МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ
для студентов специальности
1-40 05 01 «Информационные системы и технологии»,
6-05-0611-01 «Информационные системы и технологии»

Витебск
2023

УДК 004.42

Составители:

П. Г. Деркаченко, В. Е. Казаков

Рекомендовано к изданию редакционно-издательским
советом УО «ВГТУ», протокол № 9 от 29.05.2023.

Объектно-ориентированное программирование : методические указания по выполнению лабораторных работ / УО «ВГТУ» ; П. Г. Деркаченко, В. Е. Казаков. – Витебск, 2023. – 31 с.

Методические указания предназначены для выполнения лабораторных работ по дисциплине «Объектно-ориентированное программирование» для студентов специальности 1-40 05 01 «Информационные системы и технологии», 6-05-0611-01 «Информационные системы и технологии». Методические указания способствуют закреплению на практике теоретических знаний студентов, полученных на лекционных занятиях.

Издание в электронном виде расположено в репозитории библиотеки УО «ВГТУ».

УДК 004.42

СОДЕРЖАНИЕ

Лабораторная работа 1
Основные структуры программирования. Типы данных. Операторы

Цель работы: ознакомиться с ядром языка Java, синтаксисом и управляющими конструкциями языка программирования Java.

Типы данных

В языке программирования Java имеется 8 базовых типов данных, называемых также примитивными или элементарными типами:

Таблица 1.1 – Базовые типы данных

Наименование типа	Описание типа
Byte	Целые числа в диапазоне от -127 до 128
Short	Целые числа в диапазоне от -32 768 до 32 767
Int	Целые числа в диапазоне от -2 147 483 648 до 2 147 483 647
Long	Целые числа в диапазоне от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
Float	Одиночное с плавающей точкой в диапазоне от 1.4e-045 до 3.4e+038
Double	Двойное с плавающей точкой в диапазоне от 4.9e-324 до 1.8e+308
Char	Используется для представления символов в Unicode. Поэтому является 16-битным. Диапазон допустимых значений от 0 до 65536
Boolean	Предназначен для хранения логических значений. Переменные этого типа могут принимать значения: true (истинно) или false (ложно)

Существует также «пустой» тип – Void. Этот тип имеет метод, у которого нет возвращаемого значения. Аналогом такого метода может являться процедура в языке Delphi.

Кроме примитивных типов, существуют так называемые ссылочные типы. Можно сказать, что ссылочные типы – это типы, определенные пользователем.

Арифметические операции представлены следующими операторами:

Таблица 1.2 – Арифметические операции

Обозначение операции	Описание операции
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
++	Инкремент
--	Декремент
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием
%=	Деление по модулю с присваиванием

Составной оператор с присваиванием позволяет записать выражение вида a = a + 4 как a + = 4. Такого рода операторы предоставляют два преимущества. Во-первых, они позволяют уменьшить объем вводимого кода, во-вторых их реализация в системе времени выполнения Java эффективнее реализации эквивалентных длинных форм.

Таблица 1.3 – Операции сравнения

Обозначение операции	Описание операции
>	Больше
<	Меньше
= =	Равно
!==	Неравно
>=	Больше или равно
<=	Меньше или равно

Управляющие конструкции

Операторы выбора.
Оператор if.
Общий вид:

```
if (условие) {
        оператор1;
} else {
        оператор2;
}
```

Здесь каждый оператор – это одиночный или составной оператор, Условие – это любое выражение, возвращающее значение типа boolean. В качестве условия может выступать также сама переменная типа boolean. Выражение else не обязательно. **Тернарный оператор** позволяет в некоторых случаях записывать условное выражение в кратком виде. К примеру, переменная должна быть инициализирована тем или иным значением в зависимости от условия. Такого рода инициализацию можно осуществить таким образом:

<тип> переменная = (условие) ? значение_1 : значение_2;

При этом, если условие истинно, переменная станет равна значение_1, а если ложно – значение_2.

Таким же образом можно возвращать результат работы метода в зависимости от условия:

return (условие) ? значение_1 : значение_2;

Вложенный оператор if – это оператор if, являющийся целью другого оператора if или else. В программах вложенные операторы if встречаются очень часто. При использовании вложенных операторов if важно помнить, что оператор e1se всегда связан с ближайшим оператором if, расположенным в одном с ним блоке и еще не связанным с другим оператором e1se.

Например:

```
if (i = = 10) {
        if (j < 20) {
                a = b;
        }
        if (k > 100) {
                c = d;
        } else {
                a = c;
        }
} else {
        a = d;
}
```

Многозвенная (множественная) структура if-else-if:

```
if (условие) {
        оператор1;
} else if (условие) {
        оператор2;
} else if (условие) {
        оператор3;
…
}else {
        операторN;
}
```

Запись нескольких условий в операторе (составных условий) if осуществляется при помощи следующих операторов:

Таблица 1.4 – Операторы объединения условий

Обозначение оператора		Описание оператора
Полная форма	Сокращенная форма	
|	||	Логическое или
&	&&	Логическое и

Примеры записи:
if (условие_1 || условие_2)
if (условие_1 && условие_2)
if (условие_1 | условие_2)
if (условие_1 & условие_2)

Первые две записи отличаются тем, что в случае, если в операторе || первое условие окажется истинным, либо ложным в операторе &&, второе условие уже проверяться не будет. В последних же двух записях оба условия будут проверяться в любом случае.

Оператор switch – один из операторов ветвления в Java. Он предлагает простой способ направления потока выполнения команд по различным ветвям кода в зависимости от значения управляющего выражения. Часто он оказывается эффективнее применения длинных последовательностей операторов if-e1se-if. Общая форма оператора switch имеет следующий вид:

```
switch (выражение) {
    case значениel:
        // последовательность операторов1
        break;
    case значение2:
        // последовательность операторов2
        break;
    …
    case значениеN:
        // последовательность операторов3
        break;
    defau1t:
        // последовательность операторов, выполняемая по умолчанию
}
```

Выражение, стоящее в скобках после switch должно иметь тип short, int или char. Тип каждого значения, указанного в операторах case должен быть совместим с типом выражения или переменной, стоящей в скобках после switch. Для управления оператором switch можно использовать также значение перечислимого типа. Каждое значение в case должно быть уникальной

символьной константой (то есть постоянным значением, а не переменной). Дублирование значений case не допускается.

Оператор switch работает следующим образом. Значение выражения сравнивается с каждым из значений констант в операторах case. При обнаружении совпадения программа выполняет последовательность кода, следующую за данным оператором case. Если значения ни одной из констант не совпадает со значением выражения, программа выполняет оператор default. Однако этот оператор не обязателен. При отсутствии совпадений со значениями case и отсутствии оператора defau1t программа не выполняет никаких дальнейших действий. Оператор break внутри последовательности switch служит для прерывания последовательности операторов. Как только программа доходит до оператора break, она продолжает выполнение с первой строки кода, следующей за всем оператором switch. Этот оператор оказывает действие «выхода» из оператора switch. Если его нет, программа выполняет следующую после совпадения ветвь case.

Операторы цикла

Операторами цикла Java являются for, whi1e и do whi1e.

Цикл while – наиболее часто используемый оператор цикла Java. Он имеет следующую общую форму:

```
while (условие) {
   / / тело цикла
}
```

Или вторая форма:

```
do {
   / / тело цикла
} whi1e (условие);
```

Условием может быть любое булевское выражение. Тело цикла будет выполняться до тех пор, пока условие истинно. Когда условие становится ложным, управление передается строке кода, непосредственно следующей за циклом. Отличие второй формы цикла от первой состоит в следующем:

1) это – цикл с постусловием (первая форма – с предусловием);

2) данная форма цикла всегда выполняется хотя бы 1 раз.

Цикл for

Общая форма традиционного оператора for выглядит следующим образом:

```
for (инициализация; условие; повторение) {
   // тело
}
```

Цикл for действует следующим образом. При первом запуске цикла программа выполняет инициализационную часть цикла. В общем случае это выражение, устанавливающее значение управляющей переменной цикла, которая действует в качестве счетчика. Важно понимать, что выражение

инициализации выполняется только один раз. Затем программа вычисляет условие, которое должно быть булевским выражением. Как правило, выражение сравнивает значение управляющей переменной с целевым значением. Если условие истинно, программа выполняет тело цикла. Если оно ложно, выполнение цикла прерывается. Затем программа выполняет часть повторения цикла (итерационное выражение). Обычно это выражение, которое увеличивает или уменьшает значение управляющей переменной. Затем программа повторяет цикл, при каждом прохождении, вначале вычисляя условное выражение, затем выполняя тело цикла и выполняя выражение повторения. Процесс повторяется до тех пор, пока условие повторения не станет ложным. Инициализационное, итерационное выражения, условие или любые две части цикла, а также все они могут вовсе отсутствовать. Таким образом, цикл for может выглядеть так:

```
for ( ; ; ) {
  ...
}
```

Для досрочного прерывания цикла можно использовать оператор break. Когда программа встречает оператор break внутри цикла, она прекращает выполнение цикла, и управление передается оператору, следующему за циклом. Если при работе цикла возникает ситуация, когда надо пропустить одну или несколько операций в теле цикла и сразу же перейти к следующей итерации, используется оператор continue.

Контрольные вопросы:

1. Какие существуют типы данных в языке программирования Java?
2. Какой тип возвращает метод, у которого нет возвращаемого значения?
3. Какие операции можно осуществлять в языке программирования Java?
4. Что такое инкремент, декремент, составной оператор с присваиванием?
5. Какие операторы выбора существуют в языке программирования Java?
6. Что такое тернарный оператор? Как записываются множественные, составные условия?
7. Какие операторы циклов существуют в языке программирования Java? Их характеристики, форма записи, принцип работы?

Варианты индивидуальных заданий

1. Написать программу преобразования шкалы Цельсия в шкалу Фаренгейта и наоборот. 0 по Цельсию равен 32 по Фаренгейту. 1 градус Цельсия равен 1.8 по Фаренгейту (2 чел).

2. Написать программу вычисления частного и остатка от деления двух целых чисел.

3. Написать программу вычисления наибольшего общего делителя двух целых чисел. Программа должна использовать цикл while (2 чел).

4. Написать программу решения квадратного уравнения. Корни только

вещественные.

Квадратное уравнение – это уравнение вида $ax^2 + bx + c = 0$, где a не равно 0.

Для решения квадратного уравнения можно использовать формулы:

$$x_+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{и} \quad x_- = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \tag{1.1}$$

где $D = b^2 - 4ac$ – дискриминант многочлена $ax^2 + bx + c$. Если $D > 0$, то уравнение имеет два различных вещественных корня. Если $D = 0$, то оба корня вещественны и равны. Если $D < 0$, то оба корня являются комплексными числами.

5. Написать программу проверки, является ли число простым. Использовать цикл for.

6. Даны произвольные числа a, b и c. Если нельзя построить треугольник с такими длинами сторон, то напечатать 0, иначе напечатать 3, 2 или 1 в зависимости от того, равносторонний это треугольник, равнобедренный или какой-либо иной.

7. Написать программу идентификации треугольника (остроугольный, прямоугольный, тупоугольный, равнобедренный, равносторонний) по его элементам (по двум его углам; по трем его сторонам; по двум сторонам и углу между ними).

8. Написать программу вычисления факториала числа.

9. Написать программу вычисления корня уравнения вида: $ax^2 + bx + c$ методом бисекций.

10. Написать программу табулирования функции на отрезке a b c шагом h по формуле:

$$y = \begin{cases} f_1(x), a \le x < z \\ f_2(x), z \le x \le b \end{cases}. \tag{1.2}$$

Лабораторная работа 2
Работа с массивами

Цель работы: приобретение навыков работы с одномерными и двумерными массивами.

Массив, в отличие от простой переменной, представляет собой не одно значение, а множество значений, объединенных одним именем. Все значения из этого множества должны иметь один и тот же тип. Каждое из значений массива называется элементом массива. Доступ к элементам массива производится

посредством указания имени массива и номера элемента массива, заключенного в квадратные скобки. Номер элемента массива называется индексом элемента массива. Использование элемента массива не отличается от использования простой переменной, имеющей тот же тип, что и элемент массива. При объявлении массива сначала указывается его тип, затем идет имя переменной, а пара квадратных скобок указывает, что используемый тип является именно массивом: **int[] a;** количество пар квадратных скобок указывает на размерность массива. Для многомерных массивов допускаются следующие записи: **int[] a[]; int a[][]; int[][] a;** эти записи означают, что переменная а имеет тип «двумерный массив, основанный на int». Чтобы создать экземпляр массива, нужно воспользоваться ключевым словом new, после чего указывается тип массива и в квадратных скобках указывается длина массива: **int a[]=new int[5];** длина массива обязательно должна указываться при создании и уже никак не может быть изменена. Поскольку для экземпляра массива длина является постоянной характеристикой, для всех массивов существует специальное поле length типа int, позволяющее узнать ее значение. Значение индекса массива также имеет тип int. Допустимо, чтобы одна и та же переменная ссылалась на массивы разной длины, то есть:

```
int i[]=new int[5];
 ...
i=new int[7];
```

// переменная та же, длина массива другая. Индексация элементов массива происходит от 0 (индекс первого элемента) до length – 1 (индекс последнего элемента).

Инициализация массивов

– с помощью цикла for:

```
…
int p[] = new int[5];
for (int i = 0; i < p.length; i++) {
   p[i] = i;
}
```

таким образом, каждый элемент массива инициализируется его индексом;

– с помощью инициализатора:

В этом случае ключевое слово new не используется, а ставятся фигурные скобки, и в них перечисляются через запятую значения всех элементов массива: **int i[]={1, 3, 5}; int j[]={}; // эквивалентно new int[0]**. Длина массива вычисляется автоматически, исходя из количества введенных значений;

– вручную:

```
…
int p[] = new int[3];
p[0] = 4;
p[1] = 3;
p[2] = 2;
```

Такой способ требует большого количества исходного кода, однако он может быть применим для небольших массивов с фиксированными значениями.

Аналогично, для создания многомерных массивов можно использовать инициализаторы: **int i[][] = {{1, 2}, {3, 4}, {5, 6}};**

Часто массив заполняют в цикле:

```
int table[][]=new int[5][5];
for (int i=0; i<5; i++) {
   for (int j=0; j<5; j++) {
      table[i][j]=i*j;
     System.out.print(pithagor_table[i][j]+ "\t");
   }
   System.out.println();
}
```

Результатом выполнения программы будет:

0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16

Как видно, данный массив является прямоугольным, однако так бывает не всегда. Поскольку правильно говорить, что в языке Java нет многомерных массивов, а есть массивы, элементами которых в свою очередь являются массивы, то допустима следующая инициализация (**Mass.docx**):

```
int x[][]=new int[3][5]; // прямоугольная таблица
x[0]=new int[7];
x[1]=new int[0];
x[2]=null;
```

Как видно, массив изначально инициализировался 3 массивами, размером 5, но они сразу были переопределены новыми значениями. Для таких случаев полезно использовать упрощенную форму выражения создания массивов: int **x[][]=new int[3][];**

Для перебора элементов массива, начиная с Java 1.5, используется цикл вида **for each.** Цикл *for* стиля *for-each* устраняет необходимость устанавливать счетчик цикла, задавая начальное и конечное значения, и вручную индексировать массив. Вместо этого он автоматически проходит весь массив, получая поочередно каждый элемент.

Например:

```
int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0;
for(int x: nums) sum += x;
```

В каждом проходе тела цикла переменная *x* получает значение из следующего элемента массива *nums*. Таким образом, во время первого прохода

переменная цикла *x* содержит значение 1, во время второго – значение 2 и т. д. При этом не только упрощается синтаксис, но и устраняются ошибки выхода за границы диапазона.

Контрольные вопросы:

1. Что такое массив?
2. В каких случаях необходимо использовать массивы?
3. Что такое размерность массива?
4. Что такое размер массива?
5. Что такое элемент массива? Индекс массива?
6. Какие типы данных могут использоваться в качестве индексов для массивов?
7. Как ввести массив чисел?
8. Что такое двумерный массив? Назовите способы его описания.
9. Как описать переменные для хранения следующей информации: Места в кинотеатре определяются номером ряда, номером места в ряду и признаком, указывающим, продан ли билет на данное место (хотелось бы все это хранить в одном массиве!).

Варианты индивидуальных заданий

1. Найти первый минимальный и первый максимальный из данных *n* элементов.

Отсортировать матрицу размера **5 × 5** по спирали таким образом, чтобы её минимальный элемент был в первом столбце первой строки матрицы, а максимальный – в центре.

2. Найти номер первого максимального и первого минимального из данных n элементов.

Дана матрица размера **5 × 10**. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.

3. Найти номер последнего максимального из данных n целочисленных элементов.

Дана матрица размера **5 × 10**. Найти минимальное значение среди сумм элементов всех ее строк и номер строки с этим минимальным значением.

4. Найти количество минимальных из данных n целочисленных элементов.

Дана матрица размера **5 × 10**. Найти минимальный среди максимальных элементов каждой строки.

5. Найти максимальный четный из данных *n* ненулевых целочисленных элементов. Если требуемые элементы отсутствуют, то вывести 0.

Найти седловые точки матрицы размера ***M* × *N*.** Если седловые точки отсутствуют, вывести 0. Седловой точкой матрицы назовём элемент, который одновременно является минимумом в своей строке и максимумом в своём столбце.

6. Найти минимальный положительный из данных *n* элементов. Если требуемые элементы отсутствуют, то вывести 0.

Дана целочисленная матрица размера ***M* × *N***. Различные строки матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках. Найти количество строк, похожих на первую строку.

7. Даны числа *a, b* (0 < *a* < *b*) и набор из *n* элементов. Найти минимальный из элементов, содержащихся в интервале (*a, b*). Если требуемые элементы отсутствуют, то вывести – 1.

Дана квадратная матрица порядка ***M***. Найти суммы элементов ее диагоналей, параллельных главной (начиная с одноэлементной диагонали ***A [1, M]***).

8. Дан набор из *n* целочисленных элементов. Найти количество элементов, расположенных после первого минимального.

Дана квадратная матрица порядка ***M.*** Вывести минимальные элементы каждой ее диагонали, параллельной главной (начиная с одноэлементной диагонали ***A [1, M]*).**

9. Найти номер последнего экстремального (то есть минимального или максимального) из данных *n* целочисленных элементов.

Задана квадратная матрица ***R (n, n).*** Найти номер столбца ***K*** и строки ***L*** с максимальным произведением. Сформировать вектор ***B (2n),*** элементы которого чередуются – нечетные равны сумме, а четные – разности элементов ***K***-го столбца и ***L***-й строки.

10. Дан набор из *n* целочисленных элементов. Найти количество элементов, содержащихся между первым и последним минимальным. Если в наборе имеется единственный минимальный элемент, то вывести 0.

Задана квадратная матрица ***Y (n, n).*** Определить, где больше четных элементов: выше или ниже главной диагонали?

11. Найти номера двух наибольших из данных *n* элементов.

Задана матрица ***X (n, m)***. Найти строки с максимальным и минимальным средним значениями элементов и поменять их местами.

12. Дан набор из *n* целочисленных элементов. Найти максимальное количество подряд идущих минимальных элементов.

Дана матрица размера **5 × 10**. Найти минимальный среди максимальных элементов каждой строки.

13. Дан массив размера *N*. Вывести вначале его элементы с четными индексами, а затем – с нечетными, не используя условных операторов.

Дана матрица размера **5 × 10**. В каждой строке найти количество элементов, больших среднего арифметического всех элементов этой строки.

14. В массиве A (20) поменять местами соседние четные и нечетные по номеру элементы. Дополнительные массивы не использовать.

Дана действительная квадратная матрица ***M (5, 5*).** Требуется переставить строки матрицы по возрастанию первых элементов строк.

15. Задан массив ***A (n).*** Вычислить сумму произведений всех пар соседних чисел.

Дана действительная квадратная матрица ***M (5, 5)*.** Требуется переставить строки матрицы по возрастанию первых элементов строк.

Лабораторная работа 3
Определение класса, экземпляры класса.
Поля, методы, конструкторы, инициализаторы

Цель работы: получить представление о классах и объектах.

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программного продукта в виде совокупности объектов, каждый из которых является экземпляром конкретного класса.

Объект в ООП – это обладающий именем набор данных (полей объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним. Имя объекта используется для доступа к полям и методам, составляющим этот объект. В предельных случаях объект может не содержать полей или методов, а также не иметь имени.

Любой объект относится к определенному классу.

В ***классе*** дается описание некоего объекта. Объект – конкретный экземпляр класса. Объект – синоним – экземпляр класса. В качестве примера можно привести чертеж какой-либо детали (класс) и реальную деталь, сделанную по данному чертежу (экземпляр класса или объект).

Класс определяет новый тип данных. После того, как тип определен, его можно применять для создания объектов данного типа. Для объявления класса служит ключевое слово **c1ass**. **Общая форма класса:**

```
class ИмяКласса {
        тип переменнаяЭкземпляра1;
        тип переменнаяЭкземпляра2;
        //...
        конструктор:
        ИмяКласса (список параметров) {
                // тело конструктора
        }

        тип переменнаяЭкземпляраN;

        тип имяМетода1(список параметров) {
                // тело метода
}

тип имяМетода2(список параметров) {
        // тело метода
}
```

```
    //…
    тип имяМетодаN(список параметров ) {
        // тело метода
    }
}
```

Данные, или переменные, определенные внутри класса, называются ***переменными экземпляра.*** Их также называют полями или переменными класса. Они называются переменными экземпляра, так как каждый экземпляр (или объект) класса содержит собственные копии этих переменных. Таким образом, данные одного объекта отделены и отличаются от данных другого объекта. Код содержится внутри ***методов***. Определенные внутри класса методы и переменные вместе называют членами класса. В Java-программах почти всегда имеется класс, содержащий метод main (то есть «основной»). Этот метод вызывается Java-машиной самым первым и с него начинается выполнение всего проекта Java.

Пример простого класса:

```
class Box {
    double width;
    double height;
    double depth;
}
```

Параллелепипед, который определяет три переменных экземпляра: width (ширина), height (высота) и depth (глубина).

В данном случае мы определили новый тип данных под названием Box. Это имя будет использоваться для объявления **объектов** типа Box. Важно помнить, что объявление class создает только шаблон, но не действительный объект. Чтобы действительно создать объект класса Box, нужно использовать операцию, подобную следующей: **Box myBox = new Box();** после выполнения этого оператора myBox станет экземпляром класса Box, то есть он обретет «физическое» существование в оперативной памяти компьютера. Переменная myBox называется ***ссылочной*** переменной, так как она ***ссылается*** на объект класса Box. Для создания объекта используется оператор **new**. Можно просто объявить переменную **myBox**, не создавая при этом объекта: **Box myBox;** в этом случае, поскольку реально созданного объекта класса Box пока нет, она будет содержать в себе значение по умолчанию, то есть null.

Для получения доступа к переменным экземпляра применяется операция точки (.). Эта операция связывает имя объекта с именем переменной экземпляра. Например, чтобы присвоить переменной width объекта myBox значение 100, нужно использовать следующую операцию: mybox.width = 100. Эта операция указывает компилятору, что копии переменной width, хранящейся внутри объекта myBox, нужно присвоить значение, равное 100.

В общем случае операцию точки используют для доступа как к переменным экземпляра, так и к методам внутри объекта.

Общая форма объявления ***метода*** выглядит следующим образом:

```
тип имя ( список параметров ) {
        // тело метода
}
```

Здесь тип указывает тип данных, возвращаемых методом. Он может быть любым допустимым типом, В том числе типом класса, созданным программистом. Если метод не должен возвращать значение, то он возвращает void (в переводе с англ. – пустота). Имя служит для указания имени метода. Оно может быть любым допустимым идентификатором, кроме тех, которые уже используются другими элементами в текущей области определения. Список параметров – это последовательность пар «тип-идентификатор», разделенных запятыми. По сути, параметры – это переменные, которые принимают значения *аргументов*, переданных методу во время его вызова. Если метод не имеет параметров, список параметров будет пустым. Методы, тип возвращаемого значения которых отличается от void, должны вернуть значение с помощью оператора return. Если же метод возвращает void, то он может и не содержать оператор return. В таких методах return используется только для их досрочного завершения.

Сигнатура **(**signature**)** метода определяется именем метода и его аргументами (количеством, типом, порядком следования). Если для полей запрещается совпадение имен, то для методов в классе запрещено создание двух методов с одинаковыми сигнатурами. Методы одного класса, имеющие одинаковые имена, называются ***перегруженными***. Сигнатуры у них должны быть различными, и различие может быть только в наборе аргументов.

Пример перегрузки методов:

```
class Point {
        void get() {
        }
        void get(int x) {
        }
        void get(int x, double y) {
        }
        void get(double x, int y) {
        }
}
```

Для инициализации переменных объекта непосредственно во время его создания, как правило, применяют ***конструктор***. Имя конструктора совпадает с именем класса, в котором он находится, а синтаксис аналогичен синтаксису метода. Как только он определен, конструктор автоматически вызывается

непосредственно после создания объекта, перед завершением выполнения операции ***new***.

Пример класса с конструктором:

```
class Box_3 {
        double width;
        double height;
        double depth;

        Box_3() {
                width = 10;
                height = 10;
                depth = 10;
        }
}
```

Большинство конструкторов не выводят никакой информации, а лишь выполняют инициализацию объекта.

При создании объекта можно также использовать конструктор с параметрами:

```
class Box {
        double width;
        double height;
        double depth;

        Box (double w, double h, double d) {
                 width = w;
                height = h;
                depth = d;
        }
}
```

В таком случае объект класса Box следует создавать так:
Box myBox = new Box (10, 10, 10);

Надо отметить, что класс может иметь 1 или несколько конструкторов.

Экземплярные переменные можно также инициализировать в ***блоках инициализации*** или ***инициализаторах*****:**

```
class Test {
        int x, y, z;
```

```
        // инициализатор объекта
        {
                x = 3;
                if (x>0) {
                        y=4;
                }
                z =  x >= y ? x : y;
        }
}
```

Инициализаторы не имеют имен, исполняются при создании объектов и не могут быть вызваны явно. В инициализаторах нельзя использовать переменные класса, если их объявление записано позже.

Контрольные вопросы:

1. Что такое класс в языке программирования?
2. Что такое объект в языке программирования?
3. Какие данные могут содержаться в классе?
4. В каких случаях в классе используется ключевое слово this?
5. Что такое перегрузка методов?
6. Какими способами можно инициализировать поле класса при создании объекта?
7. Что такое объектно-ориентированное программирование?
8. Что такое члены класса?
9. Что значит ключевое слово void?
10. Что такое сигнатура метода?

Варианты индивидуальных заданий

Реализовать классы, описывающие сущности предметной области индивидуального задания. Классы, а также их поля и методы должны иметь отражающую их функциональность названия.

1. Student: id, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс, Группа.

Создать набор объектов. Вывести:

а) список студентов заданного факультета;

б) список студентов, родившихся после заданного года;

в) список учебной группы.

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

Создать набор объектов. Вывести:

а) полный список покупателей;

б) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. Patient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер

медицинской карты, Диагноз.

Создать набор объектов. Вывести:

а) список пациентов, имеющих данный диагноз;

б) список пациентов, номер медицинской карты у которых находится в заданном интервале.

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

Создать набор объектов. Вывести:

а) список абитуриентов, имеющих неудовлетворительные оценки;

б) список абитуриентов, средний балл у которых выше заданного.

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

Создать набор объектов. Вывести:

а) список книг заданного автора;

б) список книг, выпущенных заданным издательством;

в) список книг, выпущенных после заданного года.

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

Создать набор объектов. Вывести:

а) список квартир, имеющих заданное число комнат;

б) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;

в) список квартир, имеющих площадь, превосходящую заданную.

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

Создать набор объектов. Вывести:

а) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;

б) сведения об абонентах, которые пользовались междугородной связью;

в) сведения об абонентах в алфавитном порядке.

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

Создать набор объектов. Вывести:

а) список автомобилей заданной марки;

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

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

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

Создать набор объектов. Вывести:

а) список товаров для заданного наименования;

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

в) список товаров, срок хранения которых больше заданного.

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

Создать набор объектов. Вывести:

а) список поездов, следующих до заданного пункта назначения;

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

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

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

Создать набор объектов. Вывести:

а) список автобусов для заданного номера маршрута;

б) список автобусов, которые эксплуатируются больше 10 лет;

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

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

Создать набор объектов. Вывести:

а) список рейсов для заданного пункта назначения;

б) список рейсов для заданного дня недели;

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

Лабораторная работа 4
Наследование, инкапсуляция, полиморфизм

Цель работы: изучить основные понятия ООП: наследование, инкапсуляция, полиморфизм.

Объектно-ориентированное программирование основано на принципах:

- ***инкапсуляции;***
- ***наследования;***
- ***полиморфизма;***

Инкапсуляция (encapsulation) – принцип, объединяющий данные и код, манипулирующий этими данными, а также защищающий, в первую очередь, данные от прямого внешнего доступа и неправильного использования. Другими словами, доступ к данным класса возможен только посредством методов этого же класса.

Наследование (inheritance) – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним свойства и методы, характерные только для него. Используя наследование, можно создать класс, который определяет характеристики, общие для набора связанных элементов.

Затем этот класс может наследоваться другими, более специализированными классами, каждый из которых будет добавлять свои уникальные характеристики. Таким образом создается ***иерархия классов.*** Наследуемый класс обычно называют суперклассом. Наследующий класс носит название подкласса. Наследование в языке Java осуществляется с помощью ключевого слова ***extends***.

Полиморфизм (polymorphism) – в более общем виде концепцию полиморфизма часто выражают фразой «один интерфейс, несколько методов». Это означает, что можно спроектировать общий интерфейс для группы связанных между собой действий.

Переопределение методов

Если в иерархии классов имя и сигнатура типа метода подкласса совпадает с атрибутами метода суперкласса, говорят, что метод подкласса ***переопределяет*** метод суперкласса. Когда переопределенный метод вызывается из подкласса, он всегда будет ссылаться на версию этого метода, определенную подклассом. Версия метода, определенная суперклассом, будет сокрыта. Рассмотрим следующий пример:

```
class Parent {
        public int getValue() {
                return 0;
        }
}

class Child extends Parent {
        public int getValue() {
                return 1;
        }
}

public class Demo {
        public static void main(String args[]) {
                Child c = new Child();
                System.out.println(c.getValue());
                Parent p = new Child();
                System.out.println(p.getValue());
        }
}
```

Результатом будет:

```
1
1
```

Можно видеть, что родительский метод полностью перекрыт, значение 0 никак нельзя получить через ссылку, указывающую на объект класса Child,

несмотря даже на то, что переменная p имеет тип Parent. В этом ключевая особенность полиморфизма – ссылочная переменная суперкласса может ссылаться на объект подкласса. При помощи перегрузки и переопределения методов в современных языках программирования реализуется принцип ***полиморфизма***.

Для реализации принципа ***инкапсуляции*** (сокрытия) данных используются ***модификаторы доступа***:

Таблица 4.1 – Модификаторы доступа

Название модификатора Доступ из:	**private**	**по умолчанию (без модификатора)**	**protected**	**public**
Этот же класс	да	да	да	да
Подкласс класса этого же пакета	нет	да	да	да
Класс этого же пакета, не являющийся подклассом	нет	да	да	да
Подкласс класса, находящегося в другом пакете	нет	нет	да	да
Класс из другого пакета, не являющийся подклассом класса данного пакета	нет	нет	нет	да

Контрольные вопросы:

1. Каковы основные принципы ООП?
2. Как реализуются эти принципы в современных языках программирования?

Варианты индивидуальных заданий

Создать и реализовать интерфейсы, также использовать наследование и полиморфизм для следующих предметных областей:

1. interface Издание -> abstract class Книга -> class Справочник и Энциклопедия.
2. interface Абитуриент -> abstract class Студент -> class Студент-Заочник.
3. interface Сотрудник -> class Инженер -> class Руководитель.
4. interface Здание -> abstract class Общественное Здание -> class Театр.
5. interface Корабль -> abstract class Военный Корабль -> class Авианосец.
6. interface Врач -> class Хирург -> class Нейрохирург.
7. interface Корабль -> class Грузовой Корабль -> class Танкер.
8. interface Мебель -> abstract class Шкаф -> class Книжный Шкаф.
9. interface Фильм -> class Отечественный Фильм -> class Комедия.
10. interface Ткань -> abstract class Одежда -> class Костюм.
11. interface Техника -> abstract class Плеер -> class Видеоплеер.
12. interface Транспортное Средство -> abstract class Общественный Транспорт -> class Трамвай.
13. interface Устройство Печати -> class Принтер -> class Лазерный Принтер.
14. interface Бумага -> abstract class Тетрадь -> class Тетрадь Для Рисования.
15. interface Источник Света -> abstract class Лампа -> class Настольная Лампа.

Лабораторная работа 5
Объектно-ориентированные библиотеки

Цель работы: научиться использовать стандартные библиотеки языка программирования java.

Стандартные библиотеки расширяют возможности языка программирования, позволяя программисту использовать готовые решения многих типичных задач. На практике часто используются такие библиотеки, как java.lang, java.io и java.util.

Библиотека java.lang

В состав пакета java.lang входят классы, составляющие основу для всех других – из-за чего он является наиболее важным из всех, входящих в Java API. Каждый класс в Java по умолчанию может использовать все классы этой библиотеки. Рассмотрим некоторые из основных классов этого пакета.

Object

Класс Object является базовым для всех остальных классов. Он определяет методы, которые поддерживаются любым классом в Java. Некоторые из этих методов:

1. **public boolean equals(Object obj)** – определяет, являются ли объекты одинаковыми. Если оператор == определяет равенство именно объектных ссылок, то метод equals() определяет, содержат ли объекты одинаковую информацию. Как правило, данный метод переопределяют для описания своего алгоритма сравнения объектов.

2. **public int hashCode()** – возвращает хеш-код (hash code) для объекта. Хеш-код – это целое число, которое с очень большой вероятностью является уникальным для данного объекта.

В классе Object реализация метода hashCode() использует для получения результата преобразование внутреннего адреса объекта в памяти, поэтому если не переопределять данный метод, то для разных объектов будут возвращены различные значения.

3. **public String toString()** – возвращает строковое представление объекта.

В классах-наследниках этот метод обычно переопределяется для получения более наглядного пользовательского представления объекта.

4. **wait(), notify(), notifyAll()** – эти методы используются для поддержки многопоточности.

5. **protected void finalize() throws Throwable** – этот метод вызывается Java-машиной перед очисткой памяти, занимаемой объектом. Объект считается пригодным для сборщика мусора, когда выполняющейся части программы не будет доступно ни одной ссылки на объект. Перед очисткой занимаемой объектом памяти будет произведен вызов его метода finalize().

Реализация этого метода в классе Object не производит никаких действий. В классах-наследниках этот метод может быть переопределен для проведения всех необходимых действий по освобождению различных занимаемых ресурсов – закрытия сетевых соединений, файлов и т. д.

Wrapper Classes (классы-обертки)

Для каждого примитивного типа Java существует свой класс-обертка. Классы-обертки содержат методы для обеспечения удобного манипулирования соответствующими примитивными типами, например преобразование к строковому виду. В таблице описаны примитивные типы и соответствующие им классы обертки:

Таблица 5.1 – Классы-обертки

Класс-обертка	Примитивный тип
Byte	byte
Short	short
Character	char
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean

Float, Double наследуются от одного класса – Number. В нем содержится код, общий для всех классов-оберток числовых типов.

Math

Класс Math состоит из набора методов, производящих наиболее популярные математические вычисления и двух констант – это число *Пи* и число E, то есть экспонента.

public static final double Matht.PI – задает число *Пи*.

public static final double Matht.E – число *e*.

System

Содержит набор полей и методов, позволяющих работать со средой, в которой выполняется приложение. К примеру, переменная out позволяет нам отображать на консоли Windows результаты работы программы с помощью команды System.out.print(result);

Throwable

Является базовым классом для всех исключений. Любой класс, который планируется использовать как исключение, должен быть от него унаследованным. Классы Error, Exception, RuntimeException – это наиболее значимые его наследники.

Runnable, Thread, ThreadGroup

Эти классы служат для поддержки многопоточности.

String, StringBuffer, StringBuilder

Данные классы служат для работы со строками.

Библиотека java.io

Организует работу с потоками ввода/вывода данных. Базовыми классами данной библиотеки являются ***InputStream*** – класс для потоков ввода, то есть чтения данных и ***OutputStream*** – класс для потоков вывода, то есть записи данных.

InputStream

Простейшая операция представлена методом **int read()**. Этот метод предназначен для считывания ровно одного байта из потока, однако возвращает при этом значение типа int. В случае если считывание произошло успешно, то возвращаемое значение лежит в диапазоне от 0 до 255. В случае, если достигнут конец потока, то возвращаемое значение равно – 1. Если же считать из потока данные не удается из-за каких-то ошибок или сбоев, возникает исключение **java.io.IOException**. Что бы узнать, сколько байт в потоке готово к считыванию, используется метод **available()**. Когда работа с входным потоком данных окончена, его следует закрыть. Для этого вызывается метод **close()**. Этим вызовом будут освобождены все системные ресурсы, связанные с потоком.

OutputStream

В этом классе для записи данных определяются три метода **write()** – один принимающий в качестве параметра int, второй byte[], и третий byte[] плюс два int-числа. Все эти методы имеют тип void. В случае возникновения ошибки записи возникает java.io.IOException. Вообще, это исключение возникает в большинстве методов, связанных с вводом-выводом. Реализация потока может быть таковой, что данные записываются не сразу, а хранятся некоторое время в памяти, так называемом буфере. Чтобы убедиться, что данные записаны в поток, а не хранятся в буфере, вызывается метод **flush()**. Когда работа с потоком закончена, его следует закрыть. Для этого вызывается метод **close()**.

Библиотека java.util

В этой библиотеке собрано большое количество вспомогательных классов. Одними из самых важных являются классы-коллекции. Коллекции представляют более гибкую, чем массивы, организацию работы с набором элементов. Они предназначены только для работы с объектами, в то время как, массивы могут содержать как простые типы, так и ссылки на объекты. Для использования простых типов в коллекциях необходимо применять классы-обертки. Коллекции подразделяют на списки (list) и множества (set). Отличие

между ними состоит в том, что в последних не разрешено наличие дубликатов, а также в них разрешено только одно пустое (null) значение.

Рассмотрим основные классы коллекций.

Списки:

ArrayList

Может содержать упорядоченную последовательность объектов (объекты хранятся в том порядке, в котором они были добавлены) имеет методы, которые позволяют получать, добавлять и удалять элементы из списка. Конструкторы данного класса:

ArrayList() – создает пустой список;

ArrayList(Collection c) – список, инициализированный элементами коллекции с **ArrayList(int capacity)** – формирует список с начальной емкостью capacity, то есть числом элементов, которые он будет содержать. Если количество элементов превысит начальное, емкость автоматически возрастет.

LinkedList

LinkedList (связный список) имеет методы, которые позволяют добавлять, удалять и получать элементы в конце и начале списка. Имеет 2 конструктора: **LinkedList()** и **LinkedList(Collection c)** – пустой и проинициализированный объектами коллекции с.

Отличие LinkedList и ArrayList: в ArrayList заметно быстрей (примерно на порядок) осуществляются операции прохода по всему списку (итерации) и получения данных. LinkedList почти на порядок быстрее осуществляет операции удаления и добавления новых элементов.

Множества:

HashSet хранит элементы в случайном порядке. Для быстрого поиска HashSet расчитывает для каждлого элемента hashCode и именно по этому ключу ищет и упорядочивает элементы внутри себя. В целом, hash-set позволяют более оптимально реализовать операции над множествами – пересечение, объединение и вычитание.

TreeSet в отличии от HashSet, хранит элементы упорядоченно, либо в «естественном» порядке (от минимального элемента к максимальному), либо в порядке, определенном пользователем.

LinkedHashSet хранит элементы в порядке их добавления. Поиск по этой коллекции происходит также по hashCode, но порядок будет всегда совпадать с очерёдностью добавления.

Кроме списков и множеств, в библиотеке java.util имеются также классы, позволяющие работать с парами значений (ключ-значение). Такие классы еще называют ***картами*** **(**map**).**

HashMap

Этот класс предназначен для хранения пар объектов ключ/значение. Элементы в объекте типа HashMap хранятся в случайном порядке. Как для ключей, так для элементов допускаются значения типа null. HashMap обеспечивает постоянное время доступа для операций get и put (то есть получения и добавления элементов). Итерация по всем элементам этой коллекции пропорциональна ее емкости. Поэтому имеет смысл не устанавливать ее размер чрезмерно большим, если достаточно часто придется осуществлять итерацию по элементам. Дублирование ключей в HashMap не допускается (за исключением ключа null).

TreeMap

Обеспечивает хранение пар ключ/значение в отсортированном порядке (по возрастанию ключей) и допускает быстрый поиск и извлечение данных.

Контрольные вопросы:

1. Для чего нужны стандартные библиотеки? Какие стандартные библиотеки часто применяются на практике?
2. Основные классы библиотеки java.lang.
3. Основные классы библиотеки java.io.
4. Основные классы библиотеки java.util.
5. Для чего нужны классы-обертки?
6. В чем отличия списков от множеств?
7. Основные классы-списки.
8. Основные классы-множества.
9. В чем особенности карт?
10. Основные классы-карты.

Варианты индивидуальных заданий

1. Ввести набор слов, разделенных пробелами, дефисами и запятыми. Распечатать этот набор, удалив из него те слова, которые встретились там более одного раза. При разработке программы использовать классы-коллекции.
2. В последовательности вещественных чисел, разделенных пробелами и запятыми определить наименьшее отрицательное число и его порядковый номер. Действия по определению минимальных значений реализовать с использованием функций (методов) с параметрами. При разработке программы использовать классы-коллекции.
3. В последовательности целых чисел, разделенных пробелами и знаками препинания определить третье положительное число и подсчитать количество цифр в нем и их сумму. Действия по определению суммы, и количества цифр, а также формирования других значений в программах реализовать с использованием функций (методов) с параметрами. При разработке программы использовать классы-коллекции.
4. В последовательности букв вывести на печать TRUE, если

количество гласных букв больше, чем согласных и FALSE – иначе. Действия по определению количества букв и сравнения реализовать с использованием функций (методов) с параметрами. При разработке программы использовать классы-коллекции.

5. В последовательности вещественных чисел найти максимальный и минимальный элементы. Действия по определению минимальных и максимальных значений реализовать с использованием функций (методов) с параметрами. При разработке программы использовать классы-коллекции.

6. В последовательности целых чисел определить сумму четных положительных чисел и произведение отрицательных меньших – 10. Действия по определению суммы, произведения и проверки четности реализовать с использованием функций (методов) с параметрами. При разработке программы использовать классы-коллекции.

7. Написать программу, которая сортирует последовательность чисел по убыванию. При разработке программы использовать классы-коллекции.

8. Создать текстовый файл, состоящий из нескольких строк. Записать в другой файл в обратном порядке символы каждой строки. При разработке программы использовать классы-коллекции.

9. Создать текстовый файл, состоящий из нескольких строк. Поменять в каждой строке первое и последнее слово в каждой строке. Записать результат в другой файл.

10. Создать текстовый файл, содержащий набор слов. Найти и вывести на экран слова, для которых последняя буква одного слова совпадает с первой буквой другого слова. При разработке программы использовать классы-коллекции.

ЛИТЕРАТУРА

1. Блинов, И. Н. Java 2. Практическое руководство / И. Н. Блинов, В. С. Романчик. – Минск : УниверсалПресс, 2005. – 400 с.

2. Ноутон, П. Java TM 2 / П. Ноутон, Г. Шилд. – Санкт-Петербург : БХВ-Петербург, 2008. – 1050 с.

3. Хорстман, К. Java 2. Библиотека профессионала : в 2 т. Т. 1 : Основы / К. Хорстман, Г. Корнел. – Москва : ИД «Вильямс», 2007. – 896 с.

4. Хорстман, К. Java 2. Библиотека профессионала : в 2 т. Т. 2 : Тонкости программирования / К. Хорстман, К. Корнел. – Москва : ИД «Вильямс», 2007. – 1168 с.

5. Эккель, Б. Философия Java / Б. Эккель. – Санкт-Перебург : БХВ-Петербург, 2009. – 640 с.

6. Арнольд, К. Язык программирования Java / К. Арнольд, Д. Гослинг, Д. Холмс. – Москва : ИД «Вильямс», 2001. – 624 с.

7. Дейтел, Х. Технологии программирования на Java 2 / Х. Дейтел, Д. Гослинг, Д. Сантри. – Санкт-Петербург : Бином-Пресс, 2011. – 464 с.

Учебное издание

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ

Составители:
Деркаченко Павел Григорьевич
Казаков Вадим Евгеньевич

Редактор *А.В. Пухальская*
Корректор *А.В. Пухальская*
Компьютерная верстка *П.Г. Деркаченко*

Подписано к печати 13.06.2023. Формат 60x90 $^{1}/_{16}$. Усл. печ. листов 1,9.
Уч.-изд. листов 2,5. Тираж 2 экз. Заказ № 159.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.

Учебное издание

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ

Составители:
Деркаченко Павел Григорьевич
Казаков Вадим Евгеньевич

Редактор *А.В. Пухальская*
Корректор *А.В. Пухальская*
Компьютерная верстка *П.Г. Деркаченко*

Подписано к печати 13.06.2023. Усл. печ. листов 1,9.
Уч.-изд. листов 2,5. Заказ № 159.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.