты заданий для самостоятельной работы	59
ЛАБОРАТОРНАЯ РАБОТА №4. НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ	67
4.1. Методические рекомендации.....	67
4.2 Контрольный пример выполнения лабораторной работы	76
4.3 Содержание отчета.....	77
4.4 Контрольные вопросы	77
4.5 Варианты заданий для самостоятельной работы	77
ЛАБОРАТОРНАЯ РАБОТА №5. КОЛЛЕКЦИИ.....	85

Цель работы. Получить практические навыки работы с коллекциями в Java. ..	85
5.1 Методические рекомендации.....	85
5.2. Содержание отчета.....	93
5.3. Варианты заданий для самостоятельной работы	93
ЛАБОРАТОРНАЯ РАБОТА №6. ВВОД-ВЫВОД. ПАКЕТ JAVA.IO.....	95
6.1. Методические рекомендации.....	95
6.2 Контрольный пример	103
6.2. Содержание отчета.....	107
6.3 Варианты заданий для самостоятельной работы	108
ЛАБОРАТОРНАЯ РАБОТА №7. РАБОТА С ФАЙЛАМИ	110
7.1. Методические рекомендации.....	110
7.2. Контрольный пример	122
7.3. Содержание отчета.....	125
7.4. Варианты заданий для самостоятельной работы	125
ЛАБОРАТОРНАЯ РАБОТА №8. ПАКЕТ JAVA.AWT.....	127
8.1 Методические рекомендации.....	127
8.2. Варианты заданий	138
8.3. Содержание отчета.....	138
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	139

ЛАБОРАТОРНАЯ РАБОТА №1. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ECLIPSE. РАЗРАБОТКА ПРОГРАММ НА JAVA

Цель работы: получить практические навыки работы с интегрированной средой разработки Eclipse; научиться писать простейшую программу на языке Java; иметь представление о методе `main()`.

1.1 Методические рекомендации

1.1.1 Общие сведения о языке Java

Java – полностью объектно-ориентированный язык программирования. В Java отсутствует понятие процедур. С помощью Java решаются различные задачи и тот же самый круг проблем, что и на других языках программирования. Java может использоваться для создания двух типов программ: приложений и апплетов. *Приложение* – программа, которая выполняется на компьютере, под его операционной системой. Приложения Java могут быть непосредственно выполнены с использованием интерпретатора Java. *Апплет* – небольшая программа, работающая с окнами, которые внедрены в страницу HTML. Чтобы выполнить Java-апплеты, нужна поддержка Java web-браузером, то есть Internet Explorer, Firefox, Chrome и т. д., или средство просмотра апплета.

Java – это интерпретируемый и компилированный язык программирования. Исходный текст (файлы с расширением `a.java`) компилируется компилятором Java (`javac`), который преобразовывает исходный текст в байт-код (файлы с расширением `a.class`). Цель проектировщиков Java состояла в том, чтобы разработать язык, посредством которого программист может писать код, который будет выполняться всегда, в любое время.

Простота

Проектировщики Java пытались разработать язык, который могли бы быстро изучить программисты. Также они хотели, чтобы язык был знаком большинству программистов для простоты перехода. С этой целью в Java проектировщиками было удалено множество сложных особенностей, которые существовали в C и C++. Особенности, такие как манипуляции указателя, перегрузка оператора и т. д., в Java не существуют.

Java не использует *goto*-инструкцию, а также не использует файлы заголовка. Конструкции подобно *struct* и *union* были удалены из Java.

Объектная ориентированность

В Java всё может быть объектом. Так, основное внимание здесь уделяется свойствам и методам, которые оперируют данными в нашем приложении, и нет концентрации только на процедурах. Свойства и методы вместе описывают состояние и поведение объекта. В Java мы будем наталкиваться на термин «метод» очень часто, с ним мы будем должны познакомиться. Термин «метод» используется для функций.

Распределенность

Java может использоваться для разработки приложений, которые работают на различных платформах, операционных системах и графических интерфейсах пользователя. Java предназначен также для поддержки сетевых приложений. Таким образом,

Java широко используется как инструмент разработки в среде, подобной Internet, где существуют различные платформы.

Устойчивость

Java – язык со строгим контролем типов, так что требуется явное объявление метода. Java проверяет код во время трансляции и во время интерпретации. Таким образом, устраняются некоторые типы ошибок при программировании. Java не имеет указателей и соответственно арифметических операций над ними. Все данные массивов и строк проверяются во время выполнения, что исключает возможность выхода за границы дозволенного. Преобразование объектов с одного типа на другой также проверяется во время выполнения.

Автоматическая обработка исключений: в традиционных средах программирования программист должен был вручную распределять память, и в конце программы имел явное количество свободной памяти. Возникали проблемы, когда программист забывал освобождать память. В Java программист может не беспокоиться о проблеме, связанной с освобождением памяти. Это делается автоматически, поскольку Java обеспечивает обработку исключений для объектов, которые не используются.

Обработка исключений упрощает задачу обработки ошибок и восстановления.

Безопасность

Вирусы – большая причина беспокойства в мире компьютеров. До появления Java программисты должны были сначала просмотреть файл перед загрузкой и выполнением. Но даже после этого они не были уверены в надежности файла. Также существует много специальных программ, о которых мы должны знать. Эти программы могут находить уязвимые данные нашей системы.

Java обеспечивает управляемую среду, в которой выполнена программа. Java никогда не предполагает, что код может быть безопасно выполнен. И так как Java – больше чем язык программирования, он обеспечивает несколько уровней контроля защиты. Со справкой этих уровней он гарантирует безопасную среду выполнения.

Первый уровень – это безопасность, обеспеченная языком Java. Свойства и методы описываются в классе, и к ним можно обратиться только через интерфейс, обеспеченный классом. Java не позволяет никаких операций с указателями и таким образом запрещает прямой доступ к памяти. Избегается переполнение массивов. Проблемы, связанные с безопасностью и мобильностью, скрыты.

На следующем уровне компилятор, прежде чем приступить к компиляции кода, проверяет безопасность кода и затем следует в соответствии с протоколами, установленными Java.

Третий уровень – это безопасность, обеспеченная интерпретатором. Прежде чем байт-код будет фактически выполнен, он является полностью укрытым верификатором.

Четвертый уровень заботится о загрузке классов. Загрузчик класса гарантирует, что класс не нарушает ограничения доступа, прежде чем он загружен в систему.

Независимость от структуры системы

Код Java может быть выполнен на разных платформах. Это достигается при смещении трансляции и интерпретации.

1 Программы Java оттранслированы в байт-код компилятором (*байт-код* – это универсальный машинный код).

2 Байт-код выполняется интерпретатором (Виртуальная Машина Java). Интерпретатор должен выполнять байт-код для каждой аппаратной платформы. Байт-код выполняется на любой версии Виртуальной Машины Java.

Мобильность

Независимость от платформы означает легкость переноса программы с одного компьютера на другой без каких-либо трудностей. Также Java – платформа независимая на обоих уровнях, то есть на первичном (исходном) и на вторичном уровне.

Java – это язык со строгим контролем типов, что означает, что необходимо объявлять тип для каждой переменной. И эти типы данных в Java одинаковы для всех платформ. Java имеет свои собственные библиотеки фундаментальных классов, которые облегчают запись кода для программиста, который может быть перемещен с одной машины на другую, без потребности перезаписи кода. Независимость от платформы на исходном уровне означает, что можно переместить исходный текст из одной системы в другую, компилируя код и работая в системе.

Платформа независимая на вторичном уровне означает, что откомпилированный двоичный файл может быть выполнен на различных платформах, не перетранслируя код, если они имеют Виртуальную Машину Java, которая функционирует как интерпретатор.

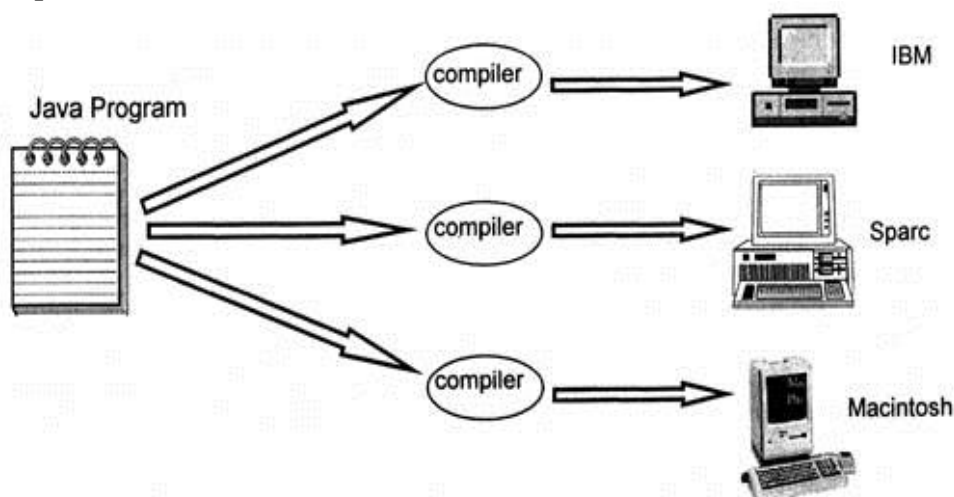


Рис. 1. Компилирование программ традиционным способом

Для программ, которые написаны на С и С++ или на любом другом языке, компилятор преобразует набор команды в машинный код или команды процессора. Эти команды являются специальными для процессора. В результате, если необходимо использовать этот код в некоторой другой системе, нужно найти компилятор для этой системы и откомпилировать код еще раз, так чтобы получился машинный код, определенный для этой машины. Рисунок 2 дает представление того, как программы Java могут быть выполнены на различных машинах.

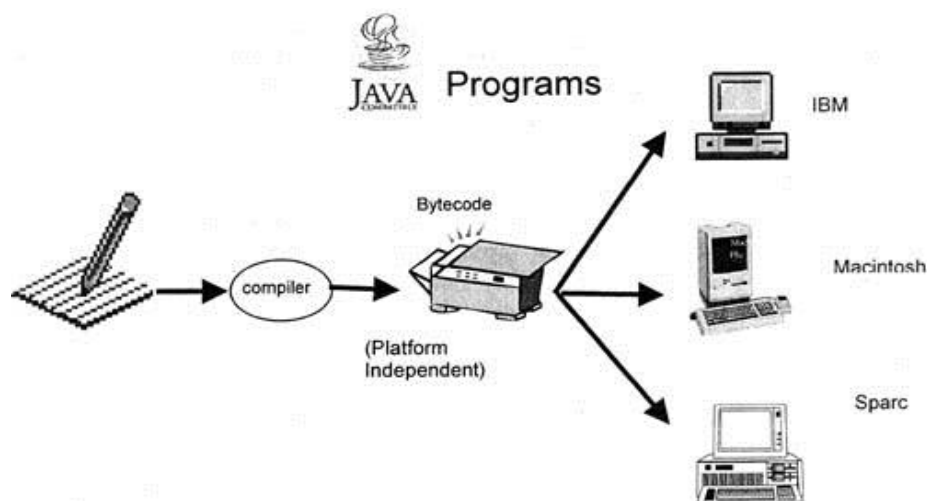


Рис. 2. Компилирование кода в Java

В отличие от C и C++, компилятор Java преобразует исходный текст в байт-коды, которые являются машинно-независимыми. Байт-коды – это только части команд Java, разрезанные на байты, которые могут быть декодированы любым процессором.

Интерпретатор Java, также именуемый как JVM (Виртуальная Машина Java), или Java Runtime Interpreter, выполняет байт-коды Java. Интерпретатор Java является частью среды разработки.

1.1.2 Средства разработки Java

JDK – средство разработки Java, используемое при разработке программ. Оно состоит из:

- классов;
- компилятора;
- отладчика;
- JRE (среды выполнения Java).

В JDK входят следующие *инструментальные средства*:

- `javac` – это команда, которая используется для компиляции исходного текста. Она конвертирует исходный файл (файл с расширением Java) в файл класса (файл с расширением `.class`);
- `java` – эта команда используется для выполнения файла класса, который исполняет классы в Виртуальной Машине Java;
- команда `appletviewer` позволяет выполнять апплеты без использования web-браузера.

1.2 Контрольный пример выполнения лабораторной работы

Для создания первой программы на языке Java запустите IDE Eclipse. В строке меню выберите *File*, пункт *New* и подпункт *Java project* (рис. 3).

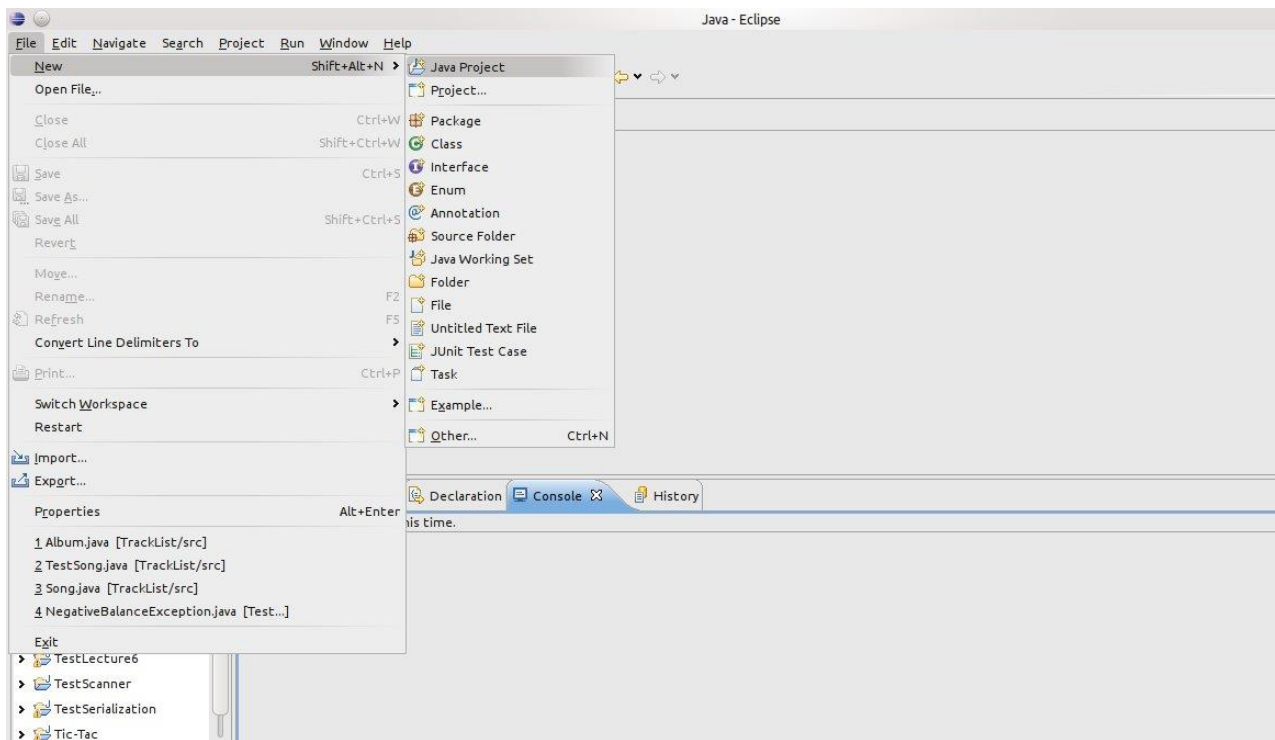


Рис. 3. Создание проекта в IDE Eclipse

В появившемся окне напишите имя создаваемого проекта, например *HelloWorld* (рис. 4)

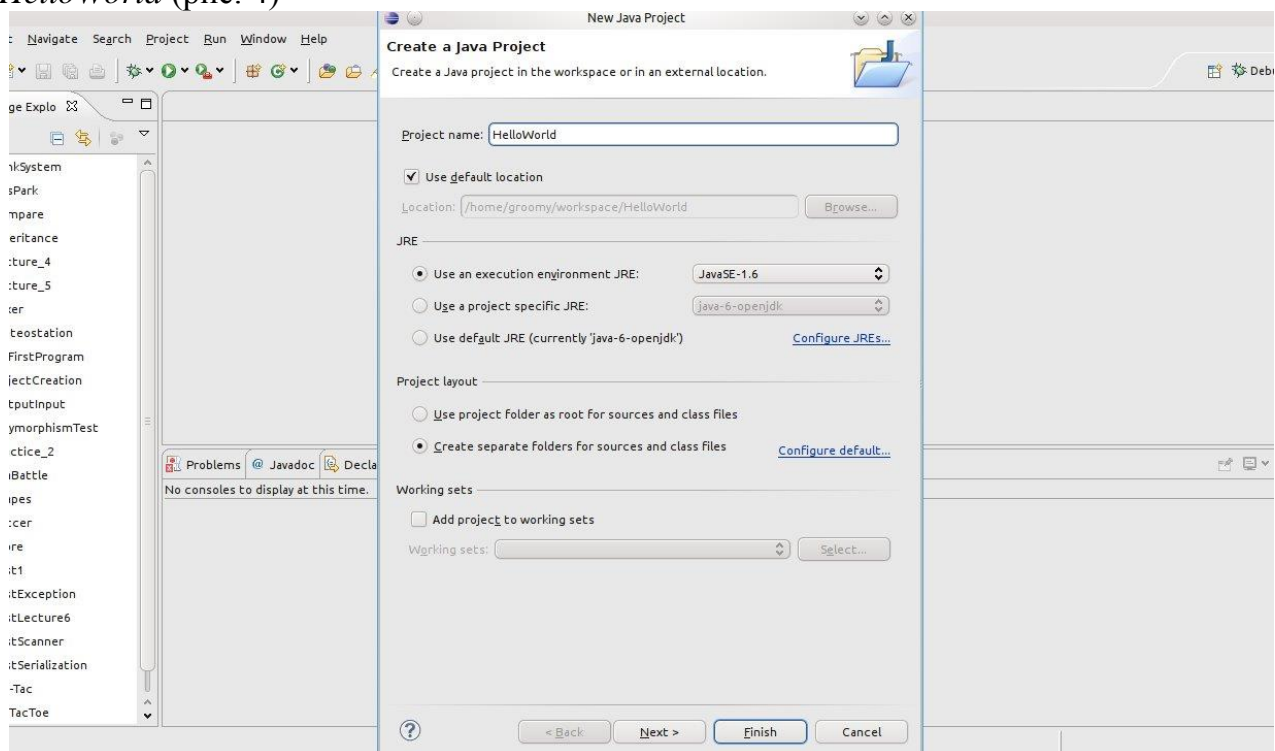


Рис. 4. Определение имени проекта в IDE Eclipse

В окне *Package Explorer* (находится слева) найдите созданный проект и нажатием правой кнопки мыши вызовите контекстное меню. В меню выберите пункты *New* → *Class* (рис. 5).

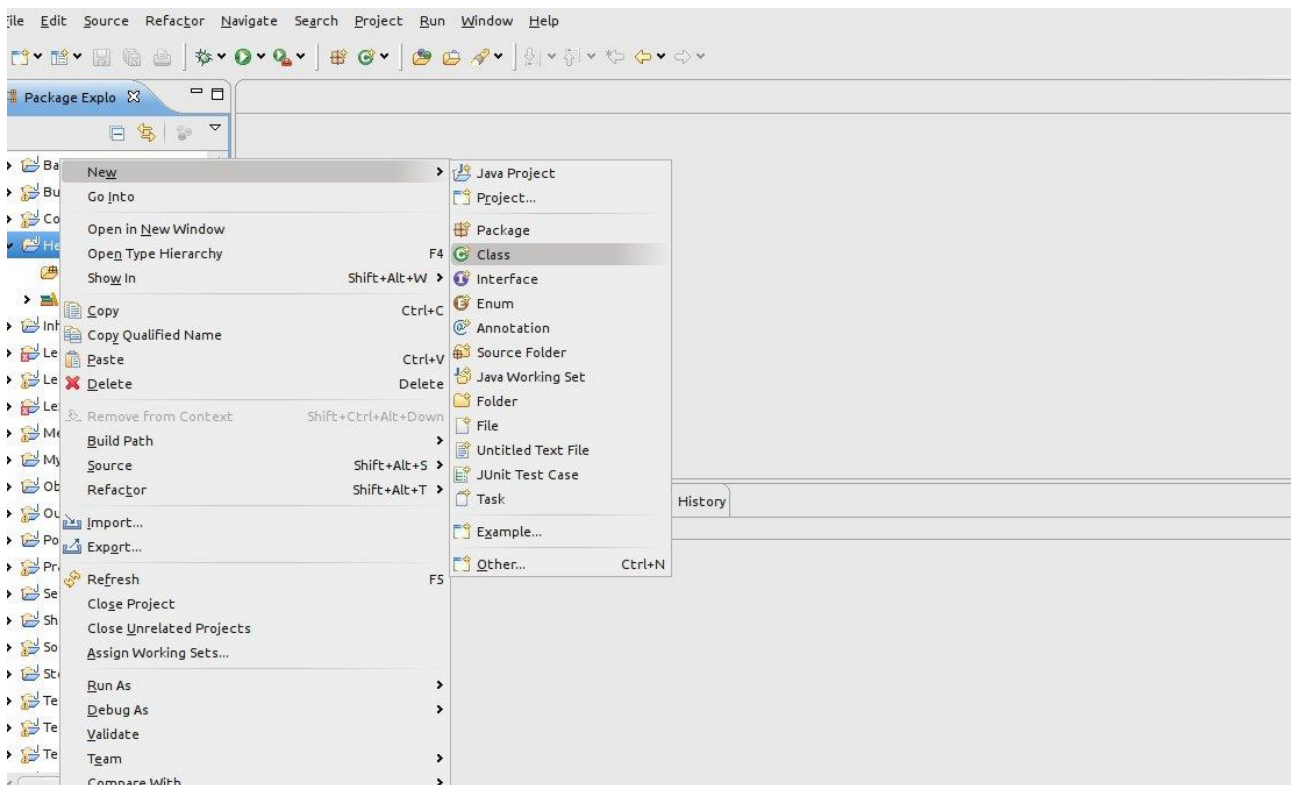


Рис. 5. Выбор меню создания нового класса в IDE Eclipse

Задайте имя класса *HelloWorld* в появившемся окне, уберите все галочки, если они выставлены, нажмите кнопку *Finish* для окончания создания класса (рис. 6).

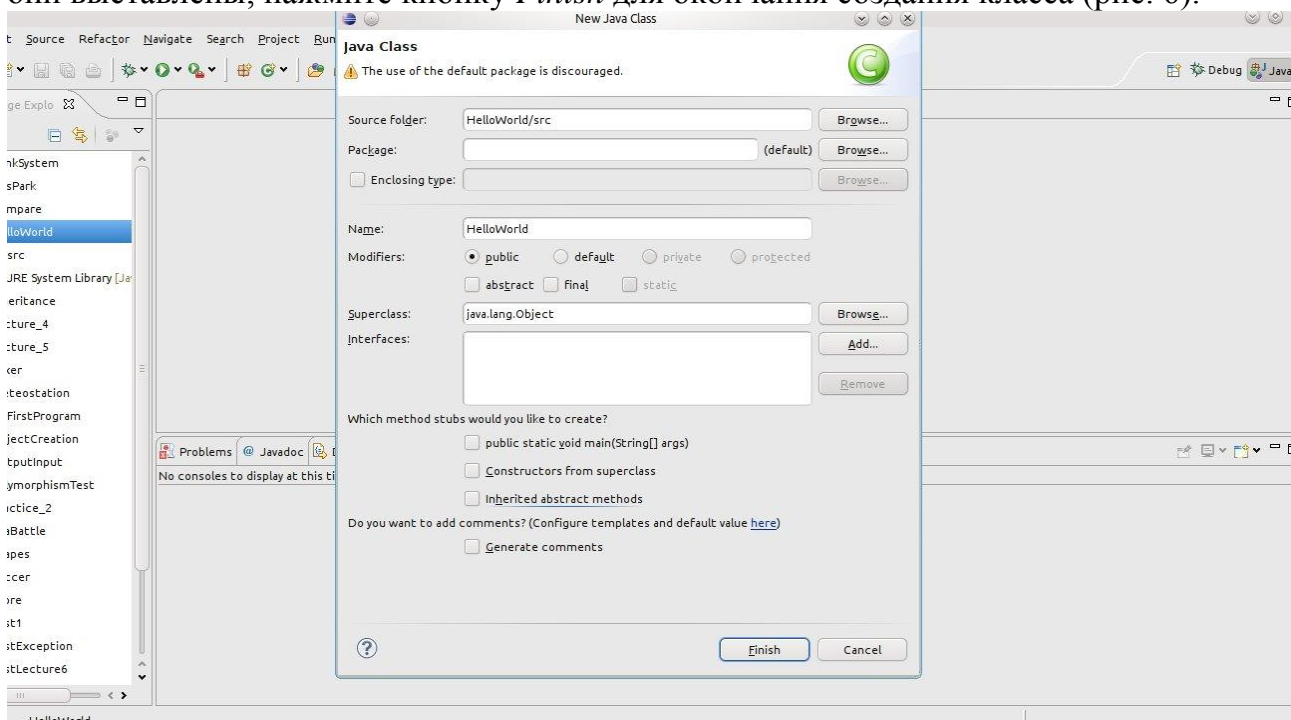


Рис. 6. Создание класса в IDE Eclipse

В появившемся окне редактирования наберите программу:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```


Для выполнения программы найдите файл в Package Explorer и, вызвав контекстное меню, выберите пункты Run as → *Java Application* (рис. 7).

Результат выполнения программы вы увидите в консоли (нижняя часть IDE Eclipse).

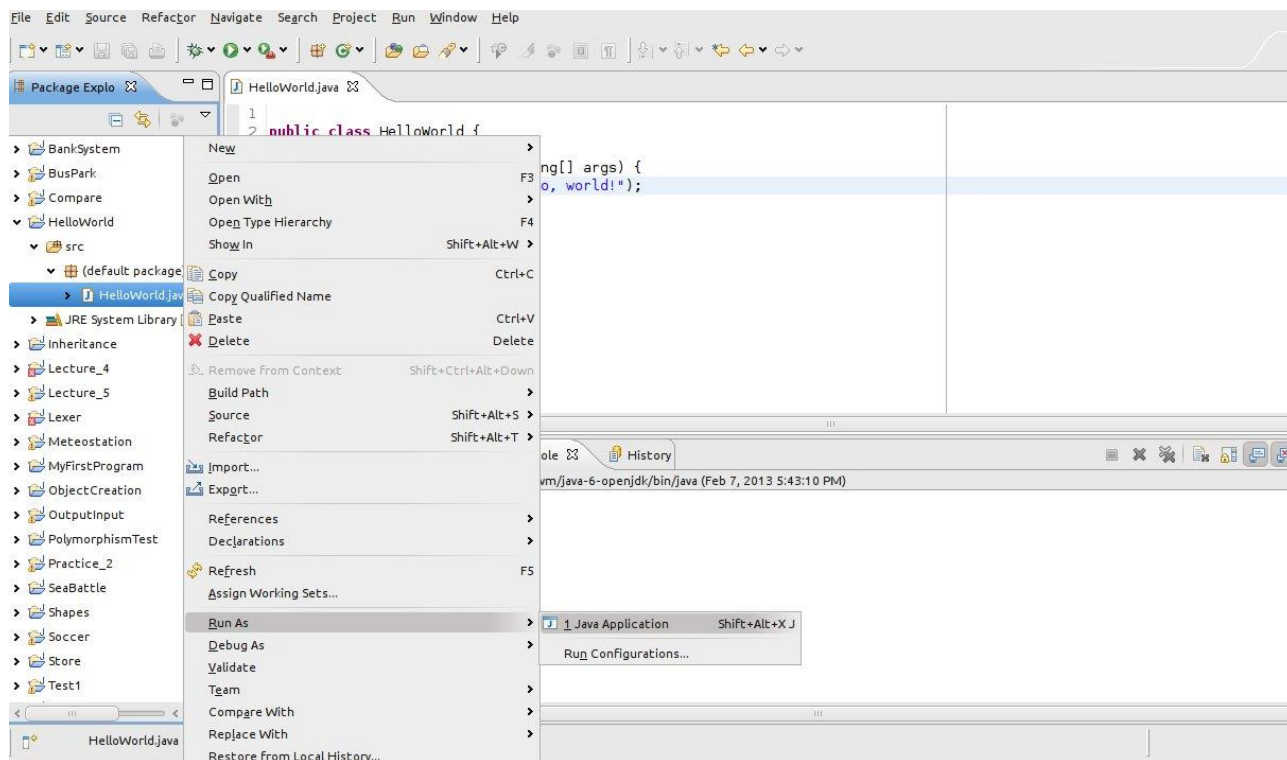


Рис. 7. Запуск приложения на языке Java в IDE Eclipse

Разберем программу более подробно.

class HelloWorld {

Эта строка объявляет класс по имени **HelloWorld**. При создании класса используется ключевое слово **class** вместе с именем класса/именем файла.

Обратите внимание: принято, чтобы имя класса начиналось с заглавной буквы.

Ключевое слово **class** используется для объявления нового класса. **HelloWorld** – идентификатор, отображающий название класса. Полное описание класса делается в пределах открытой и закрытой изогнутых фигурных скобок. Фигурные скобки указывают компилятору, где начинается и заканчивается описание класса. Открытие и закрытие изогнутой скобки формируют блок этого класса.

public static void main(String args[])

Ключевое слово **main()** – основной метод. Это – строка, с которой начинается выполнение программы. Все приложения Java должны иметь один метод **main()**. Давайте расшифруем каждое слово в коде.

Ключевое слово **public** – это спецификатор доступа. Спецификаторы доступа будут рассмотрены позже. Когда члену класса предшествует **public**, то к этому члену возможен доступ из кода, внешнего по отношению к классу, в котором описан данный метод. В данном случае *main*-метод объявлен как **public** так, чтобы JVM мог обратиться к этому методу.

Ключевое слово **static** позволяет методу *main()* вызываться без потребности создавать образец класса. К объекту класса нельзя обратиться, не создав это. Но в этом случае есть копия этого метода, доступного в памяти после того, как класс расположен, даже если не был создан образец этого класса. Это важно, потому что JVM вызывает этот метод в первую очередь. Следовательно, этот метод должен быть как **static** и не должен зависеть от экземпляров любого создаваемого класса.

Ключевое слово **void** говорит компилятору, что метод не возвращает никакого значения.

main () – метод, который исполняет специфическую задачу. Это место, с которого начинается выполнение всех приложений Java. Класс, который не имеет основного метода, может быть успешно откомпилирован, но не может быть выполнен, поскольку он не имеет отправной точки выполнения, которой является *main()*-метод.

String args [] – один из параметров, который передаётся основному методу. Любая информация, которую мы передаём методу, получена переменными, которые упомянуты в пределах круглой скобки метода. Эти переменные – параметры этого метода. Даже если мы не должны передавать никакой информации методу, название метода должно сопровождаться пустыми круглыми скобками, **args []** (переменная) – массив типа *String*. Параметры, которые передают в командной строке, сохранены в этом массиве.

Открытие и закрытие изогнутой скобки для **main** метода составляют блок метода. Функции, которые будут выполнены от основного метода, должны быть определены в этом блоке.

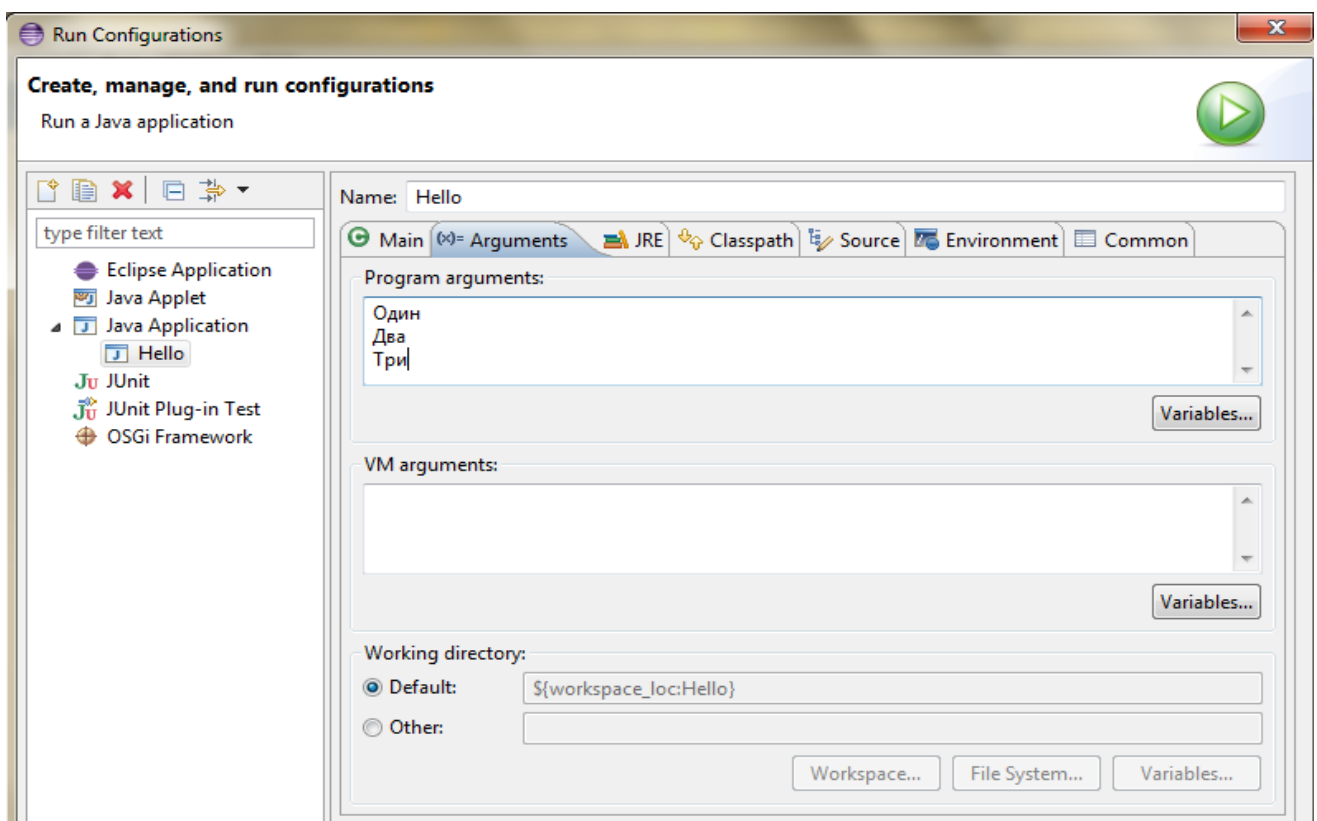
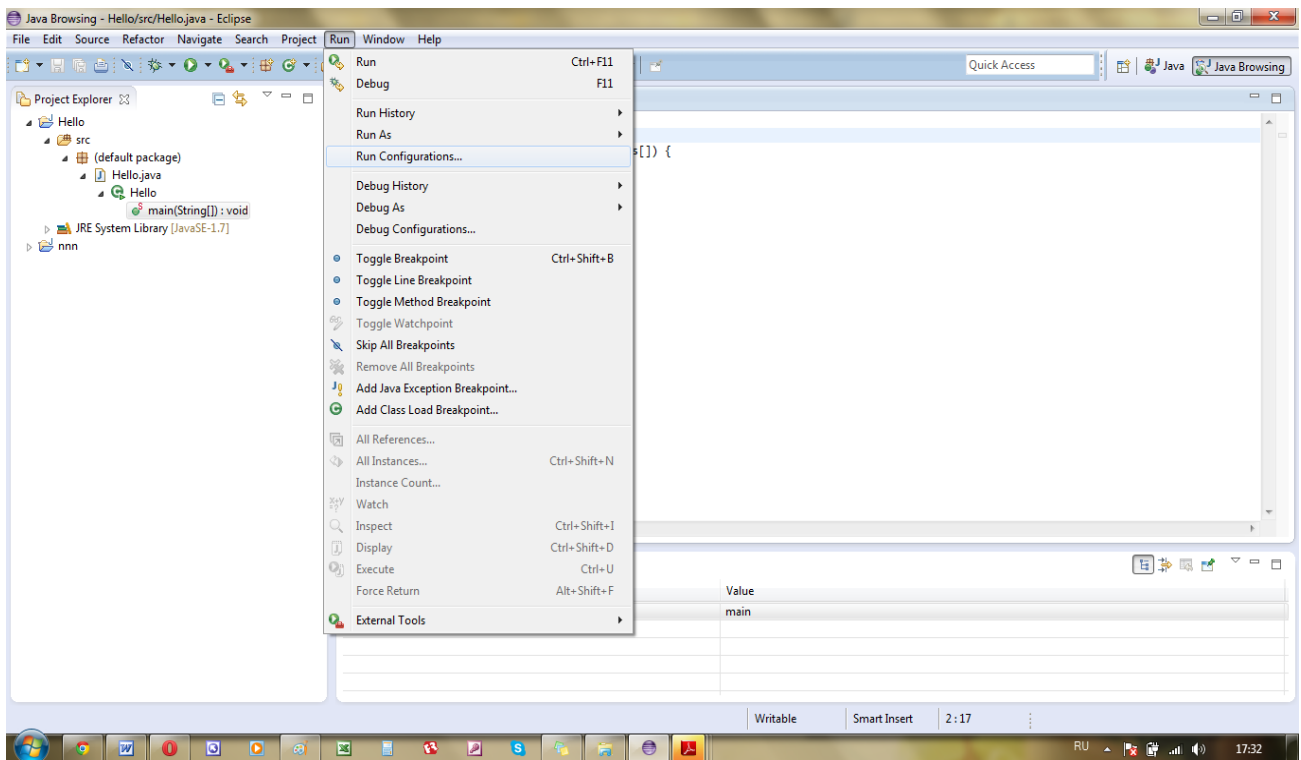
System.out.println(«Hello, World!») ;

Эта запись отображает строку на экране. Вывод строки осуществляется с помощью метода **println ()**. **println ()** отображает только строку, которая передаётся со справкой *System.out*.

System – класс, который является предопределённым и обеспечивает доступ к системе.

out – выходной поток и связан с консолью.

Чтобы задать параметры функции *main* можно воспользоваться диалоговым окном **Run/Run Configurations** и перейти на вкладку **Arguments**.



Затем, к параметрам можно обратиться по его порядковому номеру, например, так

```
System.out.println("Привет "+args[0]);
```

Все инструкции в Java заканчиваются точкой с запятой (;).

Арифметические операторы в Java

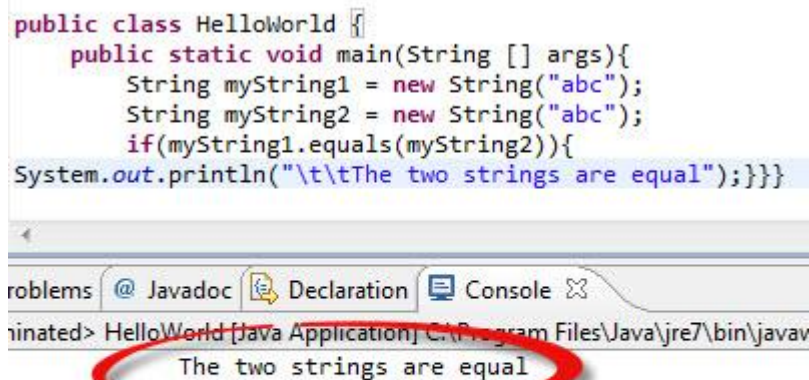
Для стандартных арифметических операций, таких как *сложение*, *вычитание*, *умножение*, *деление* в Java используются традиционные символы:

```
int a, b;  
int sum = a + b;  
int sub = a - b;  
int mult = a * b;  
int div = a / b;
```

Операторы == и != тестируют соответственно равенство или неравенство. Для примитивных типов концепция равенства или неравенства достаточно тривиальна. Для значений объектного типа, сравниваемая величина – это ссылка на объект, то есть адрес памяти и для сравнения значений объектных типов используется метод equals() :

"login".equals("login") – true, "login".equals("admin") – false

Например, сравнение двух строк в языке Java на равенство выполняется так:



```
public class HelloWorld {  
    public static void main(String [] args){  
        String myString1 = new String("abc");  
        String myString2 = new String("abc");  
        if(myString1.equals(myString2)){  
            System.out.println("\t\tThe two strings are equal");  
        }  
    }  
}
```

The screenshot shows a Java IDE with a tab for 'HelloWorld'. The console output is 'The two strings are equal', which is circled in red.

Java: Преобразование типов данных

Конвертация строки в числовой формат на языке Java производится так:

```
byte b = Byte.parseByte("123");  
short s = Short.parseShort("234");  
int i = Integer.parseInt("234");  
long l = Long.parseLong("234");  
float f = Float.parseFloat("234.4");  
double d = Double.parseDouble("233.4e10");
```

Математические функции и константы (класс Math)

Для решения задач нередко требуется использование математических функций. В Java такие функции включены в класс Math. Для того, чтобы использовать методы класса Math, нужно подключить его в начале .java файла с вашим кодом.

```
import static java.lang.Math.*;
```

Часто используемые математические функции:

- ***sqrt(a)*** — извлекает квадратный корень из числа *a*.
- ***pow(a, n)*** — *a* возводится в степень *n*.
- ***sin(a), cos(a), tan(a)*** — тригонометрические функции *sin*, *cos* и *tg* угла *a* указанного в радианах.
- ***asin(n), acos(n), atan(n)*** — обратные тригонометрические функции, возвращают угол в радианах.
- ***exp(a)*** — возвращает значение экспоненты, возведенной в степень *a*.
- ***log(a)*** — возвращает значение натурального логарифма числа *a*.
- ***log10(a)*** — возвращает значение десятичного логарифма числа *a*.
- ***abs(a)*** — возвращает модуль числа *a*.
- ***round(a)*** — округляет вещественное число до ближайшего целого.

Константы

- ***PI*** — число «пи», с точностью в 15 десятичных знаков.
- ***E*** — число «е»(основание экспоненциальной функции), с точностью в 15 десятичных знаков.

Примеры

```
System.out.println(sqrt(81)); // выведет 9.0
System.out.println(pow(2,10)); // выведет 1024.0
System.out.println(sin(PI/2)); // выведет 1.0
System.out.println(cos(PI)); // выведет -1.0
System.out.println(acos(-1)); // выведет 3.141592653589793
System.out.println(round(E)); // выведет 3
System.out.println(abs(-6.7)); // выведет 6.7
```

1.3. Самостоятельное задание

1 Реализовать программу, получающую на вход в качестве аргумента *имя человека* и выводящую “*Hello* ” + *имя*, в противном случае, если параметр не передавался, “*Hello world*”.

2 Написать программу, получающую на вход в качестве аргумента *несколько параметров*. В программе вывести “*Вы ввели*” + *N (количество параметров)* + “*параметров*”. Если параметры не передавались, вывести “*Вы не передавали параметров*”.

3 Передавать в качестве параметров *два целочисленных числа*. Вывести на экран как сами значения? так и их сумму (“*3 + 2 = 5*”). Если количество параметров не равно 2, вывести сообщение “*Неверное количество параметров*”.

4 Ввести в качестве параметров *имя пользователя и пароль*. Проверить в методе *main()* соответствие введенных значений заранее определенным значениям. В случае полного соответствия вывести сообщение “*Вас узнали. Добро пожаловать*”, в противном случае вывести сообщение “*Логин и пароль не распознаны. Доступ запрещен*”.

1.4. Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

- 1 теоретические сведения о языке Java;
- 2 реализацию программ на языке Java в соответствии с индивидуальным заданием по номеру варианта;
- 3 листинг выполнения программ, отражающий все этапы ее выполнения;
- 4 выводы о выполненной работе.

1.5. Контрольные вопросы

- 1) Объясните основное назначение метода main().
- 2) Расскажите о входном параметре метода main() String[] args.
- 3) В чем состоит разница между JDK (Java Development Kit) и JRE (Java Runtime Environment)?
- 4) Какое свойство массива возвращает его размер?
- 5) В чем отличие конструкций System.out.println() и System.out.print()?
- 6) Каким образом осуществляется конкатенация строк в Java?
- 7) Что означает ключевое слово void?
- 8) Что означает ключевое слово static?

1.6. Варианты заданий

Задание 1

1. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y. Большее из них умножить на 5. Вывести результат на экран.
2. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y. Меньшее из них разделить на 10. Вывести результат на экран.
3. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y. Если произведение этих чисел больше 100, то вычислить удвоенный куб первого числа. Вывести результат на экран.
4. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y. Если сумма этих чисел больше 20, то вычислить утроенный квадрат первого числа, в противном случае куб второго числа. Вывести результат на экран.
5. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y. Если произведение этих чисел больше 50, то вычислить удвоенный корень квадратный первого числа. Вывести результат на экран.
6. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b. Если сумма двух чисел больше 100, то вычислить удвоенный синус первого числа. Вывести результат на экран.

7. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Больше из них возвести в квадрат. Вывести результат на экран.
8. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 100, то вычислить утроенный тангенс второго числа, в противном случае первое число умножить на 5. Вывести результат на экран.
9. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 20, то вычислить котангенс второго числа, в противном случае первое число разделить на 3. Вывести результат на экран.
10. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Меньшее из них разделить на 2. Вывести результат на экран.
11. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 30, то вычислить удвоенный котангенс первого числа, в противном случае первое число разделить на 2. Вывести результат на экран.
12. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 40, то вычислить удвоенный тангенс первого числа, в противном случае второе число умножить на 4. Написать программу, получающую на вход в качестве аргумента два параметра -
13. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Из меньшего извлечь корень квадратный. Вывести результат на экран.
14. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 50, то вычислить удвоенный косинус первого числа, в противном случае второе число умножить на 3. Вывести результат на экран.
15. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Если произведение этих чисел больше 100, то вычислить удвоенный куб первого числа и второе число разделить на 2. Вывести результат на экран.
16. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Если сумма этих чисел больше 20, то вычислить утроенный квадрат первого числа и куб второго числа. Вывести результат на экран.
17. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Если произведение этих чисел больше 50, то вычислить удвоенный корень квадратный первого числа и квадрат второго числа. Вывести результат на экран.

18. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если сумма двух чисел больше 100, то вычислить удвоенный синус первого числа, а первое число умножить на 5. Вывести результат на экран.
19. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 100, то вычислить утроенный тангенс второго числа и синус второго числа, в противном случае первое число умножить на 5. Вывести результат на экран.
20. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 100, то большее возвести в квадрат, а меньшее умножить на 2. Вывести результат на экран.
21. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 20, то вычислить котангенс второго числа и тангенс первого, в противном случае первое число разделить на 3. Вывести результат на экран.
22. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Большее из них умножить на 2, меньшее разделить на 3. Вывести результат на экран.
23. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Большее из них возвести в квадрат, из меньшего извлечь корень квадратный. Вывести результат на экран.
24. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 30, то вычислить удвоенный котангенс первого числа и тангенс первого, в противном случае первое число разделить на 2. Вывести результат на экран.
25. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 40, то вычислить удвоенный тангенс первого числа и синус второго числа, в противном случае второе число умножить на 4. Вывести результат на экран.
26. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Большее из них умножить на 5, меньшее умножить на 3. Вывести результат на экран.
27. Написать программу, получающую на вход в качестве аргумента два параметра - числа x и y . Меньшее из них разделить на 10, а Большее из них возвести в квадрат. Вывести результат на экран.
28. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 50, то вычислить удвоенный косинус первого числа и квадрат второго, в противном случае второе число умножить на 3. Вывести результат на экран.

29. Написать программу, получающую на вход в качестве аргумента два параметра - действительные числа x и y , не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее – их удвоенным произведением. Вывести результат на экран.

30. Написать программу, получающую на вход в качестве аргумента два параметра - числа a и b . Если произведение двух чисел больше 20, то вычислить котангенс второго числа и тангенс первого, в противном случае первое число разделить на 3. Вывести результат на экран.

Задание №2

1. Дано a и b . Найти наибольшее **max(a,b)** и наименьшее **min(a,b)** из чисел a и b . И Найти d :

$$d = \frac{(max(a,b))^5 - 3.2 \cdot min(a,b)}{1 + min(a,b)}$$

2. Дано a, b . Найти наибольшее **max** и наименьшее **min**. Найти d :

$$d = \frac{5 \cdot max(a, b+2) - 4 \cdot min(1-a, b)}{3 + \frac{max(a, b+2)}{min(1-a, b)}}$$

3. Дано a, b . Найти наименьшее **min(a,b)** из чисел a и b . Найти $d = (min(a,b))^8$

4. Дано a, b . Найти наибольшее **max** и наименьшее **min**. Найти d :

$$d = \frac{max(2 \cdot a, b) - 10 \cdot \sqrt{min(a, 4+b)} + 4.2 min(a, 4+b)}{1 + \frac{max(2 \cdot a, b)}{min(a, 4+b)}}$$

5. Дано a, b . Найти наибольшее **max(a,b)** из чисел a и b . Найти $d := tg(max(a,b))$

6. Дано a, b . Найти наибольшее **max(a,b)** и наименьшее **min(a,b)** из чисел a и b . Найти d :

$$d = \frac{max(a,b) - 2^x min(a,b)}{Sin 2 + \frac{max(a,b)}{min(a,b)}}$$

7. Дано a, b . Найти наибольшее **max(a,b)** и наименьшее **min(a,b)** из чисел a и b . Найти d :

$$d = \frac{5 \cdot max(a,b) - 4 \cdot min(a,b)}{8.3 + \frac{max(a,b)}{min(a,b)}}$$

8. Дано a, b . Найти наибольшее $\max(a, b)$ и наименьшее $\min(a, b)$ из чисел a и b . Найти d :

$$d = \frac{\max(a, b) - 2 \cdot \sqrt{\min(a, b)} + 4.2 \min(a, b)}{1 + \frac{\max(a, b)}{\min(a, b)}}$$

9. Дано a, b . Найти наибольшее $\max(a, b)$ и наименьшее $\min(a, b)$ из чисел a и b . Найти d :

$$d = \frac{\min(a, b) - 2 \cdot \sqrt{\max(a, b)}}{1 + \frac{\max(a, b)}{\min(a, b)}}$$

10. Дано a, b . Найти наибольшее $\max(a, b)$ из чисел a и b . Найти $d := \cos(\max(a, b))$

11. Дано a, b . Найти наибольшее \max и наименьшее \min . Найти d :

$$d = \frac{\min(a, 3 \cdot b) - 2 \cdot \sqrt{\max(a, 3 \cdot b)}}{4^3 + \frac{\max(a, 3 \cdot b)}{\min(a, b)}}$$

12. Дано a, b . Найти наибольшее $\max(a, b)$ и наименьшее $\min(a, b)$ из чисел a и b . Найти d :

$$d = \frac{(\max(a, b))^5 - 3.2 \cdot \min(a, b)}{1 + \min(a, b)}$$

13. Дано a, b . Найти наибольшее $\max(a, b)$ и наименьшее $\min(a, b)$ из чисел a и b . Найти d :

$$d = \frac{\max(a, b) - 2^x \min(a, b)}{\sin 2 + \frac{\max(a, b)}{\min(a, b)}}$$

14. Дано a, b . Найти наибольшее $\max(a, b)$ из чисел a и b . Найти $d := \tan(\max(a, b))$

15. Дано a, b . Найти наибольшее \max и наименьшее \min . Найти d :

$$d = \frac{2 \cdot \max(a, b - 5) - 4 \cdot \min(1 - a, b)}{3 + \frac{\max(a, b - 5)}{\min(1 - a, b)}}$$

Задание №3

1. Наиболее рационально вычислить Y и F . Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ x(1 + 2^x) & \text{если } -3 < x \leq 4 \\ \tan x & \text{если } x > 4 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 4 \\ \sqrt[5]{\tan x} & \text{если } x > 4 \end{cases}$$

2. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 - \operatorname{tg}(2x - 1) & \text{если } x \leq -2 \\ 3x(1 + e^{x+1}) & \text{если } -2 < x \leq 1 \\ \sin^5 x & \text{если } x > 1 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -2 \\ x^2 & \text{если } -2 < x \leq 1 \\ \sqrt{\cos x} & \text{если } x > 1 \end{cases}$$

3. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ (1 + 2^{\operatorname{tg} x}) & \text{если } -3 < x \leq 0 \\ \operatorname{ctg}^2 x & \text{если } x > 0 \end{cases} \quad F = \begin{cases} e^{x+1} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 0 \\ \sqrt[3]{\operatorname{tg} x} & \text{если } x > 0 \end{cases}$$

4. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1 + 3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x + 1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[5]{x} & \text{если } x > 2 \end{cases}$$

5. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - x^5 + \operatorname{tg} 8x & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{arctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{2x+1} & \text{если } x \leq 0 \\ x^2 - \sin 4x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

6. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - 9^{x+1} & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(2x+1)} & \text{если } x \leq 0 \\ x^2 - \sin^4 x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

7. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} a - b^{x+1} & \text{если } x \leq 0 \\ (1 + 3^{xa}) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(ax+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(ax+1)} & \text{если } x \leq 0 \\ x^a - \sin^b x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

8. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ x(1 + 2^x) & \text{если } -3 < x \leq 4 \\ \operatorname{tg} x & \text{если } x > 4 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 4 \\ \sqrt[5]{\operatorname{tg} x} & \text{если } x > 4 \end{cases}$$

9. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 - \operatorname{tg}(2x - 1) & \text{если } x \leq -2 \\ 3x(1 + e^{x+1}) & \text{если } -2 < x \leq 1 \\ \sin^5 x & \text{если } x > 1 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -2 \\ x^2 & \text{если } -2 < x \leq 1 \\ \sqrt{\cos x} & \text{если } x > 1 \end{cases}$$

10. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ (1 + 2^{\operatorname{tg} x}) & \text{если } -3 < x \leq 0 \\ \operatorname{ctg}^2 x & \text{если } x > 0 \end{cases} \quad F = \begin{cases} e^{x+1} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 0 \\ \sqrt[3]{\operatorname{tg} x} & \text{если } x > 0 \end{cases}$$

11. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1 + 3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x + 1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[3]{x} & \text{если } x > 2 \end{cases}$$

12. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - x^5 + \operatorname{tg} 8x & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{arctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{2x+1} & \text{если } x \leq 0 \\ x^2 - \sin 4x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

13. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - 9^{x+1} & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(2x+1)} & \text{если } x \leq 0 \\ x^2 - \sin^4 x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

14. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} a - b^{x+1} & \text{если } x \leq 0 \\ (1 + 3^{xa}) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(ax+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(ax+1)} & \text{если } x \leq 0 \\ x^a - \sin^b x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

15. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ x(1 + 2^x) & \text{если } -3 < x \leq 4 \\ \operatorname{tg} x & \text{если } x > 4 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 4 \\ \sqrt[5]{\operatorname{tg} x} & \text{если } x > 4 \end{cases}$$

16. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ (1+2^{\lg x}) & \text{если } -3 < x \leq 0 \\ \operatorname{ctg}^2 x & \text{если } x > 0 \end{cases} \quad F = \begin{cases} e^{x+1} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 0 \\ \sqrt[3]{\operatorname{tg} x} & \text{если } x > 0 \end{cases}$$

17. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1+3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x+1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[3]{x} & \text{если } x > 2 \end{cases}$$

18. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - x^5 + \operatorname{tg} 8x & \text{если } x \leq 0 \\ (1+3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{arctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{2x+1} & \text{если } x \leq 0 \\ x^2 - \sin 4x & \text{если } 0 < x \leq 3 \\ \sqrt[3]{x^2} & \text{если } x > 3 \end{cases}$$

19. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - 9^{x+1} & \text{если } x \leq 0 \\ (1+3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(2x+1)} & \text{если } x \leq 0 \\ x^2 - \sin^4 x & \text{если } 0 < x \leq 3 \\ \sqrt[3]{x^2} & \text{если } x > 3 \end{cases}$$

20. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ x(1+2^x) & \text{если } -3 < x \leq 4 \\ \operatorname{tg} x & \text{если } x > 4 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 4 \\ \sqrt[3]{\operatorname{tg} x} & \text{если } x > 4 \end{cases}$$

21. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1+3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x+1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[3]{x} & \text{если } x > 2 \end{cases}$$

22. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 - \operatorname{tg}(2x-1) & \text{если } x \leq -2 \\ 3x(1+e^{x+1}) & \text{если } -2 < x \leq 1 \\ \sin^5 x & \text{если } x > 1 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -2 \\ x^2 & \text{если } -2 < x \leq 1 \\ \sqrt{\cos x} & \text{если } x > 1 \end{cases}$$

23. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ (1+2^{\lg x}) & \text{если } -3 < x \leq 0 \\ \operatorname{ctg}^2 x & \text{если } x > 0 \end{cases} \quad F = \begin{cases} e^{x+1} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 0 \\ \sqrt[3]{\operatorname{tg} x} & \text{если } x > 0 \end{cases}$$

24. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1 + 3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x+1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[5]{x} & \text{если } x > 2 \end{cases}$$

25. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - x^5 + \operatorname{tg} 8x & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{arctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{2x+1} & \text{если } x \leq 0 \\ x^2 - \sin 4x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

26. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} 1 - 9^{x+1} & \text{если } x \leq 0 \\ (1 + 3^x) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(2x+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(2x+1)} & \text{если } x \leq 0 \\ x^2 - \sin^4 x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

27. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} a - b^{x+1} & \text{если } x \leq 0 \\ (1 + 3^{xa}) & \text{если } 0 < x \leq 3 \\ \operatorname{ctg}(ax+1) & \text{если } x > 3 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg}(ax+1)} & \text{если } x \leq 0 \\ x^a - \sin^b x & \text{если } 0 < x \leq 3 \\ \sqrt[5]{x^2} & \text{если } x > 3 \end{cases}$$

28. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^3 + 1 & \text{если } x \leq -3 \\ x(1+2^x) & \text{если } -3 < x \leq 4 \\ \operatorname{tg} x & \text{если } x > 4 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -3 \\ x^4 & \text{если } -3 < x \leq 4 \\ \sqrt[5]{\operatorname{tg} x} & \text{если } x > 4 \end{cases}$$

29. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 - \operatorname{tg}(2x - 1) & \text{если } x \leq -2 \\ 3x(1 + e^{x+1}) & \text{если } -2 < x \leq 1 \\ \sin^5 x & \text{если } x > 1 \end{cases} \quad F = \begin{cases} e^{\sin x} & \text{если } x \leq -2 \\ x^2 & \text{если } -2 < x \leq 1 \\ \sqrt{\cos x} & \text{если } x > 1 \end{cases}$$

30. Наиболее рационально вычислить Y и F. Использовать составной оператор.

$$Y = \begin{cases} x^5 + \operatorname{arctg} 8x & \text{если } x \leq 1 \\ 5x - (1 + 3^x) & \text{если } 1 < x \leq 2 \\ \operatorname{ctg}(2x+1) & \text{если } x > 2 \end{cases} \quad F = \begin{cases} e^{\operatorname{tg} x+1} & \text{если } x \leq 1 \\ x^4 - \operatorname{tg} 4x & \text{если } 1 < x \leq 2 \\ \sqrt[5]{x} & \text{если } x > 2 \end{cases}$$

ЛАБОРАТОРНАЯ РАБОТА №2. ТИПЫ ДАННЫХ И УПРАВЛЯЮЩИЕ СТРУКТУРЫ JAVA

Цель работы: получить практические навыки работы с управляющими структурами на языке Java.

2.1 Методические рекомендации

2.1.1 Комментарии

В языке Java поддерживаются три типа комментариев: однострочный, многострочный, документационный.

1) Однострочные комментарии

Однострочный комментарий начинается символами `//` и завершается в конце строки. Компилятор игнорирует всё от `//` до конца строки.

Пример:

```
// This example demonstrates the use of single line comments
public class HelloWorld                                // Declare class as
{                                                       // with name HelloWorld
    public static void main(String args []) // Define main
    {
        System.out.println( "Hello World." ); // Prints Hello
                                                // World to console
    }                                           // End of main
} // End HelloWorld class
```

2) Многострочные комментарии

Многострочный комментарий начинается с `/*` и заканчивается `*/`. Он может занимать одну или более строк. Компилятор игнорирует всё от `/*` до `*/`.

Пример:

`/* This is a sample class which is used to demonstrate the use of multi-line comments. This comment does not appear in the java documentation */`

```
public class HelloWorld
{
    public static void main(String args [])
    {
        System.out.println("Hello World. ");
    }
}
```

3) Документационные комментарии

Документационные комментарии начинаются с `/**` и заканчиваются `/**` и могут также занимать одну или более строк. Компилятор игнорирует этот вид комментария, точно так же как игнорируются комментарии `/` и `*/`.

Документационные комментарии идут дальше, позволяя создавать отдельную документацию через использование *javadoc*-инструмента, с помощью которого пользователь может создавать документацию API в форме страниц HTML. Документаци-

онная информация собирается непосредственно от исходного текста и комментариев документа.

Также могут внедряться html-тэги в пределах комментариев документа.

Инструмент javadoc распознает только документационные комментарии, располагающиеся сразу перед классом, интерфейсом, конструктором, методом или полевыми объявлениями. Также при объявлении должен быть только один комментарий.

2.1.2 Ключевые слова

Ключевые слова – predetermined идентификаторы, зарезервированные в Java. Программисты не могут использовать их как идентификаторы. True, false, и null не ключевые слова, но они являются зарезервированными словами, так что использовать их как имена в программах нельзя.

Вот примеры категорий ключевых слов:

Примитивные типы данных:

- **byte;**
- **short;**
- **int;**
- **long;**
- **float;**
- **double;**
- **char;**
- **boolean.**

Ключевые слова выполнения цикла:

- **do;**
- **while;**
- **for;**
- **break;**
- **continue.**

Ключевые слова, выполняющие переход:

- **if;**
- **else;**
- **switch;**
- **case;**
- **default;**
- **break.**

Метод, переменная и модификаторы класса:

- **private;**
- **public;**
- **protected;**
- **final;**
- **static;**
- **abstract;**
- **synchronized;**
- **volatile;**
- **strictfp.**

Буквальные константы:

- **false;**
- **true;**
- **null.**

Ключевые слова, связанные с методом:

- **return;**
- **void.**

Ключевые слова, связанные с пакетом:

- **package;**
- **import.**

Ключевые слова, связанные с обработкой особых ситуаций:

- **try;**
- **catch;**
- **finally;**
- **throw;**
- **throws.**

Другие ключевые слова:

- **new;**
- **extends;**
- **implements;**
- **class;**
- **instanceof;**
- **this;**
- **super.**

2.1.2 Основные типы данных

Язык Java является строго *типизированным*. Это значит, что каждая переменная и каждое выражение имеют свой определенный тип. Автоматического приведения типов попросту не существует. Иногда, конечно, возникает необходимость работать с переменными разных типов, и Java позволяет это сделать, но об этом – немного позже.

Всего в Java существует восемь простых типов данных, которые условно можно разделить на четыре группы:

- **целые (*Integers*)** – описывает целые числа с учетом знака. Включает в себя **byte, short, int, long;**
- **числа с плавающей запятой (*Floating-point number*)** – описывает «дробные» числа. Их нельзя считать полноценными дробными, т. к. они имеют определенную погрешность. Это типы: **float и double;**
- **символьный тип (*Characters*)** – это тип **char**, который предназначен для описания символов (буквы, цифры, математические знаки, и др.);
- **логический тип (*Boolean*)** – это тип **Boolean**, описывающий переменные, которые могут принимать значение «правда» (true) или «ложь» (false).

Теперь подробнее о каждом типе:

Целые – хранят значения целых чисел, без дробной части.

Название	Размер (диапазон)
byt	8 бит (от –128 до 127)

e	
short	16 бит (от −32 768 до 32 767)
int	32 бита (от −2 147 483 648 до 2 147 483 647)
long	64 бита (от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807)

Важной особенностью Java является возможность присвоения переменной не только значения в десятичной системе счисления, но также и в двоичной, восьмеричной и шестнадцатеричной.

Пример использования:

int i = 10; // переменной *i* присвоить значение 10 в десятичной системе счисления

int i2 = 0x10; // переменной *i* присвоить значение 10 в шестнадцатеричной системе счисления (16 в десятичной)

int i3 = 010; // переменной *i* присвоить значение 10 в восьмеричной системе счисления (9 в десятичной)

int i4 = b10; // переменной *i* присвоить значение 10 в двоичной системе счисления (2 в десятичной)

Типы с плавающей запятой – соответствующие стандарту IEEE754-2008.

Название	Размер (диапазон)
float	32 бита (от 3.4e−38 до 3.4e+38)
double	64 бита (от 1.7e−308 до 1.7e+308)

Здесь *e* – это степень 10, на которую умножается исходное число, так называемая экспонента.

То есть 1.5e+3 можно записать, как 1.5*1000, таким образом, записи 1.5e+3 и 1500 эквивалентны.

Как говорилось ранее, использование таких чисел предполагает некоторую погрешность, которая заложена изначально в стандарте IEEE754-2008. Но вы можете не беспокоиться, для простых задач начального уровня типа float более чем достаточно, не говоря уже о double.

Пример использования:

float f = 10.5; // переменной *f* присвоить значение 10 целых и 5 десятых

Следует помнить, что в Java, как и других языках программирования, дробная часть числа отделяется от целой точкой, а не запятой.

Символьный тип используется для описания символьных констант и переменных. Во многих языках, тип **char** занимает 8 бит (1 байт) и хранит ASCII-код символа, но в Java это не так. Так как Java создавался, основываясь на опыте более ранних языков, то в нем было учтено, что 255 ASCII-символов недостаточно для написания мультязычных программ и веб-приложений. Поэтому тип **char** в Java занимает не 8, а 16 бит, что позволяет хранить в переменной этого типа любой символ Unicode. Также символ юникода можно ввести по его номеру, например: 'u0061' и 'a' имеют

одинаковое значение. Некоторые символы нельзя набрать с клавиатуры, либо они не имеют изображения. Например, чтобы присвоить переменной символ «одинарная кавычка», то запись **char ch = ‘’**; будет вызывать ошибку компиляции. Для этого используются так называемые *Escape-последовательности* (комбинация символа и других). Помимо и существуют Escape-последовательности ():

- **\\xxx** – восьмеричный символ (xxx);
- **\\uxxxx** – шестнадцатеричный символ Юникод, где xxxx – номер символа;
- **\’** – одиночная кавычка;
- **\’’** – двойная кавычка;
- **** обратный слэш;
- **\r** – возврат каретки;
- **\n** – перевод строки (новая строка);
- **\f** – перевод страницы;
- **\t** – табуляция;
- **\b** – возврат на один символ (backspace).

Пример использования:

char ch = ‘J’; // переменной ch присвоить значение ‘J’.

Логический (булевский, boolean) тип может принимать только 2 значения: *true* и *false* (англ. «правда» и «ложь»). Этот тип может показаться совсем не нужным при написании программ. Вы спросите: «Зачем мне использовать boolean, если я могу взять int, который позволит хранить 2 000 000 000 значений?» Да, можете, но в некоторых задачах удобнее использовать именно булевский тип. Например, в условных операторах.

Пример использования:

boolean b = true; // переменной b присвоить значение true («правда»).

2.1.3 Идентификаторы

Идентификаторы используются в качестве имен классов, методов и переменных, чтобы однозначно определять их при создании идентификаторов, должны быть учтены следующие пункты:

- идентификатор может быть любой последовательностью букв верхнего и нижнего регистров, чисел или символов подчеркивания и знака коммерческого \$;
- идентификатор может содержать только два специальных символа, символ подчеркивания () и знак доллара (\$). Все другие специальные символы не используются;
- идентификатор не может содержать пробел.

Вот некоторые принятые и не принятые идентификаторы.

Принятые идентификаторы:

- **x1;**
- **first_Number;**
- **countNumber;**
- **This_Is_long_Identifier.**

Непринятые идентификаторы:

- **1x;**
- **first number;**
- **x.no;**
- **44567.**

Обратите внимание: Java подобно javascript учитывает регистр, так что идентификатор *count* отличается от идентификатора *Count*.

2.1.4 Назначение переменных и инициализация

Переменная – значение, которое может измениться по мере необходимости в течение выполнения программы; они представлены символическими именами. Другими словами, значение переменной изменяется всякий раз, когда назначается новое значение. С точки зрения программиста, переменная имеет три характеристики:

- **имя;**
- **начальное значение;**
- **область видимости.**

Имя переменной также называют идентификатором. Всякий раз, когда объявляется переменная, ей присваивается значение, и это значение является значением по умолчанию. Переменные также имеют свою область, которая определяет их видимость и время жизни в различных местах программы.

Рассмотрим синтаксис объявления переменной,

- **data type identifier [=value] [, identifier[=«value»] .]**

Обратите внимание: текст, написанный между [и], является дополнительным.

Когда мы объявляем более одной переменной, мы используем запятую, чтобы отделить идентификаторы. Например,

- **int count, age; char ch.**

Переменным могут быть назначены значения, как показано ниже.

- **count = 0;**
- **age = 20;**
- **ch = "A".**

Можно объединять объявление и инициализацию вместе. Например,

- **int count = 10, age=20;**
- **char c = "A".**

Обратите внимание: объявляйте значимые идентификаторы в программе, чтобы избежать путаницы.

Переменная может быть инициализирована динамически. Следующий код показывает, как динамически инициализирована переменная.

Пример:

```
class VarInit
{
    public static void main(String args[])
```

```

    {
        int num1 = 10, num2 = 20;
        int result = num1 + num2;
        System.out.printing(«result =» + result);
    }
}

```

В этом коде результат суммы num1 и num2 присваивается result, то есть значение result не известно, пока выполнение программы не закончено. Следовательно, это называют динамической инициализацией.

2.1.5 Пакет Java.Lang

Пакет java.lang содержит классы, которые являются базовыми в языке Java. Этот пакет содержит классы, которые являются фундаментальными для дизайна языка программирования Java. Самые важные классы – **Object**, который является корнем иерархии класса, и **Class**, образцы которого представляют классы во время выполнения.

Часто необходимо представить значение примитивного типа, как будто это объект. Обертка классов **Boolean**, **Character**, **Integer**, **Long**, **Float** и **Double** реализует эту цель. **Объект типа Double**, например, содержит поле, тип которого является **double**, представляя значение таким способом, что ссылка к этому полю может быть сохранена в переменной типа ссылки. Эти классы также обеспечивают множество методов для того, чтобы преобразовать среди примитивных значений, также поддерживать такие стандартные методы подобно «равняется» и *hashCode*. **Void** класс – non-instantiable класс, который содержит ссылку на объект Class, представляющий примитивный тип void.

Класс **Math** обеспечивает математические функции типа синуса, косинуса, и квадратного корня. Классы **String** и **StringBuffer**, подобно классу Math, обеспечивают обычно используемые операции на символьных строках.

Классы **ClassLoader**, **Process**, **Runtime**, **SecurityManager** и **System** обеспечивают «системные операции», которые управляют динамической загрузкой классов, созданием внешних процессов, ведущих запросов среды типа времени дня и принуждения политики безопасности.

Некоторые статические методы доступны в **java.lang**. **Класс Math** – чтобы исполнять математические операции. Статические методы – методы, к которым можно непосредственно обратиться с помощью класса, не создавая объектную ссылку класса.

abs() – этот метод возвращает абсолютное значение числа. Аргумент может иметь тип int, float, double или long. Тип byte и short преобразуется в int, если их передают как параметры.

Например:

```
int num = -1 ;
```

```
Math.abs(num); // returns 1
```

floor() – этот метод возвращает целое число, если оно меньше или равно параметру.

Примеры:

```
Math.floor(-5.6); // returns -6.0
```

```
Math.floor(201.1); // returns 201.0
```

```
Math.floor(100); // returns 100.0
```

max() – этот метод находит большее из двух значений. Аргумент может иметь тип данных int, long, double и float.

Примеры:

Math.max(100,200); // returns 200

min() – этот метод находит меньшее из двух значений. Аргумент может иметь тип данных int, long, double и float.

Примеры:

Math.min(100,200) returns 100

random() – этот метод возвращает случайное число между 0.0 и 1.0 из типа double.

round() – этот метод округляет аргумент с плавающей точкой к самому близкому числу. Например, запись **Math.round (34.5)** возвращается 35.

2.1.6 Управляющие структуры

Компьютер хорошо умеет исполнять повторные операции. Он может исполнять такие операции неустанно 10, 100 или даже 10 000 раз. Каждый машинный язык должен иметь особенности, которые инструктируют компьютер, как исполнять такие повторные задачи, и эти задачи решаются с помощью управляющих структур.

Управляющие структуры Java:

- **if-else;**
- **switch;**
- **loops:**
- **while;**
- **do-while;**
- **for.**

Обсудим каждый вышеупомянутый пункт.

Конструкция if-else

Конструкция if-else также известна как *условная управляющая структура*, или *структура выбора*, потому что она проверяет данное условие и исполняет указанную задачу в зависимости от того, является ли условие истинным или ложным.

Синтаксис структуры if- else:

Для простой записи if-else

```
if(condition)
    statement;
else
    statement.
```

Для множественной записи if-else:

```
if(condition){
    statement;
    statement2;
    statementN;
} else {
    statement1;
    statement2;
    statementN;
}
```

Мы можем использовать «if» без «else» следующим образом:

```
if (condition){
    statements;
}
```

Обратите внимание: условие, записанное с «if» возвращает значение переменной типа boolean, то есть истина или ложь. Использование переменной типа boolean также допускается.

```
Например,
boolean buttonClick = true;
if (buttonClick) {
    statements;
} else {
    statements;
}
```

Следующий код показывает пример структуры if-else.

Пример:

```
class IfElse
{
    public static void main(String args[]) {
        int num = 10;
        if(num % 2 == 0)
            System.out.println(num + «is even number»);
        else
            System.out.println(num + «is odd number»);
    } //end of main
} // end of IfElse
```

В вышеупомянутом коде переменная num типа int и имеет значение 10. Здесь в записи структуры if-else, если условие if имеет значение True, то выполняется соответствующая запись ниже if, в противном случае выполняется условие else.

Обратите внимание: если используется только одна конструкция в «if» или в «else», то нет никакой потребности в использовании пары изогнутых фигурных скобок.

Мы можем использовать вложенную структуру if-else в программе. Следующий фрагмент кода показывает, как использовать вложенную структуру if- else.

```
if (hour >= 7 && hour < 12)
    System.out.println(«Good Morning»);
else if(hour >= 12 && hour < 16)
    System.out.println(«Good Afternoon»);
else if(hour >= 16 && hour < 20)
    System.out.println(«Good Evening»);
else
    System.out.println(«Good Night»).
```

Конструкция switch

Объявление **switch** используется, когда должны быть выполнены многократные сравнения состояний. Оператор **switch** может также заменять длинный ряд вложенных операторов **if-else-if**. Оператор **switch** может объявляться выражением или переменной.

Синтаксис использования **switch**:

```
switch (condition) {  
    case constant_valuel :  
        statement(s);  
        break;  
    case constant_value2 :  
        statement(s);  
        break;  
    case constant_valueN :  
        statement(s);  
        break;  
    default:  
        statement(s);  
}
```

В конструкции **switch**, независимо от того, какое постоянное значение дается наряду с ключевым словом «**case**», всё согласуется с результатом условия (condition). Ключевое слово «**default**» указывает на то, что, если ни одно из значений **case** не соответствует с результатами условия, данного в **switch**, тогда выполняется условие, заданное по умолчанию.

Оператор **break** используется внутри **switch**, чтобы закончить последовательность операторов. Когда встречается оператор **break**, выполнение передается к первой строке кода, которая следует за оператором **switch**. Он создает эффект досрочного выхода из **switch**.

Вот некоторые пункты, которые вы должны помнить при написании условий и постоянных значений:

- 1) значение выражения сравнивается с каждым из указанных значений в **case**-операторах;
- 2) если соответствие найдено, то выполняется кодовая последовательность, следующая после этого оператора **case**;
- 3) если ни одна из **case**-констант не соответствует значению выражения, то выполняется оператор **default**. Оператор **default** необязателен;
- 4) если согласующихся **case** нет, и **default** не присутствует, то никаких дальнейших действий не выполняется.

Следующий фрагмент кода иллюстрирует использование оператора **switch**:

```
switch(day) {  
    case 0 : System.out.println(«Sunday»);  
        break;  
    case 1 : System.out.println(«Monday»);  
        break;
```



```

    case 2 : System.out.println(«Tuesday»);
                break;
    case 3 : System.out.println(«Wednesday»);
                break;
    case 4 : System.out.println(«Thursday»);
                break;
    case 5 : System.out.println(«Friday»);
                break;
    case 6 : System.out.println(«Saturday»);
                break;
    default : System.out.println(«Invalid day of week»);
}

```

В вышеупомянутом примере, если день между 0 и 6, то буде(у)т выполнено(ы) утверждение(я), связанно(ы)е с соответствующим case, но если день (day) больше, чем 6, то буде(у)т выполняться утверждение(я), связанно(ы)е со значением по умолчанию.

Подобно вложенным if-else, мы можем использовать вложенные структуры switch. Во вложенных структурах switch, значения внешнего выключателя и внутреннего выключателя могут быть те же самые.

2.1.7 Организация циклов

Процесс повторяемого выполнения блока утверждений известен как цикл loop. Утверждения в блоке могут быть выполнены любое количество раз, от нуля до бесконечного. Если loop продолжается всегда, его называют бесконечным циклом. Java поддерживает такие особенности циклов, которые дают возможность нам развивать короткие программы, содержащие повторяющиеся процессы.

While loop

While loop также известен как итерация. Это наиболее широко используемая структура циклов loop.

Синтаксис использования while loop:

```

while(condition) {
    statements;
}

```

Если **while loop** содержит только одно утверждение, то изогнутые фигурные скобки писать не надо. Но если есть более одного утверждения, то они должны быть включены в пределы изогнутых фигурных скобок; иначе loop выполняет только первое утверждение, пока условие истинно, при этом другие утверждения просто игнорируются.

Подобно if-else, условие while loop возвращает значение булевской переменной или выражение, которое возвращает булевское значение.

Следующая программа демонстрирует использование while loop.

Пример:

```

class WhileDemo {
    public static void main(String args[]) {
        int num = 10, count = 0;
        while(num <= 15) {
            System.out.print(«num =» + num + «\t»);
            count++;
            num++;
        }
        System.out.println(«while loop executed» + count +
«times»);
    }
}

```

Do-while

Конструкция *do-while loop* работает подобно while-loop за исключением того, что в данной конструкции цикл выполняется по крайней мере один раз, даже если условие не является истиной.

Синтаксис do-while loop:

```

do{
    statements;
} while(condition);

```

Здесь do loop сначала выполняет операторы тела цикла, а затем проверяет условие после каждого выполнения. Цикл выполняется до тех пор, пока значением условия является не нуль или True (истина).

Обратите внимание: в конструкции do-while loop, точка с запятой требуется после while(condition).

For loop

For – это компактная форма **while loop** которая объединяет инициализацию переменной, проверку условия и увеличения или уменьшение значения переменной для итерации в отдельном утверждении.

Синтаксис for loop:

```

for(var init ; condition ; incre or decre value of variable)

```

Часть 1 Часть 2 Часть 3

Обратите внимание: в конструкции for loop, объявление и инициализация переменной могут быть выполнены одновременно.

Последовательность шагов, которые нужно соблюдать при выполнении конструкции for loop следующие:

- 1) инициализация переменных (Часть 1) и проверка условия перед выполнением цикла (Часть 2). Если условие истинно, то выполняется тело цикла for loop;
- 2) когда последнее утверждение for loop выполнено, тогда выполняется Часть 3, где значение переменной увеличивается или уменьшается. После этого условие проверяется снова. Если значением условия (condition) является истина (True), то

тело цикла for loop выполняется ещё раз. Инициализация переменной делается только один раз, и цикл выполняется, пока значением условия является истина.

Следующая программа демонстрирует использование for loop.

Пример:

```
class ForDemo
{
    public static void main(String args []){
        for(int i = 1; i <= 5; i + + )
            System.out.println(«value of i is» +i);
    }
}
```

В вышеупомянутом примере объявлена переменная *i* и присвоено значение 1. Это значение проверяется условием (*i* <= 5). Так как условие истина, то *i* будет напечатано. При увеличении значения переменной *i*, снова проверяется условие *i* <= 5 и выполняется соответствующее действие. Так продолжается до тех пор, пока *i* меньше или равно 5. Когда *i* = 6, то цикл заканчивается, т. к. условие *i* <= 5 ложь. Изогнутые фигурные скобки для цикла можно опустить, если есть одна инструкция, которая будет выполняться в цикле.

В цикле мы можем сделать следующее:

- инициализировать более одной переменной;
- задать более одного условия;
- приращение или уменьшение более чем одной переменной.

Следующий фрагмент кода показывает применение в цикле более одной переменной.

```
for(int i=1,j=5;i < 5 && j >= 1;i++,j--)
System.out.println( i is  + i +  j is  +j);
```

В цикле for мы можем опустить любую часть, только помещая точку с запятой (;) Например,

```
int k =10;
for(; k <= 15; k++) {
    statement1;
    statement2;
    statementN;
}
```

Вложенность циклов for:

```
for(int a = 1; a <= 2; a++){
    for(int b = 1; b <= 3; b++){
        statement1;
        statement2;
        statementN;
    }
}
```

В вышеупомянутом примере, после инициализации переменной внешнего цикла, проверяется условие, данное во внешнем цикле, если условие истинно, то выполняется внутренний цикл `for`. Внутренний цикл `for` выполняется, пока условие истинно, и как только условие оценивается как ложь, начинает выполняться снова внешний цикл `for`. Внешний цикл выполняется, пока условие истинно. Здесь каждый раз управление передается от внешнего во внутренний цикл.

Обратите внимание: когда мы объявляем переменную в цикле `for`, область действия данной переменной только внутри цикла. В результате мы не можем использовать эту переменную вне цикла.

Оператор `break` прекращает выполнение оператора цикла и передает управление следующему за ним (за циклом) оператору.

Оператор `continue` позволяет в любой точке тела цикла прервать текущую итерацию и перейти к проверке условий продолжения цикла, определенных в предложениях `for` или `while`. В результате выполнение цикла заканчивается или начинается новая итерация.

2.1.8 Работа с массивами в Java

Массив – это список значений переменных с одинаковым именем, для обращения к значениям которых используется индекс или список индексов. Как только размер массива объявлен, то он не может быть изменен. Массивы полезны, поскольку один и тот же тип данных может быть сохранен в одном месте. Данными могут быть примитивный тип данных или объект. К индивидуальным элементам массива можно обратиться с помощью имени массива и местоположения данного элемента. Местоположение также называют *индексом*.

Массив может быть объявлен и представлен в памяти, как показано ниже
`char ch[] = new char[10];`

Вышеупомянутая запись создает массив 10 символов с именем `ch`.

Этот массив представлен следующим образом:

- 1) все элементы массива `ch` сохранены последовательно;
- 2) если мы хотим обратиться к символу, сохраненному в положении 6 массива `ch`, и назначить ему значение символьной переменной, просто пишем:

`char element = ch[5];`

- 3) если мы хотим обратиться к 5-му элементу массива `ch` и назначить ему значение переменной, просто пишем:

`char element = ch[4];`

- 4) здесь 5 и 4 – индексы массива `ch`. Индекс начинается с 0. Следовательно, чтобы обратиться к 5-му символу, индекс должен быть 4.

Рассмотрим следующие различные способы создания массивов:

`char ch[]; // just a declaration.`

Здесь длина массива не определена.

`char ch[] = new char[10]; // declaration and creation.`

Ключевое слово «`new`» используется для распределения памяти для массива. Первая запись (выше) объявляет массив, а вторая запись объявляет и создает массив.

Создание массива означает, что выделяется фактическая память для сохранения элементов массива. Это также называется реализацией массива. Когда массив создается, всем элементам массива назначаются значения по умолчанию в зависимости от типа массива. Если массив типа `int`, то всем элементам массива по умолчанию присваивается нуль. Если массив имеет переменную типа `boolean`, то все элементы имеют значение `false`. Это называется *автоинициализация*.

Следующая программа показывает пример использования массива.

Пример:

```
class ArrayDemo {
    public static void main(String args[]) {
        int integer [] = new int [5] ;
        int a = 100;
        for(int i = 0; i < 5; i++)
            integer[i] = a++;
        for(int i = 0; i < 5; i++)
            System.out.print(integer[i] + «\t»);
    }
}
```

Вышеупомянутая программа создает массив, `integer`, типа `int`. Таким образом в массиве можно хранить целочисленные значения. Первый цикл `for` используется, чтобы сохранить значения в массив, а второй цикл `for` используется, чтобы напечатать все элементы массива.

Действующая инициализация

В этом типе элементы массива инициализированы во время объявления:

```
int a[] = {10,20,30,40,50};
float floatArray[] = {248.75,45.50,873.45}.
```

В действующей инициализации нет никакой потребности определять размер массива. Размер массива определен общим количеством значений, данных в пределах изогнутых фигурных скобок.

2.2 Контрольный пример выполнения лабораторной работы

Задание: написать программу, заполняющую целочисленный вектор случайными значениями в диапазоне от 100 до 200. Размер вектора 30 элементов. Вывести полученный вектор на экран. Выполнить сортировку вектора по возрастанию. Вывести результат сортировки на экран.

Напишем программу, задающую вектор случайным образом.

```
public class ControlStructures
{
    public static int SIZE = 30;
    public static int MIN = 100;
    public static int MAX = 200;

    public static void main(String[] args) {
        int[] values = new int[ControlStructures.SIZE];
```

```

        for (int i = 0; i < ControlStructures.SIZE; i++)
        {
            values[i] = ControlStructures.MIN + (int) Math.round(Math.random() *
            (ControlStructures.MAX - ControlStructures.MIN));
            System.out.print(values[i] + "\t");
        }
    }
}

```

Вынесем код, относящийся к выводу результата (печать вектора на консоль), в отдельный метод.

```

public class ControlStructures
{
    public static int SIZE = 30;
    public static int MIN = 100;
    public static int MAX = 200;

    public static void main(String[] args)
    {
        int[] values = new int[ControlStructures.SIZE];

        for (int i = 0; i < ControlStructures.SIZE; i++)
        {
            values[i] = ControlStructures.MIN + (int) Math.round(Math.random()
            * (ControlStructures.MAX - ControlStructures.MIN));
        }
        ControlStructures.printVector(values);
    }

    private static void printVector(int[] vector)
    {
        for (int i = 0; i < vector.length; i++) {
            System.out.print(vector[i] + "\t");
        }
    }
}

```

Так как метод вывода вектора на экран будет вызываться из статического метода, то метод **printVector** также объявлен статическим.

Метод **printVector** будет вызываться **только внутри класса** **ControlStructures**, поэтому его можно объявить как **private**, тем самым понизив его область видимости.

Добавим метод, осуществляющий **сортировку**. Он также будет **static** и **private**.

```

public class ControlStructures
{
    public static int SIZE = 30;
    public static int MIN = 100;
    public static int MAX = 200;

    public static void main(String[] args)

```

```

{
    int[] values = new int[ControlStructures.SIZE];

    for (int i = 0; i < ControlStructures.SIZE; i++)
    {
        values[i] = ControlStructures.MIN + (int) Math.round(Math.random() *
        (ControlStructures.MAX - ControlStructures.MIN) );
    }

    System.out.println("Initial vector:");
    ControlStructures.printVector(values);

    int[] result = ControlStructures.sortVector(values);
    System.out.println("Sorted vector:");
    ControlStructures.printVector(result);
}

private static int[] sortVector(int[] vector)
{
    boolean flag;
    int temp;
    do {
        flag = false;
        for (int i = 0; i < vector.length - 1; i++) {
            if (vector[i] > vector[i+1])
            {
                temp = vector[i];
                vector[i] = vector[i+1];
                vector[i+1] = temp;
                flag = true;
            }
        }
    } while (flag);

    return vector;
}

private static void printVector(int[] vector)
{
    for (int i = 0; i < vector.length; i++)
    { System.out.print(vector[i] + "\t");
      System.out.println();
    }
}

```

Самостоятельно доработайте программу следующим образом:

1 Реализуйте метод **fillRandomIntVector()**, на вход которому подается размер вектора, максимальное и минимальное значения, которые могут принимать элементы вектора. Метод возвращает вектор, заполненный случайными целочисленными значениями.

2 Доработайте метод **printVector()**. Добавьте к нему параметром **String message** сообщение, которое выводится перед распечаткой элементов одномерного целочисленного массива. Пример вызова:

ControlStructures.printVector(result, "Sorted vector:").

2.3 Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

1 Теоретические сведения о типах данных Java и основных управляющих структурах.

2 Реализацию программы на языке Java в соответствии с индивидуальным заданием.

3 Листинг выполнения программы, отражающий все этапы ее выполнения.

4 Выводы о выполненной работе.

2.4. Контрольные вопросы

1 Приведите пример ключевых слов.

2 Приведите пример основных типов данных в порядке возрастания их размера.

3 Назовите символы Escape-последовательности отвечающие за перевод строки и за табуляцию.

4 Приведите несколько примеров правильных и не правильных идентификаторов.

5 Значение, в каком интервале возвращает метод Math.random() и какой метод отвечает за округление числа?

6 Зачем нужна управляющая конструкция switch?

7 Зачем нужны управляющие конструкции while loop, do while, for loop?

8 Как правильно определить двумерный массив?

2.5 Варианты заданий для самостоятельной работы

Задача 1. Определить одномерный массив и заполнить его случайными значениями:

1) Составить и вывести на экран новый массив с номерами элементов исходного массива, которые равны заданному значению. Заданное значение определяется константой;

2) Поменять местами максимальный и минимальный элементы массива. Вывести измененный массив на экран;

- 3) Все элементы массива, меньшие заданного значения, и их номера записать в новые массивы. Вывести новые массивы на экран. Заданное значение определяется константой;
- 4) Определить дополнительный массив разрешенных значений. Определить и вывести на экран, сколько элементов исходного массива имеют разрешенные значения;
- 5) Определить дополнительный массив разрешенных значений. Составить массив из элементов исходного массива, имеющих неразрешенные значения. Вывести результирующий массив на экран;
- 6) Составить и вывести на экран массив с N максимальными значениями исходного массива. N определяется константой;
- 7) Переписать элементы массива в обратном порядке на том же месте. Вывести измененный массив на экран;
- 8) Определить дополнительный массив, состоящий из неповторяющихся элементов исходного массива и вывести его на экран;
- 9) Составить и вывести на экран массив номеров элементов исходного массива, встречающихся один раз;
- 10) Определить дополнительный массив, состоящий из повторяющихся элементов исходного массива и вывести его на экран.
- 11) Вычислить сколько элементов данного массива больше своего предыдущего элемента.
- 12) Найти максимальный элемент в одномерном массиве x . Затем каждый элемент в массиве разделить на максимальный элемент.
- 13) Дан массив $b(n)$. Переписать в массив $C(n)$ положительные элементы массива $b(n)$ умноженные на 5. (сжатие массива)
- 14) Дан одномерный массив $a(n)$, в котором находится единственный нулевой элемент. Найти где он находится и вычислить сумму последующих за ним элементов. Выдать на экран номер элемента и сумму.
- 15) Определить сколько раз в этом массиве меняется знак. Например, в массиве 1, -34, 8, 14, -5, -8, -78, 3 знак меняется 4 раза.
- 16) Найти минимальный элемент в одномерном массиве x . Затем каждый элемент в массиве умножить на минимальный элемент.
- 17) Дан массив $c(n)$. Переписать в массив $x(n)$ все ненулевые элементы массива умноженные на 4. (сжатие массива)
- 18) Определить номера элементов $= 5$, количество положительных элементов для всего массива и произведение возведенных в квадрат отрицательных элементов.
- 19) Найти максимальный элемент и переставить его с 1-ым элементом массива.
- 20) Дан массив из 16 двоичных цифр (0;1). Определить сколько раз в этом массиве меняется число 0 на 1 или 1 на 0. Например, в массиве 11110010001101 число меняется 6 раз.
- 21) Найти минимальный элемент в одномерном массиве x . Затем из каждого элемента массива вычесть минимальный элемент.
- 22) Дан массив $c(n)$. Переписать в массив $x(n)$ все ненулевые элементы массива возведенные в квадрат. (сжатие массива)
- 23) Известно, что в массиве $a(n)$ есть один элемент $= 5$. Найти где он находится и вычислить сумму элементов стоящих перед ним. (выдать на экран номер элемента и сумму).

- 24) Найти сумму элементов, стоящих между максимальным и минимальным элементами.
- 25) Дан массив a из n элементов. Вычислить сколько элементов данного массива больше в 3 раза своего предыдущего элемента.
- 26) Найти максимальный по модулю элемент в одномерном массиве x . Затем к каждому элементу массива прибавить этот максимальный элемент.
- 27) Дан массив $x(n)$. Переписать в массив $y(n)$ отрицательные элементы массива x деленные на 2. (сжатие массива)
- 28) Известно, что в массиве $a(n)$ есть один элемент $= 5$. Найти где он находится и вычислить произведение элементов стоящих перед ним. (выдать на экран номер элемента и сумму).
- 29) Дано два одномерных массива $y(n)$ и $x(n)$. Переставить между собой максимальный элемент из $x(n)$ и минимальный из $y(n)$ (то есть на место максимального из $x(n)$ поставить минимальный из $y(n)$ и наоборот).
- 30) Известно, что в массиве $b(n)$ есть один отрицательный элемент. Определить где он находится и вычислить произведение элементов стоящих перед ним (выдать номер и произведение).

Задача 2.

- 1) Дан массив $b(n)$. Переписать в массив $C(n)$ положительные элементы массива $b(n)$ умноженные на 5 (со сжатием, без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 2) Дан массив $a(n)$. Переписать в массив $b(n)$ только положительные элементы массива a , умноженные на 3. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 3) Дан массив $x(n)$. Переписать в массив $y(n)$ отрицательные элементы массива x умноженные на 2. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 4) Дан массив $b(n)$. Переписать в массив $C(n)$ отрицательные элементы массива $b(n)$ умноженные на 4. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 5) Дан массив $b(n)$. Переписать в массив $C(n)$ положительные элементы массива $b(n)$ деленные на 5. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 6) Дан массив $a(n)$. Переписать в массив $b(n)$ только положительные элементы массива a , деленные на 3 (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 7) Дан массив $x(n)$. Переписать в массив $y(n)$ отрицательные элементы массива x деленные на 2. (со сжатием., без пустых элементов внутри). Затем упорядочить по возрастанию новый массив.
- 8) Дан массив $b(n)$. Переписать в массив $C(n)$ отрицательные элементы массива $b(n)$. (со сжатием., без пустых элементов внутри) Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.
- 9) Дан массив $c(n)$. Переписать в массив $x(n)$ все ненулевые элементы массива умноженные на 4. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

10) Дан массив $c(n)$. Переписать в массив $x(n)$ все ненулевые элементы массива возведенные в квадрат. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

11) Дан массив $c(n)$. Переписать в массив $x(n)$ все ненулевые элементы массива (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

12) Дан массив $c(n)$. Переписать в массив x ненулевые элементы массива c разделенные на 5. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

13) Дан массив $b(n)$. Переписать в массив $C(n)$ корни квадратные из положительных элементов массива $b(n)$ деленные на 5. (со сжатием., без пустых элементов внутри) Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

14) Дан массив $b(n)$. Переписать в массив $C(n)$ корни квадратные из положительных элементов массива $b(n)$ (со сжатием., без пустых элементов внутри) Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

15) Дан массив $x(n)$. Переписать в массив $y(n)$ элементы массива x , большие 3. (со сжатием., без пустых элементов внутри). Затем упорядочить методом «выбора и перестановки» по возрастанию новый массив.

16) Дан одномерный массив $a(n)$, в котором находится единственный нулевой элемент. Найти где он находится, и упорядочить по возрастанию элементы, расположенные за ним. Выдать на экран номер элемента и упорядоченный массив.

17) Известно, что в массиве $x(n)$ есть один элемент $= 1$. Определить где он находится, и упорядочить по убыванию элементы, расположенные за ним. Выдать на экран номер элемента и упорядоченный массив.

18) В массиве $z(n)$ один отрицательный элемент. Найти где он находится, и упорядочить по возрастанию элементы, расположенные за перед ним. Выдать на экран номер элемента и упорядоченный массив.

19) Дан одномерный массив $a(n)$, в котором находится единственный элемент равный 5. Найти где он находится, и упорядочить по убыванию элементы, расположенные перед ним. Выдать на экран номер элемента и упорядоченный массив.

20) Найти максимальный и минимальный элементы в одномерном массиве x , а также их порядковые номера. Затем упорядочить по возрастанию элементы, расположенные между максимальным и минимальным элементами.

21) Найти максимальный элемент и его порядковый номер в одномерном массиве x . Затем упорядочить по возрастанию элементы, расположенные после максимального элемента.

22) Найти минимальный элемент и его порядковый номер в одномерном массиве x . Затем упорядочить по убыванию элементы, расположенные после минимального элемента.

23) Найти максимальный элемент и его порядковый номер в одномерном массиве x . Затем упорядочить по возрастанию элементы, расположенные перед максимальным элементом.

24) Найти минимальный элемент и его порядковый номер в одномерном массиве x . Затем упорядочить по возрастанию элементы, расположенные перед минимальным элементом.

25) Дан одномерный массив a . Записать в массив z все порядковые номера элементов массива a равные 1. Затем упорядочить по возрастанию элементы массива a , расположенные между двумя последними элементами равными 1.

26) Дан одномерный массив a . Записать в массив z все порядковые номера отрицательных элементов массива a . Затем упорядочить по убыванию элементы массива a , расположенные между первыми двумя отрицательными элементами

27) Дан одномерный массив a . Записать в массив $г$ все порядковые номера нулевых элементов массива a . Затем упорядочить по убыванию элементы массива a , расположенные между двумя первыми нулевыми элементами.

28) Дан одномерный массив a . Записать в массив z все порядковые номера элементов больших 1 массива a . Затем упорядочить по возрастанию элементы массива a , расположенные между любыми двумя элементами большими 1.

29) Дан одномерный массив $a(n)$, в котором находится единственный элемент, значение которого принадлежит интервалу от 2 до 5. Найти где он находится, и упорядочить по убыванию элементы, расположенные перед ним. Выдать на экран номер элемента и упорядоченный массив.

30) Найти минимальный элемент и его порядковый номер в одномерном массиве x . Затем упорядочить по возрастанию элементы, расположенные перед минимальным элементом.

Задача 3.

Определить матрицу (двумерный массив) и ее заполнить случайными значениями.

Построить вектор B , которой возвращает –

- 1) число неотрицательных элементов в i -й строке;
- 2) среднее арифметическое положительных элементов в i -м столбце;
- 3) минимальное значение в i -й строке;
- 4) максимальное значение в i -м столбце;
- 5) номер максимального значения в i -й строке;
- 6) номер минимального значения в i -м столбце;
- 7) число элементов i -й строки, значения которых меньше заданного значения;
- 8) значение элемента матрицы, не равное заданному значению;
- 9) равно 1, если значения элементов i -й строки упорядочены по возрастанию, и 0, в противном случае;
- 10) количество четных чисел в i -й строке.
- 11) сумму положительных элементов в каждом столбце матрицы.
- 12) произведение положительных элементов в каждом столбце матрицы.
- 13) количество положительных элементов в каждом столбце матрицы.
- 14) среднее арифметическое положительных элементов в каждом столбце матрицы
- 15) среднее геометрическое положительных элементов в каждом столбце матрицы
- 16) сумму отрицательных элементов в каждом столбце матрицы.
- 17) произведение отрицательных элементов в каждом столбце матрицы.
- 18) минимальный элемент в каждой строке матрицы. Затем каждую строку матрицы разделить на минимальный элемент строки.

- 19) минимальный элемент в каждой строке матрицы среди положительных элементов.
- 20) максимальный элемент в каждой строке матрицы среди отрицательных элементов.
- 21) минимальный элемент в каждом столбце матрицы. Затем каждый столбец матрицы умножить на минимальный элемент.
- 22) максимальный элемент в каждой строке матрицы. Затем к каждому элементу каждой строки прибавить максимальный элемент строки.
- 23) минимальный элемент и его номер в каждой строке матрицы. Затем из каждого элемента каждой строки вычесть номер минимального элемента строки.
- 24) номер минимального элемента в каждой строке матрицы. Затем к каждому элементу каждой строки прибавить номер минимального элемента строки.
- 25) номер максимального элемента в каждом столбце матрицы. Затем каждый элемент каждого столбца умножить на номер максимального элемента столбца.
- 26) номер максимального элемента в каждой строке матрицы. Затем каждый элемент каждой строки разделить на номер максимального элемента строки.
- 27) среднеарифметический элемент в каждой строке матрицы среди положительных элементов.
- 28) среднеарифметический элемент в каждой строке матрицы среди отрицательных элементов.
- 29) среднеарифметический элемент в каждой строке матрицы. Затем каждую строку матрицы умножить на среднеарифметический элемент строки.
- 30) среднегеометрический элемент в каждом столбце матрицы. Затем из каждого элемента каждого столбца вычесть среднегеометрический элемент столбца.

Задача 4.

1. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти номер минимального элемента её побочной диагонали.
2. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти сумму номеров минимального и максимального элементов её побочной диагонали.
3. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти сумму номеров минимального и максимального элементов её главной диагонали.
4. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти произведение минимального и максимального элементов её главной диагонали. Затем умножить побочную диагональ на максимальный элемент главной диагонали.
5. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество положительных элементов её главной диагонали. Затем умножить побочную диагональ на найденное количество.
6. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов её побочной диагонали.
7. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое положительных элементов её побочной диагонали.
8. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов параллели главной диагонали, расположенной выше над диагональю.

9. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти минимальный элемент среди положительных элементов параллели главной диагонали, расположенной выше над диагональю/
10. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое отрицательных элементов параллели главной диагонали, расположенной под диагональю.
11. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти произведение отрицательных элементов параллели побочной диагонали, расположенной над диагональю,
12. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти максимальный элемент среди отрицательных элементов параллели побочной диагонали, расположенной над диагональю,
13. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти сумму положительных элементов параллели побочной диагонали, расположенной под диагональю (ниже побочной диагонали).
14. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти сумму и количество положительных элементов параллели побочной диагонали, расположенной под диагональю (ниже побочной диагонали). Затем каждый элемент побочной диагонали умножить на количество.
15. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов параллели главной диагонали, расположенной выше над диагональю.
16. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое положительных элементов верхней треугольной матрицы, расположенной выше главной диагонали, исключая саму главную диагональ.
17. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов, верхней треугольной матрицы, расположенной выше главной диагонали,
18. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов, нижней треугольной матрицы, расположенной ниже главной диагонали, исключая саму главную диагональ.
19. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое положительных элементов, нижней треугольной матрицы, расположенной ниже главной диагонали, включая саму главную диагональ.
20. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество положительных элементов, нижней треугольной матрицы, расположенной ниже главной диагонали, включая саму главную диагональ.
21. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое элементов, нижней треугольной матрицы, расположенной ниже побочной диагонали, включая саму побочную диагональ.
22. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество нулевых элементов, нижней треугольной матрицы, расположенной ниже побочной диагонали, включая саму побочную диагональ.
23. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое отрицательных элементов, нижней треугольной матрицы, расположенной ниже побочной диагонали, исключая саму побочную диагональ.

24. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество и сумму отрицательных элементов, нижней треугольной матрицы, расположенной ниже побочной диагонали, исключая саму побочную диагональ.

25. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество элементов, равных заданному числу x и расположенных в верхней треугольной матрице, расположенной выше побочной диагонали, исключая саму побочную диагональ.

26. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти произведение элементов, расположенных в верхней треугольной матрице, расположенной выше побочной диагонали, включая саму побочную диагональ.

27. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти количество нулевых элементов, расположенных в верхней треугольной матрице, расположенной выше побочной диагонали, включая саму побочную диагональ.

28. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее арифметическое положительных элементов, верхней треугольной матрицы, расположенной выше главной диагонали,

29. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти произведение элементов, расположенных в верхней треугольной матрице, расположенной выше побочной диагонали, включая саму побочную диагональ.

30. Дан двумерный массив A , размером $(n \times n)$ (или квадратная матрица A). Найти среднее геометрическое положительных элементов верхней треугольной матрицы, расположенной выше главной диагонали, исключая саму главную диагональ.

ЛАБОРАТОРНАЯ РАБОТА №3. КЛАССЫ И ОБЪЕКТЫ В JAVA

Цель работы. Получить практические навыки разработки программ использованием объектно-ориентированного подхода на языке Java, создавать классы и объекты.

3.1. Методические рекомендации

3.1.1 Концепции объектно-ориентированного программирования

Что такое объект?

Языки структурного программирования, такие как Си и Pascal, следуют совсем иной парадигме программирования, чем объектно-ориентированные языки. Парадигма структурного программирования ориентирована на данные, что означает, что сначала создаются структуры данных, а затем пишутся команды для работы с этими данными. В объектно-ориентированных языках, таких как Java, данные и команды программы скомбинированы в *объекты*.

Объект представляет собой автономный модуль со своими атрибутами и поведением. Вместо структуры данных с полями (атрибуты), которая отражается на всей логике программы, влияющей на ее поведение, в объектно-ориентированном языке данные и логика программы объединены. Эта комбинация может быть реализована на совершенно разных уровнях детализации, от самых мелких объектов, до самых крупных.

Родительские и дочерние объекты

Родительский объект служит в качестве структурной основы для получения более сложных дочерних объектов. Дочерний объект повторяет родительский, но является более специализированным. Объектно-ориентированная парадигма позволяет многократно использовать общие атрибуты и поведение родительского объекта, добавляя к ним новые атрибуты и поведение дочерних объектов.

Связь между объектами и координация

Объекты общаются друг с другом, отправляя сообщения (на языке Java – вызовы методов). Кроме того, в объектно-ориентированных приложениях программа координирует взаимодействие между объектами для решения задачи в контексте данной предметной области.

Краткие сведения об объектах

Хорошо написанный объект:

- имеет четкие границы;
- имеет конечный набор действий;
- "знает" только о своих данных и любых других объектах, которые нужны для его деятельности.

По сути, объект – это дискретный модуль, который обладает только необходимыми зависимостями от других объектов для решения собственных задач.

Теперь посмотрим, как выглядит объект.

Объект *Person*

Начнем с примера, основанного на общем сценарии разработки приложений: физического лица, представленного объектом *Person*.

Возвращаясь к определению объекта, вспомним, что объект состоит из двух основных элементов: атрибутов и поведения. Посмотрим, как это относится к объекту *Person*.

Атрибуты (Свойства)

Какие атрибуты может иметь физическое лицо? Вот самые распространенные из них:

- имя,
- возраст,
- рост,
- вес,
- цвет глаз,
- пол.

Можно придумать еще (дополнительные атрибуты всегда можно добавить), но для начала этого достаточно.

Поведение

Физическое лицо может делать все что угодно, но поведение объекта обычно относится к контексту какого-то приложения. Например, в контексте бизнес-приложения можно задать своему объекту *Person* вопрос: "Сколько вам лет?" В ответ на это *Person* сообщит значение своего атрибута "возраст".

Внутри объекта *Person* может быть скрыта и более сложная логика, но пока предположим, что *Person* умеет отвечать на вопросы:

- Как вас зовут?
- Сколько вам лет?
- Ваш рост?
- Ваш вес?
- Какого цвета ваши глаза?
- Какого вы пола?

Состояние и строка

Состояние – важное понятие в ООП. В каждый момент времени состояние объекта представлено значением его атрибутов.

В случае объекта *Person* его состояние определяется такими атрибутами, как имя, возраст, рост и вес.

3.1.2 Структура объекта Java

Объект представляет собой дискретный модуль со своими атрибутами и поведением. Это означает, что он имеет четкие границы и состояние и может выполнять разнообразные действия, если к нему правильно обратиться. В каждом объектно-ориентированном языке есть правила определения объектов.

В языке Java объекты определяются, как показано в листинге 4.1. Листинг 3.1. Определение объекта

```
package packageName;  
  
import ClassNameToImport;  
accessSpecifier class ClassName {  
    accessSpecifier dataType variableName [= initialValue];  
    accessSpecifier ClassName([argumentList]) {
```

```

    constructorStatement(s)
}
accessSpecifier returnType methodName([argumentList]) {
    methodStatement(s)
}
// Это комментарий
/* Это тоже комментарий */
}

```

где:

- *packageName* – наименование пакета в котором находится класс;
- *ClassNameToImport* – наименование импортируемого класса или пакета;
- *accessSpecifier* – модификатор доступа;
- *ClassName* – наименование класса;
- *dataType* – тип атрибута;
- *variableName* – наименование атрибута;
- *initialValue* – начальная инициализация объекта;
- *argumentList* – список передаваемых аргументов;
- *constructorStatement* – операторы конструктора;
- *returnType* – возвращаемый тип;
- *methodName* – наименование метода;
- *methodStatement* – операторы метода.

В листинге 3.1 содержатся конструкции разного типа. Конструкции, показанные жирным шрифтом – литералы (зарезервированные слова); в любом определении объекта они должны быть точно такими же, как здесь.

Упаковка объектов

Язык Java позволяет выбирать имена объектов, такие как Account, Person или LizardMan. Время от времени может получаться так, что одно и то же имя должно выражать два разных понятия. Это называется *конфликтом имен* и происходит довольно часто. Для разрешения таких конфликтов в языке Java используются *пакеты*.

Пакет Java – это механизм организации пространства имен: ограниченная область, в которой имена уникальны, но вне которой они могут не существовать. Чтобы определить конструкцию как уникальную, нужно полностью описать ее, указав ее пространство имен.

Пакеты предоставляют также хороший способ для построения более сложных приложений из дискретных единиц функциональности.

Определение пакета

Для определения пакета используется ключевое слово **package**, за которым следует формальное имя пакета, заканчивающееся точкой с запятой (*packageName*). Часто имена пакетов разделяются точками и следуют такой схеме *де-факто*:

```
package orgType.orgName.appName.compName;
```

- Это определение пакета расшифровывается так:
- *orgType* – это тип организации, такой как com, org или net.

- **OrgName** – доменное имя организации, такое как makotogroup, sun или ibm;
- **AppName** – сокращенное имя приложения;
- **compName** – имя компонента.

Язык Java не вынуждает следовать этому соглашению о пакетах. На самом деле, определять пакет вообще не обязательно, но тогда все объекты должны иметь уникальные имена классов и будут находиться в пакете по умолчанию.

Операторы импорта

Следующим в определении объекта (возвращаясь к листингу 3.1) идет *оператор import*. Он сообщает компилятору Java, где найти классы, на которые ссылается код. Любой нетривиальный объект использует другие объекты для выполнения тех или иных функций, и оператор импорта позволяет сообщить о них компилятору Java.

Оператор импорта обычно выглядит так:

```
import ClassNameToImport;
```

За ключевым словом **import** следуют класс, который нужно импортировать, и точка с запятой. Имя класса должно быть полным, то есть включать свой пакет.

Чтобы импортировать все классы из пакета, после имени пакета можно поместить **.***. Например, следующий оператор импортирует каждый класс пакета **com.makotogroup**:

```
import com.makotogroup.*;
```

Импорт всего пакета может сделать код менее читабельным, поэтому рекомендуется импортировать только нужные классы.

Eclipse упрощает импорт

При написании кода в редакторе Eclipse можно ввести имя класса, а затем нажать **Ctrl+Space**. Eclipse определяет, какие классы нужно импортировать, и добавляет их автоматически. Если Eclipse находит два класса с одним и тем же именем, он выводит диалоговое окно с запросом, какой именно класс вы хотите добавить.

3.1.3. Объявление класса

Чтобы определить объект в языке Java, нужно объявить класс. Класс можно представить в качестве шаблона объектов, как формочка для печенья. Класс определяет базовую структуру объекта и во время выполнения программы создает экземпляр этого объекта.

Листинг 3.1 содержит следующее объявление класса:

```
accessSpecifier class ClassName {
    accessSpecifier dataType variableName [= initialValue];
    accessSpecifier ClassName(argumentList) {
        constructorStatement(s)
    }
    accessSpecifier returnType methodName(argumentList) {
        methodStatement(s)
    }
}
```

3.1.4 Соглашения об именах классов

В принципе классы можно называть как угодно, но по устоявшемуся соглашению используют *ВерблюжийСтиль* (CamelCase): начинают с прописной буквы, и первую букву каждого слова тоже делают прописной, а все остальные буквы – строчными. Имена классов должны содержать только буквы и цифры. Соблюдение этих принципов гарантирует, что ваш код будет понятен другим разработчикам, следующим тем же правилам.

У класса могут быть члены двух типов: *свойства* и *методы*.

Свойства

Значения переменных данного класса различны в разных экземплярах этого класса и определяют его состояние. Эти значения часто называют *свойства экземпляра*. Определение свойства содержит следующие элементы:

- *accessSpecifier*;
- *dataType*;
- *variableName*;
- а также может содержать *initialValue*.

Возможные значения *accessSpecifier*:

- **public**: переменную видит любой объект любого пакета;
- **protected**: переменную видит любой объект, определенный в том же пакете, или подкласс (определенный в любом пакете);
- без указателя (его еще называют доступом *для своих* или *внутри пакетным* доступом): переменную видят только объекты, класс которых определен в том же пакете;
- **private**: переменную видит только класс, ее содержащий.

Тип (*dataType*) переменной зависит от того, что представляет собой переменная – это может быть простой тип или тип другого класса.

Имя переменной (*variableName*) зависит от вас, но по общему соглашению, в именах переменных используется *ВерблюжийСтиль*, за исключением того, что начинаются они со строчной буквы. (Этот стиль иногда называют *lowerCamelCase*.)

Пример: определение класса *Person*

Прежде чем перейти к методам, приведем пример, суммирующий все, что вы усвоили до сих пор. Листинг 3.2 представляет собой определение класса *Person*.

Листинг 3.2. Определение класса *Person*

```
package com.makotogroup.intro;
public class Person {
    private String name;
    private int age;
    private int height;
    private int weight;
    private String eyeColor;
    private boolean gender;
}
```

Методы

Методы класса определяют его поведение. Иногда такое поведение – не более чем возврат (геттер, *getter*) текущего значения атрибута. В других случаях поведение может быть довольно сложным.

Существуют две категории методов: **конструкторы** и все прочие методы. Метод-конструктор используется только для создания экземпляра класса. Другие методы могут использоваться практически для любого поведения программы.

Возвращаясь к [листингу 3.1](#), он иллюстрирует способ определения структуры метода, который включает в себя такие вещи, как:

```
accessSpecifier;  
returnType;  
methodName;  
argumentList.
```

Сочетание этих структурных элементов в определении метода называется его *сигнатурой*.

Рассмотрим оба типа методов подробнее, начиная с конструкторов.

Методы-конструкторы

Конструкторы позволяют указать, как создавать экземпляр класса. В листинге 1 был показан синтаксис декларации конструктора в абстрактной форме; он еще раз приведен в листинге 3.3.

Листинг 3.3 Синтаксис декларации конструктора

```
accessSpecifier ClassName(argumentList) {  
    constructorStatement(s)  
}
```

Если конструктор отсутствует, компилятор создаст конструктор по умолчанию (или *без аргументов*). Но он не станет его генерировать, если указать другой конструктор, отличный от конструктора без аргументов.

accessSpecifier для конструктора тот же, что и для переменных. Имя конструктора должно совпадать с именем класса. Так что если мы назвали свой класс *Person*, то и имя конструктора должно быть ***Person***.

Для любого конструктора, кроме конструктора по умолчанию, создается список аргументов ***argumentList***, который содержит один или более аргументов:

```
argumentType argumentName
```

Аргументы в списке ***argumentList*** разделены запятыми, и два аргумента не могут носить одно и то же **имя.argumentType** — это либо простой тип, либо тип класса (так же, как в случае с типами переменных).

Определение класса с помощью конструктора

Теперь посмотрим, что происходит, когда появляется возможность создания объекта ***Person*** двумя способами: с использованием конструктора без аргументов и с инициализацией неполного списка атрибутов.

В листинге 3.4 показано, как создавать конструкторы и как использовать ***argumentList***.

Листинг 3.4. Определение класса Person с помощью конструктора

```
package com.makotogroup.intro;
public class Person {
    private String name;
    private int age;
    private int height;
    private int weight;
    private String eyeColor;
    private boolean gender;
    public Person() {
        // Делать больше нечего...
    }
    public Person(String name, int age, int height, String eyeColor, boolean gender)
{
    this.name = name;
    this.age = age;
    this.height = height;
    this.weight = weight;
    this.eyeColor = eyeColor;
    this.gender = gender;
}
}
```

Обратите внимание на использование ключевого слова **this** при присвоении значений переменным в [листинге 3.4](#). В Java оно означает "**this object**" (этот объект) и служит для обращения к двум переменным с одинаковыми именами (как в данном случае, когда **age** – это и параметр конструктора, и переменная класса), а также помогает компилятору при неоднозначности ссылки.

У любого класса есть **конструктор по умолчанию**, например:

```
public Person() {}
```

Но при указании нового конструктора, конструктор по умолчанию становится не доступным до тех пор, пока его явно не указать.

Другие методы

Конструктор – это метод особого рода с особой функцией. Точно так же методы многих других видов выполняют конкретные обязанности в Java-программах.

В листинге 3.1 был показан синтаксис объявления метода:

```
accessSpecifier returnType methodName([argumentList]) {
    methodStatement(s)
}
```

Другие методы, за несколькими исключениями, выглядят так же, как конструкторы. Во-первых, другие методы можно называть, как вам заблагорассудится, но существуют следующие соглашения:

- начинайте со строчной буквы;
- избегайте цифр без крайней необходимости;
- используйте только буквы.

Другие методы объекта Person

Можно посмотреть в листинге 3.5, что произойдет при добавлении к объекту Person еще нескольких методов.

Листинг 3.5. Person с несколькими новыми методами

```
package com.makotogroup.intro;

public class Person {
    private String name;
    private int age;
    private int weight;
    private boolean gender;

    public String getName() { return name; }
    public void setName(String value) { name = value; }
    // Другие комбинации геттеров/сеттеров...
}
```

Обратите внимание на комментарий в [листинге 5](#) о "комбинациях геттеров/сеттеров". *Геттер* – это метод для получения значения атрибута, а *сеттер* – для изменения этого значения. В листинге 3.5 показана одна комбинация геттер/сеттер (для атрибута Name), но аналогичным образом можно определить и другие.

Методы экземпляра и статические методы

Существуют два основных типа методов (кроме конструкторов): *методы экземпляра* (обычные методы) и *методы класса* (статические методы). Поведение метода экземпляра зависит от состояния конкретного экземпляра объекта. Статические методы иногда еще называют *методами класса*, так как их поведение не зависит от состояния какого-либо одного объекта. Поведение статического метода определяется на уровне класса.

Статические методы используются в основном для удобства, их можно представить как способ создания глобальных методов с сохранением при этом самого кода сгруппированным с классом, которому они нужны.

Метод toString()

Этот метод служит для представления объекта в виде строки. Это требуется, например, если необходимо вывести объект на экран.

Самое главное знать, что метод **toString()** есть у всех объектов и все объекты используют этот метод при работе со строками. Этот метод является методом класса Object. В случае, когда от объекта требуется результат типа **String**, например: **System.out.println(new Object())**, этот метод вызывается автоматически. Он возвращает представление объекта в виде строки и по-умолчанию состоит из двух составляющих разделенных собачкой. Эти составляющие: **имя_класса_объекта** и **хэш_кода**. Пример: **java.lang.Integer;@24d200d8**.

Но этот метод можно переопределить в своем классе и вывести более подробную информацию (листинг 3.6).

```
package com.makotogroup.intro;
public class Person {
```

```

private String name;
private int age;
private int weight;
private boolean gender;
public Person(String name, int age, int weight, boolean gender) {
    this.name = name;
    this.age = age;
    this.weight = weight;
    this.gender = gender;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public int getWeight() {
    return weight;
}
public void setWeight(int weight) {
    this.weight = weight;
}
public boolean getGender() {
    return gender;
}
public void setGender(boolean gender) {
    this.gender = gender;
}
public String toString()
{
    return "My name is " + name + " I'm " + age + " years old.";
}
}

package com.makotogroup.intro;
public class TestPerson {
public static void main(String[] args) {
    Person person = new Person("Иван", 20, 175, true);
    System.out.println(person.toString());
    System.out.println(person);
}
}

```


Оператор *new* создает экземпляр (объект) указанного класса и возвращает ссылку на вновь созданный объект. Ниже приведен пример создания и присваивание переменной *person* экземпляра класса *Person*.

```
Person person = new Person("Иван", 20, 175, true);
```

Где:

– *Person("Иван", 20, 175, true)* – является конструктором.

Можно создать несколько ссылок на один и тот же объект. Например:

```
Person person1 = new Person("Иван", 20, 175, true);
```

```
Person person2 = person1;
```

3.3 Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

- 5 Теоретические сведения о структуре объекта Java.
- 6 Реализацию программы на языке Java в соответствии с индивидуальным заданием.
- 7 Распечатанный листинг выполнения программы, отражающий все этапы ее выполнения.
- 8 Выводы о выполненной работе.

3.4 Контрольные вопросы

- 1) Что такое сигнатура метода?
- 2) Зачем нужен конструктор?
- 3) Дайте определение объекту?
- 4) Для каких целей используются пакеты в java?
- 5) Зачем необходимо ключевое слово *new*?
- 6) Зачем нужны комментарии?
- 7) Описать сигнатуру пользовательского конструктора и конструктора по умолчанию.
- 8) Зачем нужны операторы импорта?

3.5. Варианты заданий для самостоятельной работы

Создать программу на языке Java для определения класса в некоторой предметной области. Описать свойства, конструктор, методы геттеры/сеттеры, перекрыть метод *toString()* для вывода полной информации об объекте в отформатированном виде:

Вариант 1). Записная книжка контактов.

Contact – запись информации о контакте в записную книжку.

Свойства:

- | | | |
|---|---|-------------|
| <ul style="list-style-type: none">– Id – идентификатор контакта;– first-Name – имя;– lastName – фамилия;– address – адрес; | } | Конструктор |
|---|---|-------------|

- phone – телефон;
- note – запись о контакте.

Вариант 2). Система управления доставкой товара.

Order - заявка:

Свойства:

- Id – идентификатор;
 - name – название товара;
 - courier – курьер (ответственный за доставку);
 - dateTime – дата и время (String);
 - type – тип заказа (1 – срочный заказ; 2 – обычный заказ).
- } Конструктор

Вариант 3). Телепрограмма.

Show - передача:

Свойства:

- authr – ведущий;
 - name – название;
 - description – описание;
 - periodType – периодичность (1 – ежедневно; 2 – еженедельно; 3 – ежемесячно).
- } Конструктор

Вариант 4). Гостиница

Room – комната:

Свойства:

- Id – идентификатор;
 - codeNumbers – Код номера;
 - numberPeople – Количество человек;
 - comfortType – Комфортность;
 - price – цена.
- } Конструктор

Вариант 5). Реализация готовой продукции

Commodity – Товар:

Свойства:

- id – идентификатор;
 - productCode – Код товара;
 - name – Наименование;
 - wholesalePrice – Оптовая цена;
 - retailPrice – Розничная цена;
 - description – Описание;
- } Конструктор

Вариант 6). Успеваемость студентов ВУЗА

Students – Студент:

Id_studenta – номер зачетной книжки

Fam – фамилия

Name – имя

Groupa – группа

Department – кафедра

discipline- Дисциплина

mark- Оценка

NameTeacher- Фамилия преподавателя

} Конструктор

Вариант 7). Деканат

NameFaculty - факультет

Room – аудитория

corps - корпус

Telephone – контактный телефон

NameDean – фамилия декана

} Конструктор

Вариант 8). Супермаркет

Supermarket:

Свойства:

- nameotdela – название отдела;
- productCode – Код товара;
- name – Наименование товара;
- cuntry – страна-производитель;
- retailPrice – Розничная цена;
- namesource – Поставщик;

} Конструктор

Вариант 9). Военный состав

Command:

Свойства:

- фамилия;
- рота;
- звание;
- дата рождения;
- дата поступления на службу;
- часть;

} Конструктор

Вариант 10). Литература

Literature:

Свойства:

- код источника литературы;
- Тип литературы;
- название;
- год издательства;
- название издательства;

} Конструктор

- количество страниц;
- автор;

Вариант 11). Продажа путевок

Tourist:

Свойства:

- код путевки;
- фамилия клиента;
- название пансионата;
- номер;
- вид жилья;
- дата заезда;
- дата выезда;
- количество человек;
- цена;

} Конструктор

Вариант 12). Станция техобслуживания

ServiceCenter:

Свойства:

- название станции;
- адрес станции;
- название автотранспорта на ремонте;
- вид ремонта;
- дата поступления;
- дата выдачи;
- результат ремонта;
- фамилия персонала;
- сумма ремонта;

} Конструктор

Вариант 13). Медицинское обслуживание пациентов

Polyclinic:

Свойства:

- название поликлиники;
- адрес поликлиники;
- фамилия пациента;
- номер полиса;
- дата осмотра;
- фамилия врача;
- должность врача;
- диагноз;

} Конструктор

Вариант 14). Выдача литературы

Library:

Свойства:

- название библиотеки;
- название читательского зала;
- фамилия читателя;
- Название литературы;
- Дата выдачи;
- Срок выдачи;
- Сумма залога;

} Конструктор

Вариант 15). Продажа автомобилей

Car:

Свойства:

- марка автомобиля;
- Год выпуска;
- Цена автомобиля;
- Комплектация;
- Страна производитель;
- Дата продажи;
- ФИО покупателя;

} Конструктор

Вариант 16). Интернет-магазин

ElectronicShopping:

Свойства:

- Название магазина;
- Название товара;
- Страна производитель
- Вид оплаты;
- Сумма покупки;
- Дата продажи;
- ФИО покупателя;

} Конструктор

Вариант 17). Больница

Hospital:

Свойства:

- название больницы;
- Название отделения;
- ФИО пациента;
- номер полиса;
- дата поступления;
- дата выписки;
- диагноз;
- дата проведения операции;
- название операции;
- стоимость лечения;

} Конструктор

Вариант 18). Военно-морские учения

Marine:

Свойства:

- Название военной части;
- Название корабля;
- Тип корабля;
- Дата проведения учения;
- Нарботка корабля;
- Количество личного состава;
- Место проведения учений;
- Результат учений.

} Конструктор

Вариант 19). Туристические туры

Tourist trip:

Свойства:

- название тура;
- страны;
- города;
- тип передвижения;
- тип питания;
- цена тура;
- тип проживания;
- дата выезда;

} Конструктор

Вариант 20). Представление цирка

Circus:

Свойства:

- название представления;
- город
- Дата премьеры;
- Период проведения;
- Цена билета;
- Автор;
- Жанр;
- Количество актеров;

} Конструктор

Вариант 21). Аптека

Pharmacy:

Свойства:

- Название аптеки;
- ФИО пациента
- Название лекарства;
- Тип лекарства;
- Цена лекарства;
- Страна-производитель;
- Дата продажи;

} Конструктор

Вариант 22). Абонентская плата

TelephoneSubscriber:

Свойства:

- ФИО абонента;
- адрес
- телефон;
- месяц;
- год;
- количество минут;
- стоимость;

} Конструктор

Вариант 23). Стоматология

Stomatology:

Свойства:

- название стоматологии;
- адрес стоматологии;
- фамилия пациента;
- номер полиса;
- дата лечения;
- фамилия врача;
- описание работ;
- сумма выполненных услуг;

} Конструктор

Вариант 24). Техническое обслуживание подвижного состава

MaintenanceStation:

Свойства:

- название депо;
- адрес депо;
- номер вагона на ремонте;
- тип ремонта;
- дата поступления;

} Конструктор

- дата завершения ремонта;
- результат ремонта;
- фамилия сотрудника;
- сумма ремонта;

Вариант 25). Кафедра

NameChair - кафедра

Room – аудитория

corps - корпус

Telephone – контактный телефон

NameZavKaf – фамилия заведующего

Kol_teacher – количество преподавателей

} Конструктор

ЛАБОРАТОРНАЯ РАБОТА №4. НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

Цель работы. Получить практические навыки создания иерархий наследования классов. Научиться проектировать классы, используя механизм наследования. Иметь представление о полиморфизме, преимуществах его использования при проектировании классов.

4.1. Методические рекомендации

4.1.1 Общие сведения о реализации механизма наследования в языке Java Наследование

Наследование – один из трех важнейших механизмов объектно ориентированного программирования (наряду с инкапсуляцией и полиморфизмом), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса. Класс, от которого произошло наследование, называется базовым или родительским (англ. *base class*). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. *derived class*). В некоторых языках используются абстрактные классы. Абстрактный класс – это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного класса. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование. При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML. Множественное наследование – потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках (рис. 1).

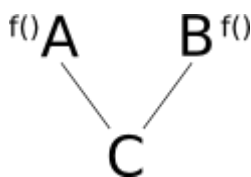


Рис. 1. Множественное наследование

В языке Java поддерживается только простое наследование: любой подкласс является производным только от одного непосредственного суперкласса. При этом любой класс может наследоваться от нескольких интерфейсов.

Наследование интерфейсов реализует некоторую замену множественному наследованию, когда вместо того чтобы один класс имел несколько непосредственных суперклассов, этот класс наследует несколько интерфейсов. Интерфейс представляет собой класс, в котором все свойства – константы (т.е. статические – `static` и неизменяемые – `final`), а все методы абстрактные, т.е. интерфейс позволяет определить некоторый шаблон класса: описание свойств и методов без их реализации.

Язык Java разрешает несколько уровней наследования, определяемых непосредственным суперклассом и косвенными суперклассами. Наследование можно использовать для создания иерархии классов.

При создании подкласса на основе одного или нескольких суперклассов возможны следующие способы изменения поведения и структуры класса:

- расширение суперкласса путем добавления новых данных и методов;
- замена методов суперкласса путем их переопределения;
- слияние методов из суперклассов вызовом одноименных методов из соответствующих суперклассов.

Пример наследования приведен в листинге 1 и на рисунке 2.

Листинг 1

```
public class Person {//неявное наследование от класса Object  
public static final String GENDER_MALE = "MALE";  
public static final String GENDER_FEMALE = "FEMALE";  
public Person() {  
    //Делать больше нечего.....  
}  
private String name;  
private int age;  
private int height;  
private int weight;  
private String eyeColor;  
private String gender;  
// ...  
}
```

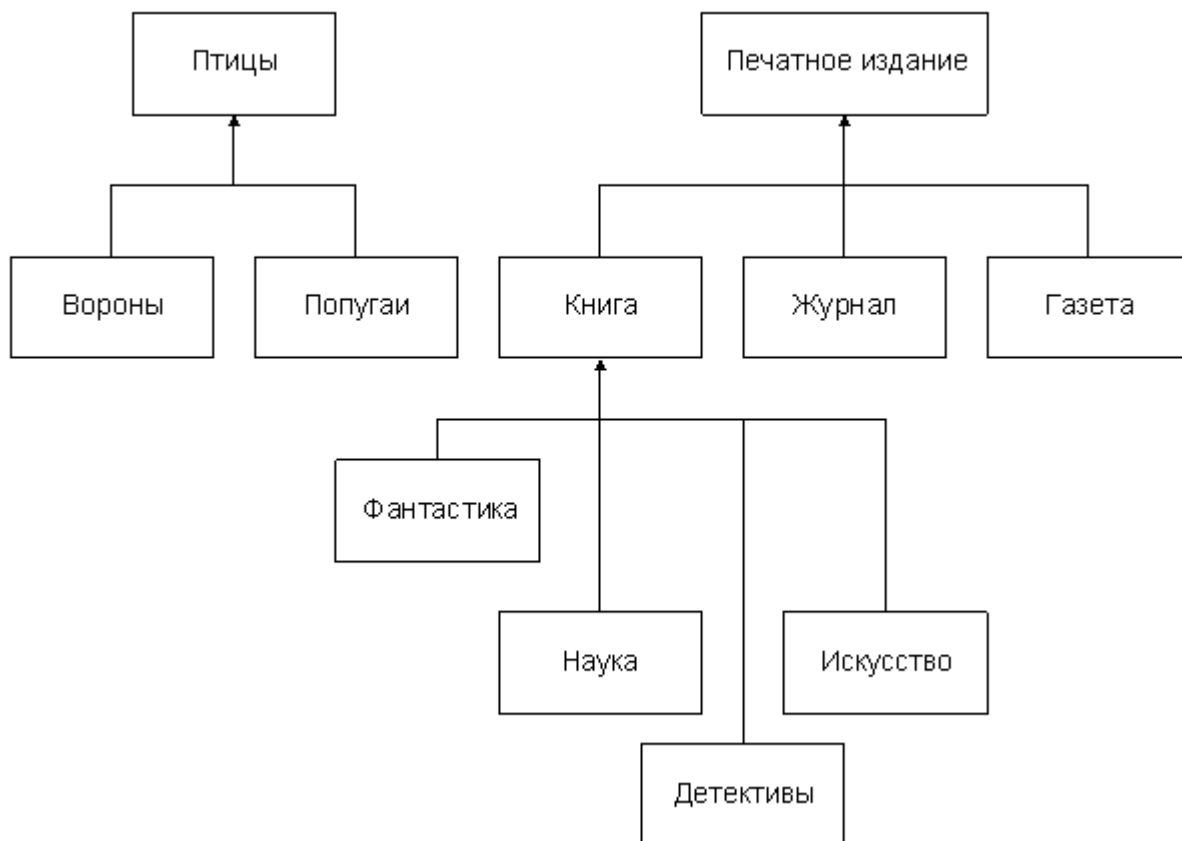


Рис. 2. Пример дерева наследования в ООП

Все классы в Java неявно наследуют свойства и поведения класса `Object`, т.е. он является предком всех классов, которые есть в JDK (Java Development Kit – комплект разработки на языке Java) и которые будут создаваться пользователем. Например: класс `Person` в листинге 1 неявно наследует свойства `Object`. Так как это предполагается для каждого класса, не нужно вводить фразу `extends Object` для каждого определяемого класса. Таким образом, класс `Person` имеет доступ к представленным переменным и методам своего суперкласса (`Object`). В данном случае `Person` может «видеть» и использовать общедоступные методы и переменные объекта `Object`, а также его защищенные методы и переменные.

4.1.2 Определение иерархии классов

Определим новый класс `Employee`, который наследует свойства `Person`. Его определение класса (или граф наследования) будет выглядеть следующим образом:

```

public class Employee extends Person {
  private String taxpayerIdentificationNumber;
  private String employeeNumber;
  private int salary;
  // ...
}
  
```

Граф наследования для `Employee` указывает на то, что `Employee` имеет доступ ко всем общедоступным и защищенным переменным и методам `Person` (так как он непосредственно расширяет его), а также `Object` (так как он фактически расширяет и этот класс, хотя и опосредованно).

Для углубления в иерархию классов еще на один шаг, можно создать третий класс, который расширяет Employee:

```
public class Manager extends Employee {  
// ...  
}
```

В языке Java любой класс может иметь не более одного суперкласса, но любое количество подклассов. Это самая важная особенность иерархии наследования языка Java, о которой надо помнить.

В Java все методы конструктора используют уточнения при переопределении методов по схеме сцепления конструкторов. В частности, выполнение конструктора начинается с обращения к конструктору суперкласса, которое может быть явным или неявным. Для явного обращения к конструктору используется оператор `super`, который указывает на вызов суперкласса (например, `super()` вызывает конструктор суперкласса без аргументов). Если же в теле конструктора явное обращение отсутствует, компилятор автоматически помещает в первой строке конструктора обращение к методу `super()`. То есть в языке Java в конструкторах используется уточнение при переопределении методов, а в обычных методах используется замещение (листинг 2).

Листинг 2

```
public class Person {  
  private String name;  
  public Person() {}  
    public Person(String name) {  
      this.name = name;  
    }  
  }  
public class Employee extends Person {  
  public Employee(String name) {  
    super(name);  
  }  
}
```

Ключевое слово `this`

Иногда в классе Java необходимо обратиться к текущему экземпляру данного класса, который в данный момент обрабатывается методом. Для такого обращения используется ключевое слово `this`. Применение этой конструкции удобно в случае необходимости обращения к полю текущего объекта, имя которого совпадает с именем переменной, описанной в данном блоке (листинг 2).

Переопределение методов.

Если например: в базовом и дочернем классах есть методы с одинаковым именем и одинаковыми параметрами. В таком случае уместно говорить о переопределении методов, т.е. в дочернем классе изменяется реализация уже существовавшего в базовом классе метода. Если пометить метод модификатором `final`, то метод не может быть переопределен. Поля нельзя переопределить, их можно только скрыть. Рассмотрим переопределение методов (листинг 3).

Листинг 3

```
public class Person {  
    private String FIO;  
    private int age;  
    public Person() {}  
    public String getFIO(){  
        return this.FIO;  
    }  
    public final setAge(int age){  
        this.age = age;  
    }  
}  
  
public class Employee extends Person {  
    public Employee(String name) {  
        super();  
    }  
    public String getFIO(){  
        return "My name is " + super.getFio();  
    }  
}
```

4.1.3 Наследование и абстракция

Если подкласс переопределяет метод из суперкласса, этот метод, по существу, скрыт, потому что вызов этого метода с помощью ссылки на подкласс вызывает версии метода подкласса, а не версию суперкласса. Это не означает, что метод суперкласса становится недоступным. Подкласс может вызвать метод суперкласса, добавив перед именем метода ключевое слово `super` (и в отличие от правил для конструкторов, это можно сделать в любой строке метода подкласса или даже совсем другого метода). По умолчанию Java-программа вызывает метод подкласса, если он вызывается с помощью ссылки на подкласс (листинг 5.3).

В контексте ООП абстрагирование означает обобщение данных и поведения до типа, более высокого по иерархии наследования, чем текущий класс. При перемещении переменных или методов из подкласса в суперкласс говорят, что эти члены абстрагируются. Основной причиной этого является возможность многократного использования общего кода путем продвижения его как можно выше по иерархии. Когда общий код собран в одном месте, облегчается его обслуживание.

Абстрактные классы и методы

Бывают моменты, когда нужно создавать классы, которые служат только как абстракции, и создавать их экземпляры никогда не придется. Такие классы называются абстрактными классами. К тому же бывают моменты, когда определенные методы должны быть реализованы по-разному для каждого подкласса, реализуемого суперклассом. Это абстрактные методы. Вот некоторые основные правила для абстрактных классов и методов:

- любой класс может быть объявлен абстрактным;
- абстрактные классы не допускают создания своих экземпляров;
- абстрактный метод не может содержать тела метода;

– класс, содержащий абстрактный метод, должен объявляться как абстрактный.

4.1.4 Использование абстрагирования

Например: необходимо запретить возможность создания экземпляра класса Employee напрямую. Для этого достаточно объявить его с помощью ключевого слова `abstract`:

```
public abstract class Employee extends Person {  
}
```

После этого при выполнении этого кода, вы получите ошибку компиляции:

```
public void someMethodSomewhere() {  
    Employee p = new Employee(); // ошибка компиляции!!  
}
```

Компилятор укажет на то, что Employee – это абстрактный класс, и его экземпляр не может быть создан.

Возможности абстрагирования

Например: Необходим метод для изучения состояния объекта Employee и проверки его правомерности. Это требование может показаться общим для всех объектов Employee, но между всеми потенциальными подклассами поведение будет существенно различаться, что дает нулевой потенциал для повторного использования. В этом случае вы объявляете метод `validate()` как абстрактный (заставляя все подклассы реализовывать его):

```
public abstract class Employee extends Person {  
    public abstract boolean validate();  
}
```

Теперь каждый прямой подкласс Employee (такой как Manager) должен реализовать метод `validate()`. При этом, как только подкласс реализовал метод `validate()`, ни одному из его подклассов не придется этого делать.

Листинг 4

```
public abstract class Employee extends Person {  
    public abstract boolean validate();  
}  
public class Manager extends Employee {  
    public boolean validate() { //метод для этого класса обязателен.  
        boolean flag = true;  
        // код вычисляющий состояние объекта  
        return flag;  
    }  
}  
public class Executive extends Manager {  
    //метод public boolean validate() можно пропустить  
}
```

4.1.5 Полиморфизм

Полиморфизм в языках программирования – это возможность объектов с одинаковой спецификацией иметь различную реализацию. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь раз-

личную реализацию – например, реализация класса может быть изменена в процессе наследования. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

В java существуют несколько способов организации полиморфизма:

1) **Полиморфизм интерфейсов.** Интерфейсы описывают методы, которые должны быть реализованы в классе, и типы параметров, которые должен получать и возвращать каждый член класса, но не содержат определенной реализации методов, оставляя это реализующему интерфейс классу (листинг 5). В этом и заключается полиморфизм интерфейсов. Несколько классов могут реализовывать один и тот же интерфейс, в то же время один класс может реализовывать один или больше интерфейсов. Интерфейсы находятся вне иерархии наследования классов, поэтому они исключают определение метода или набора методов из иерархии наследования.

Листинг 5

```
interface Shape {  
    void draw();  
    void erase();  
}  
  
class Circle implements Shape {  
    public void draw() { System.out.println("Circle.draw()");}  
    public void erase() { System.out.println("Circle.erase()");}  
}  
  
class Square implements Shape {  
    public void draw() { System.out.println("Square.draw()");}  
    public void erase() { System.out.println("Square.erase()");}  
}  
  
class Triangle implements Shape {  
    public void draw() { System.out.println("Triangle.draw()");}  
    public void erase() { System.out.println("Triangle.erase()");}  
}  
  
public class Shapes {  
    public static Shape randShape() {  
        switch((int)(Math.random() * 3)) {  
            default:  
            case 0: return new Circle();  
            case 1: return new Square();  
            case 2: return new Triangle();  
        }  
    }  
  
    public static void main(String[] args) {  
        Shape[] s = new Shape[9];  
        for(int i = 0; i < s.length; i++)  
            s[i] = randShape();  
        for(int i = 0; i < s.length; i++)  
            s[i].draw();  
    }  
}
```

2) **Полиморфизм наследования.** При наследовании класс получает все методы, свойства и события базового класса такими, какими они реализованы в базовом

классе. При необходимости в наследуемых классах можно определять дополнительные члены или переопределять члены, доставшиеся от базового класса, чтобы реализовать их иначе (листинг 6). Наследуемый класс также может реализовывать интерфейсы. В данном случае полиморфизм проявляется в том, что функционал базового класса присутствует в наследуемых классах неявно. Функционал может быть дополнен и переопределен. А наследуемые классы, несущие в себе этот функционал выступают в роли многих форм.

Листинг 6

```
class Shape {  
void draw() {}  
void erase() {}  
}  
class Circle extends Shape {  
void draw() { System.out.println("Circle.draw()");}  
void erase() { System.out.println("Circle.erase()");}  
}  
class Square extends Shape {  
void draw() { System.out.println("Square.draw()");}  
void erase() { System.out.println("Square.erase()");}  
}  
class Triangle extends Shape {  
void draw() { System.out.println("Triangle.draw()");}  
void erase() { System.out.println("Triangle.erase()");}  
}  
public class Shapes {  
    public static Shape randShape() {  
        switch((int)(Math.random() * 3)) {  
        default:  
        case 0: return new Circle();  
        case 1: return new Square();  
        case 2: return new Triangle();  
        }  
    }  
    public static void main(String[] args) {  
        Shape[] s = new Shape[9];  
        for(int i = 0; i < s.length; i++)  
            s[i] = randShape();  
        for(int i = 0; i < s.length; i++)  
            s[i].draw();  
    }  
}  
}
```

3) *Полиморфизм при помощи абстрактных классов.* Абстрактные классы поддерживают как наследование, так и возможности интерфейсов (листинг 5.4). При построении сложной иерархии, для обеспечения полиморфизма программисты часто вынуждены вводить методы в классы верхнего уровня при том, что эти методы ещё не определены. Абстрактный класс – это класс, экземпляр которого невозможно создать; этот класс может лишь служить базовым классом при наследовании. Нельзя

объявлять абстрактные конструкторы или абстрактные статические методы. Некоторые или все члены этого класса могут оставаться нереализованными, их реализацию должен обеспечить наследующий класс. Производные классы, которые не переопределяют все абстрактные методы, должны быть отмечены как абстрактные. Порожденный класс может реализовывать также дополнительные интерфейсы.

4) **Полиморфизм методов.** Способность классов поддерживать различные реализации методов с одинаковыми именами – один из способов реализации полиморфизма. Различные реализации методов с одинаковыми именами в Java называется перегрузкой методов (листинг 7). На практике часто приходится реализовывать один и тот же метод для различных типов данных. Право выбора специфической версии метода предоставлено компилятору.

Отдельным вариантом полиморфизма методов является полиморфизм методов с переменным числом аргументов, введенный в версии Java 2 5.0. Перегрузка методов здесь предусмотрена неявно, т.е. перегруженный метод может вызываться с разным числом аргументов, а в некоторых случаях даже без параметров.

Перегрузка методов как правило делается для тесно связанных по смыслу операций. Ответственность за построение перегруженных методов и выполнения ими однородных по смыслу операций лежит на разработчике.

Листинг 7

```
public class PrintStream extends FilterOutputStream
implements Appendable , Closeable {
    /*...*/
    public void print(boolean b) {
        write(b ? "true" : "false");
    }
    public void print(char c) {
        write(String.valueOf(c));
    }
    public void print(int i) {
        write(String.valueOf(i));
    }
    public void print(long l) {
        write(String.valueOf(l));
    }
    /*...*/
    public void write(int b) {
    }
}
```

5) **Полиморфизм через переопределение методов.** Если перегруженные методы с одинаковыми именами находятся в одном классе, списки параметров должны отличаться. Но если метод подкласса совпадает с методом суперкласса, то метод подкласса переопределяет метод суперкласса (листинг 3). Совпадать при этом должны и имена методов и типы входных параметров. В данном случае переопределение методов является основой концепции динамического связывания (или позднее связывание), реализующей полиморфизм. Суть динамической диспетчеризации методов состоит в том, что решение на вызов переопределенного метода принимается во время выполнения, а не во время компиляции. Однако final-методы не являются пе-

реопределяемыми, их вызов может быть организован во время компиляции и называется ранним связыванием.

Пример полиморфизма представлен в листинге 5.

4.2 Контрольный пример выполнения лабораторной работы

1) Создать класс “Компакт-диск” (MusicCD) со следующими свойствами:

- название альбома;
- имя исполнителя;
- год выпуска;
- базовая стоимость;
- длительность звучания.

Класс должен реализовывать следующие методы:

- toString() - вывод полной информации о компакт-диске;
- getPrice() - метод, реализующий расчет цены (для данного класса обычный getter).

2) Создать класс “Фильм” (MovieDVD) со следующими свойствами:

- название фильма;
- режиссер;
- год выпуска;
- жанр (1 – драма, 2 – комедия, 3 – сериал, 4 – боевик)
- страна производства;
- основные актеры;
- длительность;
- базовая стоимость.

Класс должен реализовывать следующие методы:

- toString() - вывод полной информации о фильме;
- getPrice() - метод, реализующий расчет цены в зависимости от жанра (для комедии – скидка 10%, для драмы – 15%, для сериала – 20%, для боевика полная стоимость).

3) Реализовать класс “Книга” (Book) со следующими свойствами:

- название;
- автор;
- издательство;
- год издания;
- количество страниц;
- базовая стоимость.

Класс Book должен реализовать следующие методы:

- toString() - вывод полной информации о книге;
- getPrice() - метод, реализующий расчет цены (для книг года издания до 2010 года – скидка 30 %).

4) Создать класс TestItems, содержащий метод main(), в котором создаются экземпляры всех этих классов и выводится информация о них в отформатированном виде.

Пример:

Book

Author: Bruce Eckel
Title: Thinking in Java
Year: 2006
Publisher: Printice Hall
Pages: 510
Price: \$16 (50% discount)

5) Спроектировать базовый (родительский) класс Item, включающий в себя все общие свойства классов MusicCD, MovieDVD и Book.

Классы MusicCD, MovieDVD и Book унаследовать от класса Item.

Перенести общее поведение классов в класс Item, оставив при этом частное поведение в наследниках.

6) Изменить метод main() класса TestItems, таким образом, чтобы все объекты-наследники класса Item хранились в векторе. Вывод информации о товарах, осуществить посредством прохода по элементам массива.

4.3 Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

- 1 Теоретические сведения о наследовании и полиморфизме
- 2 Реализацию наследования и полиморфизма на языке Java в соответствии с индивидуальным заданием.
- 3 Диаграмму UML, отражающую созданные классы и типы отношений между ними.
- 4 Распечатанный листинг выполнения программы, отражающий все этапы ее выполнения.
- 5 Выводы о выполненной работе (включающие описание основных преимуществ использования наследования и полиморфизма).

4.4 Контрольные вопросы

- 1 Особенности абстрактного класса.
- 2 Какой класс является абстрактным? При каком условии он обязан быть объявлен абстрактным?
- 3 Дайте определение «Наследованию» в рамках ООП.
- 4 Дайте определение «Полиморфизму» в рамках ООП.
- 5 Назовите основные способы организации «Полиморфизма».
- 6 Перечислите основные преимущества «Полиморфизму» в рамках ООП

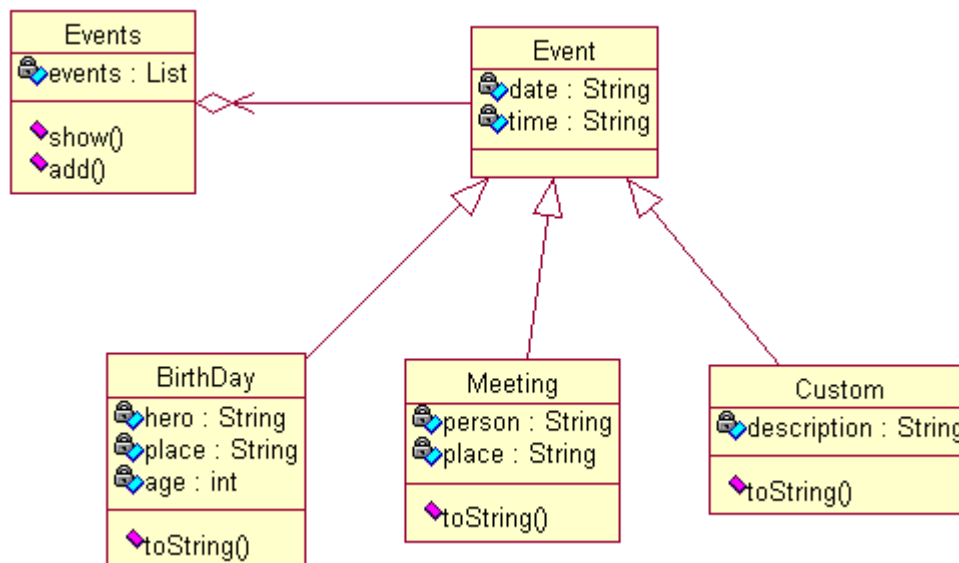
4.5 Варианты заданий для самостоятельной работы

Реализовать предметную область:

- 1 Записная книжка. Создать родительский класс «Событие» (дата, время) и дочерние классы:
 - «День рождения» (именинник, место проведения праздника и возраст);
 - «Встреча» (человек с которым назначена встреча и место встречи);

- «Другое» (описание).

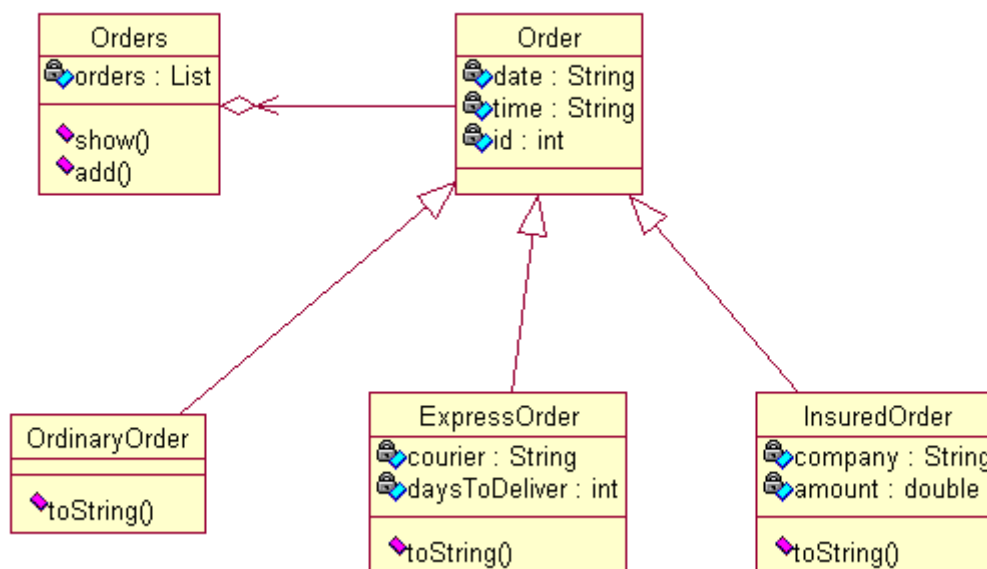
Реализовать класс для хранения списка событий с методом добавления события и методом печати списка событий.



2 Система управления доставкой товара. Создать родительский класс «Заказ» (дата, время, идентификатор) и дочерние классы:

- «Обычный заказ»;
- «Срочный заказ» (курьер, дата доставки);
- «Застрахованный заказ» (компания страхования, сумма).

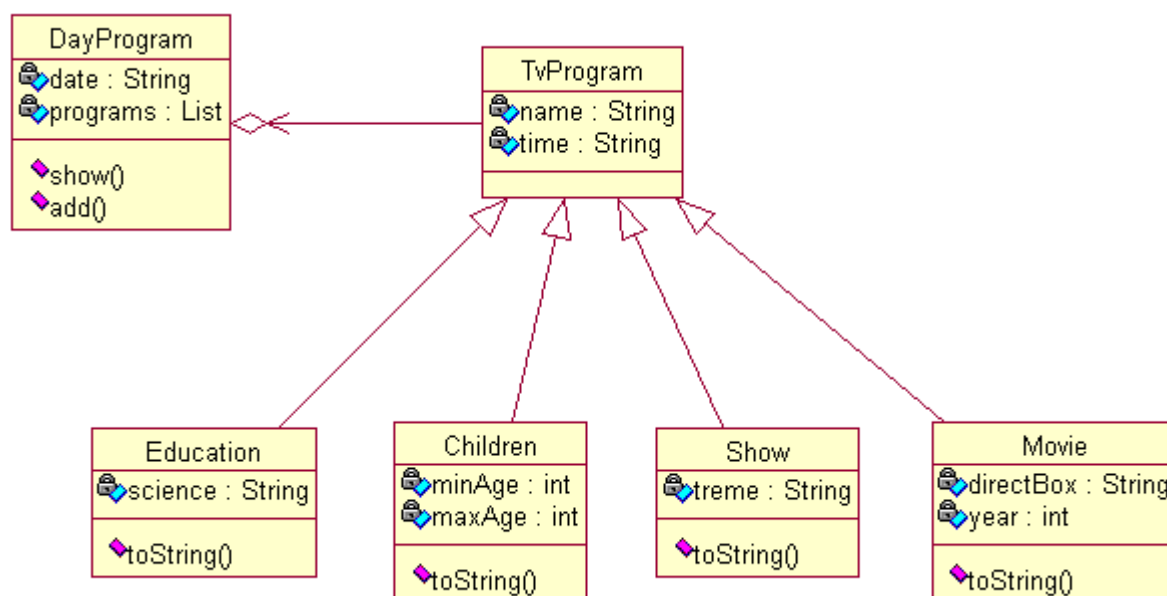
Реализовать класс для хранения списка заказов с методом добавления заказа и методом печати списка заказов.



3 Телепрограмма. Создать родительский класс «Телепрограмма» (наименование передачи, время) и дочерние классы:

- «Образовательная передача» (наименование области науки);
- «Передача для детская» (мин возраст, мах возраст);
- «Шоу» (тема);
- «Фильм» (описание, год).

Реализовать класс для хранения списка телепрограмм с методом добавления телепрограммы и методом печати списка телепрограмм.



4 Гостиница. Создать родительский класс «Комната» (идентификатор, номер, количество человек, цена) и дочерние классы:

- «Стандартная комната»;
- «Комната полулюкс»;
- «Комната люкс» (мин срок сдачи, мах срок сдачи).

Реализовать класс для хранения списка номеров с методом добавления номера и методом печати списка номеров.

5 Реализация готовой продукции. Создать родительский класс «Товар» (идентификатор, код, наименование, цена, описание) и дочерние классы:

- «Хрупкий товар» (коэффициент хрупкости);
- «Скоропортящийся товар» (маж время хранения);
- «Габаритный товар» (высота, ширина, длина).

Реализовать класс для хранения списка товаров с методом добавления нового товара и методом печати списка товаров.

Вариант 6). Учет успеваемости студентов ВУЗА. Создать родительский класс «Студент» (номер зачетной книжки, фамилия, имя, группа, кафедра, дисциплина, оценка, фамилия преподавателя) и дочерние классы:

- «Очное отделение» (бал ЭГЕ, средний бал аттестата);
- «Заочное отделение» (место работы, должность, сумма обучения);
- «Целевое обучение» (название целевого предприятия, сумма обучения).

Реализовать класс для хранения списка студентов с методом добавления нового студента и методом печати списка студентов.

Вариант 7). Автоматизация работы факультета. Создать родительский класс «Факультет» (аудитория, корпус, контактный телефон, фамилия декана) и дочерние классы:

- «Состав факультета» (ФИО, должность);

- «Кафедры» (название, направление подготовки);
- «Преподаватели кафедры» (ФИО преподавателя, кафедра, должность, стаж работы, читаемые дисциплины).

Реализовать класс для хранения списка факультетов с методом добавления нового факультета и его кафедр и преподавателей и методом печати списка факультетов с полной информацией о его составе.

Вариант 8). Продажа товаров супермаркета. Создать родительский класс «Супермаркет» (название отдела, название товара, страна-производитель, розничная цена, поставщик) и дочерние классы:

- «Игрушки» (возрастная группа, тип);
- «Фрукты» (макс. время хранения, температура хранения);
- «Габаритный товар» (высота, ширина, длина).

Реализовать класс для хранения списка товаров с методом добавления нового товара и методом печати списка товаров.

Вариант 9). Учет военного состава. Создать родительский класс «Военный состав» (фамилия, рота, звание, дата рождения, дата поступления на службу, часть) и дочерние классы:

- «Органы военного управления» (название округа, должность, выслуга лет, сумма надбавки);
- «Военная служба по контракту» (период договора, дата договора, номер протокола, сумма зарплаты);
- «Награжденные» (название награды, премия, сумма надбавки).

Реализовать класс для хранения списка военных с методом добавления нового военного и методом печати списка военных.

Вариант 10). Учет литературы. Создать родительский класс «Литература» (код источника литературы, тип литературы, название, год издательства, название издательства, количество страниц, автор) и дочерние классы:

- «научно-техническая литература» (область науки, количество экземпляров);
- «периодика» (вид периодики, период издательства);
- «справочники» (направление, том, часть).

Реализовать класс для хранения списка литературы с методом добавления нового источника и методом печати списка литературы.

Вариант 11). Учет продажи путевок. Создать родительский класс «Путевки» (код путевки, фамилия клиента, название пансионата, номер, вид жилья, дата заезда, дата выезда, количество человек, цена) и дочерние классы:

- «Зарубежные путевки» (загран паспорт, страховка);
- «Санатории» (мед.полис, диагноз, направление);
- «Детские оздоровительные» (возраст ребенка, свидетельство о рождении, пол).

Реализовать класс для хранения списка путевок с методом добавления путевки и методом печати списка путевок.

Вариант 12). Учет выполненных работ станции техобслуживания. Создать родительский класс «ТехОбслуживание» (название станции, адрес станции, название автотранспорта на ремонте, вид ремонта, дата поступления, дата выдачи, результат ремонта, фамилия персонала, сумма ремонта) и дочерние классы:

- «планово-предупредительный осмотр для легкового транспорта» (вид (плановый/капитальный), год проведения, пробег, период);
- «неисправности» (название неисправности, описание выполненных работ);
- «планово-предупредительный осмотр для грузового транспорта» (вид (ТО-1, ТО -2, ТО-3), год проведения, пробег, период, объем двигателя).

Реализовать класс для хранения списка выполненных работ с методом добавления ремонта и методом печати списка ремонтов.

Вариант 13). Медицинское обслуживание пациентов. Создать родительский класс «МедОбслуживание» (название поликлиники, адрес поликлиники, фамилия пациента, номер полиса, дата осмотра, фамилия врача, должность врача, диагноз) и дочерние классы:

- «планово-предупредительный осмотр» (вид (амбулаторный/стационарный), год проведения, период действия, результат);
- «вакцинация» (название вакцины, дата вакцинации, период действия);
- «мед.обслуживание детей и подростков» (свидетельство о рождении, пол, возраст ребенка).

Реализовать класс для хранения списка медицинского обслуживания пациентов с методом добавления и методом печати списка.

Вариант 14). Библиотека. Создать родительский класс «Библиотека» (название библиотеки, адрес, город, ФИО директора) и дочерние классы:

- «читательский зал» (название зала, количество источников литературы, этаж, кабинет);
- «читатели» (фамилия, имя, отчество, место работы, возраст, пол).
- «выдача литературы» (название читательского зала, фамилия читателя, название литературы, дата выдачи, срок выдачи, сумма залога)

Реализовать класс для хранения списка литературы с методом добавления и методом печати списка.

Вариант 15). Продажа автомобилей. Создать родительский класс «Автомобили» (марка автомобиля, год выпуска, цена автомобиля, комплектация, страна производитель, дата продажи, ФИО покупателя) и дочерние классы:

- «Поддержанные авто» (степень сохранности, ФИО владельца, пробег);
- «Спортивные» (кол-во секунд до набора скорости, объем двигателя, мощность);
- «Спецтехника» (вид (строительная, грузовая, дорожная и т.д.), масса, габаритные размеры).

Реализовать класс для хранения списка проданных автомобилей с методом добавления нового автомобиля и методом печати списка автомобилей.

Вариант 16). Учет продаж через Интернет-магазин. Создать родительский класс «Интернет-магазин» (Название магазина, Название товара, Страна производи-

тель, Вид оплаты, Сумма покупки, Дата продажи, ФИО покупателя) и дочерние классы:

- «Мебель для гостиных» (название, цена, тип мебели, производитель);
- «Мебель для кухни» (название, цена, длина, высота, ширина, материал);
- «Мебель для ванн» (название, цена).

Реализовать класс для хранения списка товаров с методом добавления нового товара и методом печати списка товаров.

Вариант 17). Создать родительский класс «Больница» (название больницы, заведующий, город, адрес) и дочерние классы:

- «отделения» (название отделения, корпус, этаж, ФИО заведующего);
- «пациенты» (название отделения, ФИО пациента, номер полиса, дата поступления, дата выписки, диагноз, дата проведения операции, название операции, стоимость лечения);
- «врачи» (название отделения, ФИО врача, должность, научное звание, стаж работы).

Реализовать класс для хранения списка больниц, с отделениями, пациентами и врачами, с методом добавления и методом печати списка.

Вариант 18). Создать родительский класс «Военно-морские учения» (название военной части, дата проведения учения, количество личного состава, место проведения учений, результат учений) и дочерние классы:

- «корабли» (название корабля, тип корабля, наработка корабля, название военной части);
- «личный состав» (ФИО военного, звание, стаж, должность, название военной части);
- «место учений» (название, область, район).

Реализовать класс для хранения списка военно-морских учений, с методом добавления и методом печати списка.

Вариант 19). Учет оформления туристических туров. Создать родительский класс «Туристические туры» (название тура, тип питания, цена тура, дата выезда) и дочерние классы:

- «Зарубежные туры» (страны, города, тип передвижения, тип проживания, длительность);
- «Походные туры» (название экскурсии, длительность, тип проживания);
- «Физкультурно-оздоровительные» (спортивное мероприятие, город, тип участия, тип проживания).

Реализовать класс для хранения списка туристических туров с методом добавления тура и методом печати списка туров.

Вариант 20). Учет представлений цирка. Создать родительский класс «Цирк» (название представления, город, дата премьеры, период проведения, цена билета, автор, жанр, количество актеров) и дочерние классы:

– «акробатические представления» (тип акробатики (силовая акробатика, парная силовая, групповая силовая, пластическая, одинарная и т.д.), количество актеров, инвентарь);

– «дрессировка» (вид дрессировки (крупные животные, мелкие животные, птицы, хищники), количество животных, количество актеров);

– «фокусы» (вид фокусов (иллюзия, манипуляция), количество актеров, инвентарь).

Реализовать класс для хранения списка представлений с методом добавления нового представления и методом печати списка представлений.

Вариант 21). Учет продажи лекарств в аптеке. Создать родительский класс «Аптека» (название аптеки, адрес, город, ФИО директора) и дочерние классы:

– «клиенты аптеки» (название аптеки, фамилия, имя, отчество клиента, процент скидки);

– «лекарственный фонд аптеки» (название аптеки, название лекарства, тип лекарства, цена лекарства, страна-производитель);

– «продажи» (название аптеки, название лекарства, цена лекарства, ФИО клиента, количество, сумма к оплате).

Реализовать класс для хранения списка лекарств с методом добавления нового лекарства и методом печати списка лекарств.

Вариант 22). Учет абонентской платы. Создать родительский класс «Абонентская плата» (Фамилия, имя, отчество абонента, адрес, город, паспортные данные) и дочерние классы:

– «международные звонки» (страна, город, длительность в минутах, цена, сумма оплаты, месяц, год);

– «городские звонки» (тариф, длительность в минутах, сумма оплаты, месяц, год);

– «подключение к Интернет» (название подключения, предоставленный объем, сумма оплаты, месяц, год).

Реализовать класс для хранения списка абонентской оплаты с методом добавления новой оплаты и методом печати всего списка.

Вариант 23). Создать родительский класс «Стоматология» (название стоматологии, адрес стоматологии, город, количество врачей) и дочерние классы:

– «отделения» (название отделения, этаж, ФИО заведующего);

– «врачи» (название отделения, ФИО врача, должность, научное звание, стаж работы).

– «история болезни пациентов» (ФИО пациента, номер полиса, дата лечения, оказанные услуги, сумма оплаты, ФИО врача)

Реализовать класс для хранения списка больниц, с отделениями, пациентами и врачами, с методом добавления и методом печати списка.

Вариант 24). Учет выполненных работ техобслуживания подвижного состава. Создать родительский класс «ТехДепо» (название депо, адрес депо, город, количество сотрудников) и дочерние классы:

- «планово-предупредительный осмотр подвижного состава» (тип ремонта (ТО-1, ТО -2, ТО-3), год проведения, месяц, номер вагона на ремонте, результат ремонта);
- «текущий ремонт» (тип ремонта (ТР-1, ТР-2, ТР-3), номер локомотива, название неисправностей, описание выполненных работ, сумма ремонта);
- «капитальный ремонт» (год проведения, месяц, номер вагона на ремонте, сумма ремонта).

Реализовать класс для хранения списка выполненных работ с методом добавления ремонта и методом печати списка ремонтов.

Вариант 25). Автоматизация работы кафедры. Создать родительский класс «Кафедра» (аудитория, корпус, контактный телефон, фамилия декана, количество преподавателей) и дочерние классы:

- «Студенческие группы» (название группы, год поступления, кафедра);
- «Студенты кафедры» (кафедра, группа, ФИО студента, пол, адрес);
- «Преподаватели кафедры» (кафедра, ФИО преподавателя, должность, стаж работы, читаемые дисциплины).

Реализовать класс для хранения списка кафедр с методом добавления новой кафедры и ее групп, студентов и преподавателей и методом печати списка кафедр с полной информацией о ее составе.

ЛАБОРАТОРНАЯ РАБОТА №5. КОЛЛЕКЦИИ

Цель работы. Получить практические навыки работы с коллекциями в Java.

5.1 Методические рекомендации

5.1.1. Коллекции объектов Java

Коллекции объектов – это очень мощный и исключительно полезный механизм. Простейшей коллекцией является массив. Но массив имеет ряд недостатков. Один из самых существенных – размер массива фиксируется до начала его использования. Т.е. необходимо заранее знать или подсчитать, сколько потребуется элементов коллекции до начала работы с ней. Зачастую это неудобно, а в некоторых случаях – невозможно.

Поэтому все современные базовые библиотеки различных языков программирования имеют тот или иной вариант поддержки коллекций объектов. Коллекции обладают одним важным свойством – их размер не ограничен. Выделение необходимых для коллекции ресурсов спрятано внутри соответствующего класса. Работа с коллекциями облегчает и упрощает разработку приложений. Отсутствие же подобного механизма в составе средств разработки вызывает серьезные проблемы, которые приводят либо к различным ограничениям в реализации либо к самостоятельной разработке адекватных средств для хранения и обработки массивов информации заранее неопределенного размера.

В Java средства работы с коллекциями весьма развиты и имеют много полезных особенностей.

Одним из широко используемых классов коллекций является **ArrayList**. Пример использования этой коллекции приводится ниже (Листинг 6.1).

Листинг 6.1

```
import java.util.*;
import java.io.*;
public class ArrayListTest {
    ArrayList lst = new ArrayList();
    Random generator = new Random();
    void addRandom() {
        lst.add(new Integer(generator.nextInt()));
    }
    public String toString() {
        return lst.toString();
    }
    public static void main(String args[]) {
        ArrayListTest tst = new ArrayListTest();
        for(int i = 0; i < 100; i++ )
            tst.addRandom();
        System.out.println("Сто случайных чисел: "+tst.toString());
    }
}
```

Рассмотрим данный пример подробнее. Здесь, кроме класса **ArrayList**, использованы еще ряд классов библиотеки Java.

– **Random** – класс из `java.util`. Расширяет возможности класса **Math** по генерации случайных чисел.

– **Integer** – так называемый **wrapper-класс (класс-обертка)** для целых (`int`). Он использован потому, что в коллекцию нельзя занести данные элементарных типов, а только объекты классов.

Класс **ArrayListTest** имеет два поля – поле `lst` класса **ArrayList** и поле **generator** класса **Random**, используемое для генерации случайных чисел.

Метод **addRandom()** генерирует и заносит в коллекцию очередное случайное число. Метод **toString()** просто обращается к методу **toString()** класса **ArrayList**, который обеспечивает формирование представления списка в виде строки.

Метод **main(...)** создает объект класса **ArrayListTest** и организует цикл заполнения коллекции 100 случайными числами. После этого он печатает результат.

Этот пример демонстрирует технику использования коллекций. Из него видно, что **добавить элемент в коллекцию можно методом `add(...)` класса **ArrayList**** и при этом мы нигде не указываем размер коллекции.

В Java коллекции объектов разбиты на три больших категории:

- **List** (список),
- **Set** (множество),
- **Map** (отображение).

List – это список объектов. Объекты можно **добавлять** в список (метод **add()**), **заменять** в списке (метод **set()**), **удалять** из списка (метод **remove()**), **извлекать** (метод **get()**). Существует также возможность организации прохода по списку при помощи итератора.

Set – множество объектов. Те же возможности, что и у **List**, но объект может входить в множество только один раз. Т.е. двойное добавление одного и того же объекта в множество не изменяет само множество.

Map – отображение или ассоциативный массив. В **Map** мы добавляем не отдельные объекты, а пары объектов (ключ, значение). Соответственно есть операции поиска значения по ключу. Добавление пары с уже существующим в **Map** ключом приводит к замене, а не к добавлению. Из отображения (**Map**) можно получить множество (**Set**) ключей и список (**List**) значений.

Коллекции – это наборы произвольных объектов.

Общим свойством коллекций является то, что они работают с **Object** (базовый класс для всех классов Java). Это означает, что мы можем добавлять в коллекцию любые объекты Java, т.к. **Object** является базовым классом для всех классов Java. При извлечении из коллекции мы тоже получаем **Object**. Зачастую это приводит к необходимости преобразования полученного из коллекции объекта к нужному классу.

Это очень удобно для реализации самих коллекций, но не всегда хорошо при программировании. Если мы составляем коллекцию из объектов класса **A** и в результате ошибки поместим туда объект класса **B**, то при выполнении программы мы получим **ClassCastException** при попытке преобразования извлеченного из коллекции элемента к классу **A**.

Идеальный вариант это, во-первых, получить сообщение об ошибке в точке ее возникновения, а не в точке ее проявления, а, во-вторых, во время трансляции программы, а не при ее выполнении.

5.1.2 Итераторы

В коллекциях широко используются итераторы. В Java **итератор** – это вспомогательный объект, используемый для прохода по коллекции объектов. Как и сами коллекции, **итераторы базируются на интерфейсе**. Это **интерфейс *Iterator***, определенный в пакете **java.util**. Т.е. любой итератор, как бы он не был устроен, имеет следующие три метода:

- **boolean hasNext()** – проверяет есть ли еще элементы в коллекции;
- **Object next()** – выдает очередной элемент коллекции;
- **void remove()** – удаляет последний выбранный элемент из коллекции.

Кроме **Iterator** есть еще **ListIterator** – это расширенный вариант итератора с доп. возможностями и **Enumerator** – это устаревший вариант, оставленный для совместимости с предыдущими версиями.

В свою очередь **интерфейс *Collection*** имеет **метод *Iterator iterator()***;

Это обязывает все классы коллекций создавать поддержку итераторов (обычно реализованы с использованием inner-классов, удовлетворяющих интерфейсу **Iterator**).

Отредактируем листинг 6.1 с использованием интерфейса *Iterator* (Листинг 6.2)

Листинг 6.2

```
public String toString() {  
    String res = "";  
    Iterator iter = lst.iterator();  
    for(int i = 0; iter.hasNext(); i++) {  
        if( i%6 == 0 )  
            res += "\n";  
        res += " " + iter.next().toString(); // !!!  
    }  
    return res;  
}
```

Метод **remove()**

В интерфейсе **Iterator** определен метод **remove()**. Он позволяет удалить из коллекции последний извлеченный из нее объект. В то же время, в интерфейсе **Collection** (а это значит – во всех классах-коллекциях) есть методы **remove(Object o)** и **removeAll()**.

Здесь нет двойственности возможностей. Дело в том, что удаление из коллекции объектов и вообще модификация коллекции параллельно с проходом по коллекции через итератор запрещено (выдается **ConcurrentModificationException**). Единственное, что разрешено – это удалить объект методом **remove()** из **Iterator** (причем, только один - последний извлеченный).

5.1.3 Классы реализации коллекций

Рассмотрим конкретные реализации коллекций, т.е. классы, которые обеспечивают описанную выше функциональность.

Коллекции-списки (List)

Реализованы в следующий трех вариантах **ArrayList**, **LinkedList**, **Vector**.

Класс *Vector* – это устаревший вариант, оставлен для совместимости с предыдущими версиями.

Класс *ArrayList* – используется чаще всего. Имеет два конструктора:

- **public ArrayList()** // Конструктор пустого списка
- **public ArrayList(Collection c)** // Строит список из любой коллекции

А также следующие методы:

- **public int size()** //Возвращает количество элементов списка;
- **public boolean isEmpty()** //Проверяет, что список пуст;
- **public boolean contains(Object elem)** //Проверяет, принадлежит ли заданный объект списку;
- **public int indexOf(Object elem)** //Ищет первое вхождение заданного объекта в список и возвращает его индекс. Возвращает -1, если объект не принадлежит списку.
- **public int lastIndexOf(Object elem)** //То же самое, но ищет последнее вхождение;
- **public Object clone()** //Создает "поверхностную" копию списка;
- **public Object[] toArray()** //Преобразует список в массив;
- **public Object get(int index)** //Возвращает элемент коллекции с заданным номером;
- **public Object set(int index, Object element)** //Заменяет элемент с заданным номером;
- **public boolean add(Object o)** //Добавляет заданный объект в конец списка;
- **public void add(int index, Object element)** //Вставляет элемент в указанную позицию списка;
- **public Object remove(int index)** //Удаляет заданный элемент списка;
- **public void clear()** //Полностью очищает список;
- **public boolean addAll(Collection c)** //Добавляет к списку (в конец) все элементы заданной коллекции;
- **public boolean addAll(int index, Collection c)** //Вставляет в список с указанной позиции все элементы коллекции;
- **protected void removeRange(int fromIndex, int toIndex)** //Удаляет из коллекции объекты в заданном интервале индексов (исключая toIndex);

*Класс **LinkedList***

LinkedList имеет практически ту же функциональность, что и **ArrayList**, и отличается от него только **способом реализации** и, как следствие, эффективностью выполнения тех или иных операций. Так добавление в **LinkedList** осуществляется в среднем быстрее, чем в **ArrayList**, последовательный проход по списку практически столь же эффективен, как у **ArrayList**, а произвольное извлечение из списка медленнее, чем у **ArrayList**.

1 Сортировка и упорядочивание. Интерфейсы Comparable и Comparator

Начиная с версии 1.5, в Java появились два интерфейса **java.lang.Comparable** и **java.util.Comparator**. Объекты классов, реализующие один из этих интерфейсов, могут быть упорядоченными.

*Интерфейс **Comparable***

В интерфейсе **Comparable** объявлен всего один метод **compareTo(Object obj)**, предназначенный для реализации **упорядочивания объектов класса**. Его удобно использовать при сортировке упорядоченных списков или массивов объектов.

Данный метод **сравнивает** вызываемый объект с **obj**. В отличие от метода **equals**, который возвращает true или false, **compareTo** возвращает:

- 0, если значения равны;
- Отрицательное значение, если вызываемый объект меньше параметра;
- Положительное значение, если вызываемый объект больше параметра.

Метод может выбросить исключение **ClassCastException**, если типы объектов не совместимы при сравнении.

Аргумент метода **compareTo** имеет тип сравниваемого объекта класса.

Классы **Byte**, **Short**, **Integer**, **Long**, **Double**, **Float**, **Character**, **String** уже реализуют интерфейс **Comparable**.

В листинге 6.3 приведен пример реализующий интерфейс.

Листинг 6.3

```
import java.util.TreeSet;
class Comp implements Comparable {
    String str;
    int number;
    Comp(String str, int number) {
        this.str = str;
        this.number = number;
    }
    @Override
    public int compareTo(Object obj) {
        Comp entry = (Comp) obj;
        int result = str.compareTo(entry.str);
        if(result != 0) {
            return result;
        }
        result = number - entry.number;
        if(result != 0) {
            return (int) result / Math.abs( result );
        }
        return 0;
    }
}
public class Example {
    public static void main(String[] args) {
        TreeSet<Comp> ex = new TreeSet<Comp>();
        ex.add(new Comp("Stive Global", 121));
        ex.add(new Comp("Stive Global", 221));
        ex.add(new Comp("Nancy Summer", 3213));
        ex.add(new Comp("Aaron Eagle", 3123));
        ex.add(new Comp("Barbara Smith", 88786));
        for(Comp e : ex) {
```

```

        System.out.println("Str: " + e.str + ", number: " + e.number);
    }
}
}
/*результат:
* Str: Aaron Eagle, number: 3123
* Str: Barbara Smith, number: 88786
* Str: Nancy Summer, number: 3213
* Str: Stive Global, number: 121
* Str: Stive Global, number: 221
*/

```

В данном примере, значения сортируются сначала по полю str, а затем по number. Это хорошо видно по двум последним строкам результата.

Для того чтобы сделать **сортировку в обратном порядке**, необходимо внести некоторые изменения в метод compareTo:

```

@Override
public int compareTo(Object obj) {
    Comp entry = (Comp) obj;
    int result = entry.str.compareTo(str); // значения меняются местами
    if(result != 0) {
        return result;
    }
    result = entry.number - number; // значения меняются местами
    if(result != 0) {
        return (int) result / Math.abs( result );
    }
    return 0;
}
/* результат:
* Str: Stive Global, number: 221
* Str: Stive Global, number: 121
* Str: Nancy Summer, number: 3213
* Str: Barbara Smith, number: 88786
* Str: Aaron Eagle, number: 3123
*/

```

Как видно данные отсортированы в обратном порядке, сначала по имени, а потом по цифрам.

Интерфейс Comparator

В интерфейсе Comparator объявлено два метода **compare(Object obj1, Object obj2)** и **equals(Object obj)**.

Метод **compare(Object obj1, Object obj2)** — так же, как и метод **compareTo** интерфейса Comparable, **упорядочивает объекты класса**. Точно так же на выходе получает 0, положительное значение и отрицательное значение.

Метод может выбросить исключение ClassCastException, если типы объектов не совместимы при сравнении.

Основным отличием **интерфейса Comparator** от **Comparable** является то, что вы можете создавать **несколько видов независимых сортировок**.

В листинге 6.4 приведен пример реализующий интерфейс.

Листинг 6.4

```
public class Product {
    private String name;
    private double price;
    private int quantity;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public int getQuantity() {
        return quantity;
    }
}

// теперь реализуем интерфейс Comparator, для сортировки по названию
class SortedByName implements Comparator<Product> {
    public int compare(Product obj1, Product obj2) {
        String str1 = ((Product) obj1).getName();
        String str2 = ((Product) obj2).getName();
        return str1.compareTo(str2);
    }
}

// а так же реализуем интерфейс Comparator, для сортировки по цене
class SortedByPrice implements Comparator<Product> {
    public int compare(Product obj1, Product obj2) {
        double price1 = ((Product) obj1).getPrice();
        double price2 = ((Product) obj2).getPrice();
        if(price1 > price2) {
            return 1;
        }
        else if(price1 < price2) {
            return -1;
        }
        else {
            return 0;
        }
    }
}
```

```

    }
}
// ну и собственно работа с нашими данными
public class ExampleProduct {
    public static void main(String[] args) {
        Product[] p = new Product[3];
        // заполним объект Product содержимым
        p[0] = new Product();
        p[0].setName("Milk");
        p[0].setPrice(7.56);
        p[0].setQuantity(56);

        p[1] = new Product();
        p[1].setName("Coffee");
        p[1].setPrice(17.00);
        p[1].setQuantity(32);

        p[2] = new Product();
        p[2].setName("Tea");
        p[2].setPrice(12.50);
        p[2].setQuantity(0);

        // выведем данные без сортировки
        System.out.println("==== no sorted =====");
        for(Product i : p) {
            System.out.println("Name: " + i.getName() +
                " price: " + i.getPrice() + " quantity: " + i.getQuantity());
        }
        // отсортируем и выведем данные по цене
        System.out.println("====sorted by price=====");
        Arrays.sort(p, new SortedByPrice());
        for(Product i : p) {
            System.out.println("Name: " + i.getName() +
                " price: " + i.getPrice() + " quantity: " + i.getQuantity());
        }
        // отсортируем и выведем данные по названию
        System.out.println("====sorted by name =====");
        Arrays.sort(p, new SortedByName());
        for(Product i : p) {
            System.out.println("Name: " + i.getName() +
                " price: " + i.getPrice() + " quantity: " + i.getQuantity());
        }
    }
}
/* результат:
===== no sorted =====
Name: Milk price: 7.56 quantity: 56
Name: Coffee price: 17.0 quantity: 32

```

Name: Tea price: 12.5 quantity: 0
===== sorted by price=====
Name: Milk price: 7.56 quantity: 56
Name: Tea price: 12.5 quantity: 0
Name: Coffee price: 17.0 quantity: 32
===== sorted by name =====
Name: Coffee price: 17.0 quantity: 32
Name: Milk price: 7.56 quantity: 56
Name: Tea price: 12.5 quantity: 0
 */

5.2. Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

1. Теоретические сведения о коллекциях.
2. Диаграмму UML, отражающую созданные классы и типы отношений между ними.
3. Распечатанный листинг выполнения программы, отражающий все этапы ее выполнения.
4. Выводы о выполненной работе.

5.3. Варианты заданий для самостоятельной работы

Вариант 1). Записная книжка контактов. Реализовать сортировку по дате и по времени.

Вариант 2). Система управления доставкой товара. Реализовать сортировку по дате и по времени.

Вариант 3). Телепрограмма. Реализовать сортировку по наименованию передачи и по времени.

Вариант 4). Гостиница. Реализовать сортировку по цене и по количеству мест (человек).

Вариант 5). Реализация готовой продукции. Реализовать сортировку по наименованию и по цене.

Вариант 6). Успеваемость студентов ВУЗА. Реализовать сортировку по фамилиям студентов и по балу ЭГЕ.

Вариант 7). Факультеты. Реализовать сортировку по наименованию факультетов и по номерам корпусов.

Вариант 8). Супермаркеты. Реализовать сортировку по цене и по стране.

Вариант 9). Военный состав. Реализовать сортировку по фамилиям и по зарплатам.

Вариант 10). Учет литературы. Реализовать сортировку по наименованию и по году издательства.

Вариант 11). Учет продажи путевок. Реализовать сортировку по дате и по названиям пансионатов.

Вариант 12). Учет выполненных работ станции техобслуживания. Реализовать сортировку по виду ремонта и по сумме ремонта.

Вариант 13). Медицинское обслуживание пациентов. Реализовать сортировку по наименованию поликлиник и по дате осмотра.

Вариант 14). Библиотека. Реализовать сортировку по фамилиям читателей и по количеству источников литературы в залах.

Вариант 15). Продажа автомобилей. Реализовать сортировку по марке автомобилей и по цене.

Вариант 16). Интернет-магазин. Реализовать сортировку по названию товара и по дате продажи.

Вариант 17). Больница. Реализовать сортировку по названиям отделений и по стоимости лечения.

Вариант 18). Военно-морские учения. Реализовать сортировку по наименованию и по количеству личного состава.

Вариант 19). Туристические туры. Реализовать сортировку по названиям туров и по цене.

Вариант 20). Представления цирка. Реализовать сортировку по городам и датам премьер.

Вариант 21). Аптека. Реализовать сортировку по наименованиям лекарств и по цене.

Вариант 22). Абонентская плата. Реализовать сортировку по фамилиям абонентов и по сумме оплаты.

Вариант 23). Стоматология. Реализовать сортировку по фамилиям пациентов и по сумме оплаты.

Вариант 24). Техобслуживание подвижного состава. Реализовать сортировку по названию и по сумме ремонта.

Вариант 25). Кафедра. Реализовать сортировку по названию кафедр и по стажу работы преподавателей.

ЛАБОРАТОРНАЯ РАБОТА №6. ВВОД-ВЫВОД. ПАКЕТ JAVA.IO

Цель работы. Лабораторная работа посвящена пакету JAVA.IO, поддерживающему операции ввода-вывода.

6.1. Методические рекомендации

Как известно всем программистам с давних времен, большинство программ не может выполнять свою работу, не имея доступа к внешним данным. Данные извлекаются из источника **ввода**. Результат программы направляется в **вывод**.

На языке Java эти понятия определяются очень широко. Например, источником ввода или местом вывода может служить **сетевое соединение, буфер памяти или дисковый файл** — всеми ими можно манипулировать при помощи классов ввода-вывода Java. Хотя физически они совершенно различны, все эти устройства описываются единой абстракцией — **поток**ом.

Поток - это логическая сущность, которая выдает или получает информацию. Поток присоединен к физическому устройству при помощи системы ввода-вывода Java. Все потоки ведут себя похоже, даже несмотря на то, что физические устройства, к которым они присоединены, в корне отличаются.

6.1.1 Класс Console

Ранее (в Java SE 6) был добавлен класс Console. Он используется для чтения и записи информации на консоли, если таковая существует, и реализует интерфейс Flushable.

Класс Console, прежде всего, введен для удобства, поскольку большая часть его функциональных возможностей доступна через объекты System.in и System.out. Однако его применение позволяет упростить некоторые виды консольных итераций, особенно при чтении строк с консоли.

Класс Console не поддерживает конструкторов. Его объект получают вызовом метода System.console ().

static System.console()

Если консоль доступна, возвращается ссылка на нее. В противном случае возвращается значение **null**. Консоль не будет доступна во всех классах, поэтому если возвращается значение **null**, консольные операции ввода-вывода невозможны.

Класс Console определяет методы, перечисленные в табл. 19.6. Обратите внимание на то, что методы ввода, такие как метод readLine (), передают исключение IOException, когда возникают ошибки ввода. Класс исключения IOException происходит от класса Error и означает сбой ввода-вывода, который происходит вне контроля вашей программы. То есть обычно вы не будете перехватывать исключение IOException. Откровенно говоря, если исключение IOException возникнет в процессе обращения к консоли, обычно это свидетельствует о катастрофическом сбое системы.

Также обратите внимание на методы readPassword (), которые позволяют приложению считывать пароль, не отображая его на экране. Читая пароли, следует "обнулять" как массив, содержащий строку, введенную пользователем, так и массив, содержащий правильный пароль, с которым нужно сравнить первую строку. Это уменьшает шансы вредоносной программы получить пароль при помощи сканирова-

ния памяти.

Таблица 6.1. Методы, определенные в классе **Console**

Метод	Описание
<code>void flush()</code>	Выполняет физическую запись буферизованного вывода на консоль
<code>Console format(String формСтрока, Object... аргументы)</code>	Выводит на консоль аргументы, используя формат, указанный в <code>формСтрока</code>
<code>Console printf(String формСтрока, Object... аргументы)</code>	Выводит на консоль аргументы, используя формат, указанный в <code>формСтрока</code>
<code>Reader reader()</code>	Возвращает ссылку на объект класса, производного от класса <code>Reader</code> , соединенный с консолью
<code>String readLine()</code>	Читает и возвращает строку, введенную с клавиатуры. Ввод прекращается нажатием клавиши <Enter>. Если достигнут конец входного потока консоли, возвращается значение <code>null</code> . В случае сбоя передается исключение <code>IOException</code>
<code>String readLine(String формСтрока, Object... аргументы)</code>	Отображает строку приглашения, форматированную в соответствии с <code>формСтрока</code> и аргументы, затем читает и возвращает строку, введенную с клавиатуры. Ввод прекращается, когда пользователь нажимает клавишу <Enter>. Если достигнут конец входного потока консоли, возвращается значение <code>null</code> . В случае сбоя передается исключение <code>IOException</code>
<code>char[] readPassword()</code>	Читает и возвращает строку, введенную с клавиатуры. Ввод прекращается нажатием клавиши <Enter>. При этом строка не отображается. Если достигнут конец входного потока консоли, возвращается значение <code>null</code> . В случае сбоя передается исключение <code>IOException</code>

Метод	Описание
<code>char[] readPassword(String формСтрока, Object... аргументы)</code>	Отображает строку приглашения, форматированную в соответствии с <code>формСтрока</code> и аргументы, а затем читает и возвращает строку, введенную с клавиатуры. Ввод прекращается, когда пользователь нажимает клавишу <Enter>. При этом строка не отображается. Если достигнут конец входного потока консоли, возвращается значение <code>null</code> . В случае сбоя передается исключение <code>IOException</code>
<code>PrintWriter writer()</code>	Возвращает ссылку на объект класса, производного от класса <code>Writer</code> , ассоциированный с консолью

Рассмотрим пример, демонстрирующий класс **Console** в действии.

```
import java.io.*;
class ConsoleDemo {
    public static void main(String args[]) {
        String str;
        Console con;
        // Получить ссылку на консоль.
        con = System.console();
        // Если нет доступной консоли, выход,
```

```

if (con == null) return;
// Прочсть строку и отобразить ее.
str = con.readLine("Введите строку: ");
con.printf("Вот ваша строка; %s\n", str);
}
}

```

Вывод этого примера.

Введите строку: Это тест. Вот ваша строка: Это тест.

6.1.2. Сериализация

Сериализация — это процесс записи состояния объекта в байтовый поток.

Она удобна, когда нужно сохранить состояние вашей программы в области постоянного хранения, такой как файл. Позднее вы можете восстановить эти объекты, используя процесс десериализации.

Сериализация также необходима в реализации дистанционного вызова методов (Remote Method Invocation — RMI).

RMI позволяет объекту Java на одной машине обращаться к методу объекта Java на другой машине. Объект может быть применен как аргумент этого дистанционного метода. Посылающая машина сериализует объект и передает его. Принимающая машина десериализует его.

Предположим, что объект, подлежащий сериализации, ссылается на другие объекты, которые, в свою очередь, имеют ссылки на еще какие-то объекты. Такой набор объектов и отношений между ними формирует ориентированный граф. В этом графе могут присутствовать и циклические ссылки. Иными словами, объект X может содержать ссылку на объект Y, а объект Y — обратную ссылку на X. Объекты также могут содержать ссылки на самих себя.

Средства сериализации и десериализации объектов устроены так, что могут корректно работать во всех этих сценариях. Если вы попытаетесь сериализовать объект, находящийся на вершине такого графа объектов, то все прочие объекты, на которые имеются ссылки, также будут рекурсивно найдены и сериализованы. Аналогично во время процесса десериализации все эти объекты и их ссылки корректно восстанавливаются.

Ниже приведен обзор интерфейсов и классов, поддерживающих сериализацию.

Интерфейс Serializable

Только объект, реализующий интерфейс Serializable, может быть сохранен и восстановлен средствами сериализации. Интерфейс Serializable не содержит никаких членов. Он просто используется для того, чтобы указать, что класс может быть сериализован. Если класс является сериализуемым, все его подклассы также сериализуемы.

Переменные, объявленные как **transient**, не сохраняются средствами сериализации. Не сохраняются также статические переменные.

Интерфейс Externalizable

Средства Java для сериализации и десериализации спроектированы так, что большая часть работы по сохранению и восстановлению состояния объекта выполняется автоматически. Однако бывают случаи, когда программисту нужно управлять

этим процессом. Например, может оказаться желательным использовать технологии сжатия и шифрования. Интерфейс `Externalizable` предназначен именно для таких ситуаций.

Интерфейс `Externalizable` определяет следующие два метода.

```
void readExternal(Obj eetInput входнойПоток)  
    throws IOException, ClassNotFoundException  
void writeExternal(ObjectOutput выходнойПоток)  
    throws IOException
```

В этих методах *входнойПоток* — это байтовый поток, из которого объект может быть прочитан, а *выходнойПоток* — байтовый поток, куда он записывается.

Интерфейс `ObjectOutput`

Интерфейс `ObjectOutput` расширяет интерфейсы `AutoCloseable` и `DataOutput`, поддерживает сериализацию объектов. Он определяет методы, показанные в табл. 6.2. Особо отметим метод `writeObject()`. Он вызывается для сериализации объекта. В случае ошибок все методы этого интерфейса передают исключение `IOException`.

Таблица 6.2. Методы, определенные в интерфейсе `ObjectOutput`

Метод	Описание
<code>void close()</code>	Закрывает вызывающий поток. Последующие попытки записи передадут исключение <code>IOException</code>
<code>void flush()</code>	Финализирует выходное состояние, чтобы очистить все буферы. То есть все выходные буферы сбрасываются

Окончание табл. 19.7

Метод	Описание
<code>void write(byte буфер[])</code>	Записывает массив байтов в вызывающий поток
<code>void write(byte буфер[], int смещение, int колБайтов)</code>	Записывает диапазон <i>колБайтов</i> байт из массива <i>буфер</i> начиная с <i>буфер[смещение]</i>
<code>void write(int b)</code>	Записывает одиночный байт в вызывающий поток. Из аргумента <i>b</i> записывается только младший байт
<code>void writeObject(Object объект)</code>	Записывает объект <i>объект</i> в вызывающий поток

Класс `ObjectOutputStream`

Класс `ObjectOutputStream` расширяет класс `OutputStream` и реализует интерфейс `ObjectOutput`. Этот класс отвечает за запись объекта в поток. Конструктор его выглядит так.

`ObjectOutputStream(OutputStream выходнойПоток) throws IOException`

Аргумент *выходнойПоток* представляет собой выходной поток, в который мо-

гут быть записаны сериализуемые объекты. Заккрытие объекта класса `ObjectOutputStream` приводит к закрытию также внутреннего потока, определенного аргументом *выходнойПоток*.

Несколько часто используемых методов класса перечислено в табл. 63. В случае ошибки все они передают исключение `IOException`. Присутствует также класс `PutField`, вложенный в класс `ObjectOutputStream`. Он обслуживает запись постоянных полей, и описание его применения выходит за рамки настоящей книги.

Таблица 6.3. Некоторые из наиболее часто используемых методов класса **ObjectOutputStream**

Метод	Описание
<code>void close()</code>	Закрывает вызывающий поток. Последующие попытки записи передадут исключение <code>IOException</code> . Внутренний поток тоже закрывается
<code>void flush()</code>	Финализирует выходное состояние, так что все буферы очищаются. То есть все выходные буферы сбрасываются
<code>void write(byte буфер[])</code>	Записывает массив байтов в вызывающий поток
<code>void write(byte буфер[], int смещение, int колБайтов)</code>	Записывает диапазон <i>колБайтов</i> байт из массива <i>буфер</i> начиная с <i>буфер[смещение]</i>
<code>void write(int b)</code>	Записывает одиночный байт в вызывающий поток. Из аргумента <i>b</i> записывается только младший байт
<code>void writeBoolean(boolean b)</code>	Записывает значение типа <code>boolean</code> в вызывающий поток
<code>void writeByte(int b)</code>	Записывает значение типа <code>byte</code> в вызывающий поток. Записываемый байт — младший из аргумента <i>b</i>
<code>void writeBytes(String строка)</code>	Записывает байты, составляющие строку <i>str</i> , в вызывающий поток
<code>void writeChar(int c)</code>	Записывает значение типа <code>char</code> в вызывающий поток

Упрощенные типы. 1.0.0

Метод	Описание
<code>void writeChars(String строка)</code>	Записывает символы, составляющие строку <i>строка</i> , в вызывающий поток
<code>void writeDouble(double d)</code>	Записывает значение типа <code>double</code> в вызывающий поток
<code>void writeFloat(float f)</code>	Записывает значение типа <code>float</code> в вызывающий поток
<code>void writeInt(int i)</code>	Записывает значение типа <code>int</code> в вызывающий поток
<code>void writeLong(long l)</code>	Записывает значение типа <code>long</code> в вызывающий поток
<code>final void writeObject(Object объект)</code>	Записывает объект <i>объект</i> в вызывающий поток
<code>void writeShort(int i)</code>	Записывает значение типа <code>short</code> в вызывающий поток

Интерфейс `ObjectInput`

Интерфейс `ObjectInput` расширяет интерфейсы `AutoCloseable` и `DataInput` и определяет методы, перечисленные в табл. 6.4. Он поддерживает сериализацию объектов. Особо стоит отметить метод `readObject()`. Он вызывается для десериализации объекта. В случае ошибок все эти методы передают исключение `IOException`. Метод `readObject()` также может передать исключение `ClassNotFoundException`.

Таблица 6.4. Методы, определенные в интерфейсе `ObjectInput`

Метод	Описание
<code>int available()</code>	Возвращает количество байтов, которые доступны во входном буфере в настоящий момент
<code>void close()</code>	Закрывает вызывающий поток. Последующие попытки чтения вызовут передачу исключения <code>IOException</code> . Внутренний поток тоже закрывается
<code>int read()</code>	Возвращает целочисленное представление следующего доступного байта ввода. При достижении конца файла возвращается значение <code>-1</code>
<code>int read(byte буфер[])</code>	Пытается прочитать до <code>буфер.длина</code> байт в <code>буфер</code> , начиная с <code>буфер[смещение]</code> , возвращая количество байтов, которые удалось прочитать. При достижении конца файла возвращается значение <code>-1</code>
<code>int read(byte буфер[], int смещение, int колБайтов)</code>	Пытается прочитать до <code>колБайтов</code> байт в <code>буфер</code> , начиная с <code>буфер[смещение]</code> , возвращая количество байтов, которые удалось прочитать. При достижении конца файла возвращается значение <code>-1</code>
<code>Object readObject()</code>	Читает объект из вызывающего потока
<code>Long skip(long numBytes)</code>	Игнорирует (т.е. пропускает) <code>колБайтов</code> байт вызывающего потока, возвращая количество действительно пропущенных байтов

Класс `ObjectInputStream`

Этот класс расширяет класс `InputStream` реализует интерфейс `ObjectInput`. Класс `ObjectInputStream` отвечает за чтение объектов из потока. Ниже показан конструктор этого класса.

`ObjectInputStream(InputStream входнойПоток)` throws `IOException`

Аргумент *входнойПоток* — это входной поток, из которого должен быть прочитан сериализованный объект. Закрытие объекта класса `ObjectInputStream` приводит к закрытию также внутреннего потока, определенного аргументом *входнойПоток*.

Несколько часто используемых методов этого класса показано в табл. 6.5. В случае ошибок все они передают исключение `IOException`. Метод `readObject()` также может передать исключение `ClassNotFoundException`. Также в классе `ObjectInputStream` присутствует вложенный класс по имени `GetField`. Он обслуживает чтение постоянных полей, и описание его применения выходит за рамки настоящей книги.

Таблица 6.5. Часто используемые методы, определенные в классе `ObjectInputStream`

Метод	Описание
<code>int available()</code>	Возвращает количество байтов, доступных в данный момент во входном буфере
<code>void close()</code>	Закрывает вызывающий поток. Последующие попытки чтения вызовут передачу исключения <code>IOException</code> . Внутренний поток тоже закрывается
<code>int read()</code>	Возвращает целочисленное представление следующего доступного байта ввода. При достижении конца файла возвращается значение <code>-1</code>
<code>int read(byte буфер[], int смещение, int колБайтов)</code>	Пытается прочитать до <code>колБайтов</code> байт в буфер, начиная с <code>буфер[смещение]</code> , возвращая количество байтов, которые удалось прочитать. При достижении конца файла возвращается значение <code>-1</code>
<code>Boolean readBoolean()</code>	Читает и возвращает значение типа <code>boolean</code> из вызывающего потока
<code>byte readByte()</code>	Читает и возвращает значение типа <code>byte</code> из вызывающего потока
<code>char readChar()</code>	Читает и возвращает значение типа <code>char</code> из вызывающего потока
<code>double readDouble()</code>	Читает и возвращает значение типа <code>double</code> из вызывающего потока
<code>double readFloat()</code>	Читает и возвращает значение типа <code>float</code> из вызывающего потока
<code>void readFully(byte буфер[])</code>	Читает буфер. длина байт в буфер. Возвращает управление, только когда все байты прочитаны
<code>void readFully(byte буфер[], int смещение, int колБайтов)</code>	Читает <code>колБайтов</code> байт в буфер, начиная с <code>буфер[смещение]</code>
	Возвращает управление, только когда прочитано <code>колБайтов</code> байт

Окончание таблицы 17.10

Метод	Описание
<code>int readInt()</code>	Читает и возвращает значение типа <code>int</code> из вызывающего потока
<code>int readLong()</code>	Читает и возвращает значение типа <code>long</code> из вызывающего потока
<code>final Object readObject()</code>	Читает и возвращает объект из вызывающего потока
<code>short readShort()</code>	Читает и возвращает значение типа <code>short</code> из вызывающего потока
<code>int readUnsignedByte()</code>	Читает и возвращает значение типа <code>unsigned byte</code> из вызывающего потока
<code>int readUnsignedShort()</code>	Читает и возвращает значение типа <code>unsigned short</code> из вызывающего потока

Пример сериализации

В следующей программе показано, как использовать сериализацию и десери-

лизацию объектов. Начинается она с создания экземпляра объекта My Class. Этот объект имеет три переменные экземпляра типа String, int и double. Именно эту информацию мы хотим сохранять и восстанавливать.

В программе создается объект класса FileOutputStream, который ссылается на файл по имени " **serial**", и для этого файлового потока создается объект класса ObjectOutputStream. Метод writeObject() этого объекта класса ObjectOutputStream используется затем для сериализации объекта. Объект выходного потока очищается и закрывается.

Далее создается объект класса FileInputStream, который ссылается на файл по имени " **serial**", и для этого файлового потока создается объект класса ObjectInputStream. Метод readObject () класса ObjectInputStream используется для последующей десериализации объекта. После этого входной поток закрывается.

Обратите внимание на то, что объект MyClass определен с реализацией интерфейса Serializable. Если бы этого не было, передалось бы исключение NotSerializableException. Поэкспериментируйте с этой программой, объявляя некоторые переменные экземпляра MyClass как transient. Эти данные не будут сохраняться при сериализации.

```
// Демонстрация сериализации
// Эта программа использует оператор try-с-ресурсами. Требуется JDK 7.
import java.io.*;
public class SerializationDemo {
    public static void main(String args[]) {
        // Сериализации объекта
        try ( ObjectOutputStream objOStrm =
            new ObjectOutputStream(new FileOutputStream(" serial")) )
        {
            MyClass object1 = new MyClass("Hello", -7, 2.7e10);
            System.out.println("object1: " + object1);
            objOStrm.writeObject(object1);
        }
        catch(IOException e) {
            System.out.println("Исключение во время сериализации : " + e);
        }
        // Десериализация объекта
        try { ObjectInputStream objIStrm =
            new ObjectInputStream(new FileInputStream("serial")) )
        {
            MyClass object2 = (MyClass) objIstrm.readObject ();
            System.out.println("object2: " + object2);
        }
        catch(Exception e) {
            System.out.println ("Исключение во время сериализации: " + e);
            System.exit (0);
        }
    }
}
```

```

class MyClass implements Serializable {
String s;
int i;
double d;
public MyClass(String s, int i, double d) {
this.s = s;
this.i = i;
this.d = d;
}
public String toString() {
return "s=" + s + " ; i=" + i + " ; d=" + d;
}
}

```

Эта программа демонстрирует идентичность переменных экземпляра объектов **object1** и **object2**. Вот ее вывод.

```

Object1: s=Hello; i=-7; d=2.7E10
object2: s=Hello; i=-7; d=2.7E10

```

6.1.3 Преимущества потоков

Потоковый интерфейс ввода-вывода в Java предоставляет чистую абстракцию для сложных и зачастую обременительных задач. Композиция классов фильтрующих потоков позволяет динамически строить собственные настраиваемые потоковые интерфейсы, которые отвечают вашим требованиям к передаче данных. Программы Java, использующие эти абстрактные высокоуровневые классы — `InputStream`, `OutputStream`, `Reader` и `Writer`, — будут корректно функционировать в будущем, даже когда появятся новые усовершенствованные конкретные потоковые классы. Эта модель работает очень хорошо, когда мы переключаемся от набора потоков на основе файлов к сетевым потокам и потокам сокетов. И наконец, сериализация объектов играет важную роль в программах Java различных типов. Классы сериализации ввода-вывода Java обеспечивают переносимое решение этой непростой задачи.

6.2 Контрольный пример

Откройте Java-проект, созданный в лабораторной работе №4. Изменим набор классов **Item**, **Book**, **MusicCD**, **MusicDVD** и **TestItem** для организации работы со сканером и возможностью ввода и вывода данных в внешние файлы.

Изменим наши классы так, чтобы данные для новых объектов классов пользователь ввод сам с клавиатуры в консоле. Также добавим для удобства общее пользовательское меню для работы с классами, в котором организуем ввод данных для создания объектов классов.

Откройте первый класс — **Book.class**. Добавим возможность в класс ввода данных пользователем с клавиатуры.

Для возможности работы со сканером подключите библиотеку **import java.util.Scanner**;

Для работы со сканером создайте следующую процедуру:

```

public static Book makeNewBook(Scanner in) {

```

```

in.skip("\r\n");
System.out.println("Программное создание объекта класса Book....");
System.out.println("Введите название книги ....");
String newnaimenovanie = in.nextLine();
System.out.println("Введите фамилию автора ....");
String newname = in.nextLine();
System.out.println("Введите год издательства ....");
int newgod = in.nextInt();
System.out.println("Введите издательство ....");
String newizdatelstvo = in.nextLine();
System.out.println("Введите цену ....");
float newprice = in.nextFloat();
System.out.println("Введите кол-во страниц ....");
int newdlitelnost = in.nextInt();

```

```

    return new Book (newnaimenovanie, newname, newgod, newizdatelstvo,
newprice, newdlitelnost); // возвращение введенных значений в конструктор
класса
}

```

Далее самостоятельно измените аналогично классы **MusicCD**, **MusicDVD**.

Затем добавим пользовательское меню в класс **TestItem**.

Подключите библиотеку **import java.util.Scanner**;

Создайте следующую процедуру, которая выведет меню на экран:

```

private static void showMenu() {
    System.out.println("\n");
    System.out.println("1 - Show collection");
    System.out.println("2 - Add new Book");
    System.out.println("3 - Add new MusicCD");
    System.out.println("4 - Add new MusicDVD");
    System.out.println("5 - Save program");
    System.out.println("6 - Load program");
    System.out.println("7 - Exit");
}

```

Далее, для того, чтобы организовать возможность пользователю выбирать элементы меню с клавиатуры реализуем работу со сканером. Для этого внутри класса **TestItem** в главной процедуре **main** объявим сканер и реализуем работу с меню. Для этого измените процедуру **main** на следующую:

```

public static void main (String args[]) {

    Scanner in = new Scanner(System.in).useDelimiter("\r\n");
    showMenu();
}

```

```

while(true) {
    int menuItem = in.nextInt();

    if (menuItem == 7) {
        System.out.println("Bye");
        System.exit(0);
    } else if (menuItem == 1) {
        //пока ничего не делаем
    } else if (menuItem == 2) {
        Book newBook = Book.makeNewBook(in);
        System.out.println("Информация    про    книгу:
"+newBook.toString());

    } else if (menuItem == 3) {
        MusicCD newMusicCD= MusicCD.makeNewMusicCD(in);
        System.out.println("Информация    про    музыку:
"+newMusicCD.toString());

    } else if (menuItem == 4) {
        MusicDVD newMusicDVD= Mu-
sicDVD.makeNewMusicDVD(in);
        System.out.println("Информация    про    фильм:
"+newMusicDVD.toString());

    } else if (menuItem == 5) {
        //    сохранить данные в файл

    } else if (menuItem == 6) {

        // извлечь данные из файла

    }
    showMenu();
}
}

```

Далее, организуем работу с **коллекциями**. Например, мы хотим вводить не по одному объекту, а хотим, чтобы каждый новый объект добавлялся в коллекцию **Book**, **MusicCD**, **MusicDVD**.

Создадим специальный класс для работы с коллекцией. Создайте и добавьте в свой пакет новый класс **Compar.class**.

```
import java.util.*;
```

```

public class Compar {

    private Comparator<Item> comparator = null; // объявление будущей
сортировки

    private ArrayList<Item> item = new ArrayList<Item>(); // создание
коллекции списка-массива типа класса Item – общего родительского класса

    public void setItem(ArrayList<Item> item) { // присвоение коллекции
        this.item = item;
    }

    public ArrayList<Item> getItem(){ // возвращение коллекции
        return item;
    }

    public void add(Item item_service) {
        // добавление элементов в коллекцию
        this.item.add(item_service);
    }

    public void setComparator(Comparator<Item> comparator) {
        this.comparator = comparator;
    }

    public void show() {
        // отображение на экран всей коллекции
        if (this.comparator != null)
            Collections.sort(this.item, this.comparator); // сортировка кол-
лекции

        Iterator<Item> iter = this.item.iterator(); // установка итератора
на начало коллекции
        while (iter.hasNext()) { // пока не конец коллекции
            System.out.println(iter.next()); // вывод объектов кол-
лекции
        }
    }
}

```

Далее изменим базовый родительский класс для реализации интерфейса сортировки по коллекции.

Для этого откройте класс **Item.class** вверху подключите библиотеку **import java.util.*;**, чтобы были доступны интерфейсы **Collection**, **ArrayList**, **Iterator**, **Comparator**.

Далее измените объявление класса на следующее:

```

public class Item implements Comparable<Item>

```


Далее необходимо описать в классе метод сортировки, допустим мы будем сортировать коллекцию по возрастанию цены продукции. Для этого добавьте еще один метод в класс:

```
public int compareTo(Item another) {  
    return (int)(this.price - another.getPrice());  
}
```

Далее изменим класс **TestItem** в главной процедуре **main**. Измените библиотеку на **import java.util.*;**, чтобы были доступны интерфейсы **Collection**, **ArrayList**, **Iterator**, **Comparator**.

Внутри процедуры **main** создайте новый объект класса **Compar**:

```
Compar compar = new Compar();
```

Затем изменим меню. Для того, чтобы обеспечить создание новых объектов и добавление их в коллекцию нужно вызвать метод **compar.add(obj)**.

Например, для объектов книг меню изменится на:

```
else if (menuItem == 2) {  
    Book newBook = Book.makeNewBook(in);  
    System.out.println("Информация про книгу:  
"+newBook.toString());  
    compar.add(newBook);  
}
```

Аналогично выполните изменение меню для объектов классов **MusicCD** и **MusicDVD**.

Далее, чтобы вывести всю отсортированную коллекцию на экран нужно вызвать метод **compar.show()**. Поэтому измените соответствующее меню на:

```
else if (menuItem == 1) {  
    compar.show(); }  
}
```

6.2. Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

1. Теоретические сведения о потоках ввода-вывода и сериализации.
2. Распечатанный листинг выполнения программы, отражающий все этапы ее выполнения.
3. Выводы о выполненной работе.

6.3 Варианты заданий для самостоятельной работы

Вариант 1). Записная книжка контактов. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 2). Система управления доставкой товара. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 3). Телепрограмма. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 4). Гостиница. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 5). Реализация готовой продукции. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 6). Успеваемость студентов ВУЗА. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 7). Факультеты. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 8). Супермаркеты. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 9). Военный состав. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 10). Учет литературы. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 11). Учет продажи путевок. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 12). Учет выполненных работ станции техобслуживания. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 13). Медицинское обслуживание пациентов. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 14). Библиотека. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню..

Вариант 15). Продажа автомобилей. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 16). Интернет-магазин. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 17). Больница. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 18). Военно-морские учения. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 19). Туристические туры. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 20). Представления цирка. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 21). Аптека. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 22). Абонентская плата. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 23). Стоматология. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 24). Техобслуживание подвижного состава. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 25). Кафедра. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

ЛАБОРАТОРНАЯ РАБОТА №7. РАБОТА С ФАЙЛАМИ

Цель работы. Изучить систему ввода-вывода Java, включая базовые методики чтения и записи файлов, обработки исключений ввода-вывода и закрытия файла.

7.1. Методические рекомендации

Как известно, большинство программ не может выполнять свою работу, не имея доступа к внешним данным. Данные извлекаются из источника **ввода**. Результат программы направляется в **вывод**.

На языке Java эти понятия определяются очень широко. Например, источником ввода или местом вывода может служить **сетевое соединение, буфер памяти или дисковый файл** — всеми ими можно манипулировать при помощи классов ввода-вывода Java. Хотя физически они совершенно различны, все эти устройства описываются единой абстракцией — **поток**.

Поток - это логическая сущность, которая выдает или получает информацию. Поток присоединен к физическому устройству при помощи системы ввода-вывода Java. Все потоки ведут себя похоже, даже несмотря на то, что физические устройства, к которым они присоединены, в корне отличаются.

Потоковая система ввода-вывода, используемая пакетом `java.io`, была частью языка Java начиная с его первого выпуска и широко используется до сих пор. Однако начиная с версии 1.4 в язык Java была добавлена вторая система ввода-вывода. Она называется **NIO** (что первоначально было акронимом от **New I/O** (новый ввод-вывод)). Система NIO расположена в пакете ***java.nio*** и его внутренних пакетах. С выпуском комплекта JDK 7 возможности системы NIO были существенно расширены, и популярность ее использования, как ожидается, возрастет. Система NIO описана в главе 20.

7.1.1. Классы и интерфейсы ввода-вывода Java

Классы ввода-вывода, определенные в пакете `java.io`, перечислены ниже.

Таблица 7.1 Классы ввода-вывода

<code>BufferedInputStream</code>	<code>FileWriter</code>	<code>PipedOutputStream</code>
<code>BufferedOutputStream</code>	<code>FilterInputStream</code>	<code>PipedReader</code>
<code>BufferedReader</code>	<code>FilterOutputStream</code>	<code>PipedWriter</code>
<code>BufferedWriter</code>	<code>FilterReader</code>	<code>PrintStream</code>
<code>ByteArrayInputStream</code>	<code>FilterWriter</code>	<code>PrintWriter</code>
<code>ByteArrayOutputStream</code>	<code>InputStream</code>	<code>PushbackInputStream</code>

CharArrayReader	InputStreamReader	PushbackReader
CharArrayWriter	LineNumberReader	RandomAccessFile
Console	ObjectInputStream	Reader
DataInputStream	ObjectInputStream. GetField	SequenceInputStream
DataOutputStream	ObjectOutputStream	SerializablePermission
File	ObjectOutputStream. PutField	StreamTokenizer
FileDescriptor	ObjectStreamClass	StringReader
FileInputStream	ObjectStreamField	StringWriter
FileOutputStream	OutputStream	Writer
FilePermission	OutputStreamWriter	
FileReader	PipedInputStream	

В пакете java.io определены следующие интерфейсы.

Closeable	FileFilter	ObjectInputValidation
DataInput	FilenameFilter	ObjectOutput
DataOutput	Flushable	ObjectStreamConstants
Externalizable	ObjectInput	Serializable

Как видите, в пакете java.io присутствует множество классов и интерфейсов. Среди них — байтовые и символьные потоки, сериализация объектов (их сохранение и восстановление). В настоящей главе рассматривается несколько наиболее широко используемых компонентов ввода-вывода. Новый класс **Console** также описан. Начнем обсуждение с одного из наиболее отличающихся классов ввода-вывода — **File**.

7.1.2 Класс File

Хотя большинство классов, определенных в пакете java.io, работают с потоками, класс **File** этого не делает. Он имеет дело непосредственно с файлами и файловой системой. То есть **класс File** не указывает, как извлекается и сохраняется информация в файлах; **он описывает свойства самих файлов**.

Объект класса **File** служит для получения информации и манипулирования информацией, ассоциированной с дисковым файлом, такой как права доступа, время, дата и путь к каталогу, а также для навигации по иерархиям подкаталогов.

Интерфейс **Path** и класс **Files**, добавленные в систему N10 комплектом JDK 7, являются серьезной альтернативой классу **File** во многих случаях.

Класс **File** — первичный источник и место назначения для данных во многих программах. Хотя существует несколько ограничений в части использования файлов в апплетах (из соображений безопасности), тем не менее они продолжают оставаться центральным ресурсом для хранения постоянной и разделяемой информации.

Каталог в Java трактуется как объект **класса File** с единственным дополнитель-

ным свойством — списком имен файлов, которые могут быть получены методом `list()`.

Для создания объектов класса `File` могут быть использованы следующие конструкторы.

```
File(String путьКкаталогу)  
File(String путьКкаталогу, String имяфайла)  
File(File объектКаталога, String имяфайла)  
File(URI объектURI)
```

Здесь *путьКкаталогу*—это путь к файлу; *имяфайла* — имя файла или подкаталога; *объектКаталога* — объект класса `File`, указывающий каталог; а *объектURI* — объект `URI`, описывающий файл.

В следующем примере создается три файла: **f1**, **f2** и **f3**. Первый объект класса `File` создается с путем к каталогу в единственном аргументе. Второй включает два аргумента — путь и имя файла. Третий включает путь, присвоенный файлу **f1**, и имя файла; объект файла **f3** ссылается на тот же файл, что и **f2**.

```
File f1 = new File ("/*") ;  
File f2 = new File ("/*","autoexec.bat");  
File f3 = new File(f1,"autoexec.bat");
```

Класс `File` определяет множество методов, представляющих стандартные свойства объекта класса `File`:

Например, метод `getName ()` возвращает имя файла,
метод `getParent ()` — имя родительского каталога,
метод `exists ()` — значение **true**, если файл существует, и значение **false** — если нет.

Однако класс `File` не симметричен. В нем есть несколько методов, позволяющих *проверять* свойства простого файлового объекта, у которых нет дополняющих их методов изменения этих атрибутов.

В следующем примере демонстрируется применение нескольких методов класса `File`. Здесь подразумевается, что в корневом каталоге существует каталог по имени **java** и что он содержит файл по имени **COPYRIGHT**.

```
// Демонстрация работы с File  
import java.io.File;  
class FileDemo {  
static void p(String s) {  
System.out.println (s);  
}  
public static void main(String args[]) {  
File f1 = new File("/java/COPYRIGHT");  
p("Имя файла: " + f1.getName());  
p("Путь: " + f1.getPath());  
p("Абсолютный путь: " + f1.getAbsolutePath());  
p("Родительский каталог: " + f1.getParent());
```

```

p(f1.exists() ? "существует" : "не существует");
p(f1.canWrite() ? "доступен для записи" : "не доступен для записи");
p(f1.canRead() ? "доступен для чтения" : "не доступен для чтения");
p(f1.isDirectory() ? "является каталогом" : "не является каталогом");
p(f1.isFile() ? "является обычным файлом" : "может быть именованным
каналом");
p(f1.isAbsolute() ? "является абсолютным" : "не является абсолютным");
p("Время модификации: " + f1.lastModified());
p("Размер: " + f1.length() + " байт");
}
}

```

Эта программа выдает нечто вроде следующего.

Имя файла: COPYRIGHT

Путь: \java\COPYRIGHT

Абсолютный путь: \java\COPYRIGHT

Родительский каталог: \java

существует

доступен для записи

доступен для чтения

не является каталогом

является обычным файлом

не является абсолютным

Время модификации: 1282832030047

Размер: 695 байт

Большинство методов класса File самоочевидно, но методы `isFile ()` и `isAbsolute ()` — нет.

Метод `isFile ()` возвращает значение **true**, если вызывается с файлом, и значение **false** — если с каталогом. Также метод `isFile ()` возвращает значение **false** для некоторых специальных файлов, таких как драйверы устройств и именованные каналы, поэтому этот метод может применяться для гарантии того, что данный файл действительно ведет себя как файл.

Метод `isAbsolute ()` возвращает значение **true**, если файл имеет абсолютный путь, и значение **false** — если относительный.

Класс File включает также два полезных служебных метода. Первый из них, метод `renameTo ()`, показан ниже:

boolean renameTo(File *новоеИмя*)

Здесь имя файла, указанное в параметре *новоеИмя*, становится новым именем вызывающего объекта класса File. Метод возвращает значение **true** в случае успеха и значение **false** — в случае неудачи, если файл не может быть переименован (если вы пытаетесь переименовать файл, указывая имя существующего файла).

Второй служебный метод — `delete ()`, который удаляет дисковый файл, представленный путем вызывающего объекта класса File.

boolean delete()

Также вы можете использовать метод `delete ()` для удаления каталога, если он пуст. Метод `delete ()` возвращает значение **true**, если ему удастся удалить файл, и значение **false**, если файл не может быть удален. В табл. 7.2 приведено еще несколько методов класса **File**, которые вы наверняка сочтете полезными.

Таблица 7.2. Полезные методы класса **File**

Метод	Описание
<code>void deleteOnExit()</code>	Удаляет файл, ассоциированный с вызывающим объектом, по завершении работы виртуальной машины Java
<code>long getFreeSpace()</code>	Возвращает количество свободных байтов хранилища, доступных в разделе, ассоциированном с вызывающим объектом
<code>long getTotalSpace()</code>	Возвращает емкость хранилища раздела, ассоциированного с вызывающим объектом

Окончание табл. 19.1

Метод	Описание
<code>long getUsableSpace()</code>	Возвращает количество доступных, годных к употреблению свободных байтов, находящихся в разделе, ассоциированном с вызывающим объектом
<code>boolean isHidden()</code>	Возвращает значение true , если вызывающий файл является скрытым, и значение false — в противном случае
<code>boolean setLastModified(long миллисекунд)</code>	Устанавливает временную метку для вызываемого файла в значение миллисекунд, которое представляет количество миллисекунд, прошедших с 1 января 1970 г., по UTC
<code>boolean setReadOnly()</code>	Делает вызывающий файл доступным только для чтения

Также существуют методы, помечающие файлы как доступные только для чтения, записи или выполнения. Поскольку класс **File** реализует интерфейс **Comparable**, также поддерживается метод `compareTo ()`.

Комплект JDK 7 добавляет в класс **File** новый метод по имени `toPath ()`, который выглядит так.

Path toPath()

Метод `toPath()` возвращает объект интерфейса **Path**, который представляет файл, инкапсулируемый вызываемым объектом класса **File**. (Другими словами, метод `toPath ()` преобразует объект класса **File** в объект интерфейса **Path**.)

Path — это новый интерфейс, добавленный в JDK 7. Он находится в пакете **java.nio.file** и является частью системы NIO. Таким образом, метод `toPath ()` формирует мост между старым классом **File** и новым интерфейсом **Path**.

7.1.3 Исключения ввода-вывода

Два исключения играют важную роль в обработке ввода-вывода. Первое из них — исключение **IOException** — имеет отношение к большинству классов ввода-вывода, описанных в данной главе, поэтому при ошибке ввода-вывода происходит передача исключения **IOException**. В большинстве случаев, если файл не может быть открыт, передается исключение **FileNotFoundException**. Класс исключения **FileNotFoundException** происходит от класса **IOException**, поэтому оба могут быть обработа-

ны в одном блоке **catch**, предназначенном для обработки исключения `IOException`. Этот подход используется для краткости в большинстве примеров кода данной главы. Однако в собственных приложениях вам может иметь смысл обрабатывать их по отдельности.

Другой класс исключения, который иногда очень важен при выполнении ввода-вывода, — это класс `SecurityException`. Когда присутствует менеджер безопасности, некоторые классы файлов передают исключение `SecurityException` при попытке открыть файл с нарушением безопасности. По умолчанию приложения, запущенные при помощи команды `java`, не используют менеджер безопасности. Поэтому примеры ввода-вывода в этой книге не отслеживают возможность передачи исключения `SecurityException`. Однако апплеты будут использовать менеджер безопасности, предоставленный браузером, и файловый ввод-вывод, выполняемый апплетом, может передать исключение `SecurityException`. В таком случае вам придется обрабатывать и это исключение.

Два способа закрытия потока

Как правило, поток следует закрыть, когда он больше не нужен. Если не сделать этого, может произойти утечка памяти и потеря ресурсов.

Начиная с JDK 7 существует два основных способа, которыми вы можете закрыть поток. Первый подразумевает явный вызов метода `close ()` для потока. Это традиционный подход, который использовался начиная с первого выпуска Java. При этом подходе метод `close ()` обычно вызывается в блоке **finally**. Таким образом, упрощенный шаблон традиционного подхода выглядит так.

```
try {  
    // открыть и использовать файл  
} catch (исключениеВвода-вывода) {  
    // ...  
} finally {  
    // закрыть файл  
}
```

Эта общая методика (или его разновидность) популярна в коде, предшествующем JDK 7.

Второй подход закрытия потока подразумевает автоматизацию процесса с использованием нового оператора `try` - с **-ресурсами**, который появился в комплекте JDK 7. Оператор `try`-с-ресурсами — это улучшенная форма оператора **try**, имеющего следующую форму.

```
try (спецификация-ресурса) {  
    // использование ресурса  
}
```

Здесь **спецификация-ресурса** — это оператор или операторы, которые объявляют и инициализируют ресурс, такой как файл или другой связанный с потоком ресурс. Он состоит из объявления переменной, в котором переменная инициализируется ссылкой на управляемый объект. Когда заканчивается блок **try**, ресурс освобождается автоматически. В случае файла это означает, что он автоматически закрывается. Таким образом, нет никакой необходимости вызывать метод `close ()` явно.

Вот три ключевых пункта, касающихся оператора *try с-ресурсами*.

- Ресурсы, управляемые оператором try-с-ресурсами, должны быть объектами классов, реализующих интерфейс `AutoCloseable`.
- Ресурс, объявленный в блоке **try**, является неявно финальным.
- Вы можете управлять несколькими ресурсами, отделив каждый из них в объявлении точкой с запятой.

Кроме того, не забывайте, что область видимости объявленного ресурса ограничивается оператором *try - с -ресурсами*.

Основное преимущество оператора *try-с-ресурсами* заключается в том, что ресурс (в данном случае — поток) закрывается автоматически по завершении блока **try**. Таким образом, невозможно забыть закрыть поток, например. Кроме того, подход с оператором try-с-ресурсами дает обычно более краткий и понятный исходный код, который проще поддерживать.

Благодаря своим преимуществам, оператор try-с-ресурсами, как ожидается, будет интенсивно использоваться в новом коде. В результате большая часть кода этой главы (и книги) будет использовать его. Однако поскольку существуют миллионы строк кода, написанного до появления комплекта JDK 7, программисты должны быть знакомы с традиционным подходом закрытия потока. Например, вам, весьма вероятно, придется работать с устаревшим кодом, который применяет традиционный подход, или в среде, которая использует версию Java, предшествующую комплекту JDK 7. Может сложиться ситуация, когда автоматизированный подход окажется неприменимым из-за других аспектов вашего кода. Поэтому несколько примеров ввода-вывода в этой книге демонстрируют традиционный подход, и вы сможете увидеть его в действии.

Еще один момент: примеры, которые используют оператор *try-с -ресурсами*, следует компилировать в комплекте ОК 7 или позже. Они не будут работать со старым компилятором. Примеры, которые используют традиционный подход, могут быть откомпилированы прежними версиями Java.

Помните! Поскольку оператор *try-с-ресурсами* упрощает процесс освобождения ресурсов и устраняет возможность их случайного пропуска, этот подход рекомендуется для нового кода, когда его использование возможно.

7.1.4 Классы потоков

Основанный на потоках ввода-вывода, Java построен на базе абстрактных классов: `InputStream`, `OutputStream`, `Reader` и `Writer`. Они используются для создания некоторых конкретных подклассов потоков.

Хотя ваши программы реализуют свои операции ввода-вывода через конкретные подклассы, классы верхнего уровня определяют базовые функциональные возможности, общие для всех потоковых классов.

Классы `InputStream` и `OutputStream` предназначены для байтовых потоков, а абстрактные классы `Reader` и `Writer` — для символьных. Классы байтовых и символьных потоков формируют отдельные иерархии. В целом классы символьных потоков следует использовать, имея дело с символами строк, а классы байтовых потоков — работая с байтами или другими двоичными объектами.

Далее в этой главе мы будем рассматривать как байтовые, так и символьные потоки.

Поскольку классы байтовых потоков берут свое начало с классов `InputStream` и `OutputStream`, с них и начнем обсуждение.

Класс **InputStream**

Класс **InputStream**—это абстрактный класс, определяющий модель Java байтового потокового ввода. Он реализует интерфейсы **AutoCloseable** и **Closeable**. При ошибках ввода-вывода большинство методов этого класса передает исключение **IOException**. (Сюда не входят методы **mark()** и **markSupported()**) В табл. 7.3 перечислены методы класса **InputStream**.

Таблица 7.3. Методы, определенные в классе **InputStream**

Метод	Описание
<code>int available()</code>	Возвращает количество байтов ввода, которые доступны в данный момент для чтения
<code>void close()</code>	Закрывает источник ввода. Последующие попытки чтения передадут исключение IOException
<code>void mark(int колБайтов)</code>	Помещает метку в текущую точку входного потока, которая остается корректной до тех пор, пока не будет прочитано <i>колБайтов</i> байт
<code>boolean markSupported()</code>	Возвращает значение true , если методы mark() и reset() поддерживаются вызывающим потоком
<code>int read()</code>	Возвращает целочисленное представление следующего доступного байта в потоке. При достижении конца файла возвращается значение -1
<code>int read(byte буфер[])</code>	Пытается читать до <i>колБайтов</i> в <i>буфер</i> , возвращая количество успешно прочитанных байтов. По достижении конца файла возвращает значение -1
<code>int read(byte буфер[], int смещение, int колБайтов)</code>	Пытается читать до <i>колБайтов</i> в <i>буфер</i> , начиная с <i>буфер[смещение]</i> и возвращая количество успешно прочитанных байтов. По достижении конца файла возвращает значение -1
<code>void reset()</code>	Сбрасывает входной указатель в ранее установленную метку
<code>long skip(long колБайтов)</code>	Игнорирует (т.е. пропускает) <i>колБайтов</i> байт ввода, возвращая количество действительности проигнорированных байтов

Большинство методов, описанных в табл. 19.1, реализуется производными классами класса **InputStream**. Методы **mark()** и **reset()** - исключения; обратите внимание на их использование или отсутствие такового в каждом производном классе и следующих обсуждениях.

Класс **OutputStream**

Класс **OutputStream** — это абстрактный класс, определяющий потоковый байтовый вывод. Реализует интерфейсы **AutoCloseable**, **Closeable** и **Flushable**. Большинство методов этого класса возвращает **void** и передает исключение **IOException** в случае ошибок ввода-вывода. В табл. 7.4 перечислены методы класса **OutputStream**.

Большинство методов, описанных в табл. 7.3 и 7.4, реализовано подклассами классов **InputStream** и **OutputStream**. Методы **mark()** и **reset()** являются исключениями; имейте это в виду, когда ниже речь пойдет о каждом подклассе.

Таблица 19.3. Методы, определенные в классе **OutputStream**

Метод	Описание
<code>int close()</code>	Закрывает выходной поток. Последующие попытки записи передадут исключение <code>IOException</code>
<code>void flush()</code>	Финализирует выходное состояние, очищая все буферы, т.е. очищает буферы вывода
<code>void write(int b)</code>	Записывает единственный байт в выходной поток. Обратите внимание на то, что параметр имеет тип <code>int</code> , а это позволяет вызывать метод <code>write()</code> с выражениями без необходимости приведения их обратно к типу <code>byte</code>
<code>void write(byte буфер[])</code>	Записывает полный массив байтов в выходной поток
<code>void write(byte буфер[], int колБайтов)</code>	Записывает диапазон из <i>колБайтов</i> байт из массива <i>буфер</i> начиная с <i>буфер[смещение]</i>

Класс **FileInputStream**

Класс **FileInputStream** создает объект класса **InputStream**, который вы можете использовать для чтения байтов из файла. Так выглядят два его наиболее часто используемых конструктора.

FileInputStream(String *путьКфайлу*)

FileInputStream(File *объектФайла*)

Каждый из них может передать исключение `FileNotFoundException`. Здесь *путьКфайлу* — полное путевое имя файла, а *объектФайла* — объект класса **File**, описывающий файл. В следующем примере создается два объекта класса **FileInputStream**, использующих один и тот же дисковый файл и оба эти конструктора.

```
FileInputStream f0 = new FileInputStream("/autoexec.bat")
```

```
File f = new File("/autoexec.bat");
```

```
FileInputStream f1 = new FileInputStream(f);
```

Хотя первый конструктор, вероятно, используется чаще, второй позволяет подробно исследовать файл с помощью методов класса **File**, прежде чем присоединять его к входному потоку. При создании объект класса **FileInputStream** открывается также и для чтения. Класс **FileInputStream** переопределяет шесть методов абстрактного класса **InputStream**. Методы **mark()** и **reset()** не переопределяются, и все попытки использовать метод **reset()** с объектом класса **FileInputStream** приводят к передаче исключения `IOException`.

Следующий пример показывает, как прочесть один байт, массив байтов и диапазон из массива байтов. Также он иллюстрирует использование метода **available()** для определения оставшегося количества байтов, а также метода **skip()** — для пропуска нежелательных байтов. Программа читает свой собственный исходный файл, который должен присутствовать в текущем каталоге. Обратите внимание на то, что здесь используется новый оператор **JDK 7 try-c-ресурсами** для автоматического закрытия файла, когда он больше не нужен.

```

// Демонстрация применения FileInputStream.
// Эта программа использует оператор try-c-ресурсами. Требуется JDK 7.
import Java.io.*;
class FileInputStreamDemo {
public static void main(String args[]) {
int size;
// Для закрытия потока используется try-c-ресурсами.
try ( FileInputStream f =new FileInputStream("FileInputStreamDemo. Java") )
{
System.out.println("Total Available Bytes: " + (size = f.available()));
int n = size/40;
System.out.println("First " + n + " bytes of the file one read() at a time");
for (int i=0; i < n; i++) {
System.out.print((char) f.read(S) );
}
System.out.println("\n Still Available: " + f.available());
System.out.println("Reading the next " + n + " with one read(b[])");
byte b[] = new byte[n];
if (f.read(b) != n) {
System.err.println("couldn't read " + n + " bytes.");
}
System.out.println(new String(b, 0, n));
System.out.println("\n Still Available: " + (size = f.available()));
System.out.println("Skipping half of remaining bytes with skip()");
f.skip(size/2);
System.out.println("Still Available: " + f.available());
System.out.println("Reading " + n/2 + " into the end of array");
if (f.read(b, n/2, n/2) != n/2) {
System.err.println ("couldn't read " + n/2 + " bytes.");
}
System.out.println(new String(b, 0, b.length));
System.out.println("\n Still Available: " + f.available());
}
catch(IOException e) {
System.out.println("I/O Error: " + e);
}
}
}

```

Так выглядит вывод этой программы.

Total Available Bytes: 1785

First 44 bytes of the file one read() at a time

// Demonstrate FileInputStream.

// This pr

Still Available: 1741

Reading the next 44 with one read(b[])

ogram uses try-with-resources. It requires J

Still Available: 1697

Skipping half of remaining bytes with skip()

Still Available: 849

Reading 22 into the end of array

ogram uses try-with-rebyte[n];

if (

Still Available: 827

Этот несколько надуманный пример демонстрирует чтение тремя способами, пропуск ввода и проверку количества доступных данных в потоке.

Класс **FileOutputStream**

Класс **FileOutputStream** создает объект класса **OutputStream**, который вы можете использовать для записи байтов в файл. Он реализует интерфейсы **AutoCloseable**, **Closeable** и **Flushable**. Вот четыре его наиболее часто используемых конструктора.

FileOutputStream(String *путькфайлу*)

FileOutputStream(File *объектФайла*)

FileOutputStream(String *путькфайлу*, boolean *добавить*)

FileOutputStreamFile *объектФайла*, boolean *добавить*)

Они могут передать исключение **FileNotFoundException**. Здесь *путькфайлу*— полное путевое имя файла, а *объектФайла* — объект класса **File**, описывающий файл. Если параметр *добавить* содержит значение **true**, файл открывается в режиме добавления.

Создание объекта класса **FileOutputStream** не зависит от того, существует ли указанный файл. Класс **FileOutputStream** создает его перед открытием, когда вы создаете объект. В случае попытки открытия файла, доступного только для чтения, будет передано исключение.

В следующем примере создается буфер байтов. Сначала создается объект класса **String**, а затем используется метод **getBytes()** для извлечения его эквивалента в виде байтового массива. Затем создается три файла. Первый — **file1.txt** — будет содержать каждый второй байт примера. Второй — **file2.txt** — полный набор байтов. Третий — **file3.txt** — будет содержать только последнюю четверть.

// Демонстрация применения **FileOutputStream.**

// Эта программа использует традиционный подход закрытия файла.

```
import java.io.*;
class FileOutputStreamDemo {
    public static void main(String args[]) {
        String source = "Now is the time for all good men\n" +
            "to come to the aid of their country\n" +
            "and pay their due taxes.";
        byte buf[] = source.getBytes();
        FileOutputStream f0 = null;
        FileOutputStream f1 = null;
        FileOutputStream f2 = null;
        try {
            f0 = new FileOutputStream("file1.txt");
            f1 = new FileOutputStream("file2.txt");
```

```

f2 = new FileOutputStream("file3.txt");
// запись в первый файл
for (int i = 0; i < buf.length; i += 2) f0.write (buf [i] );
// запись во второй файл
f1.write(buf);
// запись в третий файл
f2.write(buf, buf.length-buf.length/4, buf.length/4);
}
catch(IOException e) {
System.out.println("An I/O Error Occurred");
}
finally {
try {
if (f0 != null) f0.close();
}
catch(IOException e) {
System.out.println("Error Closing file1.txt");
}
try {
if(f1 != null) f1.close();
} catch (IOException e) {
System.out.println("Error Closing file2.txt");
}
try {
if (f2 != null) f2 .closed ;
} catch(IOException e) {
System.out.println("Error Closing file3.txt");
}
}
}
}

```

Так будет выглядеть содержимое каждого файла после выполнения этой программы. Сначала будет представлен файл **file1.txt**.

Nwi h iefralgo e
t oet h i ftercuty n a hi u ae.

Затем файл file2.txt.
Now is the time for all good men
to come to the aid of their country
and pay their due taxes.

И наконец, файл file3.txt.
nd pay their due taxes.

Как видно из комментариев вверху, приведенная выше программа демонстрирует пример использования традиционного подхода закрытия файла, когда он больше не нужен. Этот подход применялся всеми версиями Java до JDK 7 и широко исполь-

зается в устаревшем коде. Как можно заметить, здесь есть немного довольно неуклюжего кода, требуемого для явного вызова метода `close()`, поскольку каждый его вызов может передать исключение `IOException`, если операция закрытия потерпит неудачу.

7.2. Контрольный пример

Откройте Java-проект, созданный в лабораторной работе №6. Изменим набор классов **Item**, **Book**, **MusicCD**, **MusicDVD** и **TestItem** для организации возможности ввода и вывода данных в внешние файлы.

Для этого варианты все классы сохраняемых объектов должны реализовывать интерфейс `java.io.Serializable`, например изменим базовый класс:

```
import java.io.*;
public class Item implements Comparable<Item>, java.io.Serializable
```

Аналогично самостоятельно реализуйте интерфейс `java.io.Serializable` в классах **Book**, **MusicCD**, **MusicDVD**.

Затем необходимо создать в классе-родителе **Item.class** создать два метода для сохранения (метод `save()`) и извлечения (метод `load()`) объектов из внешнего файла.

```
public void save(DataOutputStream out) throws IOException {
}

public Item load(DataInputStream in) throws IOException {
    return null;
}
```

Эти методы пока будут только обрабатывать исключения в случае ошибки, а саму реализацию пропишем в классе реализующего коллекции **Compar.class**. Аналогично самостоятельно реализуйте интерфейс `java.io.Serializable` в этом классе.

Далее создадим два метода для сохранения и загрузки коллекции в текстовый файл:

```
public void save(DataOutputStream out) throws IOException {

    out.writeInt(this.item.size());
    Iterator<Item> iter = this.item.iterator();
    while (iter.hasNext()) {
        iter.next().save(out); // сохранить объект в текстовый
        файл
    }
}

public Compar load(DataInputStream in) throws IOException {
    int size = in.readInt();
    for (int i = 0; i < size; i++) {
```



```

        String x = in.readUTF();

        if (x.equals("Book")){
            Booknew book = new Booknew();
            this.item.add(book.load(in)); // добавить объ-
ект в коллекцию
        }
    }
    return this;
}

```

Далее откройте класс, который будет все запускать **TestItem.class**. Аналогично самостоятельно реализуйте интерфейс **java.io.Serializable** в этом классе.

Создадим в нем два метода **Saving** и **Loading**:

```

public static void Saving(Compar compar){

    DataOutputStream out = null;
    ObjectOutputStream out1 = null;
    try {
        out = new DataOutputStream(new BufferedOut-
putStream(new FileOutputStream(new File("test.txt"))));
        out1 = new ObjectOutputStream(new BufferedOut-
putStream(new FileOutputStream(new File("test1.txt"))));
        try {
            compar.save(out);
            out1.writeObject(compar.getItem());
        } catch (IOException e) {
            System.out.println("Maybe disk is full, or you haven't permissions to
write");
        }

        try {
            out.close();
            out1.close();
        } catch (IOException e) {
            System.out.println("Ошибка");
        }
    } catch (FileNotFoundException e) {
        System.out.println("Отсутствует файл.");
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}

public static Compar Loading(Compar compar){
    DataInputStream in = null;
    ObjectInputStream in1 = null;
    try {
        in = new DataInputStream(new BufferedInputStream(new FileIn-
putStream(new File("test.txt"))));

```

```

        in1 = new ObjectInputStream(new BufferedIn-
putStream(new FileInputStream(new File("test1.txt"))));
        try {
            compar.setItem((ArrayList<Item>)in1.readObject());
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
            System.out.println("Maybe disk is full, or you haven't permissions to
write");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        try {
            in.close();
            in1.close();
        } catch (IOException e) {
            System.out.println("ошибка");
        }
    } catch (FileNotFoundException e) {
        System.out.println("Отсутствует файл.");
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    return compar;
}

```

Далее изменим меню для запуска созданных методов. Для этого измените следующие пункты меню:

```

else if (menuItem == 5) {
    Saving(compar);
}
else if (menuItem == 6) {
    compar.getItem().clear();
    Loading(compar);
    compar.show();
}

```

При запуске программы, сначала добавляйте элементы в коллекцию, например, нажав 2 элемент меню.

Затем просмотрите результат, нажав на 1 элемент меню.

Затем сохранитесь в текстовый файл под именем **test1.txt**. Откройте этот файл в своей папке и проверьте, что данные действительно были туда записаны.

Затем нажмите на 6 элемент меню, с помощью которого будут добавлены объекты в коллекцию из текстового файла. Затем просмотрите результат, нажав на 1 элемент меню и убедитесь, что все объекты из текстового файла были добавлены.

7.3. Содержание отчета

Отчет о выполнении лабораторной работы должен включать:

1. Теоретические сведения о потоках в Java, чтения и записи файлов, обработки исключений ввода-вывода и закрытия файлов.
2. Распечатанный листинг выполнения программы, отражающий все этапы ее выполнения.
3. Выводы о выполненной работе.

7.4. Варианты заданий для самостоятельной работы

Вариант 1). Записная книжка контактов. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 2). Система управления доставкой товара. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 3). Телепрограмма. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 4). Гостиница. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 5). Реализация готовой продукции. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 6). Успеваемость студентов ВУЗА. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 7). Факультеты. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 8). Супермаркеты. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 9). Военный состав. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 10). Учет литературы. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 11). Учет продажи путевок. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 12). Учет выполненных работ станции техобслуживания. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из

текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 13). Медицинское обслуживание пациентов. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 14). Библиотека. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 15). Продажа автомобилей. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 16). Интернет-магазин. Выполнить преобразование класса в коллекцию. Создать пользовательское меню. Организовать добавление объектов в коллекцию и вывод отсортированных объектов коллекции на экран с помощью меню.

Вариант 17). Больница. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 18). Военно-морские учения. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 19). Туристические туры. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 20). Представления цирка. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 21). Аптека. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 22). Абонентская плата. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 23). Стоматология. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 24). Техобслуживание подвижного состава. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

Вариант 25). Кафедра. Выполнить преобразование класса в коллекцию. Организовать сохранение объектов из коллекции в текстовый файл и загрузку объектов из текстового файла в коллекцию. Изменить пользовательское меню для добавления работы с внешними файлами.

ЛАБОРАТОРНАЯ РАБОТА №8. ПАКЕТ JAVA.AWT

Цель работы. Получить прикладные навыки создания неконсольных приложений на языке Java, а также использования классов и методов пакета java.awt для отображения графических фигур на апплете.

8.1 Методические рекомендации

В данной лабораторной работе вы получите первое представление о создании неконсольных приложений на языке java, а также научитесь использовать классы и методы пакета java.awt для отображения графических фигур на апплете.

Лабораторная работа направлена на приобретение навыка создания апплетов, их инициализации и запуска как из среды разработки (**Eclipse**, <http://eclipse.org>), так и с помощью инструмента разработчика **appletviewer**.

В данной теме вы получите знания и практический опыт использования ранее разработанных классов в новых проектах, в частности при создании **апплета**.

Требования к результатам выполнения лабораторной работы:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- компиляцию, запуск программ выполнять различными способами;
- по завершении выполнения задания составить отчет о проделанной работе.

Теоретический материал

Эта лабораторная работа начинается с рассмотрения базовых библиотек Java, которые являются неотъемлемой частью языка и входят в его спецификацию [1], а именно описывается пакет **java.awt**, предоставляющий технологию **AWT** для создания **графического (оконного) интерфейса пользователя – GUI**. Ни одна современная программа, предназначенная для пользователя, не обходится без удобного, понятного, в идеале – красивого пользовательского интерфейса. С самой первой версии в Java существует специальная технология для создания GUI. Она называется **AWT, Abstract Window Toolkit**.

Поскольку Java-приложения предназначены для работы на разнообразных платформах, реализация графического пользовательского интерфейса (GUI) должна быть либо одинаковой для любой платформы, либо, напротив, программа должна иметь вид, типичный для данной операционной системы. В силу ряда причин, для основной библиотеки по созданию GUI был выбран второй подход [1]. Во-первых, это лишней раз показывало гибкость Java – действительно, пользователи разных платформ могли работать с одним и тем же Java-приложением, не меняя своих привычек. Во-вторых, такая реализация обеспечивала большую производительность, поскольку была основана на возможностях операционной системы. В частности, это означало и более компактный, простой, а значит, и более надежный код.

Библиотеку называли **AWT – Abstract Window Toolkit**. Слово abstract в названии указывает, что все стандартные компоненты не являются самостоятельными, а работают в связке с соответствующими элементами операционной системы [1].

GUI всегда собирается из готовых строительных блоков, хранящихся в библиотеках. В Java их называют общим термином компонент (component), поскольку все они являются подклассами **java.awt.Component**.

Класс Component

Абстрактный класс **Component** является базовым для всех компонентов AWT и описывает их основные свойства. Визуальный компонент в AWT имеет прямоугольную форму, может быть отображен на экране и может взаимодействовать с пользователем [1].

Рассмотрим основные свойства этого класса.

Положение

Положение компонента описывается двумя целыми числами (тип `int`) `x` и `y`. В Java ось `x` проходит горизонтально, направлена вправо, а ось `y` – вертикально, но направлена вниз, а не вверх, как принято в математике. Для описания положения компонента предназначен специальный класс – **Point** (точка). В этом классе определено два `public int` поля `x` и `y`, а также множество конструкторов и вспомогательных методов для работы с ними. Класс **Point** применяется во многих типах AWT, где надо задать точку на плоскости.

Для компонента эта точка задает положение левого верхнего угла.

Установить положение компонента можно с помощью метода **setLocation()**, который может принимать в качестве аргументов пару целых чисел, либо **Point**. Узнать текущее положение можно с помощью метода **getLocation()**, возвращающего **Point**, либо с помощью методов **getX()** и **getY()** [1].

Размер

Как было сказано, компонент AWT имеет прямоугольную форму, а потому его размер описывается также двумя целочисленными параметрами – **width** (ширина) и **height** (высота). Для описания размера существует специальный класс **Dimension** (размер), в котором определено два `public int` поля **width** и **height**, а также вспомогательные методы.

Установить размер компонента можно с помощью метода **setSize**, который может принимать в качестве аргументов пару целых чисел, либо **Dimension**. Узнать текущие размеры можно с помощью метода **getSize()**, возвращающего **Dimension**, либо с помощью методов **getWidth()** и **getHeight()**.

Совместно положение и размер компонента задают его границы. Область, занимаемую компонентом, можно описать либо четырьмя числами (`x`, `y`, **width**, **height**), либо экземплярами классов **Point** и **Dimension**, либо специальным классом **Rectangle** (прямоугольник).

Задать границу объекта можно с помощью метода **setBounds**, который может принимать четыре числа, либо **Rectangle**. Узнать текущее значение можно с помощью метода **getBounds()**, возвращающего **Rectangle**.

Видимость

Существующий компонент может быть как виден пользователю, так и быть скрытым. Это свойство описывается булевым параметром **visible**. Методы для управления – **setVisible**, принимающий булевский параметр, и **isVisible**, возвращающий текущее значение.

Доступность

Даже если компонент отображается на экране и виден пользователю, он может не взаимодействовать с ним. В результате события от клавиатуры, или мыши не будут получаться и обрабатываться компонентом. Такой компонент называется

disabled. Если же компонент активен, его называют **enabled**. Как правило, компонент некоторым образом меняет свой внешний вид, когда становится недоступным (например, становится серым, менее заметным), но, вообще говоря, это необязательно (хотя очень удобно для пользователя). Для изменения этого свойства применяется метод **setEnabled**, принимающий булевский параметр (true соответствует **enabled**, false – **disabled**), а для получения текущего значения – **isEnabled**.

Цвета

Разумеется, для построения современного графического интерфейса пользователя необходима работа с цветами. Компонент обладает двумя свойствами, описывающими цвета, – **foreground** и **background** цвета. Первое свойство задает, каким цветом выводить надписи, рисовать линии и т.д. Второе – задает цвет фона, которым закрашивается вся область, занимаемая компонентом, перед тем, как прорисовывается внешний вид.

Для задания цвета в AWT используется специальный класс **Color**. Этот класс обладает довольно обширной функциональностью, поэтому рассмотрим основные характеристики.

Цвет задается 3 целочисленными характеристиками, соответствующими модели RGB, – красный, зеленый, синий. Каждая из них может иметь значение от 0 до 255 (тем не менее, их тип определен как `int`). В результате (0, 0, 0) соответствует черному, а (255, 255, 255) – белому.

Класс **Color** является неизменяемым, то есть, создав экземпляр, соответствующий какому-либо цвету, изменить параметры RGB уже невозможно. Это позволяет объявить в классе **Color** ряд констант, описывающих базовые цвета: белый, черный, красный, желтый и так далее. Например, вместо того, чтобы задавать синий цвет числовыми параметрами (0, 0, 255), можно воспользоваться константами **Color.blue** или **Color.BLUE** (второй вариант появился в более поздних версиях).

Для работы со свойством компонента **foreground** применяют методы **setForeground** и **getForeground**, а для **background** – **setBackground** и **getBackground**.

Класс Container

Контейнер описывается классом **Container**, который является наследником **Component**, а значит, обладает всеми свойствами графического компонента. Однако основная его задача – группировать другие компоненты. Для этого в нем объявлен целый ряд методов.

Для добавления служит метод **add**, для удаления – **remove** и **removeAll** (последний удаляет все компоненты). Добавляемые компоненты хранятся в упорядоченном списке, поэтому для удаления можно указать либо ссылку на компонент, который и будет удален, либо его порядковый номер в контейнере.

Также определены методы для получения компонент, присутствующих в контейнере, – все они довольно очевидны, поэтому перечислим их с краткими пояснениями:

getComponent(int n) – возвращает компонент с указанным– порядковым номером;

getComponents() – возвращает все компоненты в виде массива;–

getComponentCount() – возвращает количество компонент;–

getComponentAt(int x, int y) или (**Point p**) – возвращает компонент,– который включает в себя указанную точку;

findComponentAt(int x, int y) или (Point p) – возвращает видимый– компонент, включающий в себя указанную точку.

Положение компонента (location) задается координатами левого верхнего угла. Важно, что эти значения отсчитываются от левого верхнего угла контейнера, который таким образом является центром системы координат для каждого находящегося в нем компонента. Если важно расположение компонента на экране безотносительно его контейнера, можно воспользоваться методом **getLocationOnScreen**.

Благодаря наследованию контейнер также имеет свойство **size**. Этот размер задается независимо от размера и положения вложенных компонент. Таким образом, компоненты могут располагаться частично или полностью за пределами своего контейнера (что это означает, будет рассмотрено ниже, но принципиально это допустимо).

Раз контейнер наследуется от **Component**, он сам является компонентом, а значит, может быть добавлен в другой, вышестоящий контейнер. В то же время компонент может находиться лишь в одном контейнере. Это означает, что все элементы сложного пользовательского интерфейса объединяются в иерархическое дерево. Такая организация не только облегчает операции над ними, но и задает основные свойства всей работы AWT. Одним из них является принцип отрисовки компонентов.

Алгоритм отрисовки компонента

Рассмотрим алгоритм отрисовки отдельного компонента, что определяет его внешний вид? Для этой задачи предназначен метод **paint**. Этот метод вызывается каждый раз, когда необходимо отобразить компонент на экране. У него есть один аргумент, тип которого – абстрактный класс **Graphics**. В этом классе определено множество методов для отрисовки простейших графических элементов – линий, прямоугольников и многоугольников, окружностей и овалов, текста, картинок и т.д.

Наследники класса **Component** переопределяют метод **paint** и, пользуясь методами **Graphics**, задают алгоритм прорисовки своего внешнего вида:

```
public void paint(Graphics g) {  
    g.drawLine(0, 0, getWidth(), getHeight());  
    g.drawLine(0, getHeight(), getWidth(), 0);  
}
```

Ключевым классом при выполнении всех графических операций является **Graphics**. Назначение класса:

- определяет поверхность рисования;
- определяет методы рисования;
- определяет атрибуты для методов рисования.

Рассмотрим некоторые методы класса **Graphics**.

drawLine(x1, y1, x2, y2)

Этот метод отображает линию толщиной в 1 пиксел, проходящую из точки (x1, y1) в (x2, y2).

drawRect(int x, int y, int width, int height)

Этот метод отображает прямоугольник, чей левый верхний угол находится в точке (x, y), а ширина и высота равняются width и height соответственно. Правая сторона пройдет по линии x+width, а нижняя – y+height.

fillRect(int x, int y, int width, int height)

Этот метод закрашивает прямоугольник. Левая и правая стороны прямоугольника проходят по линиям x и $x+width-1$ соответственно, а верхняя и нижняя – y и $y+height-1$ соответственно. Таким образом, чтобы зарисовать все пиксели компонента, необходимо передать следующие аргументы: `g.fillRect(0, 0, getWidth(), getHeight());`

drawOval(int x, int y, int width, int height)

Этот метод рисует овал, вписанный в прямоугольник, задаваемый указанными параметрами. Очевидно, что если прямоугольник имеет равные стороны (т.е. является квадратом), овал становится окружностью.

fillOval(int x, int y, int width, int height)

Этот метод закрашивает указанный овал.

drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Этот метод рисует дугу – часть овала, задаваемого первыми четырьмя параметрами. Дуга начинается с угла `startAngle` и имеет угловой размер `arcAngle`. Начальный угол соответствует направлению часовой стрелки, указывающей на 3 часа. Угловой размер отсчитывается против часовой стрелки. Таким образом, размер в 90 градусов соответствует дуге в четверть овала (верхнюю правую). Углы "растянуты" в соответствии с размером прямоугольника. В результате, например, угловой размер в 45 градусов всегда задает границу дуги по линии, проходящей из центра прямоугольника в его правый верхний угол.

fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Этот метод закрашивает сектор, ограниченный дугой, задаваемой параметрами.

drawString(String text, int x, int y)

Этот метод выводит на экран текст, задаваемый первым параметром. Точка (x , y) задает позицию самого левого символа. Для наглядности приведем пример:

```
g.drawString("abcdefgh", 15, 15);
```

```
g.drawLine(15, 15, 115, 15);
```

Методы repaint и update

Кроме **paint** в классе **Component** объявлены еще два метода, отвечающие за прорисовку компонента. Как было рассмотрено, вызов **paint** инициируется операционной системой, если возникает необходимость перерисовать окно приложения, или часть его. Однако может потребоваться обновить внешний вид, руководствуясь программной логикой. Например, отобразить результат операции вычисления, или работы с сетью. Можно изменить состояние компонента (значение его полей), но операционная система не отследит такое изменение и не инициирует перерисовку.

Для программной инициализации перерисовки компонента служит метод **repaint**. Конечно, у него нет аргумента типа **Graphics**, поскольку программист не должен создавать экземпляры этого класса (точнее, его наследников, ведь **Graphics** – абстрактный класс). Метод **repaint** можно вызывать без аргументов. В этом случае компонент будет перерисован максимально быстро. Можно указать аргумент типа `long` – количество миллисекунд. Система инициализирует перерисовку спустя указанное время. Можно указать четыре числа типа `int` (x , y , $width$, $height$), задавая прямоугольную область компонента, которая нуждается в перерисовке. Наконец, можно указать все 5 параметров – и задержку по времени, и область перерисовки.

Если перерисовка инициируется приложением, то система вызывает не метод **paint**, а метод **update**. У него уже есть аргумент типа **Graphics** и по умолчанию он

лишь закрашивает всю область компонента фоновым цветом (свойство **background**), а затем вызывает метод **paint**. Зачем же было вводить этот дополнительный метод, если можно было сразу вызвать **paint**? Дело в том, что поскольку перерисовка инициируется приложением, для сложных компонентов становится возможной некоторая оптимизация обновления внешнего вида.

Например, если изменение заключается в появлении нового графического элемента, то можно избежать повторной перерисовки остальных элементов — переопределить метод **update** и реализовать в нем отображение одного только нового элемента. Если же компонент имеет простую структуру, можно оставить метод **update** без изменений.

Апплеты

Кроме приложений, язык Java позволяет создавать **апплеты (applets)**. Это программы, работающие в среде другой программы — браузера.

Апплеты не нуждаются в окне верхнего уровня — им служит окно браузера.

Они не запускаются JVM — их загружает браузер, который сам запускает JVM для выполнения апплета. Эти особенности отражаются на написании программы апплета [2].

С точки зрения языка Java, **апплет** — это всякое расширение класса **Applet**, который, в свою очередь, расширяет **класс panel**. Таким образом, апплет — это панель специального вида, контейнер для размещения компонентов с дополнительными свойствами и методами.

Менеджером размещения компонентов по умолчанию, как и в классе **Panel**, служит **FlowLayout**. Класс **Applet** находится в пакете **java.applet**, в котором кроме него есть только три интерфейса, реализованные в браузере.

Надо заметить, что не все браузеры реализуют эти интерфейсы полностью. Поскольку JVM не запускает апплет, отпадает необходимость в методе **main()**, его нет в апплетах. В апплетах редко встречается конструктор. Дело в том, что при запуске первого создается его контекст. Во время выполнения конструктора контекст еще не сформирован, поэтому не все начальные значения удастся определить в конструкторе.

Начальные действия, обычно выполняемые в конструкторе и методе **main()**, в апплете записываются в метод **init()** класса **Applet**. Этот метод автоматически запускается исполняющей системой Java браузера сразу же после загрузки апплета.

Вот как он выглядит в исходном коде класса **Applet**:

```
public void init(){} 
```

Метод **init()** не имеет аргументов, не возвращает значения и должен переопределяться в каждом апплете — подклассе **класса Applet**. Обратные действия — завершение работы, освобождение ресурсов — записываются при необходимости в метод **destroy()**, тоже выполняющийся автоматически при выгрузке апплета. В классе **Applet** есть пустая реализация этого метода.

Кроме методов **init()** и **destroy()** в классе **Applet** присутствуют еще два пустых метода, выполняющихся автоматически. Браузер должен обращаться к методу **start()** при каждом появлении апплета на экране и обращаться к методу **stop()**, когда апплет уходит с экрана. В методе **stop()** можно определить действия, приостанавливающие работу апплета, в методе **start()** — возобновляющие ее. Надо сразу же заметить, что не все браузеры обращаются к этим методам как должно [2].

Метод **paint(Graphics g)** вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод **paint(Graphics g)**.

Перерисовка содержимого апплета выполняется методом **update()**. Для инициации **update()** предусмотрены несколько вариантов метода **repaint**, который в свою очередь вызывает метод **update**:

```
repaint();  
repaint(time);  
repaint(x, y, w, h);  
repaint(time, x, y, w, h);
```

Приведем пример простого апплета, выводящего текст на экран.

```
import java.awt.*;  
import java.applet.*;  
public class HelloWorld extends Applet{  
    public void paint(Graphics g)  
    {  
        g.drawString("Hello, XXI century World!!!", 10, 30);  
    }  
}
```

Эта программа записывается в файл **HelloWorld.java** и компилируется как обычно: **javac HelloWorld.java**.

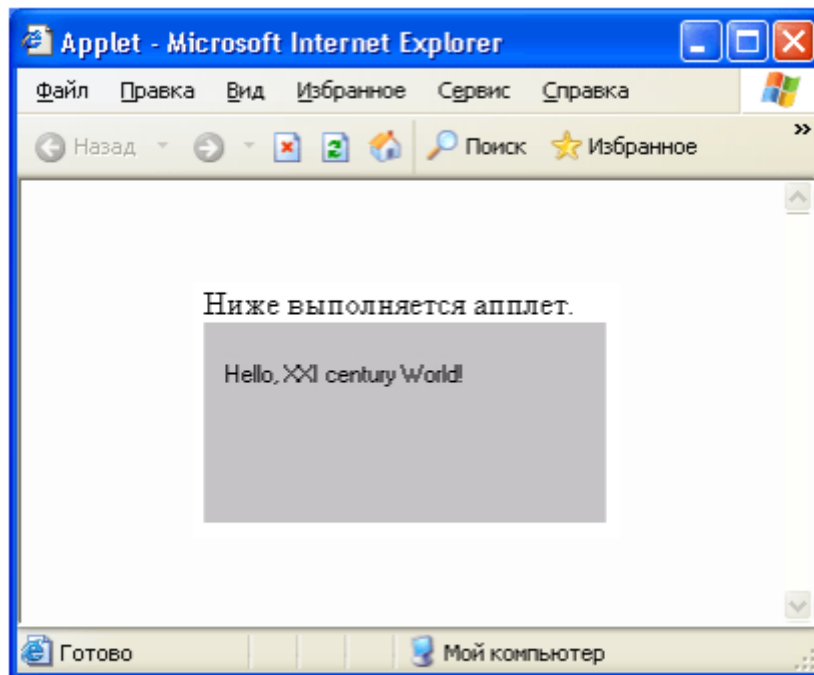
Компилятор создает файл **HelloWorkLclass**, но воспользоваться для его выполнения интерпретатором **java** теперь нельзя — нет метода **main()**. Вместо интерпретации надо дать указание браузеру для запуска апплета.

Все указания браузеру даются пометками, тегами (tags), на языке HTML (HyperText Markup Language). В частности, указание на запуск апплета дается в теге **<applet>**. В нем обязательно задается имя файла с классом апплета параметром **code**, ширина **width** и высота **height** панели апплета в пикселах. Полностью текст HTML для запуска нашего апплета на странице браузера приведен ниже:

```
<html>  
<head><title> Applet</title></head> <body>  
Ниже выполняется апплет.<br>  
<applet code = "HeiioWorid.class" width = "200" height = "100">  
</applet>  
</body>  
</html>
```

Этот текст заносится в файл с расширением **html** или **htm**, например, **HelloWorld.html**. Имя файла произвольно, никак не связано с апплетом или классом апплета. Оба файла — **HelloWorld.html** и **HelloWorld.class** — помещаются в один каталог на сервере, и файл **HelloWorld.html** загружается в браузер, который может находиться в любом месте Internet. Браузер, просматривая HTML-файл, выполнит тег и загрузит апплет. После загрузки апплет появится в окне браузера, как показано ниже [2].

(Hello.zip)



Апплет HelloWorld в окне Internet Explorer

В этом простом примере можно заметить еще две особенности апплетов. Во-первых, размер апплета задается не в нем, а в теге **<applet>**. Это очень удобно, можно менять размер апплета, не компилируя его заново. Можно организовать апплет невидимым, сделав его размером в один пиксел. Кроме того, размер апплета разрешается задать в процентах по отношению к размеру окна браузера, например

```
<applet code = "HelloWorld.class" width = "100%" height = "100%">
```

Во-вторых, как видно на рисунке, у апплета серый фон. Такой фон был в первых браузерах, и апплет не выделялся из текста в окне браузера. Теперь в браузерах принят белый фон, его можно установить обычным для компонентов методом **setBackground(Color.white)**, обратившись к нему в методе **init ()**. В настоящее время синтаксис тега **<APPLET>** таков :

```
<APPLET  
CODE = appletFile  
WIDTH = pixels  
HEIGHT = pixels  
[ARCHIVE = jarFiles]  
[CODEBASE = codebaseURL]  
[ALT = alternateText]  
[NAME = appletInstanceName]  
[ALIGN = alignment]  
[VSPACE = pixels]  
[HSPACE = pixels]  
>  
[HTML-текст, отображаемый при отсутствии поддержки Java]  
</APPLET>
```

CODE = appletClassFile ; **CODE** – обязательный атрибут, задающий– имя файла, в котором содержится описание класса апплета. Имя файла задается относительно codebase, то есть либо от текущего каталога, либо от каталога, указанного в атрибуте **CODEBASE**.

WIDTH = pixels

HEIGHT = pixels ; **WIDTH** и **HEIGHT** - обязательные атрибуты,– задающие размер области апплета на HTML -странице.

ARCHIVE = jarFiles ; Этот необязательный атрибут задает список jar– -файлов (разделяется запятыми), которые предварительно загружаются в Web -браузер. В них могут содержаться классы, изображения, звук и любые другие ресурсы, необходимые апплету. Архивирование наиболее необходимо именно апплетам, так как их код и ресурсы передаются через сеть.

CODEBASE = codebaseURL ;

CODEBASE – необязательный атрибут,– задающий базовый URL кода апплета; является каталогом, в котором будет выполняться поиск исполняемого файла апплета (задаваемого в признаке **CODE**). Если этот атрибут не задан, по умолчанию используется каталог данного HTML -документа. С помощью этого атрибута можно на странице одного сайта разместить апплет, находящийся на другом сайте.

ALT = alternateAppletText ; Признак **ALT** – необязательный атрибут,– задающий короткое текстовое сообщение, которое должно быть выведено (как правило, в виде всплывающей подсказки при нахождении курсора мыши над областью апплета) в том случае, если используемый браузер распознает синтаксис тега <applet>, но выполнять апплеты не умеет. Это тоже самое, что HTML – текст, который можно вставлять между <applet> и </applet> для браузеров, вообще не поддерживающих апплетов.

NAME = appletInstanceName ; **NAME** – необязательный атрибут,– используемый для присвоения имени данному экземпляру апплета. Имена апплетам нужны для того, чтобы другие апплеты на этой же странице могли находить их и общаться с ними, а также для обращений из Java Script.

ALIGN = alignment

VSPACE = pixels

HSPACE = pixels ; Эти три необязательных атрибута предназначены– для того же, что и в теге **IMG**. **ALIGN** задает стиль выравнивания апплета, возможные значения: **LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM**.

В состав JDK любой версии входит программа **appletviewer**. Это простейший браузер, предназначенный для запуска апплетов в целях отладки. Если под рукой нет Internet-браузера, можно воспользоваться им. **Appletviewer** запускается из командной строки:

appletviewer HelloWorld.html

На рисунке **appletviewer** показывает апплет **HelloWorld**.

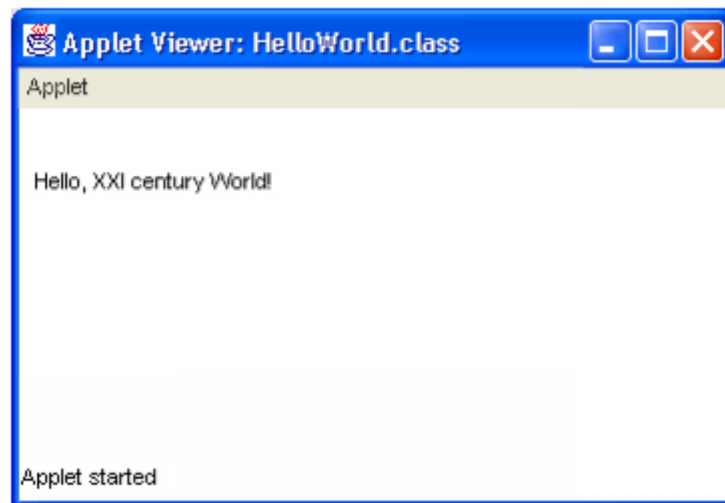


Рисунок. Апплет HelloWorld в окне программы appletviewer

Приведем пример невидимого апплета. В нижней строке браузера — строке состояния (**status bar**) — отражаются сведения о загрузке файлов.

Апплет может записать в нее любую строку **str** методом **showstatus(string str)**.

// Файл RunningString.Java

```
import java.awt.*;
import java.applet.*;
public class Runningstring extends Applet{
private Boolean go;
public void start(){
go = true;
sendMessage("Эта строка выводится апплетом"); }
```

```
public void sendMessage(String s){
String s1 = s+" ";
while(go){
showStatus(s);
try{
Thread.sleep(200); }
catch(Exception e){}
s = s1.substring(1)+s.charAt(0);
s1 =s; } }
public void stop(){ go = false; } }
```

(Running.zip)

Следующий апплет имеет более сложную структуру: по экрану движется круг, который упруго отражается от границ области. Движение происходит непрерывно. Особенность данного апплета состоит в том, что он реализован с помощью дополнительного потока **Thread**, который отвечает за движение круга. Класс апплета реализует интерфейс **Runnable**. Программный код приведен ниже (ссылка на файл **bouncingcircle.java**).

```
import java.applet.*;
```

```

import java.awt.*;
//Создаем собственный класс,который наследуется от класса Applet
//Данный класс реализует методы интерфейса Runnable

public class BouncingCircle extends Applet implements Runnable {
int x = 150, y = 50, r = 50; // Центр и радиус круга
int dx = 11, dy = 7; // Движение круга по горизонт и вертик
Thread animator; // Нить, которая осуществляет анимацию
volatile boolean pleaseStop; // Флаг остановки движения

//Метод для рисования окружности красным цветом
public void paint(Graphics g) {
g.setColor(Color.red); //установка цвета для g
g.fillOval(x-r, y-r, r*2, r*2); //прорисовка круга
}

//Метод двигает круг и "отражает" его при ударе круга о стенку,
//затем вызывает перерисовку.
//Данный метод вызывается многократно анимационным потоком

public void animate() {
Rectangle bounds = getBounds(); //Получение размера окна программы
if ((x - r + dx < 0) || (x + r + dx > bounds.width)) dx = -dx;
if ((y - r + dy < 0) || (y + r + dy > bounds.height)) dy = -dy;

// Изменение координат круга, по сути - движение.
x += dx; y += dy;

//Просим браузер вызвать метод paint() для отрисовки
// круга в новой позиции
repaint(); }

/*Это метод из интерфейса Runnable. Это тело потока исполнения,
осуществляющего анимацию. Сам поток создается и запускается методом
start() */
public void run() {
while(!pleaseStop) { // Цикл до тех пор, пока не будет
//команды остановиться.
animate(); // Обновляем положение и перерисовываем
try { Thread.sleep(100); } // Ждем 100 миллисекунд
catch(InterruptedException e) {} // Игнорируем прерывания } }

//Запускаем анимацию при запуске апплета браузером
public void start() {
animator = new Thread(this); // Создаем поток исполнения
pleaseStop = false; // Пока не просим его остановиться
animator.start(); // Запускаем поток

```

```
}  
//Останавливаем анимацию, когда браузер останавливает апплет  
  
public void stop() { pleaseStop = true; } }
```

Данный апплет работает следующим образом:
(BouncingCircle)

8.2. Варианты заданий

1. Изучить предлагаемый теоретический материал;

2. Создайте следующие программы:

1) Запустить апплет, рассмотренный в примере презентации (HelloWorld), используя два метода запуска – через html-страницу и через appletviewer

2) Модифицировать код программы bouncingcircle таким образом, чтобы вместо круга движение осуществлял экземпляр реализованного ранее (в лабораторной работе №2) класса ColorableRect(), как показано ниже: (BouncingBox). о Модифицировать код предыдущей программы таким образом, чтобы движение осуществляли сразу 10 экземпляров класса Rectangle, 10 класса DrawableRect и 10 экземпляров класса ColorableRect. Все созданные объекты должны храниться в одном массиве с типом класса родителя Rectangle. Ниже показано, как должен выглядеть апплет. (BouncingBox2).

8.3. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- 1 **Васильев, А.Н.** Java. Объектно-ориентированное программирование для магистров и бакалавров: базовый курс по объектно-ориентир. программированию : учеб. пособие / А.Н. Васильев. – М. ; СПб. ; Нижний Новгород : Питер, 2013. – 396 с.
- 2 **Вязовик, Н.А.** Программирование на Java. [электронный ресурс] <http://www.intuit.ru/department/pl/javapl>.
- 3 Введение в программирование Java [Электронный ресурс]. (<https://www.ibm.com/developerworks/ru/java/newto/>).
- 4 **Дж. Стивен Перри.** Введение в Java-программирование: Часть 2. Конструкции реальных приложений [Электронный ресурс] (<https://www.ibm.com/developerworks/ru/edu/j-introtojava2/index.html>).
- 5 **Зыков, С.В.** Программирование. Объектно-ориентированный подход: учебник и практикум для академического бакалавриата / С.В. Зыков. – М. Научная школа: Национальный исследовательский университет «Высшая школа экономики», 2017. – 155 с. [электронный ресурс]. <https://biblio-online.ru/book/E006A65E-B936-4856-B49E-1BA48CF1A52F>.
- 6 **Николаев, Е.И.** Объектно-ориентированное программирование: учебное пособие. – Ставрополь: Изд-во СКФУ, 2015. – 255 С. [электронный ресурс] <http://www.knigafund.ru/books/200468/>
- 7 **Тузовский, А.Ф.** Объектно-ориентированное программирование: учебное пособие для прикладного бакалавриата / Тузовский А.Ф. – Томск. Научная школа: Национальный исследовательский Томский политехнический , 2017. – 206 с. [электронный ресурс] <https://biblio-online.ru/book/BDEEFB2D-532D-4306-829E-5869F6BDA5F9>.
- 8 **Хорстманн К.С., Корнелл Г.** Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
- 9 **Эккель, Б.** Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
- 10 **Шилдт, Г.** Java. Полное руководство, 8-е изд.: Пер.с англ. – М.: ООО «И.Д. Вильямс», 2012 . – 1104 с.
- 11 **Шилдт, Г.** Java 8. Руководство для начинающих, 9-е изд.: Пер.с англ. – М.: ООО «И.Д. Вильямс», 2016 . – 720 с.
- 12 **Нимейер, П.** Программирование на Java. – М.: Эксмо, 2014. – 1214 с.