МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра "ПОВТ и АС"

Управляющие конструкции Java

Методические указания к лабораторной работе
по дисциплинам "Программирование", "Объектно-ориентированное программирование"

Ростов-на-Дону 20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.
УДК 512.3

Управляющие конструкции Java: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – 8 с.

Рассмотрены все управляющие конструкции (операторы) Java, условная, циклические, передачи управления. Даны задания по выполнению лабораторной работы. Методические указания предназначены для студентов направлений 090304 "Программная инженерия", 020303 "Математическое обеспечение и администрирование информационных систем".

Ответственный редактор:

Издательский центр ДГТУ, 20

I. Консольный ввод/вывод

Поток вывода на стандартное устройство (по умолчанию консольное окно) представлен в Java классом PrintStream из пакета java.io. Уже созданный объект класса PrintStream можно получить через статическое поле класса System - System.out.

Класс PrintStream предлагает методы print(String str), println(String str) для вывода строк, и метод printf для форматированного вывода в стиле языка Си. Метод printf перегружен. Есть версия, использующая текущие языковые настройки (locale) и версия, в которой locale задается явно в качестве первого аргумента. Если locale не указывается явно, первым аргументом printf будет форматная строка, в противном случае она будет вторым аргументом. В форматной строке могут содержаться обычные символы и специальные обозначения - спецификаторы формата. Обычные символы просто выводятся в консольное окно, спецификаторы формата указывают на то, каков тип выводимых значений, и, быть может, какова минимальная ширина поля вывода, точность представления чисел и т.п.

Спецификаторы типа:

%b логическое (булево) значение true или false
%h хэш-код
%c символ
%s строка
%d целое десятичное значение (типов byte, short, int, long, BigInteger
%o целое десятичное значение
%x целое десятичное значение
%f действительное значение (float, double) в форме с фиксированной точкой
%e действительное значение в экспоненциальной форме
%a действительное значение в шестнадцатеричной системе счисления

Минимальная ширина поля вывода (количество позиций, занимаемых выводимым значением) задается как целое значение перед символом типа.

```
System.out.printf("%5d", 12);
```

Выводит три пробела и 12

```
  12
```

Если перед шириной поля вывода поместить символ ноль, то вместо пробелов будут выведены незначащие нули

```
System.out.printf("%05d", 12);
```

Выводит

00012

Точность вывода задается после точки, следующей за шириной поля ввода в виде целого значения. Для действительных чисел точность задает число цифр в дробной части числа (с округлением), например, %5.2f - как минимум пять позиций на экране, две цифры после запятой, или %.2f - нужное число позиций (нет минимального ограничения) и две цифры в дробной части. Для строки - максимальное число символов, выводимых на экран.

```
double x = 125.125;
System.out.printf("%.2f %.2e\n", x, x);
```

Выводит

125,13 1,25e+02

В отличие от Си, выводить значения, указанные в printf можно в произвольном порядке, для этого используются номера этих значений, отсчитываемые от единицы и указываемые в форме 1$, 2$ и т.д. сразу после значка %. Например,

```
int x = 123;
float f = 9.87e2f; // суффикс f - константа типа float
System.out.printf("%2$.2f %1$d\n", x, f);
```

Выводит

987,00 123

Если установлена русская locale то, в частности, при выводе действительных значений дробная часть будет отделяться от целой не точкой, а запятой. Чтобы в качестве разделителя использовалась точка можно указать другую locale. В классе Locale (из пакета java.util) есть предопределенные locale, например, Locale.US.

```
int x = 123;
float f = 9.87e2f;
System.out.printf(java.util.Locale.US,"%2$.2f %1$d\n", x, f);
```

Выводит

987.00 123

Стандартный поток ввода в Java представлен классом java.io.InputStream. В классе System есть статическое поле System.in - уже созданный объект класса InputStream. В классе InputStream есть методы для ввода отдельных байт или массивов байт. С другой стороны, есть класс Scanner (из пакета java.util), который работает как надстройка над каким-нибудь потоком ввода - стандартным, файловым, строкой. Этот класс представляет методы для ввода значений различных стандартных типов, но при создании объекта такого

класса ему нужно передать конкретный поток, который и будет поставлять реально вводимые данные.

```
Scanner scan = new Scanner( System.in );
```

Методы класса Scanner для чтения данных разных типов:

String next()	возвращает строку (до первого пробельного символа)
String nextLine()	возвращает строку (до Enter)
boolean nextBoolean()	логическое значение
byte nextByte()	целое значение
short nextShort()	целое значение
int nextInt()	целое значение
long nextLong()	целое значение

Для всех целых значений есть версии с одним аргументом - целым значением, задающим систему счисления. Например,

```
Scanner scan = new Scanner(System.in);
byte b = scan.nextByte(2);
```

При вводе

101

Выводит

5

float nextFloat()	действительное значение
double nextDouble()	действительное значение

При вызове методов nextByte и т.д. для ввода чисел, в случае, если пользователь задает какие-то символы не являющиеся допустимыми для чисел, генерируется исключение java.util.InputMismatchException. Чтобы избежать возникновения исключительной ситуации нужно использовать методы Scanner, позволяющие узнать есть ли в потоке данные, которые можно интерпретировать как значения нужного типа:

boolean hasNext()	есть строка без пробельных символов
boolean hasNextLine()	есть строка
boolean hasNextBoolean()	есть логическое значение
boolean hasByte()	есть byte
boolean hasShort()	есть short
boolean hasInt()	есть int
boolean hasLong()	есть long
boolean hasFloat()	есть float
boolean hasDouble()	есть double

Например,

```
Scanner scan = new Scanner(System.in);
byte b = 0;
if(scan.hasNextByte())    b = scan.nextByte();
```

II. Управляющие конструкции

Синтаксис управляющих конструкций Java - операторов (statements) похож на синтаксис Си/C++.

Так же точно задаются пустой оператор ; составной оператор (блок) {} и оператор-выражение *expression*; например, x = 5; // оператор-выражение

Оператор условия

Оператор условия в Java подобен таковому в Си/C++ или Паскале, но, в отличие от Си, в Java есть логический тип и запись вида if(1) недопустима. Например,

```
// тип выражения boolean
if( x < 0 && y >= 0 ) System.out.println("I квадрант");
```

или

```
if( x >= 0 ) absx = x; else absx = -x;
```

Операторы циклов

Оператор цикла с параметром в простейшем случае выглядит так же как в Си/C++:

```
for(int i=0; i<10; i++) System.out.println(""+i);
        |     |       |
        |     |       изменение параметра цикла
        |  условие продолжения выполнения тела цикла (пока истинно)
      начальное значение параметра цикла
```

И, хотя в Java и нет операции "запятая", в цикле for, как и в Си/C++ можно объявлять и изменять несколько переменных одного типа

```
for(int i =0, j=10; i<10; ++i, --j)
        System.out.printf("%5d%5d\n", i, j);
```

Синтаксис for в стиле foreach тоже подобен синтаксису C++

```
int[] a = {1, 2, 3, 4, 5};
for(int x : a)
    System.out.printf("%5d\n", x);
```

Цикл с предусловием:

```
// пока логическое значение истинно выполнять оператор
while(логическое выражение) оператор;
```

Например,

```
int i = 0;
while(i < 10) {
    System.out.println(""+i);
    ++i;
}
```

Цикл с постусловием:

```
// выполнять оператор пока логическое значение истинно
do оператор; while(логическое выражение);
```

Например,

```
int i = 0;
do {
    System.out.println(""+i);
    ++i;
} while( i < 10 );
```

Оператор-переключатель (оператор выбора)

```
// выражение может иметь тип byte, short, int, char, String или enum
switch(выражение) {
    case константное_выражение1: операторы1
    …
    case константное_выражениеN: операторыN
    default: операторы
}
```

Если значение выражения совпадает с каким-либо из константных выражений выполняются ВСЕ операторы, начиная с соответствующего case. Если нет - все операторы, начиная с тех, которые расположены после default. Значения

константных выражений не могут совпадать. Взаиморасположение конструкций case между собой и с конструкцией default произвольное. Если default отсутствует и совпадения ни с одним константным выражением нет выполнение предается за оператор switch.

Оператор return

Возвращает управление из вызванного метода в метод его вызвавший. Если метод возвращает значение, оно должно быть указано в операторе return. Например, метод, проверяющий на четность переданный ему аргумент

```
import java.util.Scanner;

public class Test {
	public static boolean isEven(int value) {
		return (value & 1) == 0;
	}

	public static void main(String[] args) {
		Scanner scan = new Scanner(System.in);
		int i  = 0;
		// пока вводятся целые значения
		while(scan.hasNextInt()) {
			i = scan.nextInt();
			System.out.printf("Is %d even? %b\n", i, isEven(i));
		}
	}
}
```

Оператор break

Прерывает выполнение оператора цикла или оператора-переключателя.

В Java есть форма break с меткой. Тогда прерывается тот цикл, который помечен. Имеет смысл для вложенных циклов. Например,

```
int limit = 0;
Scanner scan = new Scanner(System.in);
if(scan.hasNextInt()) limit = scan.nextInt();
m:
for(int i=0; i<100; ++i)    // или в одной строке m: for(int i=0; i<100; ++i)
```

```
for(int j=100; j<0; --j)
        for(int k=0; k<20; ++k) {
                if(2*i+3*j+k > limit) break m; // выход из цикла по i
                fun(); // вызов некоторого метода
        }
```

Оператор continue
Прерывает текущую и инициирует начало следующей итерации цикла.
В Java есть форма continue с меткой. Тогда происходит переход к следующей итерации того цикла, который помечен.

В Java отсутствует оператор безусловного перехода goto.

III. Регулярные выражения

Регулярные выражения (regular expressions) используются в задачах проверки некоторого текста на соответствие определенному шаблону. Основные формы - это проверка на совпадение всего текста или поиск наличия в тексте некоторых фрагментов, соответствующих шаблону. В дальнейшем найденные фрагменты можно извлечь из текста или же заменить их на другие фрагменты. Регулярное выражение представляет собой тот самый шаблон, определяющий искомый текст. Очевидный вариант шаблона - точный набор символов, который мы ищем в тексте. Например, проверка учетной информации предполагает сравнение строки текста, которую задает пользователь в качестве своего имени (login) на подлинность. Тогда образцом выступает сама строка, подлежащая проверке. Собственно, задача сводится к проверке на совпадение содержимого двух текстовых строк. В более сложных случаях шаблон описывается некоторыми правилами, например, логин должен содержать не менее шести символов и это могут быть буквы или цифры. Используя специальные предопределенные символы и конструкции можно составить текстовую строку, содержащую регулярное выражение, соответствующее этим правилам. Например, "[\\w\\d]{6,}". Синтаксис регулярных выражений в различных языках программирования разный. Java использует синтаксис, принятый в языке Perl. В языке программирования, поддерживающем обработку регулярных выражений существуют классы, предоставляющие методы для работы с регулярными выражениями. В Java, например, класс String содержит метод boolean matches(String regex),

позволяющий проверить соответствует ли некоторый текст, содержащийся в строке заданному шаблону regex. Например,

```
System.out.println("abraca12dabra".matches("[\\w\\d]{6,}"));
```

Выведет true, так как строка "abraca12dabra" содержит только буквы и цифры и в ней не меньше шести символов.

```
System.out.println("abr+aca12dabra".matches("[\\w\\d]{6,}"));
System.out.println("abr".matches("[\\w\\d]{6,}"));
```

Выведет false, так как в первом случае строка "abr+aca12dabra" содержит символ +, отличный от буквы и цифры, а, во втором случае в строке "abr" меньше шести символов.

Как уже сказано выше, регулярное выражение может содержать набор обычных символов, или же полностью состоять из таких символов.

```
System.out.println("admin".matches("admin"));
```

Выведет true (регистр символов важен)

Помимо обычных символов в регулярных выражениях Java могут использоваться специальные символы и конструкции:

.	представляет один, любой символ
[abcd]	представляет один символ из указанного набора

Примеры, [aAbB123x=], можно использовать диапазоны [a-zA-Z] - любая латинская буква, для текста на русском языке нужно помнить, что буквы ё и Ё расположены вне диапазонов а-я и А-Я, поэтому одну, но любую русскую букву можно представить так [а-яА-ЯёЁ], о же относится и к цифрам [0-9] - любая арабская цифра.

[^abcd]	представляет любой символ, не входящий в указанный набор
\d	любая цифра
\D	любой символ - не цифра
\w	любая буква
\W	любой символ - не буква

Так как в последних обозначениях используется значок \, при помещении регулярного выражения в строку нужно его экранировать (удвоить). Например, если нужно убедиться, что текст состоит из одного символа и этот символ цифра, строка содержащее нужное регулярное выражение может выглядеть так

```
"\\d"
```

или так

```
"[0-9]"
```

Этому регулярному выражению соответствуют строки "1", "9", "0", но не соответствуют строки "12", "x".

Так как точка - это специальный знак, для проверки наличия точки в тексте нужно либо использовать \. либо заключить ее в квадратные скобки. Например, для проверки того, что строка состоит из трех символов, причем первый какая-то буква, второй точка, а третий цифра можно сформировать такую строку

"\\w\\.\\d"

или

"\\w[.]\\d"

Чтобы проверить, что текст состоит ровно из 7 цифр можно составить выражение "\\d\\d\\d\\d\\d\\d\\d" но удобнее использовать квантификаторы.

{n} символ (или группа символов, если они заключены в круглые скобки) перед квантификатором должна повторяться n раз

{n,m} m>n, повторов может быть не меньше n и не больше m

{n,} повторов не меньше n

Для значений {0,} - ни одного или сколько угодно повторов можно использовать специальный символ *

Для значений {1,} - один или сколько угодно повторов специальный символ +

Для {0,1} - специальный символ ?

Например, не менее шести символов, букв или цифр

"[\\w\\d]{6,}"

любое количество любых символов или ни одного символа (пустая строка)

".*"

С помощью круглых скобок можно выделять группы символов.

Специальный символ | показывает, что может быть выбрана одна из групп, указанных слева и справа от него, например, символ '0' или '1'

"0|1"

один символ из набора '1', '2', '3' или из набора 'A', 'B', 'C'

"[1-3]|[A-C]"

Пример шаблона для поиска следующей последовательности символов: от одной до семи цифр включительно и вся последовательность может быть заключена в круглые скобки или не заключена в них

"**(**[(]\\d{1,7}[)]**)|(**\\d{1,7}**)**"

последнюю пару круглых скобок в данном случае можно опустить, так как в них содержится всего одна конструкция с квантификатором

"**(**[(]\\d{1,7}[)]**)|**\\d{1,7}"

Сами круглые скобки это специальные символы, поэтому для трактовки их как обычных символов нужно заключить их в квадратные скобки [(] и [)] или экранировать с помощью \(и \)

"(\\(\\d{1,7}\\))|\\d{1,7}"

Помимо метода matches в классе String есть еще метод String[] split(String regexpr). Он используется для того, чтобы разбить строку на части путем удаления указанных символов-разделителей. Метод возвращает массив строк - "слов" получившихся в результате такого разбиения. Например,

```
String[] word = "Hello World".split(" ");
for(String w : word)
    System.out.println(w);
```

Выведет

Hello

World

Так как разделители задаются как регулярное выражение - удаляются те последовательности символов, которые соответствуют этому регулярному выражению, которое может быть сколь угодно сложным.

На самом деле, работу с регулярными выражениями в Java обеспечивают классы Pattern и Matcher из пакета java.util.regex. Методы String лишь используют некоторые возможности этих классов. Для работы с регулярными выражениями обычно эти классы используют напрямую, а не косвенно, через методы String, но для достаточно простых задач работа напрямую со строками проще.

Общая схема работы с Pattern и Matcher такова:

```
import java.util.regex.*;
…
    Pattern p = Pattern.compile("[\\w\\d]{6,}"); // нет открытых конструкторов
```

Статический метод compile создает объект класса Pattern и хранит регулярное выражение в откомпилированном виде. У этого класса есть также статический метод matches

```
Boolean flag = Pattern.matches("[\\w\\d]{6,}", "ab12cd"); // true
```

и нестатический метод split

```
String[] s = p.split("***ab12cd+++aaa---abracadabra1234…");
for(String w : s)
    System.out.println(w);
```

Выводит

***

+++aaa---

…

Класс Matcher содержит методы для работы с регулярными выражениями, текст к которому применяется выражение и хранит результаты этого применения. У этого класса также нет открытых конструкторов и объект класса создает метод matcher класса Pattern.

```
Matcher m = p.matcher("ab12cd "); // текст
```

В классе Matcher также есть метод match для проверки соответствия всего текста заданному регулярному выражению, но также и другие методы, например, find для поиска соответствия выражению фрагментов текста (поиск подстроки в строке). Метод find возвращает true, если подходящий фрагмент найден. Используя другие методы Matcher, например, start() и end() можно получить сам найденный фрагмент, в случае start и end позицию начального вхождения фрагмента в текст и позицию за вхождением последнего символа фрагмента. Чтобы перейти к следующему фрагменту нужно вызвать find снова.

```
Pattern p = Pattern.compile("[\\w\\d]{6,}");
String text = "***ab12cd+++aaa---abracadabra1234…";
Matcher m = p.matcher(text);
while(m.find()) {
        int begin = m.start();
        int end = m.end();
        System.out.println(text.substring(begin, end));
}
```

Выводит

```
ab12cd
abracadabra1234
```

Если в регулярном выражении заданы группы - наборы обычных и специальных символов, заключенные в круглые скобки, можно получить соответствующие найденные фрагменты текста с помощью метода group.

Есть еще много нерассмотренных возможностей. Чтобы разобраться с ними смотрите, например, книгу Фицджеральда, источник [5] в списке литературы, и документацию Java.

ЗАДАНИЕ 1. Вывести на экран в виде таблицы (протабулировать) значения функций sin(x), e^x / x * lg(x) для значений x из интервала [π/15..pi], меняющихся с шагом π/15. Для столбца со значениями аргумента нужно использовать представление с фиксированной точкой, ширина столбца - 10 позиций, точность - 5 знаков после запятой. Для столбца со значениями функции: экспоненциальное представление, ширина 15 позиций, точность 7 знаков.

ЗАДАНИЕ 2. Реализовать алгоритм нахождения наибольшего отрицательного элемента двумерного массива, содержащего произвольное число строк и столбцов и, быть может, различное число элементов в каждой строке.

ЗАДАНИЕ 3. Написать приложение для реализации следующего алгоритма: упорядочить по возрастанию элементы каждой строки матрицы 3x3. Вывести матрицу на экран до сортировки и после.

ЗАДАНИЕ 4. Создать класс с методом, реализующим следующий алгоритм: на плоскости размещены две окружности, центр первой в координатах x1, y1 и ее радиус r1, а центр и радиус другой в x2, y2, r2, конкретные значения передаются методу через его аргументы или хранятся в полях класса. Определить пересекаются ли окружности в одной точке (касаются), в двух точках, совпадают, не пересекаются и ни одна из окружностей не является вложенной в другую, вторая окружность вложена в первую, первая вложена во вторую. Метод должен возвращать целое значение, соответствующее возникшей ситуации. Протестировать работу метода, запрашивая у пользователя (ввод с клавиатуры) параметры окружностей и выводя сообщение о их взаимоположении.

Java поддерживает перечислимые типы. В определении типа задаются имена, которые и являются допустимыми значениями для переменных этого типа. Значения перечислимого типа можно сравнивать друг с другом. Например,

```
enum MyColor { White, Black, Red, Green, Blue }
public class Test {
        public static void main(String[] args) {
                MyColor c = MyColor.Red;
                System.out.println( c ); // Red
                System.out.println( MyColor.Green ); // Green
```

```
            if(c == MyColor.Red) System.out.println("Red color");
            else System.out.println("Unknown color");
        }
}
```

ЗАДАНИЕ 5. Измените класс из предыдущего задания так, чтобы в нем был определен подходящий перечислимый тип, и чтобы метод возвращал значение этого перечислимого типа.

ЗАДАНИЕ 6. Написать метод, реализующий алгоритм численного интегрирования левыми прямоугольниками. В программе-тесте протабулировать функцию $y(x) = e^x - x^3$ на интервале, который задает пользователь (например, от 0 до 4) с постоянным шагом так, чтобы получить значения аргумента x и функции y в 101 точке (два массива). Передать массивы методу интегрирования. Посчитать значение интеграла аналитически и сравнить результат, возвращаемый методом с точным значением.

ЗАДАНИЕ 7. Написать приложение для преобразования целого числа из десятичного представления в представление в заданной системе счисления (от 2 до 8 включительно). Для проверки правильности работы программы используйте методы классов-оболочек toString(value, base), где value - преобразуемое десятичное значение, base - основание системы счисления, в которую нужно переводить value. Например, String val3 = Integer.toString(12, 3); // 12 в троичную систему счисления (в своем алгоритме использовать toString нельзя!).

ЗАДАНИЕ 8. Реализуйте вычисление значения полинома n-й степени по схеме Горнера. Суть схемы в том, что запись полинома

$P(x) = a_n * x^n + a_{n-1} * x^{n-1} + \ldots + a_0$ преобразуется в

$P(x) = (\ldots((a_n * x + a_{n-1}) * x + a_{n-2}) * x + \ldots) * x + a0$

Полином в программе представляется массивом его коэффициентов (массив из n+1 элемента), а сами вычисления выполняются в цикле так, что в начале $P = a_n * x + a_{n-1}$, а затем на каждой итерации $P = P * x + a_i$.

ЗАДАНИЕ 9. 1) Сформируйте регулярное выражение для проверки того, содержит ли заданная строка представление одиннадцатизначного (федерального) телефонного номера российского оператора связи. В начале должны быть либо символы +7, либо 8. Кроме того, номер может содержать в определенных позициях пробелы, дефисы, круглые скобки. Например, правильными будут следующие варианты

+79043781661	+7 904 378 1661	+7 904 378 16 61
+7-904-378-16-61	+7(904)3781661	+7(904) 378-16 61
89043781661	8 904 378-16-61	

Круглыми скобками могут быть выделены только три цифры после префикса +7 или 8. Пробелы могут быть после префикса, после трех цифр, следующих за префиксом, после следующих трех цифр, после следующих двух цифр. На тех же позициях могут встречаться дефисы. Проверьте корректность полученного регулярного выражения на приведенных выше примерах и для других вариантов (допустимых и не допустимых) представления телефонных номеров.

2) Сформируйте регулярное выражение для проверки того, содержится ли где-то в заданной строке представление семизначного (муниципального) телефонного номера Ростова-на-Дону. Номер должен начинаться с цифры 2 или 3, эта цифра может быть отделена от остальной части номера пробелом или дефисом (а может быть и не отделена ничем), далее каждая из трех пар цифр также может отделяться пробелом или дефисом. Например, содержимое следующей строки должно соответствовать построенному регулярному выражению

"Мои номера 220-30-40 и 8904-378-16-61 не считая служебных"

ЗАДАНИЕ 10. Используйте регулярные выражения из предыдущего задания для того, чтобы извлечь из строки

"Мои номера 220-30-40 и 8904-378-16-61 не считая служебных"

все содержащиеся в ней номера телефонов.

Литература

1. К.Арнольд, Дж.Гослинг, Д.Холмс "Язык программирования Java", 3-е изд. – М.: Вильямс. – 2001, 624с.
2. М.Холл, Л.Браун "Программирование для Web. Библиотека профессионала". – М.: Вильямс. – 2002, 1264с.
3. Н.А.Прохоренок "Основы Java", 2-е изд. – СПб.:BHV-Петербург. – 2019, 768с.
4. П.Ноутон, Г.Шилдт "Java 2". – СПб.:BHV-Петербург. – 2001, 1072с.
5. М.Фицджеральд "Регулярные выражения. Основы". – М.: Вильямс. – 2015, 144с.

Редактор А.А. Литвинова

ЛР № 04779 от 18.05.01.	В набор	В печать
Объем 0,5 усл.п.л., уч.-изд.л.	Офсет.	Формат 60х84/16.
Бумага тип №3.	Заказ №	Тираж 140. Цена

Издательский центр ДГТУ

Адрес университета и полиграфического предприятия:

344010, г. Ростов-на-Дону, пл. Гагарина, 1.