ЛАБОРАТОРНАЯ РАБОТА №1.
Разработка консольного приложения для изучения типов данных и операторов. Документирование кода.

1. Цель работы

Получить общее представление о создании программ на языке Java и познакомиться с его основными понятиями. Изучить синтаксические единицы, основные операторы и структуру кода программы. Освоить способы компиляции исходного кода и запуска программы.

2. Методические указания

Лабораторная работа направлена на приобретение навыка написания программ на языке Java, а также умения выполнять компиляцию и запуск программы как из среды разработки (Eclipse, http://eclipse.org), так и из командной строки.

Требования к результатам выполнения лабораторного практикума:

- 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.

3. Теоретический материал

Язык *Java* ворвался в Интернет в конце 1995 года и немедленно завоевал популярность. Он обещал стать универсальным средством, обеспечивающим связь пользователей с любыми источниками информации, независимо от того, где она расположена – на *Web*-сервере, в базе данных, хранилище данных и т.д. Этот хорошо разработанный объектно-ориентированный язык программирования поддержали все производители программного обеспечения. Он имеет встроенные средства, позволяющие решать задачи повышенной сложности такие как: работа с сетевыми ресурсами, управление базами данных, динамическое наполнение *web*-страниц, многопоточность приложений.

Инсталляция набора инструментальных средств *Java Software Development Kit.*

JDK долгое время был базовым средством разработки приложений. Он не содержит никаких текстовых редакторов, а оперирует только с уже существующими *java*-файлами. Компилятор представлен утилитой *javac (java compiler),* виртуальная машина реализована программой *java*. Для тестовых запусков аплетов есть специальная утилита *appletviewer*.

Пакет *Java Software Development Kit* можно загрузить с *web*-страницы:

http://java.sun.com/javase/downloads/index.jsp. Способы инсталляции на разных платформах (*Solaris, Windows, Linux*) отличаются друг от друга.

После инсталляции пакета *JSDK* нужно добавить имя каталога *jdk\bin* в список путей, по которым операционная система может найти выполняемые файлы. Правильность установки пакета можно проверить, набрав команду *java–version*. На экране должно появиться, примерно, следующее:

java version "1.5.0_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_01-b08)
Java HotSpot(TM) Client VM (build 1.5.0_01-b08, mixed mode, sharing)

Среда разработки программ

Для написания программ на языке *Java* достаточно использовать самый простой текстовый редактор, однако применение специализированных средств разработки (Eclipse, Java WorkShop, Java Studio и др.) предоставляет большой набор полезных и удобных функций. Существует несколько способов компиляции и запуска на выполнение программ, написанных на языке *Java*: из командной строки или из другой программы, например, интегрированной среды разработки. Для компиляции программы из командной строки

необходимо вызвать компилятор, набрав команду *javac* и указав через пробел имена компилируемых файлов:

javac file1.java file2.java file3.java

При успешном выполнении этапа компиляции в директории с исходными кодами появятся файлы с расширением *.class*, которые являются *java* байт-кодом. Виртуальная *Java*-машина (*JVM*) интерпретирует байт-код и выполняет программу. Для запуска программы необходимо в *JVM* загрузить основной класс, т.е. класс, который содержит функцию *main(String s[]).*

Например, если в файле *file1.java* есть функция *main(),* которая располагается в классе *file1*, то для запуска программы после этапа компиляции необходимо набрать следующее:

java file1

Компиляцию и запуск программ из интегрированных сред разработки необходимо осуществлять в соответствии с документацией на программный продукт.

Анализ программы

Технология *Java*, как платформа, изначально спроектированная для Глобальной сети *Internet*, должна быть многоязыковой, а значит, обычный набор символов *ASCII* (*American Standard Code for Information Interchange*, Американский стандартный код обмена информацией), включающий в себя лишь латинский алфавит, цифры и простейшие специальные знаки (скобки, знаки препинания, арифметические операции и т.д.), недостаточен. Поэтому для записи текста программы применяется более универсальная кодировка *Unicode*. Например, если в программу нужно вставить знак с кодом 6917, необходимо его представить в шестнадцатеричном формате (1B05) и записать: \u1B05.

Компилятор, анализируя программу, сразу разделяет ее на:

- пробелы (*white spaces*);
- комментарии (*comments*);
- основные лексемы (*tokens*).

Пробелами в данном случае называют все символы, разбивающие текст программы на лексемы. Это как сам символ пробела (*space*, \u0020, десятичный код 32), так и знаки табуляции и перевода строки. Они используются для разделения лексем, а также для оформления кода, чтобы его было легче читать. Например, следующую часть программы (вычисление корней квадратного уравнения):

```
double a = 1, b = 5, c = 6;
double D = b * b - 4 * a * c;
if (D >= 0) {
double x1 = (-b + Math.sqrt (D)) / (2 * a);
double x2 = (-b - Math.sqrt (D)) / (2 * a);
}
можно записать и в таком виде:
double a=1,b=5,c=6;double D=b*b-4*a*c;if(D>=0)
{double x1=(-b+Math.sqrt(D))/(2*a);double
x2=(-b-Math.sqrt(D))/(2*a);}
```

В обоих случаях компилятор сгенерирует абсолютно одинаковый код.

Единственное соображение, которым должен руководствоваться разработчик, - легкость чтения и дальнейшей поддержки такого кода.

Комментарии не влияют на результирующий бинарный код и используются только для ввода пояснений к программе. В *Java* комментарии бывают двух видов: строчные и блочные. Строчные комментарии начинаются с *ASCII*-символов // и длятся до конца текущей строки, например:

```
int y=1994; // год рождения
```

Блочные комментарии располагаются между *ASCII*-символами /* и */, могут

занимать произвольное количество строк. Кроме этого, существует особый вид блочного комментария – комментарий разработчика (/**комментарии*/). Он применяется для автоматического создания документации кода [1].

Лексика языка

Лексика описывает, из чего состоит текст программы, каким образом он записывается и на какие простейшие слова (лексемы) компилятор разбивает программу при анализе. Лексемы (или *tokens* в английском варианте) – это основные "кирпичики", из которых строится любая программа на языке *Java* [1]. Ниже перечислены все ***виды лексем*** в *Java*:

- идентификаторы (*identifiers*);
- ключевые слова (*key words*);
- литералы (*literals*);
- разделители (*separators*);
- операторы (*operators*).

Идентификаторы – это имена, которые даются различным элементам языка для упрощения доступа к ним. Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные. Длина имени не ограничена. Идентификатор состоит из букв и цифр. Имя не может начинаться с цифры.

Ключевые слова – специальные лексемы, зарегистрированные в системе для внутреннего использования, такие как *abstract, default, if, private, this, boolean, implements, protected, static, try, void, native* и др.

Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений, а также *null*-литералов. В *Java* определены следующие ***виды литералов***:

- целочисленный (*integer*);
- дробный (*floating-point*);
- булевский (*boolean*);
- символьный (*character*);
- строковый (*string*);
- null-литерал (*null-literal*).

Целочисленные (тип *int* занимает 4 байта, тип *long* – 8) литералы позволяют задавать целочисленные значения в десятеричном, восьмеричном и шестнадцатеричном виде. Запись нуля можно осуществить следующими способами:

- 0 (10-ричная система)
- 00 (8-ричная)
- 0x0 (16-ричная)

Если в конце литерала не стоит указателя на тип, то литерал по умолчанию имеет тип *int*.

Дробные литералы (тип *float* занимает 4 байта, тип *double* – 8) представляют собой числа с плавающей десятичной точкой. Дробный литерал состоит из следующих составных частей (по умолчанию имеет тип *double*):

- целая часть;
- десятичная точка (используется *ASCII*-символ точка);
- дробная часть;
- показатель степени (состоит из латинской *ASCII*-буквы «*E*» в произвольном регистре и целого числа с опциональным знаком «+» или «–»);
- окончание-указатель типа (*D* или *F*).

Символьные литералы. Представляют собой один символ и заключаются в одинарные кавычки *‘s’, ‘a’*. Допускается запись через *Unicode* ‘\u0041’ – латинская буква “*A*”.

Строковые литералы состоят из набора символов и записываются в двойных кавычках: “символьный литерал”. *Null* литерал может принимать всего одно значение: *null*. Это литерал ссылочного типа, причем эта ссылка никуда не ссылается.

Разделители – специальные символы, используемые в конструкциях языка “()”, “[]”, “{}”, “;”, “,”, “.”.

Операторы используются в различных операциях – арифметических, логических, битовых, операциях сравнения и присваивания.

Пример простой программы *“Hello, world!”* выглядит следующим образом:

```
public class Test {
   /**
    * Основной метод, с которого начинается
    * выполнение любой Java программы.
    */
   public static void main(String args[])
   {
      System.out.println("Hello, world!");
   }
}
```

Типы данных

Java является строго типизированным языком. Это означает, что любая переменная и любое выражение имеют известный тип еще на момент компиляции. Такое строгое правило позволяет выявлять многие ошибки уже во время компиляции. Компилятор, найдя ошибку, указывает точную строку и причину ее возникновения, а динамические «баги» необходимо сначала выявить тестированием, а затем найти место в коде, которое их породило.

Все типы данных разделяются на две группы. Первую составляют 8 простых или примитивных (от английского *primitive*) типов данных [1-3]. Они подразделяются на три подгруппы:

- целочисленные: *byte, short, int, long, char*;
- дробные: *float, double;*
- булевский: *boolean*.

Булевский тип представлен всего одним типом *boolean*, который может хранить всего два возможных значения – *true* и *false*. Величины именно этого типа получаются в результате операций сравнения.

Вторую группу составляют объектные или ссылочные (от английского *reference*) типы данных. Это все классы, интерфейсы и массивы.

Переменные

Переменные используются в программе для хранения данных. Любая переменная имеет три базовых характеристики: имя, тип, значение. Имя уникально идентифицирует переменную и позволяет к ней обращаться в программе. Тип описывает, какие величины может хранить переменная. Значение – текущая величина, хранящаяся в переменной на данный момент. Значение может быть указано сразу (инициализация), а в большинстве случаев задание начальной величины можно и отложить.

```
Int a;
int b=0, c=2+3;
double d=e=5.5;
```

Ниже приведены данные по всем целым и дробным типам:

Название типа	**Длина (байт)**	**Область значений**
Целые типы		

byte	1	-128..127
short	2	-32768..32767
int	4	-2147483648..2147483647
long	8	-9223372036854775808..9223372036854775807
char	2	0..65535
Дробные типы		
float	4	3.40282347e+38f; 1.40239846e-45f
double	8	1.79769313486231570e+308; 4.94065645841246544e-324

```
/* # 1 # Пример. Типы данных, литералы и операции над ними */
public class Primer1 {
   public static void main(String[] args) {
      byte b = 1, b1 = 1 + 2;
      final byte B = 1 + 2;
      // b = b1 + 1; // ошибка приведения типов int в byte
      /*
       * переменная b1 на момент выполнения кода b = b1 + 1; может измениться, и
       * выражение b1 + 1 может превысить допустимый размер byte- типа
       */
      b = (byte) (b1 + 1);
      b = B + 1; // работает
      /*
       * B - константа, ее значение определено, компилятор вычисляет значение
       * выражения B + 1, и если его размер не превышает допустимого для byte типа, то
       * ошибка не возникает
       */
      // b = -b; // ошибка приведения типов
      b = (byte) -b;
      // b = +b; // ошибка приведения типов
      b = (byte) +b;
      int i = 3;
      // b = i; // ошибка приведения типов, int больше, чем byte
      b = (byte) i;
      final int D = 3;
      b = D; // работает
      /*
       * D –константа. Компилятор проверяет, не превышает ли ее значение допустимый
       * размер для типа byte, если не превышает, то ошибка не возникает
       */
      final int D2 = 129;
      // b=D2; // ошибка приведения типов, т.к. 129 больше, чем допустимое 127
      b = (byte) D2;
      b += i++; // работает
      b += 1000; // работает
      b1 *= 2; // работает
```

```
        float f = 1.1f;
        b /= f; // работает
        /*
         * все сокращенные операторы автоматически преобразуют результат выражения к
         * типу переменной, которой присваивается это значение. Например, b /= f;
         * равносильно b = (byte)(b / f);
         */
    }
}
```

Документирование кода

В языке Java используются блочные и однострочные комментарии /* */ и //, аналогичные комментариям, применяемым в C++. Введен также новый вид комментария /** */, который может содержать описание документа с помощью дескрипторов вида:

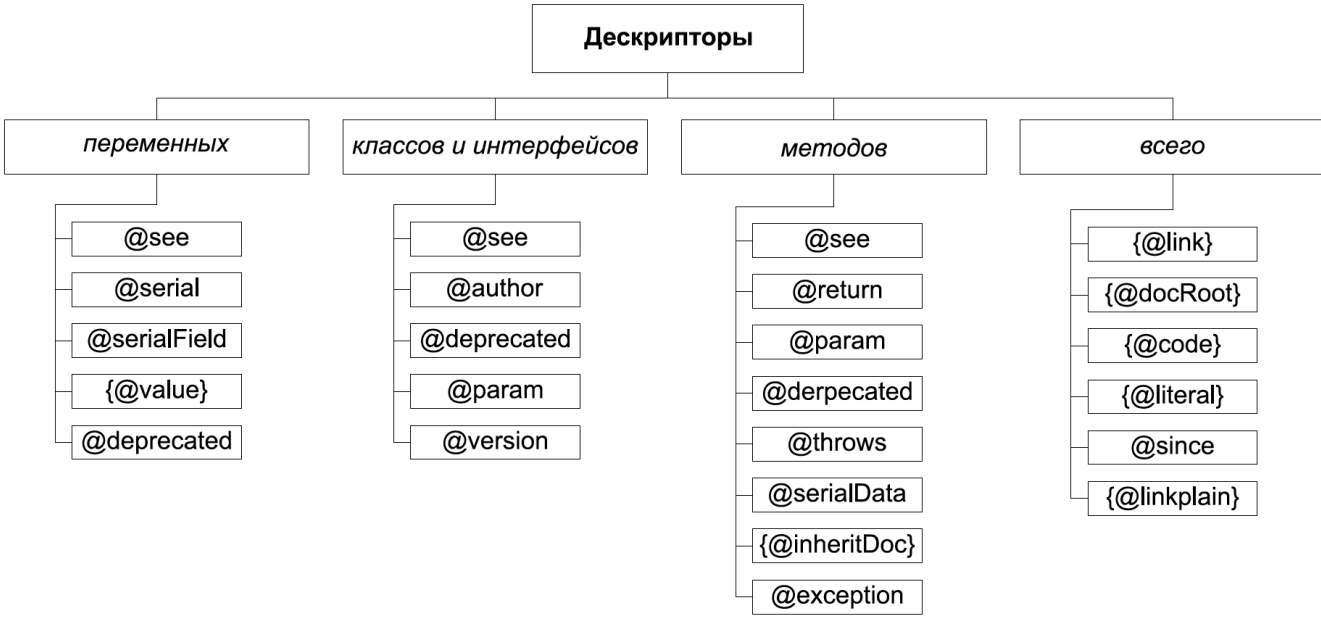

Рисунок 1.1. Дескрипторы документирования кода

@author – задает сведения об авторе;
@version – задает номер версии класса;
@exception – задает имя класса исключения;
@param – описывает параметры, передаваемые методу;
@return – описывает тип, возвращаемый методом;
@deprecated – указывает, что метод устаревший и у него есть более совершенный аналог;
@since – определяет версию, с которой метод (член класса, класс) присутствует;
@throws – описывает исключение, генерируемое методом;
@see – что следует посмотреть дополнительно.

Из *java*-файла, содержащего такие комментарии, соответствующая утилита javadoc.exe может извлекать информацию для документирования классов и сохранения ее в виде *html*-документа. В качестве примера и образца можно рассматривать исходный код языка *Java* и документацию, сгенерированную на его основе.

/* # 2 # фрагмент класса Object с дескрипторами документирования # Object.java */

```
package java.lang;
```

```
/**
* Class {@code Object} is the root of the class hierarchy.
* Every class has {@code Object} as a superclass. All objects,
* including arrays, implement the methods of this class.
*
* @author unascribed
* @see java.lang.Class
* @since JDK1.0
*/
public class Object {
/**
* Indicates whether some other object is "equal to" this one.
* <p>
* MORE COMMENTS HERE
* @param obj the reference object with which to compare.
* @return {@code true} if this object is the same as the obj
* argument; {@code false} otherwise.
* @see #hashCode()
* @see java.util.HashMap
*/
public boolean equals(Object obj) {
return (this == obj);
}
/**
* Creates and returns a copy of this object.
* MORE COMMENTS HERE
* @return a clone of this instance.
* @exception CloneNotSupportedException if the object's class does not
* support the {@code Cloneable} interface. Subclasses
* that override the {@code clone} method can also
* throw this exception to indicate that an instance cannot
* be cloned.
* @see java.lang.Cloneable
*/
protected native Object clone() throws CloneNotSupportedException;
// more code here
}
```

Всегда следует помнить, что точные названия классов, их полей и методов улучшают восприятие кода и уменьшают размер комментариев. Наличие комментария должно еще больше облегчить скорость восприятия разработанного кода.

Код системы будет читаться чаще и больше по времени, чем требуется на его создание. Комментарии помогут программисту, сопровождающему код, быстрее разобраться в нем и грамотнее использовать или изменять его.

Классы–оболочки

Кроме базовых типов данных, в языке *Java* широко используются соответствующие классы-оболочки (wrapper-классы) из пакета *java.lang: Boolean,Character, Integer, Byte, Short, Long, Float, Double*. Объекты этих классов могут хранить те же значения, что и соответствующие им базовые типы.

Объект любого из этих классов представляет собой полноценный экземпляр в динамической памяти, в котором хранится его неизменяемое значение.

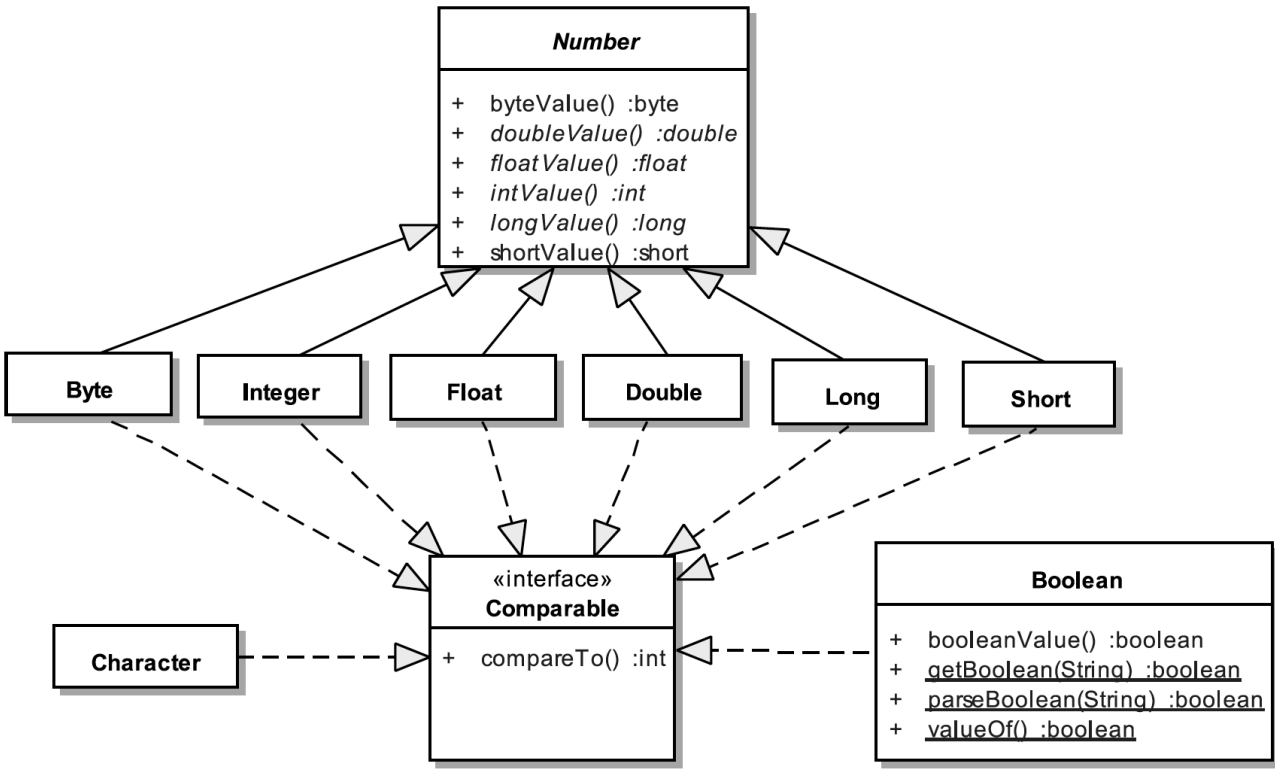

Рисунок 1.2 Иерархия классов-оболочек

Значения базовых типов хранятся в стеке и не являются объектами. Классы, соответствующие числовым базовым типам, находятся в библиотеке *java.lang,* являются наследниками абстрактного класса *Number* и реализуют интерфейс *Comparable<T>*. Этот интерфейс определяет возможность сравнения объектов одного типа между собой с помощью метода *int compareTo(T ob)*. Объекты классов-оболочек по умолчанию получают значение *null*.

Создаются экземпляры интегральных или числовых классов с помощью одного из двух конструкторов с параметрами типа *String* и соответствующего базового типа.

Объект класса-оболочки может быть преобразован к базовому типу методом *munValue*() или обычным присваиванием.

Класс *Character* не является подклассом *Number*, этому классу нет необходимости поддерживать интерфейс классов, предназначенных для хранения результатов арифметических операций. Вместо этого класс *Character* имеет целый ряд специфических методов для обработки символьной информации.

У класса *Character*, в отличие от других классов оболочек, не существует конструктора с параметром типа *String*.

```
Float ft = new Float(1.7); // double в Float
Short s = new Short((short)5); // int в Short
Short sh = new Short("5"); // String в Short
double d = s.doubleValue(); // Short в double
byte b = (byte)(float)ft; // Float в byte
Character ch = new Character('3');
int i = Character.digit(ch.charValue(), 10); /* Character в int */
```

Конструкторы классов-оболочек с параметром типа *String* и их методы *valueOf(String str), decode(String str)* и *parseTun(String str)* выполняют действия по

преобразованию значения, заданного в виде строки, к значению соответствующего объектного типа данных. Исключение составляет класс *Character*. При преобразовании строки к конкретному типу может возникнуть ошибка формата данных, если строка не соответствует этому типу данных. Для устойчивой работы приложения все операции по преобразованию строки в типизированные значения желательно заключать в блок *try-catch* для перехвата и обработки возможного исключения.

Четыре стандартных способа преобразования строки в число:

```
/* # 3 # преобразование строки в целое число # StringToInt.java */
package by.bsac.transformation;
public class StringToInt {
public static void main(String[ ] args) {
String arg = "71"; // 071 или 0x71или 0b1000111
try {
int value1 = Integer.parseInt(arg); // возвращает int
int value2 = Integer.valueOf(arg); // возвращает Integer
int value3 = Integer.decode(arg); // возвращает Integer
int value4 = new Integer(arg); /* создает Integer,
для преобразования применяется редко */
} catch (NumberFormatException e) {
System.err.println("Неверный формат числа " + e);
}
}
}
```

У приведенных способов есть определенные различия при использовании разных систем счисления и представления чисел.

Обратное преобразование из типизированного значения (в частности *int*) в строку можно выполнить следующими способами:

```
int value = 71;
String arg1 = Integer.toString(value); // хороший способ
String arg2 = String.valueOf(value); // хороший способ
String arg3 = "" + value; // плохой способ
```

Существует два класса для работы с высокоточной арифметикой – *java.math.BigInteger* и *java.math.BigDecimal*, которые поддерживают целые числа и числа с фиксированной точкой произвольной длины.

Начиная с версии 5.0, введен процесс автоматической инкапсуляции данных базовых типов в соответствующие объекты оболочки и обратно (автоупаковка/автораспаковка). При этом нет необходимости в явном создании соответствующего объекта с использованием оператора new:

```
Integer iob = 71; // эквивалентно Integer iob = new Integer(71);
```

При инициализации объекта класса-оболочки значением базового типа преобразование типов в некоторых ситуациях необходимо указывать явно, то есть код

```
Float f = 7; // правильно будет (float)7 или 7F вместо 7
```

вызывает ошибку компиляции.

С другой стороны, справедливо:

```
Float f = new Float("7");
```

Автораспаковка – процесс извлечения из объекта-оболочки значения базового типа. Вызовы методов *intValue(), doubleValue()* и им подобных для преобразования объектов в значения базовых типов становятся излишними.

Допускается участие объектов в арифметических операциях, однако не следует этим злоупотреблять, поскольку упаковка/распаковка является ресурсоемким процессом:

```
// autoboxing & unboxing:
Integer i = 71; // создание объекта+упаковка
```

++i; // распаковка+операция+создание объекта+упаковка
int j = i; // распаковка

Однако следующий код генерирует исключительную ситуацию *NullPointerException* при попытке присвоить базовому типу значение *null* объекта класса *Integer*, литерал null – не объект и не может быть преобразован к значению «ноль»:

Integer j = null; // объект не создан! Это не ноль!
int i = j; // генерация исключения во время выполнения

Несмотря на то, что значения базовых типов могут быть присвоены объектам классов-оболочек, сравнение объектов между собой происходит по ссылкам. Для сравнения значений объектов следует использовать метод *equals()*.

```
int i = 127;
Integer a = i; // создание объекта+упаковка
Integer b = i;
System.out.println("a==i " + (a == i)); // true – распаковка и сравнение значений
System.out.println("b==i " + (b == i)); // true
System.out.println("a==b " + (a == b)); /* false(ссылки на разные объекты) */
System.out.println("equals ->" + a.equals(i)
+ b.equals(i)
+ a.equals(b)); // true, true, true
```

Метод *equals*() сравнивает не значения объектных ссылок, а значения объектов, на которые установлены эти ссылки. Поэтому вызов *a.equals(b)* возвращает значение *true*.

Значение базового типа может быть передано в метод *equals()*. Однако ссылка на базовый тип не может вызывать методы:

i.equals(a); // ошибка компиляции

Стало возможным создавать объекты и массивы, сохраняющие различные базовые типы без взаимных преобразований, с помощью ссылки на класс *Number*, а именно:

Number n1 = 1; // идентично new Integer(1)
Number n2 = 7.1; // идентично new Double(7.1)

Практическое применение таких объектов крайне ограничено.

При автоупаковке значения базового типа возможны ситуации с появлением некорректных значений и непроверяемых ошибок.

Переменная базового типа всегда передается в метод по значению, а переменная класса-оболочки – по ссылке.

Операторы управления

В языке *Java*, как и в любом другом языке программирования, есть условные операторы и циклы для управления потоком. Блок, или составной оператор, произвольное количество простых операторов языка *Java*, заключенных в фигурные скобки. Блоки определяют область видимости своих переменных. Блоки могут быть вложенными один в другой. Однако невозможно объявить одинаково названные переменные в двух вложенных блоках.

```
Public static void main(String [] args)
{ int n;
...
{
int k;
int n; // Ошибка – невозможно переопределить переменную n во вложенном цикле
} //переменная k определена только в этом блоке
}
```

Условные операторы

Условный оператор в языке *Java* имеет вид:

```
if (условие) оператор
// или
if (условие) {
оператор1;
оператор2; }
```

Все операторы, заключенные в фигурные скобки, будут выполнены, если значение условия истинно. Общий случай условного оператора выглядит так:

```
if (условие) оператор1 else оператор2
if (yourSale >= target)
{ performance=”Удовлетворительно”;
Bonus = 100 + 0.01*( yourSale – target);
}
else
{ performance=”Неудовлетворительно”;
Bonus =0;
}
```

Многовариантное ветвление представлено в виде повторяющихся операторов

```
 if … else if…
if (sale >=2*target)
{ performance=”Отлично”;
}
else if (sale >=1.5*target)
{ performance=”Удовлетворительно”;
}
else {System.out.println(“Вы уволены”);}
```

Многовариантное ветвление – оператор *switch*

Конструкция *if/else* может оказаться неудобной, если необходимо сделать выбор из многих вариантов. Например, создавая систему меню из трех альтернатив, можно использовать следующий код.

```
String input = “1”;
int choice = Integer.parseInt(input);
switch (choice){
case 1:
…
break;
case 2:
…
break;
case 3:
…
break;
default: // неверный выбор
…
break; }
```

Выполнение начинается с метки *case*, соответствующей значению переменной *choice*, и продолжается до следующего оператора *break* или конца оператора *switch*. Если ни одна метка не совпадает со значением переменной, выполняется раздел *default*. Метка *case* должна быть целочисленной!

Неопределенные циклы

Существует два вида повторяющихся циклов, которые лучше всего подходят, если

вы точно не знаете, сколько повторений должно быть выполнено. Первый из них, цикл *while*, выполняет тело цикла, только пока выполняется его условие.

while (условие) {операторы;}

Условие цикла *while* проверяется в самом начале. Следовательно, возможна ситуация, когда код, содержащийся в блоке, не будет выполнен ни разу. Если необходимо, чтобы блок выполнялся хотя бы один раз, проверку условия нужно перенести в конец. Это можно сделать с помощью цикла

do/while.

do оператор while (условие);

Определенные циклы

Цикл *for* – распространенная конструкция для выполнения повторений, количество которых контролируется счетчиком, обновляемым на каждой итерации.

```
for (int i = 1; i <= 10; i++){
System.out.println(i);
}
```

Первый элемент оператора *for* обычно выполняет инициализацию счетчика, второй формулирует условие выполнения тела цикла, а третий определяет способ обновления счетчика.

```
for (int i = 10; i > 0; - i){
System.out.println("Обратный отсчет …" + i);
}
```

Цикл полного перебора используется для перебора элементов массива или коллекции:

for (ТипДанных имя : имяОбъекта) { / операторы */ }*

Оператор продолжения continue

Позволяет начать новую итерацию цикла не доходя до конца текущей итерации. По умолчанию итерация относится к телу цикла, в котором вызывается оператор. Также можно указать метку блока, с которой начать новую итерацию.

continue;

Оператор выхода из блока break

Позволяет выйти из текущего блока в операторе выбора или из тела цикла. Если указана метка блока, то выходит из того блока.

break;

Пакеты

Программа на *Java* представляет собой набор пакетов (*packages*).

Каждый пакет может включать вложенные пакеты, а так же может содержать классы и интерфейсы. Каждый пакет имеет свое пространство имен, что позволяет создавать одноименные классы в различных пакетах.

Имена бывают простыми (*simple*), состоящими из одного идентификатора, и составными (*qualified*), состоящими из последовательности идентификаторов, разделенных точкой. Составное имя любого элемента пакета составляется из составного имени этого пакета и простого имени элемента.

Простейшим способом организации пакетов и типов является обычная файловая структура. Например, исходный код класса *space.sunsystem.Moon* хранится в файле *space\sunsystem\Moon.java*

Запуск программы на *JAVA* стоит производить из директории, в которой содержатся пакеты. Было бы ошибкой запускать *Java* прямо из папки *space\sunsystem* и пытаться

обращаться к классу *Moon*, несмотря на то, что файл-описание лежит именно в ней. Необходимо подняться на два уровня директорий выше, чтобы *Java*, построив путь из имени пакета, смогла обнаружить нужный файл.

Модуль компиляции

Модуль компиляции (*complication unit*)-хранится в текстовом *.java*-файле и является единичной порцией входных данных для компилятора. Состоит из трех частей:

– Объявление пакета;

– *Import*-выражения;

– Объявления верхнего уровня;

Объявление пакета указывает, какому пакету будут принадлежать все объявляемые ниже типы. Используется ключевое слово *package*, после которого указывается полное имя пакета. Например, в файле *java/lang/Object.java* идет: *package java.lang;* что служит одновременно объявлением пакета *lang*, вложенного в пакет *java*, и указанием, что объявляемый ниже класс *Object*, находится в этом пакете. Так складывается полное имя класса *java.lang.Object.*

Область видимости типа – пакет, в котором он располагается. Внутри этого пакета допускается обращение к типу по его простому имени. Из всех других пакетов необходимо обращаться по составному имени.

Для решения этой проблемы вводятся *import*-выражения, позволяющие импортировать типы в модуль компиляции и далее обращаться к ним по простым именам. Существует два вида таких выражений:

- импорт одного типа: *import java.net.URL;*

- импорт пакета: *import java.awt.*;*

4. Порядок выполнения работы

- изучить теоретический материал;

- напишите программы на языке Java:

1. Программа, в которой перебираются числа от 1 до 500 и выводятся на экран. Если число делится на 5, то вместо него выводится слово *fizz*, если на 7, то *buzz*. Если число делится на 5 и на 7, то выводить слово *fizzbuzz*. Примечание*: остаток от деления в *Java* обозначается через символ %.

2. Программа, в которой все переданные во входную строку аргументы выводятся на экран в обратной порядке. Например, если было передано 2 аргумента – *make install*, то на экран должно вывестись *llatsni ekam*. Примечание*: для разбора слова по буквам необходимо использовать функцию *charAt()*. Например, *str.charAt(i)* вернет символ с позиции *i* в слове, записанном в строковую переменную *str*. Команда *str.length()* возвращает длину слова *str*.

3. Создайте программу, вычисляющую числа Фибоначчи. Числа Фибоначчи – последовательность чисел, в котором каждое следующее число равно сумме двух предыдущих. Начало этой последовательности – числа 1, 1, 2, 3, 5, 8, 13…

4. Создайте программу, вычисляющую факториал целого числа.

- выполните индивидуальные задания.

5. Индивидуальные задания

1. Ввести с консоли 3 целых числа. На консоль вывести: Четные и нечетные числа.

2. Ввести с консоли 3 целых числа. На консоль вывести: Наибольшее число.

3. Ввести с консоли 3 целых числа. На консоль вывести: Числа, которые делятся на 3 или на 9.

4. Ввести с консоли 3 целых числа. На консоль вывести: Числа, которые делятся на 5 и на 7.

5. Ввести с консоли 3 целых числа. На консоль вывести: Определить среднее

значение наибольшего и наименьшего числа.

7. Ввести с консоли 3 целых числа. На консоль вывести: Наибольший общий делитель этих чисел.

8. Ввести с консоли 3 целых числа. На консоль вывести: Простые числа.

9. Ввести с консоли 3 целых числа. На консоль вывести: Отсортированные числа в порядке возрастания и убывания.

10. Ввести с консоли 3 целых числа. На консоль вывести: Наименьшее общее кратное этих чисел.

11. Ввести с консоли 3 целых числа. На консоль вывести: Наименьшее число.

12. Ввести с консоли 3 целых числа. На консоль вывести: Числа, входящие в заданный промежуток.

13. Ввести с консоли 3 целых числа. На консоль вывести: Найти количество положительных чисел.

14. Ввести с консоли 3 целых числа. На консоль вывести: Найти среднее из них (то есть число, расположенное между наименьшим и наибольшим).

15. Ввести с консоли 3 целых числа. На консоль вывести: Если их значения упорядочены по возрастанию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное. Вывести новые значения переменных.

16.: Даны координаты точки, не лежащей на координатных осях OX и OY. Определить номер координатной четверти, в которой находится данная точка.

17Даны три целых числа, одно из которых отлично от двух других, равных между собой. Определить порядковый номер числа, отличного от остальных.

18. Ввести с консоли 3 целых числа. На консоль вывести: Найти сумму двух наибольших из них.

19. Дано целое число. Вывести его строку-описание вида «отрицательное четное число», «нулевое число», «положительное нечетное число».

20. Даны три числа. Найти сумму двух наименьших из них.

21. Даны четыре целых числа, одно из которых отлично от трех других, равных между собой. Определить порядковый номер числа, отличного от остальных.

22. Ввести с консоли 3 целых числа. На консоль вывести: Определить среднее значение наибольшего и среднего числа.

23. Ввести с консоли 3 целых числа. На консоль вывести: Определить среднее значение среднего и наименьшего числа.

24. Дано целое число. Если оно является отрицательным, то прибавить к нему 1; в противном случае не изменять его. Вывести полученное число.

25. Дано целое число. Если оно равно 0, то прибавить к нему 1; в противном случае не изменять его. Вывести полученное число.

26. Дано целое число. Если оно является положительным, то прибавить к нему 1; в противном случае не изменять его. Вывести полученное число.

27. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной большее из этих значений, а если равны, то присвоить переменным нулевые значения. Вывести новые значения переменных A и B.

28. Даны две переменные вещественного типа: A, B. Перераспределить значения данных переменных так, чтобы в A оказалось меньшее из значений, а в B – большее. Вывести новые значения переменных A и B.

29. Дано целое число. Если оно является положительным, то прибавить к нему 1; в противном случае не изменять его. Вывести полученное число.

30. Пользователь вводит порядковый номер пальца руки. Необходимо вывести его название на экран.

6. Контрольные вопросы

1. В чем разница JDK и JRE?

2. Назовите типы данных Java и классы-оболочки. Чем они отличаются?

3. Для чего применяют документирование кода с помощью дескрипторов? Перечислите основные дескрипторы.

4. Перечислите операторы управления, используемые в Java.