УНИВЕРСИТЕТ ИТМО

А.В. Лямин, Е.Н. Череповская

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ. КОМПЬЮТЕРНЫЙ ПРАКТИКУМ

Санкт-Петербург
2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.В. Лямин, Е.Н. Череповская

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ. КОМПЬЮТЕРНЫЙ ПРАКТИКУМ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 11.03.02 «Инфокоммуникационные технологии и системы связи», 09.03.02 «Информационные системы и технологии» и другим информационным направлениям в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ бакалавриата

Санкт-Петербург

2017

Лямин А.В., Череповская Е.Н. Объектно-ориентированное программирование. Компьютерный практикум. – СПб: Университет ИТМО, 2017. – 143 с.

Рецензенты:
Чежин М.С., к.т.н., доцент
Меженин А.В., к.т.н., доцент

Компьютерный практикум содержит краткие справочные материалы и задания для изучения языков программирования Python и Java, включая операторы ветвления и циклов, форматированный ввод и вывод данных, работу со структурами данных и файлами, объектно-ориентированное программирование и программирование многопоточных приложений, веб-программирование. Закрепление полученных навыков осуществляется на задачах разного уровня сложности из области информатики. Предназначается для студентов бакалавриата, обучающихся по направлениям 11.03.02 «Инфокоммуникационные технологии и системы связи», 09.03.02 «Информационные системы и технологии» и другим информационным направлениям.

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Содержание

Лабораторная работа № 1: Ввод, вывод и типы данных

1. Запустите онлайн среду разработки "Tutorials Point" с рабочим пространством для языка программирования Python версии 3 по ссылке:

https://www.tutorialspoint.com/execute_python3_online.php

Интерфейс среды разработки состоит из трех областей: с левой стороны располагается каталог текущих файлов проекта, с правой стороны – редактор кода программы, под редактором кода находится терминал для ввода команд. В ходе выполнения лабораторных работ будет необходимо создавать программы и запускать их командами из терминала. Для сохранения файлов проекта можно использовать локальный диск ("Project" → "Download project") или диск Google ("Project" → "Save project" → "Save on Google Drive").

2. Создайте программу **hello-world.py** по следующему шаблону:

```
print("Hello World!");
```

Функция **print()** служит для вывода информации в терминал (консоль, командную строку).

3. Запустите программу, введя в терминале следующую команду:

```
python hello-world.py
```

В терминале будет выведено сообщение "Hello World!".

Запуск программ

Программы, написанные на языке программирования Python, запускаются с помощью команды **python <имя_файла>**. Python является некомпилируемым языком программирования, однако существуют специальные расширения для его компиляции в код на языке программирования Си. Следует отметить, что отсутствие точки с запятой в конце команды не является ошибкой для данного языка программирования.

4. Создайте новую программу **lab_01_01.py** по следующему шаблону:

```
# однострочный комментарий
'''
Многострочный
комментарий
'''

'''
    Пример работы с целыми числами
'''
a = 15
b = 7
print("a = ", a)
print("b = ", b)
print("a + b = ", a + b) # сложение чисел
print("a - b = ", a - b) # вычитание чисел
print("a * b = ", a * b) # умножение чисел
print("a / b = ", a / b) # деление чисел
print("a // b = ", a // b) # деление с округлением
вниз до целого
print("a % b = ", a % b) # остаток от деления
print("a ** b = ", a ** b) # возведение в степень
# приведение типа int к float
a1 = float(a)
print("a1 = ",a1)
print("\n")

'''
    Пример работы с вещественными числами
'''
c = 3.5
d = 6.9
print("c = ", c)
print("d = ", d)
print("c + d = ", c + d) # сложение чисел
print("c * d = ", c * d) # умножение чисел
print("c / d = ", c / d) # деление чисел

d1 = 2.3e-5
print("d1 = ",d1)
print("c + d1 = ", c + d1) # сложение чисел
# приведение типа float к int
d2 = int(d)
print("d2 = ",d2)
```

```
print("\n")

'''
    Операторы присваивания
'''
e = 7
print("e = 7: ", e)
e += 3
print("e += 3: ", e)
e -= 3
print("e -= 3: ", e)
e *= 15
print("e *= 5: ", e)
e /= 4
print("e /= 4: ", e)
e //= 2
print("e //= 2: ", e)
e %= 5
print("e %= 5: ", e)
e **= 3
print("e **= 3: ", e)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
a =  15
b =  7
a + b =  22
a - b =  8
a * b =  105
a / b =  2.142857142857143
a // b =  2
a % b =  1
a ** b =  170859375
a1 =  15.0


c =  3.5
d =  6.9
c + d =  10.4
c * d =  24.150000000000002
c / d =  0.5072463768115941
```

```
d1 =  2.3e-05
c + d1 =  3.500023
d2 =  6


e = 7:  7
e += 3:  10
e -= 3:  7
e *= 5:  105
e /= 4:  26.25
e //= 2:  13.0
e %= 5:  3.0
e **= 3:  27.0
```

Типы данных

В языке программирования Python используется динамическое определение типа данных. К простым типам данных относятся целые и вещественные числа, для которых доступны следующие арифметические операции:

- **+** (сложение)
- **-** (вычитание)
- ***** (умножение)
- **/** (деление)
- **//** (деление с округлением вниз до целого)
- **%** (остаток от деления)
- ****** (возведение в степень)

Для присваивания переменной определенного значения используется оператор присваивания: **=**. Также могут быть использованы сокращенные операции для автоматического присваивания переменной результата выполненного над ней выражения:

- **+=** (сложение с присваиванием)
- **-=** (вычитание с присваиванием)
- ***=** (умножение с присваиванием)
- **/=** (деление с присваиванием)
- **//=** (деление с округлением вниз до целого с присваиванием)
- **%=** (остаток от деления с присваиванием)
- ****=** (возведение в степень с присваиванием)

При использовании сокращенных операций результат выполнения **a += b** будет идентичен результату выполнения выражения **a = a + b**.

Для вещественных чисел язык Python поддерживает запись чисел в экспоненциальном формате, например, **`0.5e4`**, что эквивалентно **`0.5*10⁴`**.

При необходимости привести целый тип к вещественному или же строковые данные к целому числу используется приведение типов. Для этого используются функции **`int()`**, **`float()`** и т.д., с указанием числа, которое необходимо привести к нужному формату в качестве аргумента функции в скобках.

5. Модифицируйте программу **`lab_01_01.py`**, изменив значения переменных **`a`** и **`b`**. Ознакомьтесь с результатом.

6. Модифицируйте программу **`lab_01_01.py`**, создав переменную **`z`** со значением 3 и переменную **`y`** со значением 5. Осуществите операции сложения, вычитания, умножения, деления, деления с округлением вниз, вычисления остатка от деления и возведения в степень над созданными переменными, выводя результаты на экран.

7. Дополните код программы **`lab_01_01.py`**. Создайте переменную **`x`** со значением 105 и **`v`** со значением 58. Осуществите вывод результата деления переменной **`x`** на переменную **`v`** на экран с приведением типа к вещественному и без приведения типов. Ознакомьтесь с результатом.

Логические и побитовые операции

В языке программирования Python дополнительно к основным существует булевский тип данных, для которого доступны два значения:

- **`True`**
- **`False`**

Для переменных и результатов выражений булевского типа доступны следующие логические операции:

- **`not a`** – логическое НЕ
- **`a and b`** – логическое И
- **`a or b`** – логическое ИЛИ
- **`a == b`** – эквиваленция
- **`a != b`** – проверка неравенства двух значений
- **`>`**, **`>=`**, **`<`**, **`<=`**

Python поддерживает возможность записи двусторонних условий,

например:
0 <= h <= 10
Для работы с двоичными представлениями чисел предусмотрены побитовые операции:

- **x & y** – побитовое И
- **x | y** – побитовое ИЛИ
- **x ^ y** – побитовое исключающее ИЛИ
- **~x** – побитовая инверсия числа **x**
- **x >> n** – побитовый сдвиг числа **x** вправо на **n** бит
- **x << n** – побитовый сдвиг числа **x** влево на **n** бит

Для двоичного представления числового значения используется функция **bin()**, при этом число передается в качестве аргумента функции, например **bin(5)**.

8. Создайте новую программу **lab_01_02.py** по следующему шаблону:

```
'''
    Логические операции
'''
f = True
g = False
print("f: ", f)
print("not f: ", not f)
print("f and g: ", f and g)
print("f or g: ", f or g)
print("f == g: ", f == g)
print("f != g: ", f != g)
print("\n")
h = 3
i = 5
print("h = ", h)
print("i = ", i)
print("h > i: ", h > i)
print("h < i: ", h < i)
print("h >= i: ", h >= i)
print("0 < h <= i: ", 0 < h <= i)
print("\n\n")

'''
```

```
    Побитовые операции
'''
j = 7
k = 20
print("j = %d; j in binary format: %s" % (j, bin(j))
)
print("k = %d; k in binary format: %s" % (k, bin(k))
)
print("j & k: %d; binary: %s" % (j & k, bin(j & k)) )
# побитовое AND
print("j | k: %d; binary: %s" % (j | k, bin(j | k)) )
# побитовое OR
print("j ^ k: %d; binary: %s" % (j ^ k, bin(j ^ k)) )
# побитовое XOR
print("~k: %d; binary: %s" % (~k, bin(~k)) ) #
инверсия двоичного числа
print("k>>1: %d; binary: %s" % (k>>1, bin(k>>1)) ) #
сдвиг на один бит вправо
print("k<<1: %d; binary: %s" % (k<<1, bin(k<<1)) ) #
сдвиг на один бит влево
print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
f:  True
not f:  False
f and g:  False
f or g:  True
f == g:  False
f != g:  True

h =  3
i =  5
h > i:  False
h < i:  True
h >= i:  False
0 < h <= i:  True

j = 7; j in binary format: 0b111
k = 20; k in binary format: 0b10100
```

```
j & k: 4; binary: 0b100
j | k: 23; binary: 0b10111
j ^ k: 19; binary: 0b10011
~k: -21; binary: -0b10101
k>>1: 10; binary: 0b1010
k<<1: 40; binary: 0b101000
```

9. Дополните код программы **lab_01_02.py**. Создайте переменные **A** и **B** с неравными целочисленными значениями, переменные **C** и **D** со значениями **True** и **False** соответственно.

10. Дополните код программы **lab_01_02.py**. Осуществите вывод на экран значений следующих логических выражений:

- **¬(C∧D)**
- **C∧D∨¬(C∧D)**
- **¬C∨D**

11. Дополните код программы **lab_01_02.py**. Осуществите вывод в консоль значений следующих выражений:

- **A<=B**
- **A>B ∨ A==B**
- **¬(A<B)**

12. Дополните код программы **lab_01_02.py**. Создайте переменную **s** со значением 154 и переменную **p** со значением 6. Осуществите вывод ее значения в десятичном и двоичном виде на экран.

13. Дополните код программы **lab_01_02.py**. Выполните операцию побитового ИЛИ над переменными **s** и **p** с записью результата в переменную **s**. Осуществите вывод значения в десятичном и двоичном виде на экран.

14. Дополните код программы **lab_01_02.py**. Выполните операцию побитового сдвига вправо на 2 бита над переменными **s** и **p** с записью результатов в соответствующие переменные. Осуществите вывод значений в десятичном и двоичном виде на экран.

15. Создайте новую программу **lab_01_03.py** по следующему шаблону:

```
'''
    Форматированный ввод/вывод данных
'''
m = 10
pi = 3.1415927
print("m = ",m)
print("m = %d" % m)
print("%7d" % m)
print("pi = ", pi)
print("%.3f" % pi)
print("%10.4f\n" % pi)
print("m = {}, pi = {}".format(m,pi))
ch = 'A'
print("ch = %c" % ch)
s = "Hello"
print("s = %s" % s)
print("\n\n")

code = input("Enter your position number in group: ")
n1, n2 = input("Enter two numbers splitted by space:
").split()
d, m, y = input("Enter three numbers splitted by
\'.\': ").split('.')
print("{} + {} =
{}".format(n1,n2,float(n1)+float(n2)))
print("Your birthday is %s.%s.%s and you are %d in
the group list" % (d,m,y,int(code)) )
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
m =  10
m = 10
     10
pi =  3.1415927
3.142
    3.1416

m = 10, pi = 3.1415927
```

```
ch = A
s = Hello

Введите Ваш номер по списку группы:  10
Введите два числа через пробел:  4 7
Введите дату Вашего рождения:  01.01.1234
4 + 7 = 11.0
Ваш день рождения 01.01.1234 и Вы 10 в списке учебной группы
```

Форматированный вывод

Функция **print()** служит для вывода информации на экран. В языке программирования Python существует неформатированный вывод значений на экран, когда тип выводимого на экран значения определяется автоматически. В этом случае используется следующий вариант записи команды:

```
print("m = ", m)
```

При необходимости форматированного вывода значений в терминал используются различные спецификаторы для разных типов данных, в частности:

- **%i** или **%d** для целых чисел
- **%ld** для длинных целых чисел
- **%f** для вещественных чисел
- **%lf** для длинных вещественных чисел
- **%c** для символьного типа данных
- **%s** для строковых данных

В этом случае используется следующий вариант записи команды:

```
print("m = %d" % m)
```

Знак **%** перед именем переменной в аргументе функции предшествует перечисляемым аргументам.

Функция **print()** по умолчанию осуществляет вывод на новой строке. Для дополнительного перевода строки при выводе информации следует использовать **'\n'**. Символ **'\'** используется также для вывода на экран некоторых специальных символов, например, **'\'**, **'''**, **'"'**.

При выводе целочисленных и вещественных числовых значений возможно изменение базового спецификатора с целью округления выводимого числа:

- **%d** – вывод целого числа без форматирования
- **%3d** – вывод целого числа шириной как минимум 3 знака
- **%f** – вывод вещественного числа без форматирования
- **%4**f – вывод вещественного числа шириной как минимум 4 знака
- **%.5f** – вывод вещественного числа с точностью до 5 знака после запятой
- **%3.2f** – вывод вещественного числа шириной как минимум 3 знака с точностью до 2 знака после запятой

Пример наиболее часто используемых форматов округления:

Команда	Результат
print("%d",305)	**305**
print("%4d",17)	** 17**
print("%f",7.2)	**7.2**
print("%.1f",10.87)	**10.9**
print("%3.2f",3.14159)	**3.14**

При выводе нескольких аргументов может быть использован форматированный вывод, тогда все необходимые для вывода аргументы должны быть указаны в скобках после знака **%** в функции **print()**:

```
print("m = %d, n = %f" % (m,n))
```

Если тип выводимого значения точно не известен или же форматирование не требуется возможен вывод без указания формата (места вставки значений обозначаются пустыми фигурными скобками **{}**). В этом случае тип значения будет определен автоматически. Для этого используется функция **format()**, в качестве аргументов которой передаются значения для подстановки в выводимую строку. Функция **format()** вызывается от главного аргумента функции **print()**:

```
print("m = {}, n = {}".format(m,n))
```

Общий вид команды вывода в Python представляется как:

```
print(*objs, sep=' ', end='\n', file=sys.stdout,
```

flush=False),

где:

- ***objs** – список выводимых значений;
- **sep** – разделитель, используемый при выводе, в частности для осуществления вывода значений в одну строку;
- **end** – символ, печатаемый по окончании вывода значений;
- **file** – поток вывода данных (консоль / файл);
- **flush** – автоматический выброс данных в поток.

Форматированный ввод

Ввод информации из терминала в языке программирования Python осуществляется с использованием функции **input()**. Для записи введенного значения в переменную необходимо присвоить возвращаемого функцией значение переменной, например:

```
a = input("Введите число: ")
```

В этом случае введенное число будет записано в переменную `a`.

При необходимости ввода нескольких значений с клавиатуры и их записи в различные переменные используется функция **split()**, вызываемая от возвращаемого функцией значения. Отсутствие аргумента функции **split()** означает, что разделитель для вводимых значений будет установлен как пробел (по умолчанию). В случае использования другого разделителя – необходимо указать его в скобках. Все значения будут считаны в строковом формате, для перевода их к числовому виду, необходимо выполнять приведение типов. Переменные, использующиеся для записи значений, в этом случае указывается перед оператором присваивания через запятую. Формат команды ввода в этом случае будет следующим:

```
a,b = input("Введите число: ").split()
```

или, в случае использования разделителя `'/'`

```
a,b = input("Введите два числа через \'/\': ").split('/')
```

16. Дополните код программы **lab_01_03.py**. Осуществите форматированный вывод в терминал значений переменных **m** и **pi**, подставив вместо многоточий их значения, в формате: "**m = ..; pi = ..**". При этом значение **m** должно быть выведено шириной 4 знака, а значение **pi** с точностью до 3 знаков после запятой.

17. Дополните код программы **lab_01_03.py**. Осуществите вывод в терминал значений переменных **m** и **pi**, подставив вместо многоточий их значения, в формате: "**m = ..; pi = ..**", используя автоматическое определение типов переменных с помощью функции **format()**.

18. Дополните код программы **lab_01_03.py**. Осуществите ввод значения номера Вашего курса обучения с клавиатуры, записав введенное значение в переменную **year**. Выведите значение переменной в терминал.

19. Дополните код программы **lab_01_03.py**. Осуществите ввод значений Ваших баллов ЕГЭ по русскому языку, математике и профильному предмету, записав введенные значения в переменные **r1**, **m1**, **p1**. В качестве разделителя при вводе значений используйте запятую. Выведите значения переменных в терминал.

20. Дополните код программы **lab_01_03.py**. Осуществите преобразование в десятичную систему счисления введенных двенадцатиразрядных чисел в системе счисления с основанием, которое равно увеличенному на два остатку от деления Вашего дня рождения на 8, и выведите результат преобразования в терминал.

21. Дополните код программы **lab_01_03.py**. Осуществите умножение и деление введенного с клавиатуры числа на два с использованием операции побитого сдвига влево и вправо соответственно. Выведите результат в терминал.

Лабораторная работа № 2: Операторы ветвления, циклы, списки и строки

1. Создайте новую программу **lab_02_01.py** по следующему шаблону:

```
'''
    Условия
'''
# if..else
num = int(input("How many times have you been to the Hermitage? "))
if num > 0:
    print("Wonderful!")
    print("I hope you liked this museum!")
else:
    print("You should definitely visit the Hermitage!")

# if..elif..else
course = int(input("What is your course number?"))
if course == 1:
    print("You are just at the beginning!")
elif course == 2:
    print("You learned many things, but not all of them!")
elif course == 3:
    print("The basic course is over, it's time for professional disciplines!")
else:
    print("Oh! You need to hurry! June is the month of thesis defense")

x = 5
y = 12
if y % x > 0 : print("%d cannot be evenly divided by %d" % (y,x))

z = 3
x = "{} is a divider of {}".format(z,y) if y%z==0 else "{} is not a divider of {}".format(z,y)
print(x)
```

```
print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
How many times have you been to the Hermitage?  2
Wonderful!
I hope you liked this museum!
What is your course number? 1
You are just at the beginning!
12 cannot be evenly divided by 5
3 is a divider of 12
```

Условный оператор

В языке программирования Python существует лишь один тип оператора ветвления: **if** … **elif** … **else**:

```
if <условие>:
    <операторы>
[elif <условие>:
    <операторы>]
[else:
    <операторы>]
```

В квадратных скобках **[ ]** указаны необязательные блоки. В качестве проверяемого условия (**<условие>**) может выступать простое или составное логическое выражение, использующее различные логические операции. Одним из основных отличий языка программирования Python от других языков программирования является отсутствие явных границ для вложенных операторов (**<операторы>**) в условиях, циклах и других операторах и блоках. В Python уровень вложенности показывается с помощью отступов, указанных перед тем или иным оператором. Вложенность блока операций в **if** или **else**, определяется отступом перед каждым из операторов.

В случае, если оператор ветвления содержит лишь один вложенный оператор, он может быть записан в следующем виде:

```
if a>b: c=1
```

Для проверки выполнения условия также может быть использован тернарный оператор ветвления, т.е. сокращенная форма условия, который представляется в следующем формате:

```
c = 1 if a>b else 0
```

Следует отметить, что в языке программирования Python отсутствует возможность использования **switch..case**.

2. Дополните код программы **lab_02_01.py**. Создайте переменную **p**, в которую будет записано значение количества выполненных за год лабораторных по различным дисциплинам. Осуществите вывод значения переменной в терминал, если значение больше 10. Оператор ветвления запишите двумя способами: в одну строку и в несколько строк. Ознакомьтесь с результатом.

3. Дополните код программы **lab_02_01.py**, создав переменные **a** со значением 157 и **b** со значением 525. Осуществите проверку следующих условий и выполнение соответствующих действий:

- если **a>b**, рассчитайте остаток от деления **a** на **b** и выведите значение на экран;
- если **a<b**, рассчитайте остаток от деления **b** на **a** и выведите значение на экран;
- если **a==b**, рассчитайте произведение чисел **a** и **b** и выведите значение на экран.

Осуществите проверку, изменяя значения переменных в соответствии с условиями. Ознакомьтесь с результатом.

4. Создайте новую программу **lab_02_02.py** по следующему шаблону:

```
'''
    Циклы
'''
# while
print("Numbers < 10 (while):")
i = 0
while (i<10):
    print(i, end=" ") # print in one line
    i += 1
print("\n")
```

```
# for
print("Numbers < 10 (for):")
for i in range(0,10):
    print(i, end=" ")
else:
    print("\nThe next number is 10\n")

# break
sum = 0
for i in range(0,100):
    if i > 10:
        print("\nWe reached the end, final sum: ",
sum)
        break
    sum += i

# continue
i = 0
while i<=15:
    if i % 3 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1

print("\n")

# pass
print("Let's print numbers again!")
for i in range(0,10):
    pass
    print(i, end=" ")

print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Numbers < 10 (while):
0 1 2 3 4 5 6 7 8 9
```

```
Numbers < 10 (for):
0 1 2 3 4 5 6 7 8 9
The next number is 10

We reached the end, final sum:  55
1 2 4 5 7 8 10 11 13 14

Let's print numbers again!
0 1 2 3 4 5 6 7 8 9
```

Циклы

В языке программирования Python существует два типа циклов: цикл с итератором и цикл с предусловием. Вложенные операторы должны быть обозначены отступом. Цикл с итератором представляется в следующем виде:

```
for <переменная> in <объект>:
    <операторы>
```

Здесь **<переменная>** – переменная-итератор, которая представляет собой определенный объект или элемент из списка (**<объект>**), одно из свойств объекта (**<объект>**) и т.д. По окончании выполнения вложенных операторов (**<операторы>**) и переходе к следующей итерации, происходит переход к следующему объекту, свойству и т.д. Например, для написания цикла на 10 итераций можно использовать запись: **for i in range(0,10):** …, где **range** – функция получения списка значений заданного интервала.

Цикл с предусловием реализуется в следующем формате:

```
while <условие>:
    <операторы>
```

Одним из основных отличий языка Python является возможность использования **else** для циклов обоих видов, что позволяет определить операторы, которые будут выполнены, если условие цикла дало ложный результат. Для цикла с предусловием данный блок указывается следующим образом:

```
while <условие>:
    <операторы>
else:
    <операторы>
```

Также внутри циклов могут быть использованы следующие ключевые слова:

- **break** – остановка цикла
- **continue** – переход на следующую итерацию цикла
- **pass** – не несет смыслового значения, может быть использован для упрощения идентификации части кода, которую необходимо дополнить позже

5. Дополните код программы **lab_02_02.py**. Осуществите вывод на экран чисел в диапазоне от 0 до 500, кратных 7, с использованием двух видов циклов. По окончании работы циклов в блоке **else** выведите сообщение **"All numbers were printed!"**. Ознакомьтесь с результатом.

6. Дополните код программы **lab_02_02.py**. Модифицируйте написанные на предыдущем шаге циклы, добавив пропуск вывода значений кратных 14 и остановку цикла по достижении значения 300 (с помощью **break**). Ознакомьтесь с результатом.

7. Дополните код программы **lab_02_02.py**. Осуществите вывод на экран следующей таблицы с использованием двух видов циклов:

```
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
```

8. Создайте новую программу **lab_02_03.py** по следующему шаблону:

```
'''
    Списки
'''
a = [1,2,3,4,5]
print("a[0]: ", a[0])
print("List a[0:5]: ", a[0:5])
```

```
print("List a[:]: ", a[:])
print("List a: ", a)
print("List a[1:5]: ", a[1:])
print("List a[0:4]: ", a[:4])
print("Length of list a: ", len(a))

print("\nList a (by index):")
# получение элементов списка в цикле (1)
for i in range(0,len(a)):
    print(a[i], end=" ")
print("\nList a (by elements):")
# получение элементов списка в цикле (2)
for elem in a:
    print(elem, end=" ")
print("\n")

b = []
b.append(20) # добавление элемента в конец
b.extend(a) # добавление элементов списка a в b
print("List b (extended): ",b)
b.insert(3,5) # добавить элемент на позицию
print("List b (insert element): ",b)
b.remove(5) # удалить первый элемент, равный 5
print("List b (remove element): ",b)
c1 = b.pop() # удалить и вернуть последний элемент
print("List b (pop last element): ",b)
print("c1: ",c1)
c2 = b.pop(3) # удалить и вернуть эл. с позиции
print("List b (pop 3rd element): ",b)
print("c2: ",c2)
bCopy = b.copy() # создать копию списка
print("List b: ",b)
print("List bCopy: ",bCopy)
b.reverse() # поменять элем. местами с конца в начало
print("List b (reversed): ",b)
b.sort(reverse=True) # сортировка элементов списка
print("List b (sorted): ",b)
b.clear(); # очистка списка
print("List b (cleared): ",b)
print("\n")

# сравнение списков
a1 = [1,2,4]
a2 = [1,2,3]
```

```
print("a1 == a2: ", a1 == a2)
a1.append(-1)
print("a1 == a2: ", a1 == a2)
print("a1 > a2: ", a1 > a2)
print("a1 < a2: ", a1 < a2)
print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
a[0]:  1
List a[0:5]:  [1, 2, 3, 4, 5]
List a[:]:  [1, 2, 3, 4, 5]
List a:  [1, 2, 3, 4, 5]
List a[1:5]:  [2, 3, 4, 5]
List a[0:4]:  [1, 2, 3, 4]
Length of list a:  5

List a (by index):
1 2 3 4 5
List a (by elements):
1 2 3 4 5

List b (extended):  [20, 1, 2, 3, 4, 5]
List b (insert element):  [20, 1, 2, 5, 3, 4, 5]
List b (remove element):  [20, 1, 2, 3, 4, 5]
List b (pop last element):  [20, 1, 2, 3, 4]
c1:  5
List b (pop 3rd element):  [20, 1, 2, 4]
c2:  3
List b:  [20, 1, 2, 4]
List bCopy:  [20, 1, 2, 4]
List b (reversed):  [4, 2, 1, 20]
List b (sorted):  [20, 4, 2, 1]
List b (cleared):  []

a1 == a2:  False
a1 == a2:  False
a1 > a2:  True
a1 < a2:  False
```

Списки

Одним из основных типов данных в языке программирования Python являются списки (Lists) – индексированные совокупности переменных разного типа. Новый пустой список может быть создан с использованием любого из следующих операторов:

```
a = []
a = list()
```

Для создания заполненного списка, его значения указываются в квадратных скобках:

```
a = [1, 2, 3, "keyboard", "mouse", [0,1] ]
```

Получение количества элементов списка осуществляется с помощью функции **`len(a)`**, где **`a`** – имя списка. Доступ к элементам списка осуществляется по индексу. Могут быть использованы следующие форматы записи, в зависимости от поставленной задачи:

- **`a[0]` – получение элемента на позиции 0**
- **`a[0:5]` – получение первых пяти элементов**
- **`a[-1]` – получение последнего элемента списка**
- **`a[-2]`** – получение элемента, находящегося на 2-й позиции с конца списка
- **`a[:3]`** – получение элементов от начала списка до 3 позиции
- **`a[2:]`** – получение элементов от 2 позиции до конца списка
- **`a[:]`** – получение всех элементов списка
- **`a[0:len(a)]`** – получение всех элементов списка
- **`a`** – получение всех элементов списка, например: **`print(a)`**

Для удаления элемента из списка используется ключевое слово **`del`**, например, **`del a[3]`** для удаления элемента списка или **`del a`** для удаления списка целиком.

Для работы со списком могут быть использованы функции, описанные далее в таблице. В квадратных скобках указаны аргументы, которые могут быть опущены.

Функция	Описание	Пример
append(*element*)	Добавление элемента ***element*** в конец списка	**a.append(1)**
extend(*list*)	Добавление в список элементов списка ***list***	**a.extend([2,3,4])**
insert(*index*,*element*)	Вставка элемента ***element*** на позицию ***index***	**a.insert(1,23)**
remove(*element*)	Удаление первого встретившегося элемента ***element***	**a.remove(3)**
pop([*index*])	Удаление и возвращение последнего элемента списка (если функция вызвана без аргументов) или элемента с указанной позиции ***index***	**el = a.pop()** **el = a.pop(2)**
copy()	Создание и возвращение копии элементов списка	**aC = a.copy()**
reverse()	Изменение положения элементов списка (и конца в начало)	**a.reverse()**
sort([*reverse={True|False}*])	Сортировка элементов по возрастанию (***reverse=False***) или по убыванию (***reverse=True***)	**a.sort()**
clear()	Очистка списка: удаление всех элементов из списка	**a.clear()**

В таблице приведены основные функции. Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном структурам данных:
https://docs.python.org/3/tutorial/datastructures.html

Для сравнения списков используются стандартные операторы сравнения. Элементы списка поочередно сравниваются между собой по лексикографическому принципу. Если все элементы списков равны между собой – списки считаются равными.

9. Дополните код программы **lab_02_03.py**. Создайте список **list1**, заполненный 10 значениями, введенными с клавиатуры. Отсортируйте список по убыванию, используя стандартную функцию, после чего выведите последние пять значений списка на экран.

10. Дополните код программы **lab_02_03.py**. Создайте список **list2**, являющий копией списка **list1**, значения которого перенесены из конца списка в начало. Используйте стандартные функции. Выведите список **list2** на экран до и после реверсирования значений.

11. Создайте новую программу **lab_02_04.py** по следующему шаблону:

```
'''
    Строки
'''
group = input("What is your group number? ")
print("Your group is ",group)
print("\nString:")
# получение символов строки в цикле (1)
for i in range(0,len(group)):
    print(group[i], end=" ")
print("\nString:")
# получение символов строки в цикле (2)
for elem in group:
    print(elem, end=" ")
print("\n")

s = "Hello, World!"
print(s)
num = s.count('o')
print("Count 'o': ",num)

ind = s.find("world")
print("Find 'world' (by find()): ",ind)
ind = s.find("World")
print("Find 'World' (by find()): ",ind)
ind = s.index("World")
print("Find 'World' (by index()): ",ind)
print("Find 'World' (by in operation): ", "World" in
s)
print("\nReplace substring:")
```

```
s1 = s.replace("Hello", "hello")
print(" Old: {} \n New: {} ".format(s, s1))
l1 = s.split(",")
print("List l1 (splitted): ",l1)
l2 = s.partition(",")
print("List l2 (partitioned): ",l2)
s2 = s
print("Copy s in s2; s2: ",s2)
sep = "|"
l = ["h","e","l","l","o"]
s = sep.join(l)
print(s)
print("\n")

st = "Привет, мир!"
st1 = st.encode("utf-8")
print("Encoding utf-8:\n",st1)
st2 = st.encode("cp1251")
print("Encoding cp1251:\n",st2)
st3 = st2.decode("cp1251")
print("Decoding cp1251:\n",st3)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
What is your group number?  P0001
Your group is  P0001

String:
P 0 0 0 1
String:
P 0 0 0 1

Hello, World!
Count 'o':  2
Find 'world' (by find()):  -1
Find 'World' (by find()):  7
Find 'World' (by index()):  7
Find 'World' (by in operation):  True

Replace substring:
 Old: Hello, World!
 New: hello, World!
```

```
List l1 (splitted):  ['Hello', ' World!']
List l2 (partitioned):  ('Hello', ',', ' World!')
Copy s in s2; s2:  Hello, World!
h|e|l|l|o


Encoding utf-8:
 b'\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82,
\xd0\xbc\xd0\xb8\xd1\x80!'
Encoding cp1251:
 b'\xcf\xf0\xe8\xe2\xe5\xf2, \xec\xe8\xf0!'
Decoding cp1251:
 Привет, мир!
```

Строки

Строки в языке программирования Python представляют собой списки символов. Они могут быть объявлены тремя способами:

- с использованием одинарных кавычек: **'строка'**
- с использованием двойных кавычек: **"строка"**
- с использованием трех кавычек одинарного или двойного типа: **'''строка'''** или **"""строка"""**

Доступ к символам строки осуществляется аналогично доступу к элементам списка, используя позицию символа в строке. Длину строки можно получить с помощью функции **len(s)**, где **s** – строка. Копирование значений строк происходит с использованием оператора присваивания. Конкатенация строк осуществляется с помощью операции сложения:

```
s1 = s + "abc"
```

Для работы со строками могут быть использованы функции, описанные далее в таблице. В квадратных скобках указаны аргументы, которые могут быть опущены.

Функция	Описание	Пример
count(*substr*)	Подсчет количества повторов подстроки ***substr*** в строке	**num = s.count('o')**

find(*substr*)	Поиск подстроки в строке. Возвращает позицию первого вхождения подстроки ***substr*** в строку. Если подстрока ***substr*** не найдена, возвращает **-1**. Другим вариантом проверки вхождения подстроки ***substr*** в строку является запись: ***substr* in s**, где **s** – строка	**ind = s.find("ab")**
index(*substr*)	Поиск подстроки ***substr*** в строке. Возвращает позицию первого вхождения подстроки ***substr*** в строку. Если подстрока ***substr*** не найдена, возвращает ошибку.	**ind = s.index("ab")**
replace(*oldS*, *newS*)	Возвращает строку, для которой выполнена замена подстроки ***oldS*** в предыдущей строке на новую подстроку ***newS***	**s1 = s.replace("ab", "AB")**
split([*sep*])	Разделение строки на элементы списка по разделителю ***sep***, если он указан, или по пробелу в ином случае.	**l1 = s.split(",")**
partition(*sep*)	Разделение строки на элементы списка, состоящего из трех элементов – часть строки до разделителя, разделитель (***sep***), часть строки после разделителя.	**l2 = s.partition(",")**
join(*iterable*)	Формирование строки из любой итерируемой структуры данных ***iterable*** (может быть указан разделитель в качестве основной строки).	**s = ";".join(l)**
encode(*encoding*)	Возвращает новую строку, являющуюся копией первой в кодировке ***encoding***	**st1 = st.encode("utf-8")**

decode(*encoding*)	Возвращает новую строку, являющуюся декодированной копией первой из кодировки ***encoding***	**st1 = st.decode("utf-8")**

В таблице приведены основные функции. Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:
https://docs.python.org/3/library/stdtypes.html#str

12. Дополните код программы **lab_02_04.py**. Создайте строковую переменную **season**, заполнив ее названием текущего сезона, введенным с клавиатуры. Смените кодировку данной строки на **utf-8**. Выведите закодированное и декодированное значения на экран.

13. Дополните код программы **lab_02_04.py**. Создайте переменную **sh** со значением *"When shall we three meet again; In thunder, lightning, or in rain?"*. Осуществите замену слов *"thunder"*, *"lightning"* и *"in rain"* на подстроки *"C"*, *"Erlang"* и *"Java main"* соответственно. Осуществите вывод итоговой строки на экран.

14. Создайте программу **lab_02_05.py**, которая выводит на экран все возможные уникальные строки, составленные из символов строки введенной с клавиатуры.

15. Создайте программу **lab_02_06.py**, которая выводит на экран дополнительный код введенного с клавиатуры шестнадцатеричного числа на восемь разрядов.

16. Создайте программу **lab_02_07.py**, которая преобразует введенное с клавиатуры двенадцатеричное число в систему с основанием 14 и выводит результат преобразования на экран.

Лабораторная работа № 3: Кортежи, множества, словари, функции, работа с файлами

1. Создайте файл **lab_03_01.py**:

```
'''
    Кортежи
'''
# создание кортежа
a1 = tuple()
a2 = 1, 2, 3, "abc"
a3 = (1, 2, 3, "abc")
print("Tuple a1 = ", a1)
print("Tuple a2 = ", a2)
print("Tuple a3 = ", a3)

# создание кортежа из других структур данных
l = [1, 2, 3, "abc"] # из списка
a4 = tuple(l)
print("Tuple a4 from list l = ", a4)
a5 = tuple("Hello, World!") # из строки
print("Tuple a5 from string = ", a5)

# вложенность кортежей
a6 = a2, a3
print("Tuple a6 formed by a2 and a3 = ", a6)

# объединение кортежей
a7 = a2 + a3
print("Tuple a7 by combining a2 and a3 = ", a7)

# доступ к элементам кортежей
print("a6[0]: ", a6[0])
print("a6[0][3]: ", a6[0][3])
# a6[0][3] = "cba"
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Tuple a1 =  ()
```

```
Tuple a2 =  (1, 2, 3, 'abc')
Tuple a3 =  (1, 2, 3, 'abc')
Tuple a4 from list l =  (1, 2, 3, 'abc')
Tuple a5 from string =  ('H', 'e', 'l', 'l', 'o',
',', ' ', 'W', 'o', 'r', 'l', 'd', '!')
Tuple a6 formed by a2 and a3 =  ((1, 2, 3, 'abc'),
(1, 2, 3, 'abc'))
Tuple a7 by combining a2 and a3 =  (1, 2, 3, 'abc',
1, 2, 3, 'abc')
a6[0]:  (1, 2, 3, 'abc')
a6[0][3]:  abc
```

Кортежи

В языке программирования Python кортеж (Tuple) представляет собой неизменяемый список. Для него доступны все операции, что и для списков, за исключением влияющих на значения списка, например, добавление, изменение, удаление элементов и т.д. Кортежи записываются в круглых скобках и могут быть объявлены как:

```
a1 = tuple()
a2 = 1, 2, 3, "abc"
a3 = (1, 2, 3, "abc")
```

Если кортеж состоит из одного элемента, при объявлении кортежа без использования функции **tuple()** необходимо ставить запятую после значения элемента, например: **a=(5,)**. Функция **tuple()** также позволяет создать кортеж из существующего списка или строки.

Для объединения кортежей используется операция сложения: **a = b + c**. Для вложения кортежей в общий кортеж используется следующая форма записи: **a = b, c**. Доступ к элементам кортежа осуществляется так же, как и доступ к элементам списка.

2. Модифицируйте код программы **lab_03_01.py**. Раскомментируйте строку '**# a6[0][3] = "cba"**'. Объясните поведение программы.

3. Дополните код программы **lab_03_01.py**. Создайте кортеж **k1**, содержащий значения дня, месяца и года Вашего рождения, введенные с клавиатуры, и кортеж **k2** со значениями Ваших фамилии, имени и отчества. Объедините кортежи **k1** и **k2**, записав результат в **k3**.

Осуществите вывод значения переменной **k3** на экран. Ознакомьтесь с результатом.

4. Дополните код программы **lab_03_01.py**. Создайте кортеж **k4**, в который будут вложены кортежи **k1** и **k2**. Осуществите вывод значения переменной **k4**, а также второго элемента второго вложенного кортежа на экран. Ознакомьтесь с результатом.

5. Создайте файл **lab_03_02.py**:

```
'''
    Множества
'''
# создание множества
b1 = set()
print("Set b1 = ", b1)
b2 = {"bear", "fox", "squirrel", "woodpecker",
"woodpecker", "wolf", "hedgehog"}
print("Set b2 = ", b2)
# создание множества из строки
b3 = set("abcdabcdefg")
print("Set b3 from string: ", set(b3))
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Set b1 =  set()
Set b2 =  {'fox', 'squirrel', 'bear', 'hedgehog',
'woodpecker', 'wolf'}
Set b3 from string:  {'g', 'a', 'b', 'c', 'd', 'f',
'e'}
```

Множества

Множество (Set) – это контейнер уникальных значений. Значения множества указываются в фигурных скобках. Множество создается с использованием функции **set()** или с указанием набора значений в фигурных скобках. Функция **set()** позволяет создать пустое множество,

а также множество из строки, списка или другой итерируемой структуры данных. Если в наборе значений присутствуют повторяющиеся, все повторы будут удалены. Пустое множество не может быть создано с указанием пустых фигурных скобок. Для множеств не предусмотрено обращение к отдельному элементу. Например, для проверки принадлежности элемента **element** множеству **set1** и проходу по множеству, можно использовать:

```
if element in set1:
    <операторы>
```

и

```
for element in set1:
    <операторы>
```

6. Дополните код программы **lab_03_02.py**. Создайте строковую переменную **s** со значением *"Electricity is the set of physical phenomena associated with the presence of electric charge. Lightning is one of the most dramatic effects of electricity"*. Создайте множество **set1** из строки **s**. Осуществите вывод множества **set1** на экран.

7. Дополните код программы **lab_03_02.py**. Для множества **set1** осуществите проход по всем его элементам с выводом на экран гласных букв. Ознакомьтесь с результатом.

8. Создайте файл **lab_03_03.py**:

```
'''
    Операции над множествами
'''
print("Check 'bear' in b2 = ", "bear" in b2)
b4 = set("123456135")
b5 = set("12367")
print("Set b4: {0}, \nSet b5: {1}".format(b4,b5))
print("b4 - b5: ", b4 - b5) # присутствие в первом
множестве, но не во втором
print("b4 difference b5 (b4-b5): ",
b4.difference(b5))
print("b4 | b5: ", b4 | b5) # присутствие хотя бы в
одном множестве
print("b4 union b5 (b4 | b5): ", b4.union(b5))
print("b4 & b5: ", b4 & b5) # присутствие в обоих
множествах
```

```
print("b4 intersection b5 (b4&b5): ",
b4.intersection(b5))
print("b4 ^ b5: ", b4 ^ b5) # присутствие только в
одном из множеств
# проверка на непересечение множеств
print("b4 and b5 are disjoint: ", b4.isdisjoint(b5))

b4.update(b5) # добавить элементы другого множества
print("add b5 to b4: ", b4)
b4.add("abc") # добавить элемент
print("add 'abc' to b4: ", b4)
b4.remove("5") # удалить элемент
print("remove element '5' from b4: ", b4)
b4.clear() # очистить множество
print("clear b4: ", b4)
print("\n ")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Check 'bear' in b2 =  True
Set b4: {'4', '6', '5', '2', '1', '3'},
Set b5: {'3', '2', '7', '6', '1'}
b4 - b5:  {'4', '5'}
b4 difference b5 (b4-b5):  {'4', '5'}
b4 | b5:  {'4', '7', '6', '5', '2', '1', '3'}
b4 union b5 (b4 | b5):  {'4', '7', '6', '5', '2',
'1', '3'}
b4 & b5:  {'3', '1', '6', '2'}
b4 intersection b5 (b4&b5):  {'3', '1', '6', '2'}
b4 ^ b5:  {'4', '7', '5'}
b4 and b5 are disjoint:  False
add b5 to b4:  {'4', '7', '6', '5', '2', '1', '3'}
add 'abc' to b4:  {'4', '7', '6', '5', 'abc', '2',
'1', '3'}
remove element '5' from b4:  {'4', '7', '6', 'abc',
'2', '1', '3'}
clear b4:  set()
```

Операции над множествами

В языке программирования Python существуют следующие операции, выполняемые над множествами:

- Проверка наличия элемента **element** в множестве **set1: element in set1**
- Разность множеств (включаются элементы, присутствующие в первом множестве, но не во втором):
  - **set3 = set1 - set2**
  - **set3 = set1.difference(set2)**
- Объединение множеств (присутствие элемента хотя бы в одном множестве):
  - **set3 = set1 | set2**
  - **set3 = set1.union(set2)**
- Пересечение множеств (присутствие элемента в обоих множествах):
  - **set3 = set1 & set2**
  - **set3 = set1.intersection(set2)**
- Симметричная разность (присутствие элементов только в одном из множеств):
  - **set3 = set1 ^ set2**
  - **set3 = set1.symmetric_difference(set2)**
- Проверка на непересечение множеств: **set1.isdisjoint(set2)**

Изменение значений множеств можно проводить с помощью следующих операций:

- Обновление множества добавлением элементов другого множества: **set1.update(set2)**
- Добавление элемента **element** в множество **set1**: **set1.add(element)**
- Удаление элемента **element** из множества **set1**: **set1.remove(element)**
- Очистка множества **set1**: **set1.clear()**

Также существуют неизменяемые множества – **frozenset**, для которых недоступны операции, связанные с изменением значений. Неизменяемое множество может быть также получено из изменяемого множества или любой другой итерируемой структуры данных путем использования функции **frozenset()**, аргументом которой является структура данных:

```
fset = frozenset({"1","2","3"})
```

Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:

https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset

9. Дополните код программы **lab_03_03.py**. Создайте два множества **set1** и **set2** из строк *"qetuwrt"* и *"asfrewgq"* соответственно. Поочередно выполните операции разности, объединения, пересечения и симметричной разности, выводя значения на экран. Добавьте в множество **set1** элементы множества **set2** с использованием функции **update()**, в множество **set2** элементы "t" и "u" с использованием функции **add()**. Повторно выполните операции разности, объединения, пересечения и симметричной разности, выводя значения на экран. Сравните полученные результаты.

10. Дополните код программы **lab_03_03.py**. Создайте неизменяемое множество **set3** из множества **set1**. Удалите из множества **set3** элемент "q". Запустите программу. Ознакомьтесь с результатом и объясните, почему так произошло.

11. Создайте файл **lab_03_04.py**:

```
'''
    Словари
'''
d1 = {
    "day": 18,
    "month": 6,
    "year": 1983
}
d2 = dict(bananas=3,apples=5,oranges=2,bag="basket")
d3 = dict([("street","Kronverksky pr."), ("house",
49)])
d4 = dict.fromkeys(["1","2"], 3)
print("Dict d1 = ", d1)
print("Dict d2 by dict()= ", d2)
print("Dict d3 by dict([])= ", d3)
print("Dict d4 by fromkeys = ", d4)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Dict d1 =  {'month': 6, 'year': 1983, 'day': 18}
Dict d2 by dict()=  {'apples': 5, 'bag': 'basket',
'bananas': 3, 'oranges': 2}
Dict d3 by dict([])=  {'house': 49, 'street':
'Kronverksky pr.'}
Dict d4 by fromkeys =  {'2': 3, '1': 3}
```

Словари

Словари в языке программирования Python представляют собой неупорядоченные коллекции объектов, доступ к которым осуществляется по ключу. Альтернативные названия – ассоциативные массивы, хэш-таблицы.

Существует три способа создания словаря:

- Указание пар "ключ-значение" в фигурных скобках: **`d = {"a": 1, "b": 2}`**
- С помощью функции **`dict()`**:
  - **`d = dict(a = 1, b = 2)`**
  - **`d = dict([("a",1), ("b",2)])`**
- Заполнение одинаковыми значениями по ключам: **`d = dict.fromkeys(["a","b"], 1)`**

12. Дополните код программы **`lab_03_04.py`**. Создайте словарь **`startDict`** с ключами **`ready`**, **`set`**, **`go`** и значениями **`3`**, **`2`** и **`1`** соответственно тремя разными способами, добавляя индекс к имени переменной. Выведите получившиеся словари на экран.

13. Дополните код программы **`lab_03_04.py`**. Создайте словарь **`dict1`** с ключами **`key1`** и **`key2`**, заполнив их одинаковым значением, введенным с клавиатуры. Осуществите вывод словаря **`dict1`** на экран.

14. Создайте файл **`lab_03_05.py`**:

```
'''
    Операции со словарями
'''
d5 = d2.copy() # создание копии словаря
print("Dict d5 copying d2 = ", d5)
```

```
# получение значения по ключу
print("Get dict value by key d5['bag']: ", d5["bag"])
print("Get dict value by key d5.get('bag'): ",
d5.get('bag'))
print("Get dict keys d5.keys(): ", d5.keys()) #
список ключей
print("Get dict values d5.values(): ", d5.values()) #
список значений
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Dict d5 copying d2 =  {'bananas': 3, 'bag': 'basket',
'apples': 5, 'oranges': 2}
Get dict value by key d5['bag']:  basket
Get dict value by key d5.get('bag'):  basket
Get dict keys d5.keys():    dict_keys(['bananas',
'bag', 'apples', 'oranges'])
Get dict values d5.values():    dict_values([3,
'basket', 5, 2])
```

Операции со словарями

В языке программирования Python существуют следующие операции, выполняемые со словарями:

- Копирование словаря **d1**: **d2 = d1.copy()**
- Получение значения словаря **d1** по ключу **'key'**:
  - **a = d1['key']**
  - **a = d1.get('key')**
- Удаление пары "ключ-значение" по ключу: **del d1['key']**
- Получить набор ключей словаря **d1**: **d1.keys()**
- Получить набор ключей словаря **d1**: **d1.values()**

Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:

https://docs.python.org/3/library/stdtypes.html#dict

15. Дополните код программы **lab_03_05.py**. Создайте словарь **myInfo**, содержащий информацию о Ваших фамилии, имени, отчестве, дне, месяце, годе рождения и университете в полях **surname**, **name**, **middlename**, **day**, **month**, **year**, **university** соответственно. Получите и поочередно выведите на экран списки ключей и значений словаря **myInfo**. Ознакомьтесь с результатом.

16. Создайте файл **lab_03_06.py**:

```
'''
    Функции
'''
def dictUpdate(a):
    a.update([("x",5)])
    print("dict in function: ",a)
    return
def dictNoUpdate(a):
    a = a.copy()
    a.update([("y",3)])
    print("dict in function: ",a)
    return
def returnFunc(a):
    def f1(a):
        print("returned f1(a): ",a)
    return f1
d= {"v":7}
dictUpdate(d)
print("dict out of function: ",d)
dictNoUpdate(d)
print("dict out of function: ",d)
f = returnFunc(d)
print("f: ", f)
f(2)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
dict in function:  {'v': 7, 'x': 5}
dict out of function:  {'v': 7, 'x': 5}
dict in function:  {'v': 7, 'y': 3, 'x': 5}
dict out of function:  {'v': 7, 'x': 5}
```

```
f:  <function returnFunc.<locals>.f1 at
0x7f6d7a778048>
returned f1(a):  2
```

Функции

Функции представляют собой объекты, принимающие на вход некоторые аргументы, и возвращающие определенные значения. В языке программирования Python объявление функции происходит с использованием ключевого слова **def**:

```
def <имя_функции>(<аргумент_1>, <аргумент_2>):
    <операторы>
    return <значение>
```

Аргументы функции (**<аргумент_1>, <аргумент_2>**) указываются в скобках после имени функции (**<имя_функции>**). Такие функции называются именными. Возвращаемое значение (**<значение>**) указывается после ключевого слова **return**. Аргументы в функции передаются по ссылкам, вследствие чего, во избежание перезаписи значений, в блоке операторов (**<операторы>**) необходимо аккуратно обрабатывать входные аргументы. В качестве возвращаемого значения функции может быть также указана внутренняя функция, например:

```
def returnFunc(a):
    def f1(a):
        print("returned f1(a): ",a)
    return f1
```

Функция может не иметь возвращаемого значения. В этом случае, после слова **return** значение не указывается. При попытке вывода возвращаемой функции в терминал, будет выведена ссылка на объект функции.

17. Дополните код программы **lab_03_06.py**. Создайте функцию **returnMod**, возвращающую функцию, которая осуществляет расчет остатка от деления переданного аргумента на 15 и вывод этого значения на экран. Вызовите функцию **returnMod** и осуществите запись возвращенного значения в переменную **mod15**. С использованием переменной **mod15** добейтесь выполнения расчета остатка от деления и вывода значения на экран. Ознакомьтесь с результатом.

18. Создайте файл **lab_03_07.py**:

```
'''
    Аргументы функции
'''
def sum(x, y, z=1):
    return x + y + z

print("sum(1,2,3): ",sum(1,2,3))
print("sum(1,2): ",sum(1,2))
print("sum(x=1,y=3): ",sum(x=1,y=3))

# переменное количество аргументов
def printArgs(*args):
    print("args of printArgs(): ",args)
    return
# переменное количество аргументов и аргументов-ключевых слов
def printArgsnKwargs(m,*args,**kwargs):
    print("main argument of printArgsnKwargs(): ",m)
    print("args of printArgsnKwargs(): ",args)
    print("args of printArgsnKwargs(): ",kwargs)
    return

printArgs("Hello World!", 1, 3, 5)
printArgsnKwargs("Earth", 7.125, radius=6371, pos=3)
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
sum(1,2,3):  6
sum(1,2):  4
sum(x=1,y=3):  5
args of printArgs():  ('Hello World!', 1, 3, 5)
main argument of printArgsnKwargs():  Earth
args of printArgsnKwargs():  (7.125,)
args of printArgsnKwargs():  {'pos': 3, 'radius': 6371}
```

Аргументы функции

Количество аргументов функции может быть переменным. Для указания базового значения аргумента, который в некоторых случаях может отсутствовать, его значение указывается после имени аргумента и знака равно, например:

```
def sum(x, y, z=0):
    return x + y + z
```

В случае, если количество аргументов функции неизвестно, можно использовать следующую запись, позволяющую получить список аргументов позже в теле функции:

```
def func(*args):
    <операторы>
    return <значение>
```

Часть аргументов может быть представлена парой "ключ-значение", аналогично записи пропускаемого аргумента, описанной ранее. Число таких аргументов, называемых аргументами-ключевыми словами (keyword arguments, или **kwargs**), может быть неизвестном, аналогично обычным аргументам. В этом случае, необходимо использовать следующую форму записи:

```
def printArgsnKwargs(*args,**kwargs):
    <операторы>
    return <значение>
```

19. Дополните код программы **lab_03_07.py**. Создайте функцию **checkArgs()** с неизвестным количеством аргументов и аргументов-ключевых слов. Функция **checkArgs** проверяет количество переданных аргументов и аргументов-ключевых слов и:

- Если количество аргументов меньше либо равно трем и количество аргументов-ключевых слов строго меньше трех – выводит их на экран.
- Иначе – выводит предупреждение о превышении количества передаваемых аргументов.

20. Создайте файл **lab_03_08.py**:

```
'''
    Анонимные функции, lambda-выражения
'''
lfunc = lambda x, y, z = 1: x + y + z
print("lfunc(1,2,3): ",lfunc(1,2,3))
print("lfunc(1,2): ",lfunc(1,2))
print("lfunc(x=1,y=3): ",lfunc(x=1,y=3))
print("lambda result: ", \
    (lambda                 a,b,sep=",                    ":
sep.join((a,b)))("Hello","World!"))
print("\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
lfunc(1,2,3):  6
lfunc(1,2):  4
lfunc(x=1,y=3):  5
lambda result:  Hello, World!
```

Анонимные функции, lambda-выражения

Анонимные функции могут содержать лишь одно выражение, однако выполняются они быстрее, чем именные функции. Анонимные функции представляют собой так называемые **lambda**-выражения, определяемые ключевым словом **lambda**, например:

```
f = lambda <аргумент_1>, <аргумент_2>,…: <оператор>
```

Анонимные функции так же, как и именные могут содержать переменное количество аргументов. Для анонимных функций результат выполнения оператора (**<оператор>**) является возвращаемым значением. Этот тип функций также может быть использован и без записи в переменную с помощью вызова при объявлении. В этом случае, сначала в скобках пишется **lambda**-выражение, после чего в скобках перечисляются аргументы функции:

```
(lambda a,b,sep=", ":
sep.join((a,b)))("Hello","World!")
```

21. Дополните код программы **lab_03_08.py**. Создайте **lambda**-выражение, присвоив его переменной **lam**, осуществляющее проверку равенства остатка от деления введенного с клавиатуры числа, передаваемого в качестве аргумента, на 3. При равенстве остатка от деления на три нулю функция выводит значение на экран.

22. Создайте файлы со следующим содержимым:

Имя файла	*Содержимое*
file1.txt	Hello, World!
file2.txt	Yesterday all my troubles seemed so far away. Now it looks as though they're here to stay. Oh, I believe in yesterday.
file3.txt	

23. Создайте файл **lab_03_09.py**:

```
'''
    Работа с файлами
'''
file1 = open("file1.txt", "r") # открыть на чтение
st = file1.read(1) # считать 1 символ
st += file1.read() # считать до конца файла
print("File1: ", st)
file1.close()

# считать построчно
file2 = open("file2.txt", "r")
print("File2: ")
i = 0
for line in file2: # итерация по строкам файла
    print("Line {}:
{}".format(i,line.replace("\n","")))
    i += 1
file2.close()

# запись в файл
file3 = open("file3.txt","w")
for s in "Strings can be easily written to file":
    file3.write(s)
    if s == " ":
        file3.write("\n")
print("File 3 was written successfully")
```

```
file3.close()
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
File1:  Hello, World!
File2:
Line 0: Yesterday all my troubles seemed so far away.
Line 1: Now it looks as though they're here to stay.
Line 2: Oh, I believe in yesterday.
File 3 was written successfully
```

Работа с файлами

Для открытия файла используется функция **`open()`**, аргументами которой является полное имя файла, а также спецификатор, указывающий режим открытия файла:

```
f = open(<имя_файла>, <спецификатор>)
```

Спецификатор	*Описание*
`'r'`	Открытие файла на чтение
`'w'`	Открытие файла на запись с перезаписью размещенных ранее данных. В случае отсутствия файла – файл создается автоматически
`'x'`	Открытие файла на запись. В случае отсутствия файла – выдается ошибка
`'a'`	Открытие файла на дополнение
`'b'`	Открытие файла в двоичном виде
`'t'`	Открытие файла в виде текста (по умолчанию)
`'+'`	Открытие файла на чтение и запись

Режимы могут комбинироваться между собой, например при открытии файла на чтение в двоичном режиме используется запись **`'rb'`**.

Для чтения данных из файла используется функция **`read()`**, вызываемая от объекта, созданного при открытии файла, которая может принимать на вход количество символов, которое необходимо считать. При отсутствии аргумента файл считывается до конца:

```
s = f.read(4)
s = f.read()
```

Для записи значения **s** в файл используется функция **write()**, вызываемая от объекта, созданного при открытии файла:

```
f.write(s)
```

Считывание файла построчно может проходить в цикле:

```
for line in f:
    <операторы>
```

Закрытие файла после завершения работы с ним происходит с помощью функции **close()**:

```
f.close()
```

24. Создайте файл **lab_03_10.py**. Создайте словарь **textDict** для содержимого файла **text1.txt**, который приведен далее, по следующему принципу: ключи словаря – слова текста, значения словаря – частота появления слова в отрывке. Запишите словарь **textDict** в файл с именем **textDict.txt**.

Файл **text1.txt**:

```
Around the new position a circle, somewhat larger than in the former instance, was now described, and we again set to work with the spades. I was dreadfully weary, but, scarcely understanding what had occasioned the change in my thoughts, I felt no longer any great aversion from the labor imposed. I had become most unaccountably interested --nay, even excited.
```

25. Создайте файл **lab_03_11.py**. Создайте функции **encodeHuffman(fileIn, fileOut)** и **decodeHuffman(fileIn, fileOut)**, осуществляющие кодирование и декодирование текста с использованием метода Хаффмана соответственно. **fileIn** – полное имя файла с исходным текстом, **fileOut** – полное имя файла, куда необходимо записать результат. Функции возвращают **True**, если ошибок не возникло, и **False** в ином случае. Осуществите проверку работы функций на произвольном тексте.

26. Дополните код программы **lab_03_11.py**. Создайте функции **encodeLZ(fileIn, fileOut)** и **decodeLZ(fileIn, fileOut)**, осуществляющие кодирование и декодирование текста с использованием

метода Лемпеля-Зива соответственно. **`fileIn`** – полное имя файла с исходным текстом, **`fileOut`** – полное имя файла, куда необходимо записать результат. Функции возвращают **`True`**, если ошибок не возникло, и **`False`** в ином случае. Осуществите проверку работы функций на произвольном тексте.

27. Подберите два текста, для одного из которых лучше подходит метод Хаффмана, а для второго – метод Лемпеля-Зива. Рассчитайте коэффициенты сжатия информации. Выведите тексты и коэффициенты их сжатия разными методами на экран.

Лабораторная работа № 4: Объектно-ориентированное программирование

1. Создайте файл **lab_04_01.py**:

```
import time

class Ticket:

    def __init__(self, date, name,deadline):
        self.createDate = date
        self.owner = name
        self.deadline = deadline

    def __del__(self):
        print("Delete                                ticket:
",time.asctime(self.createDate))

    def display(self):
        print("Ticket:")
        print("                                createDate:
",time.asctime(self.createDate))
        print(" owner: ",self.owner)
        print("                                    deadline:
",time.asctime(self.deadline))

# создание объект класса
ticket1 = Ticket(time.localtime(),"Ivan Ivanov", \
                    time.strptime("17.12.2017",
"%d.%m.%Y"))
# вызов метода
ticket1.display()
# получение значения атрибута
print("Owner: ", ticket1.owner)
print("Owner(getattr): ", getattr(ticket1,"owner"))
# проверка наличия атрибута
print("hasattr: ", hasattr(ticket1,"owner"))
setattr(ticket1,"owner","Alexei Petrov") # установка
значения атрибута
print("Owner(setattr): ", ticket1.owner)
#   delattr(ticket1,"owner")   #   удаление   значения
атрибута
# print("delattr: ", ticket1.owner)
```

```
# del ticket1 # удаление объекта
# print(ticket1)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Ticket:
 createDate:  Thu Mar  9 14:53:55 2017
 owner:  Ivan Ivanov
 deadline:  Sun Dec 17 00:00:00 2017
Owner:  Ivan Ivanov
Owner(getattr):  Ivan Ivanov
hasattr:  True
Owner(setattr):  Alexei Petrov
Delete ticket:  Thu Mar  9 14:53:55 2017
```

Классы и объекты классов

Python является объектно-ориентированным языком программирования. В объектно-ориентированном программировании основными понятиями являются «класс» и «объект» класса. Класс представляет собой категорию, к которой можно отнести множество объектов, обладающих характерными признаками и выполняющих одинаковые действия. Объявление класса происходит с использованием ключевого слова **`class`** и указанием имени класса (имена классов начинаются с большой буквы). Класс может содержать атрибуты и методы (функции, описывающие действия его объектов). В языке Python объявление класса происходит следующим образом:

```
class Ticket:

    def __init__(self, date, name, deadline):
        pass

    def __del__(self):
        pass

    def display(self):
        pass
```

В качестве первого аргумента методов выступает ссылка на текущий объект класса **self** (аналогичный **this** в большинстве объектно-ориентированных языках программирования). При вызове методов в передаваемых аргументах объект класса не указывается, например:

```
t = Ticket(date, name, deadline)
```

где **t** – объект класса **Ticket**.

Существует два базовых метода, предусмотренных для всех создаваемых классов:

- **__init__** – конструктор класса.
- **__del__** – деструктор класса

Атрибуты класса задаются внутри его методов. Атрибуты, заданные вне методов, являются общими для всех объектов класса и могут изменять свои значения в случае их редактирования.

Вызов методов класса осуществляется с указанием имени метода через точку после имени объекта, от которого вызывается метод. Также существуют методы, предоставляющие возможности работы с атрибутами классов:

- **hasattr(obj,attr)** – проверка наличия атрибута **attr** у объекта **obj**
- **setattr(obj,attr,value)** – установка значения **value** атрибута **attr** у объекта **obj**
- **getattr(obj,attr)** – получение значения атрибута **attr** у объекта **obj**
- **delattr(obj,attr)** – удаление атрибута **attr** у объекта **obj**

Методы класса объявляются как функции внутри класса.

Работа с подключаемой библиотекой

Для подключения внешних библиотек в языке программирования Python используется ключевое слово **import**. Подключение самостоятельной библиотеки или модуля происходит следующим образом:

```
import name1, name2, …
```

где **name1, name2,** … – имена подключаемых библиотек/модулей.

Для подключения библиотеки из какого-либо пакета используется следующий формат записи:

from package import name1, name2, …

где **package** – имя пакета, **name1, name2,** … – имена подключаемых библиотек/модулей.

Для сокращения названий подключаемых библиотек можно указывать собственные названия после ключевого слова **as**:

```
from package import name1 as N
```

где **N** – собственное название подключаемой библиотеки.

Для работы с датами и временем используется библиотека **time**. Основными функциями данной библиотеки являются:

- **time()** – получение времени в системе Unix-время
- **localtime()** – получение текущего времени машины в виде кортежа
- **asctime(timeT)** – автоматическое приведение времени, представленного кортежем (**timeT**), к формату:
  Thu Mar 9 14:53:55 2017
- **strptime(str, format)** – создание кортежа с данными времени по строке **str**, представленной в формате **format**
- **strftime(timeT, format)** – представление кортежа с данными времени **timeT** в виде строки в формате **format**

Со списком форматов для функций **strptime()** и **strftime()**, а также списком остальных функций библиотеки можно ознакомиться по ссылке:
https://docs.python.org/3/library/time.html

2. Модифицируйте код программы **lab_04_01.py**. Раскомментируйте строки:
'**# delattr(ticket1,"owner") # удаление значения атрибута**
 # print("delattr: ", ticket1.owner)'.
Объясните поведение программы. Исправьте программу с помощью стандартных функций так, чтобы ошибка не возникала.

3. Модифицируйте код программы **lab_04_01.py**. Раскомментируйте строки:
'**# del ticket1 # удаление объекта**
 # print(ticket1)'.
Объясните поведение программы.

4. Дополните код программы **lab_04_01.py**. Получите текущее время сервера (или компьютера) и выведите его на экран в аналогичном формате: **9 Mar 2017 14:53:55**.

5. Дополните код программы **lab_04_01.py**. Создайте объект типа **time** по следующей строке с использованием соответствующей функции библиотеки **time**: **17.07.2017 10:53:00**.

6. Создайте файл **lab_04_02.py**:

```
class Worker:
    'doc class Worker'
    count = 0

    def __init__(self,name,surname):
        self.name = name
        self.surname = surname
        Worker.count += 1

    def display(self):
        print("Worker:")
        print("{} {}".format(self.name,self.surname))

w1 = Worker("Ivan", "Ivanov")
print("w1.count: ", w1.count)
w2 = Worker("Alexei", "Petrov")
print("w2.count: ", w2.count)
print("w1.count: ", w1.count)
print("Worker.count: {0} \n".format(Worker.count))
print("Worker.__name__: ", Worker.__name__)
print("Worker.__dict__: ", Worker.__dict__)
print("Worker.__doc__: ", Worker.__doc__)
print("Worker.__bases__: ", Worker.__bases__)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
w1.count:  1
w2.count:  2
w1.count:  2
Worker.count: 2

Worker.__name__:  Worker
Worker.__dict__:   {'count': 2, 'display': <function
Worker.display   at   0x7fbc0afb5a60>,   '__module__':
'builtins',   '__dict__':   <attribute   '__dict__'   of
'Worker'      objects>,      '__init__':      <function
Worker.__init__   at   0x7fbc0afb5ae8>,   '__weakref__':
<attribute   '__weakref__'   of   'Worker'   objects>,
'__doc__': 'doc class Worker'}
Worker.__doc__:  doc class Worker
Worker.__bases__:  (<class 'object'>,)
```

Общие атрибуты класса

Общие атрибуты класса доступны всем объектам данного класса. Значения, установленные для общих атрибутов класса, могут быть впоследствии отредактированы, что повлечет за собой их изменения во всех объектах класса. Общие атрибуты задаются вне методов класса:

```
class Worker:
    count = 0
```

Доступ к общим атрибутам класса может быть получен как от объектов класса, так и из самого класса, например:

```
w1.count
Worker.count
```

Описание класса (документация) добавляется в виде строки сразу после начального объявления класса:

```
class Worker:
    'doc class Worker'
    ...
```

Для получения информации о классе используются следующие базовые атрибуты:

- **__name__** – получение имени класса в виде строки
- **__dict__** – получение сведений о всех доступных методах и общих атрибутах класса
- **__doc__** – получение строки с описанием (документацией) класса
- **__bases__** – классы, от которых был унаследован данный класс. По умолчанию все классы наследуются от общего класса, представляющего объекты (**object**)

7. Дополните код программы **lab_04_02.py**. Создайте класс **Animal**, обозначив для него в конструкторе атрибуты **name**, **age** и общий атрибут **id**, который будет увеличиваться на единицу при создании каждого нового объекта класса. Определите также метод **display()**, осуществляющий вывод на экран информации по объекту класса в следующем формате, подставив нужные значения:

Animal *id*:
Name: *name*
Age: *age*

Создайте три объекта класса **Animal**, заполнив их произвольными значениями **name** и **age**. Осуществите вывод информации об объектах на экран с помощью функции **display()**.

8. Дополните код программы **lab_04_02.py**. По выведенной на прошлом шаге информации об объектах объясните поведение атрибута **id**.

9. Модифицируйте код программы **lab_04_02.py**. Измените название атрибута **id** на **count**. Модифицируйте программу так, чтобы, определив атрибут **id**, он являлся уникальным для каждого объекта, изменяясь на единицу с увеличением количества объектов класса.

10. Создайте файл **lab_04_03.py**:

```
class Geometric:
    def calculateArea(self):
        print("Calculating area")

class Square(Geometric):
    def __init__(self,a):
        self.side = a
```

```
    def _perimeter(self):
        print("Perimeter of Square {}:
{}\n".format(self.side, self.side*4))
    def calculateArea(self):
        print("Area of Square {}:
{}\n".format(self.side, pow(self.side,2)))

geom = Geometric()
geom.calculateArea()
sq = Square(5)
sq.calculateArea()
sq._perimeter()

print("Check subclass: ",
issubclass(Square,Geometric))
print("Check instance sq->Square: ",
isinstance(sq,Square))
print("Check instance sq->Geometric: ",
isinstance(sq,Geometric))
print("Check instance sq->dict: ",
isinstance(sq,dict))

print("Geometric.__bases__: ", Geometric.__bases__)
print("Square.__bases__: ", Square.__bases__)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Calculating area
Area of Square 5: 25

Perimeter of Square 5: 20

Check subclass:  True
Check instance sq->Square:  True
Check instance sq->Geometric:  True
Check instance sq->dict:  False
Geometric.__bases__:  (<class 'object'>,)
Square.__bases__:  (<class 'Geometric'>,)
```

Наследование

Наследование между классами в языке программирования Python реализуется следующим образом:

```
class Ancestor:
    def func(self):
        pass

class Descendant(Ancestor):
    def func(self):
        pass
```

Имя родительского класса (**Ancestor**) указывается в скобках при объявлении дочернего (**Descendant(Ancestor)**).

Для показания приватности метода в начале его названия добавляется символ подчеркивания "**_**". Однако, следует отметить, что в Python не предусмотрено полностью приватных атрибутов и методов: доступ к ним в любом случае может быть получен из основного кода программы. Для того, чтобы метод/атрибут казался более закрытым, допускается добавление двух символов подчеркивания перед названием метода "**__**", но в этом случае доступ к методу/атрибуту может быть получен с использованием имени класса "**_Class__method**".

Переопределение методов аналогично многим объектно-ориентированным языкам программирования осуществляется путем указания функции родительского класса с тем же именем и аргументами внутри дочернего класса.

Для наследования атрибутов родительского класса, указанных в конструкторе родительского класса, необходимо внутри конструктора дочернего класса первоначально вызвать конструктор родительского класса, например:

```
class Ancestor:
    def __init__(self,param):
        self.param = param
    def func(self):
        pass

class Descendant(Ancestor):
    def __init__(self, param, number):
```

```
        Ancestor.__init__(self, param)
        self.number = number
    def func(self):
        pass
```

Здесь:
Ancestor.__init__(self, param) – строка вызова конструктора родительского класса.
Также может быть использована следующая запись:
super(Descendant, self).__init__(param)

Для проверки наследования класса (**Descendant**) от известного родительского класса (**Ancestor**) используется функция **issubclass(Descendant, Ancestor)**. Для проверки принадлежности объекта (**obj**) какому-либо классу (**Class**) используется функция **isinstance(obj,Class)**. Также для проверки наследования классов может быть использован атрибут **__bases__**.

11. Дополните код программы **lab_04_03.py**. Создайте класс **Circle**, унаследованный от класса **Geometric**, для которого определите атрибут **radius**. Сделайте атрибут **radius** приватным (с использованием двойного подчеркивания). Переопределите унаследованный метод **calculateArea()**, рассчитав площадь окружности (значение числа Пи доступно в библиотеке **math**).

12. Создайте файл **lab_04_04.py**. Создайте класс **Encoder**, определив для него методы **encode()** и **decode()**, аргументом которых является строка, а выходными данными – закодированная и декодированная строка соответственно. Создайте классы **HuffmanEncoder** и **LZEncoder**, унаследованные от класса **Encoder**, определив для них атрибут **compressionCoef** в конструкторах классов. Для созданных классов **HuffmanEncoder** и **LZEncoder** переопределите методы **encode()** и **decode()**, реализующие кодирование и декодирование поданных в качестве аргументов строк, используя методы Хаффмана и Лемпеля-Зива соответственно. Определите приватный метод **setCompressionCoef()** (с использованием двойного подчеркивания), осуществляющий расчет коэффициентов сжатия для методов Хаффмана и Лемпеля-Зива в соответствующих классах. Метод **setCompressionCoef()** должен вызываться при работе внутри класса в методе **encode()**. Определите общедоступный метод

getCompressionCoef(), позволяющий получить значение коэффициента сжатия. Осуществите проверку методов классов на следующей строке:

```
Python is a widely used high-level programming language for
general-purpose programming, created by Guido van Rossum and
first released in 1991.
```

13. Создайте файл **lab_04_05.py**. Создайте класс **Person**, определив для него в конструкторе атрибуты **firstname**, **lastname** и **age**, а также метод **display()**, выводящий информацию об объекте класса на экран. Создайте класс **Student**, унаследованный от **Person**, определив для него дополнительные атрибуты **studentID**, являющийся уникальным номером для каждого объекта класса, и **recordBook**, содержащий информацию о количестве пятерок, четверок, троек и двоек студента в виде списка. В классе **Student** дополните метод **display()** выводом значений атрибутов **studentID** и **recordBook** на экран. Создайте три объекта класса **Student** для проверки работы методов и выведите информацию о них на экран.

14. Дополните код программы **lab_04_05.py**. Создайте класс **Professor**, унаследованный от **Person**, определив для него дополнительные атрибуты **professorID**, являющийся уникальным номером для каждого объекта класса, и **degree**, содержащий информацию о научной степени в виде строки. В классе **Professor** дополните метод **display()** выводом значений атрибутов **professorID** и **degree** на экран. Создайте три объекта класса **Professor** для проверки работы методов и выведите информацию о них на экран.

15. Создайте файл **lab_04_06.py**. Создайте класс **HammingEncoder**, конструктор которого принимает на вход количество информационных разрядов **dataBits**, на основе которых рассчитывает значение количества контрольных разрядов – **controlBits**. **dataBits** и **controlBits** являются атрибутами класса. Реализуйте метод **encode(str)**, служащий для кодирования строки двоичных символов **str** с использованием кода Хэмминга с установленными параметрами **dataBits** и **controlBits**. Реализуйте метод **decode(str)**, служащий для определения кода ошибки закодированной строки **str**.

16. Создайте файл **lab_04_07.py**. Создайте следующие классы и реализуйте в них указанные методы:

- Класс **Row**
  Атрибуты:
  - **id** – идентификатор строки.
  - **collection** – список значений переменных для текущего значения функции.
  - **value** – значение функции.

  Методы:
  - **__init__(collection, value)** – конструктор класса, принимающий на вход значения **collection** и **value**.
- Класс **Table**
  Атрибуты:
  - **rows** – список объектов класса **Row**.
  - **rowsNum** – количество строк таблицы.

  Методы:
  - **__init__(rowsNum)** – конструктор класса, принимающий на вход значение **rowsNum**.
  - **addRow(row)** – добавление строки (объект **row** класса **Row**) в список. Если в списке уже находится строка с таким же идентификатором, должна выдаваться ошибка.
  - **setRow(row)** – изменение строки (объект **row** класса **Row**). Если в списке нет строки с таким же идентификатором, должна выдаваться ошибка.
  - **getRow(rowId)** – получение строки с идентификатором **rowId**. Возвращается объект класса **Row**.
  - **display()** – вывод таблицы на экран в следующем формате:

```
id  x1  x2   f(x1,x2)
1   0   0  |    1
2   0   1  |    0
3   1   0  |    0
4   1   1  |    1
```

- Класс **LogicFunction**
  Атрибуты:
  - **variablesNum** – количество переменных функции
  - **table** – объект класса **Table**. Таблица истинности логической функции.

  Методы:
  - **__init__(variablesNum,table)** – конструктор класса, принимающий на вход значения **variablesNum** и **table**.

- **`getExpression()`** – вычисляет и возвращает минимальную формулу логической функции.
- **`getTable()`** – получение значения **`table`**.
- **`printTable()`** – вывод значения **`table`** на экран.

Лабораторная работа № 5: Веб-программирование с использованием фреймворка Django

1. Запустите онлайн среду разработки "Codeanywhere" с рабочим пространством для программирования на Python с использованием фреймворка Django версии 1.9 по ссылке:

`https://codeanywhere.com/`

Для входа в систему необходимо иметь учетную запись на одном из следующих сервисов: Google+, GitHub, Bitbucket, Facebook. После входа в аккаунт аутентифицируйте приложение, перейдя по ссылке в письме, присланном на электронную почту аккаунта. Из предложенного списка платформ в среде "Codeanywhere" (окно "Connection Wizard") необходимо выбрать Django (Ubuntu 14.04) и в поле "Name", расположенном в верхней части окна, ввести имя проекта. Нажатием кнопки "Create" Вы подтверждаете создание проекта с указанными параметрами. Будет создано два каталога:

- **`django`** – содержит файлы фреймворка Django;
- **`projectname`** – содержит файлы проекта.

Фреймворк Django

Django является свободно распространяемым фреймворком для написания веб-приложений на языке программирования Python. В качестве основы для проектирования веб-приложений используется шаблон проектирования MVC (Model-View-Controller), включающий в себя:

- Model (Модель) – служит для предоставления запрошенных пользователем данных, может изменять свое состояние в зависимости от полученных от контроллера команд.
- View (Представление) – служит для отображения данных модели пользователю, может изменять выдаваемые данные в зависимости от изменений модели.
- Controller (Контроллер) – служит для обработки действий пользователя, управляет изменениями, происходящими в модели.

При взаимодействии с интерфейсом веб-приложения пользователь отдает команды контроллеру (например, добавить запись в базу данных), контроллер обрабатывает команду и обновляет модель (запись добавлена в базу данных), после чего страница веб-приложения (представление) обновляется (запись отображается пользователю).

Основные файлы проекта

В список основных файлов проекта входят следующие:

- **manage.py** – устанавливает настройки для проекта из указанного файла настроек;
- **projectname/settings.py** – файл настроек проекта, хранящийся в каталоге **projectname**;
- **projectname/urls.py** – файл настроек URL (ссылок) для каждого представления, хранящийся в каталоге **projectname**;
- **projectname/wsgi.py** – устанавливает настройки для проекта при необходимости использования WSGI-технологии;
- **projectname/__init__.py** – файл, указывающий Python, что текущий каталог является пакетом Python.

Рассмотрим настройки проекта, размещаемые в файле **settings.py**. Изначально в настройках указываются:

- **BASE_DIR** – путь к основной папке проекта;
- **SECRET_KEY** – секретный ключ, служащий для обеспечения дополнительной безопасности данных;
- **DEBUG** – флаг, идентифицирующий запуск проекта в режиме тестирования;
- **ALLOWED_HOSTS** – позволяет указать варианты записи домена и субдоменов веб-приложения для обеспечения дополнительной безопасности;
- **INSTALLED_APPS** – список всех приложений проекта, включая базовые приложения Django (sessions, messages и т.д.). Для подключения собственных разработанных приложений необходимо указать путь к папкам приложений, используя точку в качестве разделителя;
- **MIDDLEWARE_CLASSES** – список подключенных приложений фреймворка **MIDDLEWARE**, позволяющих использовать базовые обработчики некоторых действий пользователя, например, связывать запрос пользователя с текущим идентификатором сессии;
- **ROOT_URLCONF** – содержит путь к основному файлу списка URL (**urls.py**);
- **TEMPLATES** – описывает настройки шаблонов серверной (backend) и клиентской (представления) частей проекта, путь к папкам, в которых хранятся шаблоны, рекомендуется делать относительным с записью в следующем виде: **os.path.join(BASE_DIR,**

'templates'), где **join** – стандартная функция объединения строк для двух путей: главного пути **os.path**, пути **BASE_DIR**, и пути к нужному каталогу **templates**, переданных в качестве аргумента. Пользовательские шаблоны настраиваются в параметре **DIRS**;

- **WSGI_APPLICATION** – путь к приложению WSGI;
- **DATABASES** – словарь, описывающий основные настройки используемой базы данных:
  - **ENGINE** – драйвер базы данных, указывается один из существующих в Django драйверов:
    - Для MySQL: **'django.db.backends.mysql'**
    - Для PostgreSQL: **'django.db.backends.postgresql'**
    - Для SQLite: **'django.db.backends.sqlite3'**
    - Для Oracle: **'django.db.backends.oracle'**
  - **NAME** – имя базы данных;
  - **USER** – имя пользователя для автоматического подключения к базе данных;
  - **PASSWORD** – пароль для автоматического подключения к базе данных;
  - **HOST** – адрес, на котором расположена база данных;
  - **PORT** – порт, на котором расположена база данных;
- **AUTH_PASSWORD_VALIDATORS** – словарь, приводящий описание валидаторов, отвечающих за проверку сложности паролей пользователей;
- **TIME_ZONE** – временная зона сервера
- **LANGUAGE_CODE** – язык приложений проекта, необходимо, чтобы флаг **USE_I18N** был установлен как **True**;
- **USE_I18N** – флаг, показывающий статус включения системы перевода Django (подстановки нужного языка);
- **USE_L10N** – флаг, показывающий статус включения системы локализации Django, если флаг установлен как **True**, то время, дата и т.п. будут отображаться в соответствии с установленной локализацией;
- **USE_TZ** – флаг использования параметра **TIME_ZONE**;
- **STATIC_URL** – путь к статическим файлам, например, CSS-стилям. Для указания множества папок собственных статических файлов необходимо использование параметра **STATICFILES_DIRS**.

Для добавления дополнительных страниц и адресов ссылок в проект необходимо в файл **urls.py** добавить соответствующую ссылку, например:

url(r'^admin/', admin.site.urls) – подключение всех ссылок указанных в файле **urls.py** каталога "**admin/site/**" (базовый каталог Django из пакета **django.contrib**).

Для указания конкретной функции, вызываемой при переходе по ссылке, используют следующий формат записи:

```
url(r'^auth/$', views.auth, name='auth'),
```

где:

- **r'^auth/$'** – ссылка для перехода
- **views.auth** – путь к функции **auth** в файле **views**, подключаемом к **urls.py**
- **name='auth'** – сокращенное для перехода по ссылкам напрямую из кода проекта

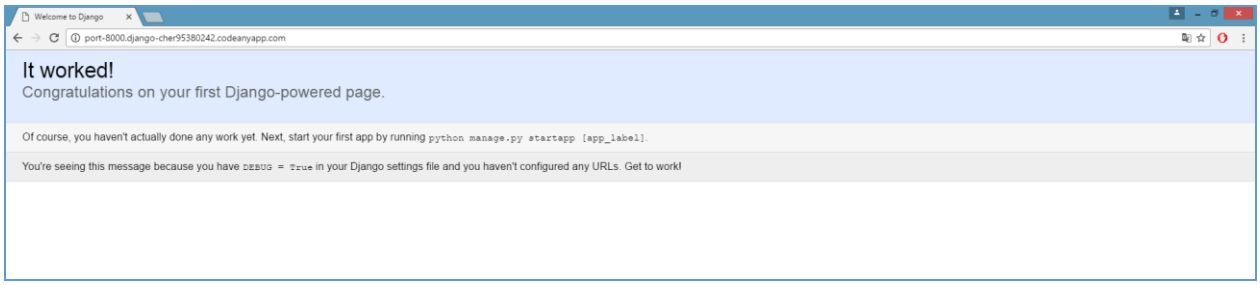
Для подключения обработчиков представлений для указания имен функций используется стандартная запись с указанием имени файла (**views**), содержащего нужные функции:

```
from projectname import views
```

Запуск проекта из консоли осуществляется с помощью команды

```
python manage.py runserver
```

2. Запустите созданный проект. При работе в среде Codeanywhere можно использовать функцию "Run project". Перейдите по ссылке проекта, указанной в конфигурации и справке к проекту, и убедитесь, что проект успешно запускается, при этом на странице должно отображаться следующее:



3. Создайте каталог **templates** в каталоге проекта. Добавьте в каталог файл **index.html**:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}
            Hello
        {% endblock %}
    </title>
</head>
<body>
    <div class="header">
        Hello World from render!
    </div>
</body>
</html>
```

Установите путь к каталогу в параметре **TEMPLATES.DIRS** в настройках проекта.

4. Создайте файл **views.py**:

```
from django.http import HttpResponse
from django.shortcuts import render

def index(request):
    return HttpResponse("Hello, world!")

def indexRender(request):
    return render(request, 'index.html', {})
```

5. Дополните файл **urls.py**, добавьте ссылки:

- пустая ссылка для главной страницы, которую обрабатывает функция **index**
- ссылка "hello", которую обрабатывает функция **indexRender**

6. Запустите проект. Перейдите по ссылкам на главную страницу и страницу hello. Удостоверьтесь, что при переходе по ссылкам выводится следующая информация:

- при переходе на главную страницу:
  Hello, world!
- при переходе на страницу hello:
  Hello World from render!

Работа с представлениями

Для обработки представлений и отображения их пользователю необходимо создать файл, содержащий функции обработки запросов пользователя и необходимых данных (например, **views.py**). Обработчики представляют собой функции, возвращающие в качестве результата, например, готовое представление или же результат выполнения переадресации на другую страницу. В качестве аргументов обязательно принимают объект, представляющий запрос пользователя, из которого в дальнейшем могут быть получены значения переданных на сервер параметров. Для этого используются следующие функции:

- **HttpResponse** – отправляет ответ сервера пользователю, ответ указывается в скобках после имени функции: **HttpResponse(…);**
- **HttpResponseRedirect** (пакет **django.http**) – осуществляет переадресацию на указанный в скобках адрес: **HttpResponseRedirect(…);**
- **render** – рендеринг (подготовка) страницы для выдачи пользователю, в скобках после имени функции указывается объект запроса, адрес шаблона, словарь передаваемых параметров, например:
  render(request, 'photoorg/albumimages.html', {'isAuthor': isAuthor, 'album': album, 'images': imagesToDisplay})
- **reverse** (пакет **django.core.urlresolvers**) – получение ссылки на страницу по описанию, также могут быть переданы дополнительные аргументы.

Для выполнения переадресации на другую страницу, ссылку на которую необходимо получить, используется следующий формат записи:

```
HttpResponseRedirect(reverse('photos:images',
args=(albumid,)))
```

Чаще всего в виде представлений в веб-приложениях выступают HTML-страницы, сформированные вручную или же с помощью некоторого шаблона. В Django используется собственный шаблонизатор, позволяющий удобно работать с передаваемыми данными и форматом страниц.

Для хранения шаблонов страниц веб-приложений создается папка с шаблонами, путь к которой указывается в соответствующем параметре в настройках. Для работы с шаблоном необходимо вставить необходимый блок кода в фигурные скобки вида **{%** … **%}**. Подстановка значения переданных переменных осуществляется путем указания имени переменной в двойных фигурных скобках, например **{{surname}}** или **{{person.surname}}**.

Наиболее часто используемые функции шаблонов:

- **{% load staticfiles %}** – обеспечивает подгрузку статических файлов, например, стилей, к странице, прописывается в начале шаблона;
- **{% static "styles.css" %}** – подключает статический файл (например, файл CSS-стилей **styles.css**) к странице, находя его в указанных с помощью **STATICFILES_DIRS** папках;
- **{% extends 'templatePath' %}** – указывает странице, что текущий шаблон является расширением шаблона с путем **templatePath**, может использоваться с целью упрощения кода, например, не будет необходимости добавлять в каждый шаблон шапку главной страницы и т.д., прописывается в начале шаблона;
- **{% url mainProcessor:urlName %}** – подстановка (например, в тег **<a>**) ссылки **urlName** (сокращенное имя ссылки) из **mainProcessor** (имя приложения проекта);
- **{% block blockTitle %}** … **{% endblock %}** – позволяет определять блоки с определенным именем (**blockTitle**) и указанным содержимым на странице, при наследовании шаблонов может использоваться для сохранения местоположения блока на странице, но изменения его содержимого (в этом случае прописывается в начале шаблона);
- **{% for variable in container %}** … **{% endfor %}** – цикл **for** для итерации по массиву переданных в шаблон данных, например:

```
    {% for a in publicAlbums %}
        <option value="{{ a.id }}">{{ a.title }}</option>
    {% endfor %}
```

Здесь **a** – объект последовательности общедоступных альбомов (**publicAlbums**), для которого создается элемент страницы **<option>** с подстановкой соответствующего идентификатора (**a.id**) и названия (**a.title**).

- **{% if condition %}…{% elif %}…{% else %} {% endif %}** – условие (блоки **elif** и **else** являются необязательными), например:

```
{% if albumList %}
    Number of albums: {{ albumList.length }}
{% else %}
    No albums.
{% endif %}
```

С полным списком доступных блоков шаблонов Django можно ознакомиться по ссылке:

https://docs.djangoproject.com/en/1.9/ref/templates/builtins/

Формат JSON

JSON является форматом, удобным для передачи, обработки, изменения и доступа к данным, который представляется аналогично ассоциативному массиву парами ключ-значение:

```
{
   "firstname": "Иван",
   "lastname": "Иванов",
   "address": {
       "streetAddress": "Kroverksky pr. 49",
       "city": "Saint Petersburg",
       "postalCode": 197101
   },
   "education": [
       "school": {
           "number": 123,
           "name": "Global school"
       },
```

```
        "university": {
            "name": "ITMO",
            "yearStart": 2011,
            "yearEnd": 2015
        },
    ]
}
```

Поля (ключи) объекта JSON представляют собой строки, в то время как их значения, указанные через двоеточие, могут являться следующими типами данных:

- число;
- строка;
- одномерный массив;
- литералы: **`true`**, **`false`**, **`null`**;
- объекты, также представляемые в аналогичном JSON формате

Для работы с форматом JSON в языке Python существует специальный пакет **`json`**, предоставляющий функции для разбора, создания и обработки данных этого формата, например:

- **`dumps(obj, [..])`** – производит сериализацию (перевод) объекта **`obj`** в формат JSON, также при вызове функции могут быть указаны дополнительные параметры, ознакомиться с которыми можно по приведенной ниже ссылке;
- **`loads(jsonObj, [..])`** – производит десериализацию данных в формате JSON в списки, словари и т.д., также при вызове функции могут быть указаны дополнительные параметры

Стандартное расширение файла для данных типа JSON: **`.json`**

С полным списком функций и их аргументов можно ознакомиться по ссылке:
`https://docs.python.org/3/library/json.html`

7. Создайте файл **`json_data.json`** со структурой согласно описанию:

Каждый человек (руководитель, сотрудник, студент) в описанной структуре характеризуется фамилией, именем, отчеством, должностью и имеет табельный номер. Каждое подразделение (ректорат, бухгалтерия, мегафакультет, факультет, кафедра) помимо указанных в описании данных характеризуется идентификатором, названием, имеет руководителя

(информация о руководителе представляется согласно приведенному описанию человека) и сотрудников, представленных в виде списка описывающих их объектов.
Университет характеризуется названием, местоположением (с указанием отдельно индекса, города и адреса) и имеет руководителя. В университете существует два типа подразделений: административные и научно-образовательные. К административным подразделениям относятся ректорат и бухгалтерия, описанные характеристиками подразделений, указанными выше. К научно-образовательным подразделениям относятся мегафакультеты, каждый из которых также имеет список факультетов. Факультет помимо базовых параметров также содержит список кафедр факультета, для которых список сотрудников представлен списком преподавателей кафедры. Также кафедра имеет список образовательных программ, каждая из которых характеризуется названием, номером, руководителем, дисциплиной (указывается название дисциплины) и годом обучения. Год обучения представляет собой структуру, для которой прописывается непосредственно год обучения в виде строки (ограничиться годом 2016/2017) и список учебных групп для конкретного года обучения, каждая из которых включает название учебной группы и список обучающихся в ней студентов, для которых вместо должности указывается размер получаемой стипендии или null в случае ее отсутствия.

Заполните структуру данными по Университету ИТМО (идентификаторы могут быть придуманы самостоятельно). Приведите не менее 3 различных примеров вложенных объектов для каждого объекта-списка.

8. Создайте шаблон **universityInfo.html** и соответствующие ему ссылку **universityInfo** и обработчик запроса пользователя. Выведите на страницу данные по университету в указанном формате, подставив нужные значения, полученные из файла **json_data.json**. Формат вывода данных:

```
Университет: ..
Ректор: ..
Местоположение: индекс, город, адрес
Кол-во административных подразделений: ..
Кол-во научно-образовательных подразделений: ..
Кол-во мегафакультетов: ..
Кол-во факультетов: ..
Кол-во кафедр: ..
```

9. Создайте шаблон **disciplineInfo.html** и соответствующие ему ссылку **disciplineInfo** и обработчик запроса пользователя. Выведите

на страницу данные по дисциплине в следующем формате, подставив вместо многоточий нужные значения, полученные из файла **json_data.json**:

```
Номер программы: ..
Название программы: ..
Дисциплина: ..
Год обучения: ..
Кол-во групп: ..
```

10. Создайте шаблон **groupsInfo.html** и соответствующие ему ссылку **groupsInfo** и обработчик запроса пользователя. Выведите на страницу данные по студентам всех групп в указанном формате, подставив нужные значения, полученные из файла **json_data.json**. Формат вывода:

Группа	Табельный номер	Студент	Стипендия
Q1234	120	Фамилия Имя Отчество	10 000
...	...	...	...

11. Создайте шаблон **departmentsInfo.html** и соответствующие ему ссылку **departmentsInfo** и обработчик запроса пользователя. Выведите на страницу данные по кафедрам в указанном формате, подставив нужные значения, полученные из файла **json_data.json**. Формат вывода данных:

```
ID: ..
Кафедра: ..
Заф. кафедры: Фамилия Имя Отчество (таб.номер)
Преподаватели:
  1.Фамилия Имя Отчество (таб.номер, должность)
  2.Фамилия Имя Отчество (таб.номер, должность)
    …
Образовательные программы:
  • Номер программы (Название, Дисциплина)
```

Между каждыми двумя кафедрами делайте пропуск двух строк при выводе информации.

12. Создайте шаблон **universityStructure.html** и соответствующие ему ссылку **universityStructure** и обработчик запроса пользователя. Выведите на страницу структуру университета с указанием идентификаторов подразделений и их названий, а также ФИО руководителей (Фамилия И.О.) в указанном формате, подставив нужные значения, полученные из файла **json_data.json**. Формат вывода данных:

```
Университет .. (Фамилия И.О. руководителя)
|_Административные подразделения:
| |_Ректорат (ID,..)
| |_Бухгалтерия (ID,..)
|
|_Научно-образовательные подразделения:
  |_Мегафакультет .. (ID,..)
  | |_Факультет .. (ID,..)
  | | |_Кафедра .. (ID,..)
  | | |_Кафедра .. (ID,..)
  | | |_ ...
  | | |_Кафедра .. (ID,..)
  | |_Факультет .. (ID,..)
  | | ...
  | |_Факультет .. (ID,..)
  | ...
  |_Мегафакультет .. (ID,..)
  | ...
  |_Мегафакультет .. (ID,..)
```

Лабораторная работа № 6: Введение в Java

1. Запустите онлайн среду разработки "Tutorials Point" с рабочим пространством для языка программирования Java версии 8 по ссылке:

https://www.tutorialspoint.com/compile_java8_online.php

Интерфейс среды разработки состоит из трех областей: с левой стороны располагается каталог текущих файлов проекта, с правой стороны – редактор кода программы, под редактором кода находится терминал для ввода команд. В ходе выполнения лабораторных работ будет необходимо создавать программы и запускать их командами из терминала. Для сохранения файлов проекта можно использовать локальный диск ("Project" → "Download project") или диск Google ("Project" → "Save project" → "Save on Google Drive").

Язык программирования Java

Java является объектно-ориентированным языком программирования. Практически все данные в Java представляют собой объекты, за исключением примитивных типов данных. Это строго типизированный язык программирования с трансляцией разработанных приложений в байт-код, что позволяет использовать Java на любой платформе. Java обладает большим количеством преимуществ, включая автоматическое управление памятью, набор встроенных коллекций (массивы, списки, стеки и т.д.), средства создания сетевых приложений и многое другое.

Простейшая программа на языке Java:

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

В приведенном примере объявлен общедоступный (**public**) класс **HelloWorld**, имеющий общедоступный статический (**static**) метод **main** без возвращаемого значения (**void**), который осуществляет вывод на экран текста **"Hello, World!"**.

Компиляция программ осуществляется с использованием команды **javac**,

после которой указывается имя файла:
javac HelloWorld.java

Запуск программ осуществляется с использованием команды **java**, после которой указывается имя запускаемого класса:
java HelloWorld

Для вывода информации в консоль используется класс **System**, реализующий стандартные методы работы с входными и выходными потоками данных. Для вывода информации в консоль может быть использован метод **println()** поля класса для выходного потока **out**, осуществляющий вывод переданного в скобках значения с добавлением перевода строки.

Наличие главного метода класса определяет, является ли данный класс запускаемым. Главный метод характеризуется спецификатором доступа **public** (является общедоступным), модификатором **static** (метод статичен), типом **void** (метод без возвращаемого значения) именем **main** и аргументом **String[] args**, представляющим собой массив значений, переданных в командной строке при вызове метода.

2. Создайте файл **FirstClass.java**:

```
public class FirstClass {
    public static void main(String[] args) {
        int i = 5;
        i += 7;
        System.out.println("i = " + i);
        double d = (double) i / 8;
        System.out.println("d = " + d);
        char c1 = 'n';
        char c2 = 110;
        char c3 = 111;
        System.out.println("c1=" + c1 + " & " + "c2="
+ c2);
        boolean b1 = (c1 == c2);
        boolean b2 = (c1 == c3);
        System.out.println(b1);
        System.out.println(b2);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
i = 12
d = 1.5
c1=n & c2=n
true
false
```

Примитивные типы

Язык программирования Java оперирует понятиями объектов и классов, реализующих методы, которые могут быть выполнены над соответствующими объектами. Исключение составляют 8 примитивных типов, представленных в таблице, для которых возможно использовать лишь базовые арифметические и логические операции. Простые типы являются упрощением базовых классов, реализующих основные методы работы с данными. При необходимости можно использовать приведение типов путем указания нужного типа в скобках перед выражением.

Примитивный тип	Диапазон значений	Основной класс
boolean	**true, false**	Boolean
char	Unicode: 0 .. 2^{16}-1	Character
byte	-128 .. 127	Byte
short	-2^{15} .. 2^{15}-1	Short
int	-2^{31} .. 2^{31}-1	Integer
long	-2^{63} .. 2^{63}-1	Long
float	IEEE 754	Float
double	IEEE 754	Double
void	-	Void

Основным отличием примитивных типов от классов является отсутствие методов работы с переменными таких типов, а также формат их записи: названия классов в языке Java всегда начинаются с прописной буквы, в то время как названия примитивных типов прописываются строчными буквами.

Описание основных арифметических и логических операций можно найти в спецификации языка Java:
https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.2.2

3. Дополните файл **FirstClass.java**. Создайте целочисленную переменную **year** со значением года Вашего рождения и переменную **div** типа **double**, записав в нее результат деления значения переменной **year** на 5. Выведите значение переменной **div** на экран.

4. Дополните файл **FirstClass.java**. Создайте переменные логического типа **L1** и **L2** со значениями **true** и **false** соответственно. Поочередно осуществите вывод на экран результаты выполнения всех возможных логических операций над переменными **L1** и **L2**.

5. Создайте файл **SecondClass.java**:

```
public class SecondClass {
    public static void main(String[] args) {
        String s = "13.7";
        Double a = new Double(s);
        char c = "qwe".charAt(2); // символ со 2-й позиции
        System.out.println(a);
        System.out.println(c);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
13.7
e
```

Объекты

Классы в языке Java определяют типы создаваемых объектов. Создание объекта какого-либо класса осуществляется с помощью вызова конструктора класса с использование ключевого слова **new**. Для основных классов, представленных в таблице, и класса **String**, предназначенного для работы со строками, возможно опускать использование ключевого слова **new**. Таким образом, объекты таких классов могут быть созданы несколькими способами, представленными далее, первый из которых является общим для создания объектов всех классов. Все объекты должны быть объявлены. При объявлении объекта без указания его значения,

присваивается пустое значение **null**.

```
Integer i; // объекту присвоено значение null
int a = 5;
Integer b = new Integer(a);
Integer c = 6;
```

Для работы со строками предназначен класс `String`, объекты которого так же могут быть объявлены различными способами. Вызов методов классов осуществляется с помощью указания имени метода через точку от имени объекта данного класса:

```
String s1 = "Hello, ";
String s2 = s1.concat("World!"); // конкатенация строк
System.out.println(s2); // вывод в консоль значения s2
```

Следует отметить, что под "работой с объектами" в языке Java подразумевается работа со ссылками на объекты. Полная информация, включающая описание встроенных классов языка Java и соответствующих методов доступна по ссылке:
https://docs.oracle.com/javase/8/docs/api/overview-summary.html

6. Дополните файл **SecondClass.java**. Создайте новый объект **iS** целочисленного типа **Integer** из строки "135" двумя способами: с помощью конструктора с использованием ключевого слова **new** и с помощью метода класса **Integer** получения числового значения из строки.

7. Дополните файл **SecondClass.java**. Создайте строку **s1** "Java is one of the best languages!". Осуществите вывод на экран результатов следующих действий, выполненных с использованием методов класса **String**:

- получение символа, находящегося на 5 позиции;
- сравнение строки со строкой "Java is one of the most beautiful languages!";
- поиска в строке подстроки "best".

8. Создайте файл **ThirdClass.java**:

```
public class ThirdClass {
    public static void main(String[] args) {
        double[] arr = {0.5, 1.3, 2.7, 0.2};
        Arrays.sort(arr);
        System.out.println(Arrays.toString(arr));
        System.out.println(arr[2]);
        System.out.println(arr.length);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
[0.2, 0.5, 1.3, 2.7]
1.3
4
```

Массивы

В языке программирования Java существует множество различных коллекций. Для примера рассмотрим класс **Arrays**, реализующий методы работы с массивами в Java. Объявление массивов происходит аналогично другим языкам программирования. Индексация элементов массива начинается с 0.

```
int[] arr;
arr = new int[3]; // резервируем память на 3 элемента массива
// заполняем массив значениями
arr[0] = 5;
System.out.println(arr[0]);
// заполнение массива при инициализации
int[] a = {0, 1, 2, 3};
```

Класс **Arrays** содержит статичные методы (подробнее режимы доступа и модификаторы рассмотрены в следующем блоке), принимающие на вход объекты массивов, над которыми необходимо выполнять действия. Статичные поля и методы классов могут быть вызваны напрямую от самого класса. Для использования методов работы с массивами необходимо подключить класс **Arrays** пакета **java.util** с помощью

ключевого слова **import**:

```
import java.util.Arrays; // подключение Arrays
```

Пример работы с массивами:

```
double[] arr = {0.5, 1.3, 2.7, 0.2};
Arrays.sort(arr); // сортировка массива
System.out.println(Arrays.toString(arr));   //   вывод
значений
```

9. Дополните файл **ThirdClass.java**. Создайте массив строк **stringArray**. Заполните его значениями, осуществив разделение строки "Peter Piper picked a peck of pickled peppers" по символу пробела. Осуществите вывод на экран результатов выполнения следующих действий:

- вывод значений всего массива, используя метод класса **Arrays**;
- вывод на экран значения пятого элемента массива;
- сортировка массива, используя метод класса **Arrays**;
- поиск элемента "peppers" в массиве, используя метод класса **Arrays**.

10. Создайте файл **FourthClass.java**:

```
public class FourthClass {
    public static void main(String[] args) {
        int p = 0;
        for (int i = 0; i < 30; i++) {
            if (i % 2 == 0) {
                double d = (double) i / 4;
                System.out.print(d+"; "); // вывод в
одну строку
            }
        }
        System.out.println();
        int year = 2016;
        switch (year) {
            case 2014:
                System.out.println("You're  3rd  year
student");
                break;
            case 2015:
```

```
                System.out.println("You're  2nd  year
student");
                break;
            case 2016:
                System.out.println("You're  1st  year
student");
                break;
        }
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
6.5 7.0
You're 1st year student
```

Базовые конструкции Java

Условный оператор:

```
if (<условие>) {
    <операторы>
}
else if (<условие>) {
    <операторы>
}
else {
    <операторы>
}
```

Оператор **switch**:

```
switch (<переменная>) {
    case <значение>:
        <операторы>
        break;
    . . .
    default:
        <операторы>
        break;
}
```

Оператор цикла **for**:

```
for (<переменная>;<условие>;<оператор>) {
    <операторы>
}
```

Оператор цикла **while**:

```
while (<условие>) {
    <операторы>
}
```

Оператор цикла **do..while**:

```
do {
    <операторы>
} while (<условие>);
```

Для операторов цикла могут быть использованы операторы выхода из цикла (**break**) и перехода на следующую итерацию (**continue**).

Области видимости

Области видимости переменных примитивных типов определяются фигурными скобками **{}**. Таким образом, внутренние переменные условий, циклов и других выделенных фигурными скобками блоков будут не доступны в блоках, являющихся внешними по отношению к данным. При этом внутренние блоки могут использовать переменных внешних блоков. В случае с объектами видимость так же ограничивается блоками фигурных скобок, однако объект, созданный с использованием ключевого слова **new**, остается в памяти и после завершения блока.

Одним из главных преимуществ языка программирования Java является отсутствие необходимости удаления (разрушения) объектов после окончания работы с ними. Для этого существует "Сборщик мусора" (garbage collector), который автоматически очищает память, занимаемую объектами, созданными с помощью **new**, когда ссылка на них, записанная в переменную перестает использоваться программой.

11. Дополните файл **FourthClass.java**. Осуществите вывод на экран таблицы истинности для функции трех логических переменных **x1∧x2∨¬x3** с использованием трех видов циклов.

12. Дополните файл **FourthClass.java**. Осуществите последовательный вывод на экран чисел кратных 7 и меньших 130. Остановите работу цикла с помощью оператора остановки по достижении значения в три раза превосходящего число 23.

13. Создайте файл **Planet.java**:

```
public class Planet {
}
```

Класс

Для создания собственного класса Java необходимо создать файл **<Имя_класса>.java**. Имя файла должно полностью совпадать с именем класса. Имена классов принято начинать с прописной буквы. Объявление класса происходит с помощью ключевого слова **class**.

```
class MyClass {
    . . .
}
```

Объявление объектов определенного класса осуществляется следующим образом:

```
MyClass obj = new MyClass();
```

14. Создайте общедоступный класс **Satellite**.

15. Модифицируйте класс **Planet**:

```
public class Planet {

    String name;
    Double radius;
    Double sunDistance;

}
```

Поля класса

Для класса определяются поля определенных типов, которые должен содержать данный класс. Для каждого поля обязательно указывается его

тип. Поля класса рекомендуется определять сразу после строки объявления класса, например:

```
class MyClass {
    int century;
    int year;
    . . .
}
```

Если значения полей класса не заданы, для них будут установлены стандартные нулевые значения соответствующих типов.

Доступ к полям объекта класса осуществляется с указанием названия поля класса через точку после имени объекта:

```
obj.century = 21;
```

16. Дополните класс **Satellite**. Добавьте классу **Satellite** следующие поля:

- **name** – название;
- **radius** – содержит значение радиуса спутника;
- **period** – период обращения спутника (в часах).

17. Модифицируйте класс **Planet**:

```
public class Planet {

    String name;
    Double radius;
    Double sunDistance;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double toThousandKm(String param){
        double result = 0;
        switch (param){
            case "sunDistance":
                result =  this.sunDistance / 1000;
```

```
            case "radius":
                result =  this.radius / 1000;
        }
        return result;
    }
}
```

Методы класса

Методы класса определяют действия, которые могут быть выполнены над его объектами. Для каждого метода класса обязательно указывается тип возвращаемого значения или **void**, в случае, если метод ничего не возвращает, и имя метода. Формат описания метода представляется следующим образом:

```
<тип_возвращаемого_значения> <имя_метода>
(<аргументы>) {
    <операторы>
    [return <значение>;]
}
```

Например:

```
class MyClass {
    int century;
    int year;

    double division (double value) {
        return this.year / value;
    }
}
```

В описанном примере метод **division** возвращает остаток от деления значения поля **year** объекта класса **MyClass** на переданное методу в качестве аргумента значение. Ключевое слово **this**, использованное внутри метода представляет собой текущий объект класса, для которого был вызван метод **division**.

18. Дополните класс **Satellite**. Добавьте классу **Satellite** следующие методы:

- **getPeriod** –возвращает значение периода обращения спутника;
- **getPeriodInDays** – осуществляет перевод и возвращает значение периода обращения спутника в днях;

- **print** – осуществляет вывод информации об объекте (значения всех полей объекта класса), не возвращает значений;

19. Модифицируйте класс **Planet**:

```
public class Planet {

    String name;
    Double radius;
    Double sunDistance;

    public Planet(String name, Double radius, Double
sunDistance) {
        this.name = name;
        this.radius = radius;
        this.sunDistance = sunDistance;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double toThousandKm(String param){
        double result = 0;
        switch (param){
            case "sunDistance":
                result =  this.sunDistance / 1000;
            case "radius":
                result =  this.radius / 1000;
        }
        return result;
    }
}
```

20. Создайте файл **FifthClass.java**:

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

```
        Planet planet = new Planet("Earth-01",
6371.0, 149.6);
        System.out.println(planet.name);
        planet.setName("Earth");
        System.out.println(planet.getName());

System.out.println(planet.toThousandKm("radius"));
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Earth-01
Earth
6.371
```

Конструкторы класса

Для каждого класса по умолчанию создается пустой конструктор. Определение собственных конструкторов происходит следующим образом:

```
class MyClass {
    int century;
    int year;

    public MyClass(int century) {
        this.century = century;
    }

    double division (double value) {
        return this.year / value;
    }
}
```

В приведенном примере общедоступный конструктор **MyClass()** принимает на вход значение целочисленного типа **century** и записывает его в качестве значения поля **century** создаваемого объекта класса.

21. Дополните класс **Satellite**. Создайте конструктор для класса, принимающий на вход значения полей **name**, **radius** и **period**.

22. Дополните класс **Planet**. Добавьте поле **satellite** типа **Satellite** и метод **getSatelliteInfo**, осуществляющий вывод информации о спутнике планеты на экран и не возвращающий значений.

23. Дополните метод **main** класса **FifthClass**, создав объект класса **Satellite** и реализовав проверку всех созданных методов.

24. Есть три стержня, на которых размещены n дисков таким образом, что на диске большего диаметра находится диск меньшего диаметра. В начальный момент все диски находятся на первом стержне. Напишите программу, осуществляющую перемещение дисков с первого стержня на третий, при условии, что на диске большего диаметра всегда должен находится диск меньшего диаметра.

Лабораторная работа № 7: Инкапсуляция, полиморфизм, наследование

1. Создайте файл **Cube.java**, демонстрирующий использование различных модификаторов доступа:

```
public class Cube {
    public double width;
    private double height;
    protected double depth;

    public Cube(double width, double height, double
depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
    }

    public void print(){
        System.out.println(this.width + " | " +
this.height +
                           " | " + this.depth);
    }
}
```

2. Создайте файл **AccessExample.java**:

```
public class AccessExample {
    public static void main(String[] args) {
        // обращение к полям с разными режимами
доступа
        Cube c = new Cube(5,4,3);
        c.print();
        System.out.println(c.width);
        System.out.println(c.depth);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
5.0 | 4.0 | 3.0
```

```
5.0
3.0
```

Модификаторы доступа

В Java предусмотрен контроль пространства имен на уровне пакетов. Пакет в Java представляет собой набор Java-классов и сопутствующих файлов (например, изображений, файлов стилей и т.д.), представленный в виде отдельной директории типа **`package`**. Использование пакетов позволяет разделить написанные программистом классы конкретной программы и другие классы, находящиеся внутри пакета.

Программист может устанавливать различные режимы доступа к полям и методам класса и области их видимости с помощью следующих ключевых слов (модификаторов доступа):

- **`public`** – поля и методы, указанные как **`public`**, являются доступными из любого класса, вне зависимости от того, в каком пакете находится класс;
- **`protected`** – поля и методы, указанные как **`protected`**, являются общедоступными внутри текущего пакета и приватными в случае, если класс находится в другом пакете;
- *без модификатора* – поля и методы, указанные без модификатора, являются публичными внутри текущего пакета и приватными за его пределами;
- **`private`** – поля и методы, указанные как **`private`**, являются доступными исключительно внутри текущего класса.

	Класс	Пакет	Унаследованный класс	Другое
`public`	Доступен	Доступен	Доступен	Доступен
`protected`	Доступен	Доступен	Доступен	*Не доступен*
без модификатора	Доступен	Доступен	*Не доступен*	*Не доступен*
`private`	Доступен	*Не доступен*	*Не доступен*	*Не доступен*

В приведенном примере, в случае, если класс **`Cube`** будет помещен в отдельный пакет, доступ к полю **`depth`** пропадет и возникнет соответствующая ошибка при попытке компиляции и запуска программы.

Поля классов рекомендуется делать приватными (инкапсулировать),

создавая геттеры (методы для получения значений полей, например **getWidth()**) и сеттеры (методы для установки значений полей, например **setWidth(double width)**). Таким образом, каждый класс будет обладать набором полей и методы со скрытой реализацией, предоставляя возможность обращения извне исключительно к общедоступным полям с модификатором доступа **public**.

3. Создайте класс **Spy** с полями строкового типа **name** и **realName** с модификаторами доступа **public** и **private** соответственно, приватное поле целочисленного типа **squad**, методами **getSpyInfo()** (возвращает строку с информацией об объекте) с доступом **private** и **print()** с публичным доступом, выводящим информацию об объекте на экран. Для каждого из полей добавьте геттер и сеттер. В отдельном классе проведите испытания для всех полей и методов, выводя информацию на экран.

4. Создайте файл **Triangle.java**:

```
public class Triangle {

    public double side; // сторона

    public Triangle(double side) {
        this.side = side;
    }

    public double area(){
        return(Math.sqrt(3)/4)*Math.pow(this.side,2);
    }

    public static void checkTriangles(Triangle
triangle1, Triangle triangle2) {
        double area1 = triangle1.area();
        double area2 = triangle2.area();
        if (area1 == area2) {
            System.out.println("Треугольники равны");
        }
        else if (area1 > area2) {
            System.out.println("Первый треугольник
больше второго");
        }
        else {
```

```
            System.out.println("Второй     треугольник
больше первого");
        }
    }
}
```

5. Создайте файл **StaticExample.java**:

```
public class StaticExample {

    static int result;

    public static void main(String[] args){
        int[] array = {-3, 20, 5, 16, 27};

        // итерация по объектам коллекции
        for (int value: array) {
            // тернарный оператор
            result = (value % 4 == 0) ? 1 : 0;
            System.out.println(result);
        }

        Triangle t1 = new Triangle(3);
        Triangle t2 = new Triangle(4);
        Triangle.checkTriangles(t1,t2);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
0
1
0
1
0
Второй треугольник больше первого
```

Ключевое слово static

Доступ к полям и методам, объявленным с модификатором **static**, можно получить без создания объекта напрямую от соответствующего

класса:

```
public class ComedyBook {
    static genre = "comedy";
    …
}
```

Основные особенности модификатора **static**:

- Доступ к полям и методам, объявленным с модификатором **static**, может быть получен только из статических методов;
- Методы, объявленные с модификатором **static**, не могут быть переопределены при наследовании;
- Поля, объявленные с модификатором **static**, инициализируются после загрузки класса в память;
- Для классов, имеющих статические методы, доступен импорт (**import static ...**) данных методов в другие классы, например: **import static java.lang.Math.sqrt** позволит использовать функцию исчисления корня из числа в виде **sqrt(value)** вместо **Math.sqrt(value)**.

6. Создайте класс **StaticContainer** со целочисленным **static**-полем **counter** и публичным **static**-методом **operation()**, реализующим увеличение значения counter на 3 при каждом вызове метода. Метод **operation()** значений не возвращает.

7. Создайте запускаемый класс **StaticCheck**, в главном методе которого реализуйте, используя бесконечный цикл, ожидание получения значения поля **counter** класса **StaticContainer** более 100, после чего завершите цикл и выведите значение **counter** на экран.

8. Создайте файл **Plant.java**:

```
public class Plant {
    private String type;
    private String color;

    public Plant() {}

    public Plant(String type, String color) {
        this.type = type;
        this.color = color;
```

```
    }

    public Plant(String type) {
        this.type = type;
    }

    public void print(){
        System.out.println("type: " + this.type + ";
color: " + this.color);
    }
}
```

9. Создайте файл **OverloadExample.java**:

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

        Plant p1 = new Plant();
        Plant p2 = new Plant("tulip", "red");
        Plant p3 = new Plant("cactus");

        p1.print();
        p2.print();
        p3.print();

    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
type: null; color: null
type: tulip; color: red
type: cactus; color: null
```

Полиморфизм

Методы в Java характеризуются названием, аргументами, возвращаемым значением, режимом доступа. Полиморфизм – способность методов обрабатывать данные разных типов. Программисты могут создавать в классе методы с одинаковым названием, но имеющие при этом различные аргументы. В этом случае метод будет перегружен, при вызове метода из

внешнего контекста, компилятор определит, какой именно из методов должен быть вызван по его аргументам. Также следует учитывать, что порядок аргументов при перегрузке методов будет влиять на вызываемый метод.

10. Дополните класс **Plant**, добавив приватные поля **existenceArea** (тип **String**) и **rare** (тип **boolean**). Создайте дополнительные к существующим конструкторы с использованием вновь созданных полей класса. Осуществите проверку в главном методе класса **OverloadExample**.

11. Создайте класс **Ship** со следующими полями:
- **title (String)** – название судна;
- **captainName (String)** – имя капитана судна;
- **port (int)** – номер порта стоянки судна (целое число);
- **type (char)** – тип судна (буква латинского алфавита).

Разработайте программу испытаний различных вариантов перегрузки (допустимых и недопустимых) некоторого метода **updateShipInfo()** (**public**, **void**), обновляющего различные данные о судне, выводя список текущих аргументов метода на экран каждый раз при вызове. На вход метод принимает одно или несколько значений разных полей класса. Вызов методов осуществляйте в отдельном запускаемом классе **ShipTest**.

12. Создайте файл **Furniture.java**:

```
public class Furniture {
    String material;
    double age;
    public Furniture() {}
    public Furniture(String material, double age) {
        this.material = material;
        this.age = age;
    }
    public void print() {
        System.out.println("Furniture: ");
        System.out.println("material:       "       +
this.material + "; age: " + this.age);
    }
}
```

13. Создайте файл **Chair.java**:

```
public class Chair extends Furniture {
    private String color;

    public Chair(String material, double age, String
color) {
        super(material, age);
        this.color = color;
    }
    @Override
    public void print() {
        System.out.println("from Furniture: ");
        super.print();
        System.out.println("Chair: ");
        System.out.println("material:       "       +
this.material + "; age: " + this.age +
                           ";     color:     "     +
this.color);
    }
}
```

14. Создайте файл **InheritanceExample.java**:

```
public class InheritanceExample {
    public static void main(String[] args){
        Furniture        furniture        =        new
Furniture("wood",2.5);
        furniture.print();
        Chair  chair  =  new  Chair("aluminum",  1.3,
"beige");
        chair.print();
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Furniture:
material: wood; age: 2.5
from Furniture:
Furniture:
```

```
material: aluminum; age: 1.3
Chair:
material: aluminum; age: 1.3; color: beige
```

Наследование

Наследование в Java реализуется при помощи ключевого слова **extends** в следующем формате:

```
. . . Descendant extends Ancestor { . . . }
```

где **Descendant** – имя дочернего класса, **Ancestor** – имя родительского класса.
Может быть использовано множественное наследование с указанием двух и более родительских классов через запятую. Наследуются поля и методы родительских классов.

При наследовании необходимо учитывать следующее:

- поля и методы доступа **private** не наследуются;
- поля и методы доступа **public** должны оставаться **public** в дочерних классах;
- поля и методы доступа **protected** должны быть **public** или **protected** в дочерних классах.

При наследовании можно получить доступ к методам суперкласса (класса-родителя) напрямую из дочернего класса с использованием ключевого слова **super**, например:

```
super.printValue();
```

Все классы по умолчанию унаследованы от класса **Object**, предоставляющего следующие методы:

- **toString()** – строковое представление объекта класса;
- **equals(Object obj)** – метод сравнения текущего объекта с объектом **obj**, переданным в качестве аргумента;
- **getClass()** – возвращает имя и класса;
- **notify()** и **notifyAll()** – используются для работы с потоками, ожидающими реакцию объекта;
- **wait()** – переводят поток, обрабатывающий объект, в режим ожидания;
- **hashCode()** – возвращает хэш-код для объекта.

При необходимости указанные методы могут быть переопределены внутри любого дочернего класса.

15. Дополните класс **Chair**. Переопределите метод **toString()**, унаследованный от класса **Object** так, чтобы он возвращал информацию о всех полях объекта класса вместе с их текущими значениями. В классе **InheritanceExample** осуществите проверку переопределенного метода.

16. Используя наследование, реализуйте схему банковских счетов некоторого аккаунта в банке, предполагая наличие как минимум двух счетов для разных типов валюты. Классы должны хранить информацию о номере счета, лимите счета и текущих средствах. В поле строкового типа, объявленном с модификатором **static**, в родительском классе необходимо указать имя владельца счетов.

17. Пусть задана последовательность из n положительных чисел **w = (w1, w2, ..., wn)** и **s**. Разработайте программу, которая вычисляет бинарный вектор **x = (x1, x2, ..., xn),(xi = 0** или **1)**, чтобы:

$$s = \sum_{i=1}^{n} x_i w_i$$

Проведите тестирование программы.

18. Пусть **p=43**, а **q=61**. Постройте асимметричную систему шифрования с открытым ключом. Проведите испытание построенной системы.

Лабораторная работа № 8: Интерфейсы и абстрактные классы

1. Создайте файл **AbstractExample.java**, демонстрирующий работу с абстрактными классами и их методами:

```
// абстрактный класс
abstract class Storage {
    // сохранить данные (абстрактный метод)
    public abstract void store(String data);
    // чтение данных (абстрактный метод)
    public abstract String read();
}
// класс, реализующий абстрактный
class CD extends Storage {
    private String data = "";
    public CD() {}
    @Override // реализация
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on CD");
    }
    @Override // реализация
    public String read() {
        return this.data;
    }
}
// класс, реализующий абстрактный
class SDCard extends Storage {
    private String data = "";
    public SDCard() {}
    @Override // реализация
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on SDCard");
    }
    @Override // реализация
    public String read() {
        return this.data;
    }
}
```

```
public class AbstractExample {
    public static void main(String[] args) {
        // создание объекта CD
        CD cd = new CD();
        cd.store("Some data to CD");
        System.out.println(cd.read());
        // создание объекта SDCard
        SDCard sd = new SDCard();
        sd.store("Some data to SDCard");
        System.out.println(sd.read());
        // попытка создать объект класса Storage
        // требует реализации абстрактных методов
        Storage storage = new Storage() {
            private String data = "";
            @Override
            public void store(String data) {
                this.data += data;
                System.out.println("Data   stored   on
storage");
            }
            @Override
            public String read() {
                return this.data;
            }
        };
        storage.store("Some data to storage");
        System.out.println(storage.read());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Data stored on CD
Some data to CD
Data stored on SDCard
Some data to SDCard
Data stored on storage
Some data to storage
```

Абстрактные классы и методы

Абстрактные методы служат для объявления сигнатуры методов без его реализации (тела метода). Абстрактный метод объявляется с использованием ключевого слова **abstract**:

```
abstract void erase();
```

Если класс содержит хоть один абстрактный метод, то он считается абстрактным и должен быть объявлен с использованием ключевого слова **abstract**:

```
abstract class Storage { . . . }
```

Абстрактные классы в Java широко используются при наследовании для определения шаблонов реализации классов, унаследованных от абстрактного. В показанном примере объявлен класс **Storage** с абстрактными методами **store** и **read** для записи и чтения данных соответственно. От него унаследованы классы **CD** и **SDCard**, реализующие методы в соответствии со списком, описанным в классе **Storage**. Все абстрактные методы, указанные в абстрактном классе, обязательно должны быть реализованы в унаследованных классах. При попытке создать объект абстрактного класса, необходимо напрямую указывать реализацию его абстрактных в фигурных скобках после создания объекта, в ином случае возникнет ошибка компиляции, например:

```
Storage storage = new Storage() {
    private String data = "";
    @Override
    public void store(String data) {
        this.data += data;
        System.out.println("Data stored on storage");
    }
    @Override
    public String read() {
        return this.data;
    }
};
```

2. Создайте абстрактный класс **Gift**, объявив для него абстрактные методы:

- **buy()** – без аргументов и возвращаемого значения, выводи на экран информацию, о том, что подарок куплен;
- **give()** – принимает на вход имя получателя в виде строки, без возвращаемого значения.

Создайте следующие классы, унаследованные от **Gift**:

- **Postcard** – класс реализует абстрактные методы класса **Gift**, дополнительно к которым имеет строковое поле персонального пожелания **wish**, метод **writeWish()**, принимающий на вход текст поздравления, метод **getWish()**, возвращающий значение пожеланий (**wish**). Реализованный метод **give** выводит на экран пожелание с подставленным в него именем получателя.
- **Painting** – класс реализует абстрактные методы класса **Gift**, дополнительно к которым имеет строковые поля **title** и **author** и геттеры и сеттеры для данных полей.

Создайте запускаемый класс **GiftSharing**, в котором реализуйте проверку всех методов.

3. Создайте файл **StructureUnit.java**, описывающий интерфейс классов, описывающих структурное подразделение университета:

```
public interface StructureUnit {
    String university = "ITMO University";
    // нанять сотрудника
    void hireEmployee(String name);
    // уволить сотрудника
    void fireEmployee(String name);
    String getInfo();
}
```

4. Создайте файл **Department.java**, реализующий интерфейс **StructureUnit**:

```
import java.util.ArrayList;
import java.util.Arrays;
public class Department implements StructureUnit {
    private String title;
    private ArrayList<String> employees; // множество
сотрудников
    public Department() { this.employees = new
ArrayList<>(); }
    public void setTitle(String title) { this.title =
title; }
```

```
    // нанять сотрудника
    public void hireEmployee(String name){
this.employees.add(name); }
    // уволить сотрудника
    public void fireEmployee(String name){
this.employees.remove(name); }
    // получение информации о структурной единице
    public String getInfo(){
        int num = this.employees.size();
        return "Title: " + this.title + " | Number of
employees: " + num + " | Employees: " +
Arrays.toString(this.employees.toArray());
    }
}
```

5. Создайте файл **Faculty.java**, реализующий интерфейс **StructureUnit**:

```
import java.util.ArrayList;
import java.util.Arrays;
public class Faculty implements StructureUnit {
    String title;
    private ArrayList<Department> departments;
    private ArrayList<String> employees; // множество
сотрудников
    public Faculty() {
        this.employees = new ArrayList<>();
        this.departments = new ArrayList<>();
    }
    public void setTitle(String title) { this.title =
title; }
    // нанять сотрудника
    public void hireEmployee(String name){
        this.employees.add(name);
    }
    // уволить сотрудника
    public void fireEmployee(String name){
        this.employees.remove(name);
    }
    // добавить вложенную структ. единицу
    public void addDepartment(Department department)
{
        this.departments.add(department);
```

```
    }
    // получение информации о структурной единице
    public String getInfo() {
        int num = this.employees.size();
        String info =  "Title: " + this.title + " |
Number of employees: " + num + " | Employees: " +
Arrays.toString(this.employees.toArray());
        info += "\n Info from departments: \n";
        for (Department d : departments) info +=
d.getInfo() + "\n";
        return info;
    }
}
```

6. Создайте файл **InterfaceExample.java**:

```
public class InterfaceExample {
    public static void main(String[] args) {
        Department department = new Department();
        department.setTitle("Department of CET");
        department.hireEmployee("John");
        System.out.println(department.getInfo());
        Faculty faculty = new Faculty();
        faculty.setTitle("Faculty of SECS");
        faculty.addDepartment(department);
        faculty.hireEmployee("Jack");
        System.out.println(faculty.getInfo());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Title: Department of CET | Number of employees: 1 |
Employees: [John]
Title: Faculty of SECS | Number of employees: 1 |
Employees: [Jack]
 Info from departments:
Title: Department of CET | Number of employees: 1 |
Employees: [John]
```

Ключевое слово `final`

При использовании ключевого слова **`final`** с полями класса и переменными:

- инициализация значения переменной происходит один раз;
- ссылка на переменную, объявленную с помощью **`final`**, не может быть изменена;
- при использовании совместно с модификатором **`static`** позволяет определить константу для класса, например: **`static final double pi = 3.14`**.

Использование ключевого слова **`final`** с методами класса запрещает наследование этого метода любым классом, унаследованным от текущего. Объявление метода с модификатором **`final`**:

```
public final void f() { . . .}
```

Создание класса как **`final`** позволяет запретить наследование любых методов данного класса. Объявление класса с модификатором **`final`**:

```
public final class Finalist { . . .}
```

Интерфейсы

Интерфейсы в Java представляют собой полностью абстрактные классы. Все методы интерфейсов по умолчанию являются абстрактными методами и не требуют наличия модификатора **`abstract`** при их объявлении. Объявление интерфейса происходит при помощи ключевого слова **`interface`** следующим образом:

```
interface Food {
    Boolean eatable = true; // static, final
    void eat();
}
```

В приведенном примере создается интерфейс **`Food`** с (абстрактным) методом **`eat()`** и полем **`eatable`**. Все поля интерфейса по умолчанию являются **`static`** и **`final`**. Использование модификатора **`public`** перед ключевым словом **`interface`** позволяет сделать интерфейс общедоступным, иначе, по умолчанию, он будет виден исключительно внутри текущего пакета.

Реализация методов, указанных в интерфейсе, осуществляется в классах,

реализующих данный интерфейс, объявленных с использованием ключевого слова **implements**. Методы интерфейса, реализованные в классе в обязательном порядке должны быть объявлены с модификатором **public**.

```
public class Tomato implements Food {
    public void eat(){
        System.out.println("Tomato eaten");
    }
}
```

Таким образом, интерфейс определяет форму тех классов, которые будут реализовывать этот интерфейс. Основными отличиями интерфейсов от абстрактных классов являются:

- отсутствие неабстрактных методов;
- все поля интерфейса являются **static final**;
- все методы интерфейса по умолчанию **public** и должны оставаться **public** при реализации;
- интерфейс может расширять другой интерфейс (в том числе базовые интерфейсы Java), но не может быть унаследованным от класса;
- для интерфейса не могут быть созданы никакие объекты данного типа, объекты могут быть созданы исключительно для классов, реализующих интерфейс.

7. Логически разделите устройства из следующего списка: центральный процессор, основная память, шина, клавиатура, мышь, принтер, жесткий диск, монитор, на смысловые категории, для каждой из которых создайте свой интерфейс с основными методами, необходимыми для реализации. В каждом из реализующих классов, созданных в соответствии с указанным перечнем устройств компьютера, создайте по одному дополнительному методу, специфичному для конкретного устройства. В отдельно созданном запускаемом классе осуществите проверку всех классов и их методов.

8. Создайте файл **DocumentFactory.java**, описывающий интерфейс фабрики документов:

```
public interface DocumentFactory {
    Document getDocument();
}
```

9. Создайте файл **Document.java**, описывающий интерфейс отдельного документа:

```
public interface Document {
    void sign(String name);
    String getSignatures();
    void printInfo();
}
```

10. Создайте файл **Regulation.java**, описывающий класс документа-положения:

```
public class Regulation implements Document{
    private String signatures = "";
    @Override
    public void sign(String name) {
        if (this.signatures.length()>0)
            this.signatures += ", ";
        this.signatures += name;
    }
    @Override
    public String getSignatures() { return
this.signatures; }
    @Override
    public void printInfo() {
        System.out.println("Regulation info: \n" +
                           "Title: Regulation \n" +
                           "Signed by: " +
this.signatures);
    }
}
```

11. Создайте файл **RegulationFactory.java**, описывающий класс фабрики документов-положений:

```
public class RegulationFactory implements
DocumentFactory {
    @Override
    public Document getDocument() {
        return new Regulation();
```

```
    }
}
```

12. Создайте файл **Contract.java**, описывающий класс документа-договора:

```
public class Contract implements Document{
    private String signatures = "";
    @Override
    public void sign(String name) {
        if (this.signatures.length()>0)
            this.signatures += ", ";
        this.signatures += name;
    }
    @Override
    public String getSignatures() { return
this.signatures; }
    @Override
    public void printInfo() {
        System.out.println("Contract info: \n" +
                " Title: Contract \n" +
                " Signed by: " + this.signatures);
    }
}
```

13. Создайте файл **ContractFactory.java**, описывающий класс фабрики документов-договоров:

```
public class ContractFactory implements
DocumentFactory {
    @Override
    public Document getDocument() {
        return new Contract();
    }
}
```

14. Создайте файл **FactoriesExample.java**:

```
public class FactoriesExample {
    public static void
workWithDocument(DocumentFactory dF){
        Document doc = dF.getDocument();
        doc.sign("Paul");
```

```
        doc.sign("Jack");
        System.out.println(doc.getSignatures());
        doc.printInfo();
    }
    public static void main(String[] args){
        workWithDocument(new RegulationFactory());
        System.out.println();
        workWithDocument(new ContractFactory());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Paul, Jack
Regulation info:
 Title: Regulation
 Signed by: Paul, Jack

Paul, Jack
Contract info:
 Title: Contract
 Signed by: Paul, Jack
```

Множественная реализация интерфейсов

Разработка с использованием фабрик (множественных реализаций интерфейсов) позволяет программисту не работать напрямую с определенным типом объекта, а осуществлять создание объекта с использованием нужной ему фабрики, определяющей тип автоматически.

Фабрики позволяют:

- разработать более защищенное от случайных ошибок приложение;
- разработать фреймворки, для которых характерно наличие схожих методов (в частности это может быть использовано при программировании игр: например, шахматы и шашки, для которых игра ведется похожим образом на одном и том же поле), или же разработке систем анализа различных документов и т.д.

15. С использованием описанного подхода к проектированию приложений с помощью множественной реализацией интерфейсов реализуйте систему, осуществляющую перевод документов с различных

языков на русский с использованием отдельных словарей для каждого из языков. Ограничиться тремя различными языками.
Словарь для каждого из языков представляет собой реализацию общего интерфейса словарей с методами поиска отдельного слова, переданного в качестве аргумента (возвращает логическое значение **`true`** и выводит на экран информацию, о том, что слово было найдено в конкретном словаре).

Лабораторная работа № 9: Контейнеры

1. Создайте файл **ListExample.java**, демонстрирующий работу с контейнерами типа **List**:

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
public class ListExample {
    /**
     * Returns elements for the list of Strings that
have certain length
     *
     * Method is executed in {@link #main(String[])}
     *
     * @param len - maximum length of String element
     * @param list - List of values
     * @return list of elements, that have length
less than len
     *
     */
    public static List getListElementsUnder(int len,
List<String> list) {
        List<String> result = new ArrayList<>();
        for (int i = 0; i < list.size(); i++) {

if(list.get(i).length()<len)result.add(list.get(i));
        }
        return result;
    }
    public static void main(String[] args){
        List<String> arrayListStr = new
ArrayList<>();
        arrayListStr.add("Hello");
        arrayListStr.add("Student");

List<String>result=getListElementsUnder(6,arrayListSt
r);
        System.out.println(result);
        List<String> linkedListStr = new
LinkedList<>();
        linkedListStr.add("Math");
        linkedListStr.add("IT");
```

```
        linkedListStr.add("Physics");
        linkedListStr.add("PE");
        List<String> res =
getListElementsUnder(6,linkedListStr);
        System.out.println(res);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
[Hello]
[Math, IT, PE]
```

Список (`List`)

`List` представляет собой интерфейс контейнера с индексируемым доступом к его элементам. В отличие от обычных массивов, он является динамическим. Основными методами контейнеров типа **`List`** являются:

- **`add(Object obj)`** – осуществляет добавление объекта **`obj`**, переданного в качестве аргумента, в конец контейнера;
- **`add(int index, Object obj)`** – осуществляет добавление объекта **`obj`**, переданного в качестве аргумента, в контейнер на позицию **`index`**;
- **`addAll(Collection collection)`** – осуществляет добавление элементов коллекции **`collection`** в конец списка;
- **`addAll(int index, Collection collection)`** – осуществляет добавление элементов коллекции **`collection`** в список, начиная с позиции **`index`**;
- **`get(index)`** – получение элемента с позиции **`index`**;
- **`size()`** – получение числа элементов контейнера;
- **`remove(Object obj)`** – удаление объекта **`obj`** из контейнера;
- **`remove(int index)`** – удаление элемента с позиции **`index`**;
- **`removeAll(Collection collection)`** – удаление всех элементов коллекции **`collection`** из контейнера;
- **`contains(Object obj)`** – проверка наличия элемента **`obj`** в контейнере.

Вызов методов происходит напрямую от объекта-списка с указанием имени метода через точку после имени объекта. С остальными методами

можно ознакомиться в документации по ссылке:
https://docs.oracle.com/javase/7/docs/api/java/util/List.html

К основным контейнерам типа **List** относятся:

- **ArrayList** – данный тип контейнеров отличается быстротой произвольного доступа к элементам, однако добавление и удаление элементов из середины списка происходит с относительно небольшой скоростью.
- **LinkedList** – данный тип контейнеров характеризуется высокой скоростью вставки и удаления элементов, в том числе и из середины списка, однако скорость произвольного доступа к элементам в **LinkedList** невысокая. Может использоваться для реализации стека.

Объявление объектов типов **List**, **ArrayList** и **LinkedList** может происходить следующим образом:

```
List list = new ArrayList(); // может содержать элементы любых типов
List<String> list1 = new ArrayList<>(); // содержит строки
LinkedList<Integer> list2 = new LinkedList<>(); // содержит целые числа
```

Типизированные контейнеры

Типизированные контейнеры (Generics) – контейнеры некоторого неопределенного типа **<T>**, позволяющие применять обозначенные в них методы к различным типам данных. При таком подходе к разработке для обеспечения безопасности рекомендуется указывать тип значений, которые могут храниться в контейнере. Объявление таких контейнеров происходит следующим образом:

```
ArrayList<String> list1 = new ArrayList<>();
```

Тип значений контейнера указывается в треугольных скобках **<>** после типа контейнера.

Документация Java (JavaDoc)

Большинство сред разработки поддерживают автоматическое генерирование HTML-документации на основе оформленной документации к вашему Java-коду. Документации к коду, написанному на языке Java носит специальное название – JavaDoc – и оформляется как комментарии перед именем метода, класса и т.д. следующим образом:

```
    /**
     * Returns elements for the list of Strings that
have certain length
     *
     * Method is executed in {@link #main(String[])}
     *
     * @param len - maximum length of String element
     * @param list - List of values
     * @return list of elements, that have length
less than len
     *
     */
     public static List getListElementsUnder(int len,
List<String> list) { . . . }
```

В отличие от обычных комментариев, комментарии, содержащие документацию, начинаются с символов **/** ... */**. Среда разработки автоматически проставляет для таких комментариев символы * слева от каждой строки. Первые несколько строк документации метода (класса, …) несут в себе общую информацию о предназначении данного метода (класса, …). Далее перечисляются основные характеристики методы в соответствии с шаблоном ключевых аннотаций (тегов), указанных в таблице:

Тег и формат	Описание	Применение
@author <имя>	Описание автора	Class, Interface, Enum
@version <версия>	Информация о версии ПО. Должен встречаться один раз на класс, интерфейс и т.д.	Class, Interface, Enum
@since <дата>	Дата первого внедрения функционала	Class, Interface, Enum, Field, Method

@see <ссылка>	Ссылка на другой элемент документации	Class, Interface, Enum, Field, Method
@param <имя> <описание>	Параметр некого метода с указанием его имени и краткого описания	Method
@return <описание>	Возвращаемое значение	Method
@exception <имя класса> <описание> @throws <имя класса> <описание>	Исключение, которое может возникнуть в некотором методе	Method
@deprecated <описание>	Использование данного функционала запрещено	Class, Interface, Enum, Field, Method
{@inheritDoc}	Копия описания из метода родительского класса	Overriding Method
{@link <ссылка>}	Ссылка	Class, Interface, Enum, Field, Method
{@value #<static-поле>}	Значение переменной	Static Field
{@code <литерал>}	Форматирует литерал в HTML-тег **<code> {@literal} </code>**	Class, Interface, Enum, Field, Method
{@literal <литерал>}	Буквенный текст, не содержащий HTML-элементов	Class, Interface, Enum, Field, Method

После генерирования подготовленной документации для классов, методов и т.д., будет получена веб-страница с документацией.

2. Создайте класс **Device** с полями, хранящими значение идентификатора и емкости устройства. Добавьте метод **getInfo()**, возвращающий информацию о полях объекта в виде строки.

3. Создайте класс **ListCheck**, в котором определите методы:

- **printList(..)**, принимающий в качестве аргумента контейнер **List<Device>** и осуществляющий вывод информации о содержащихся в контейнере объектах;
- **checkElement(..)**, принимающий в качестве аргумента контейнер **List<Device>**, элемент типа **Device** и некоторое целое число **number** и осуществляющий проверку кратности идентификатора элемента **Device** переданному числу **number**. Перед проверкой на кратность необходимо удостоверится, что элемент **Device** присутствует в контейнере, если элемент отсутствует, на экран необходимо вывести сообщение об ошибке.

Добавьте в класс запускаемый метод, в котором заполните 50 значениями два контейнера типов **ArrayList** и **LinkedList** соответственно. Осуществите проверку реализованных методов на созданных контейнерах.

4. Дополните класс **ListCheck**, добавив документацию в формате JavaDoc для методов **printList()** и **checkElement()**, указав краткое описание метода, его аргументы и возвращаемое значение.

5. Создайте файл **IteratorExample.java**, демонстрирующий работу с итераторами:

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
public class IteratorExample {
    public static void main(String[] args){
        List<String> linkedListStr = new
LinkedList<>();
        linkedListStr.add("Math");
        linkedListStr.add("IT");
        linkedListStr.add("Physics");
        // пример итерации с использованием итератора
        Iterator iterator = linkedListStr.iterator();
        System.out.println("Before: " +
linkedListStr);
        while(iterator.hasNext()){
            String elem = iterator.next().toString();
            if (elem.length() >= 6) {
                iterator.remove(); // удаление
элемента
            }
```

```
        }
        System.out.println("After: " +
linkedListStr);
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Before: [Math, IT, Physics]
After: [Math, IT]
```

Итератор

Для итерируемых контейнеров (коллекций) предусмотрены различные варианты итераций по объектам контейнера:

1. Для индексируемых контейнеров проход по контейнеру с использованием индексом элементов.
2. С использованием цикла с итерацией по объектам контейнера без индексирования.
3. С помощью итераторов.

Итератор – это специальный объект, позволяющий пройти по контейнеру, получить значение элемента, удалить элемент и т.д. Для использования данного способа итерации необходимо определить объект-итератор для конкретного контейнера, для которого станут доступны методы, позволяющие получить текущее значение контейнера, до которого доходит итератор:

- **`hasNext()`** – проверка наличия следующего элемента в контейнере (конца длины контейнера);
- **`next()`** – получение следующего значения итератора (контейнера);
- **`remove()`** – удаляет значение контейнера, полученное при последнем вызове метода **`next()`.**

Вызов данных методов осуществляется напрямую от объекта-итератора с указанием имени метода через точку после имени объекта. Получение итератора для контейнера осуществляется с помощью метода **`iterator()`**, вызываемого от объекта-контейнера.

Пример программы с использованием итератора:

```
. . .
Iterator iterator = list.iterator();
System.out.println("Before: " + linkedListStr);
while(iterator.hasNext()){    //    проверка    наличия
значений
    String  elem  =  iterator.next().toString();  //
получение значения
    if (elem.length() >= 6) {
        iterator.remove(); // удаление элемента
    }
}
```

6. Дополните класс **ListCheck**, определив в нем метод **printListWithIterator(..)**, принимающий в качестве аргумента контейнер **List<Device>** и выводящий информацию о каждом элементе контейнера на экран с использованием итератора. Осуществите проверку работы метода на созданных ранее контейнерах в главном методе класса.

7. Создайте файл **SetExample.java**, демонстрирующий работу с контейнерами типа **Set**:

```
import java.util.*; // импорт всех библиотек пакета
java.util
public class SetExample {
    public static void main(String[] args) {
        Set<Integer> hashSet = new HashSet<>();
        Set<Integer> treeSet = new TreeSet<>();
        Set<Integer> linkedHashSet = new
LinkedHashSet<>();
        Random random = new Random();
        for (int i=0; i< 20; i++) {
            int val = random.nextInt(50);
            hashSet.add(val);
            treeSet.add(val);
            linkedHashSet.add(val);
        }
        // вывод контейнеров
        System.out.println(hashSet);
        System.out.println(treeSet);
        System.out.println(linkedHashSet);
        // проверка наличия элемента
```

```
        System.out.println("Is 5 presented? " +
                hashSet.contains(5));
        // получение значений контейнера
        System.out.println("Let's compare first
elements: " +
                           hashSet.iterator().next() + "
| " +
                           treeSet.iterator().next() + "
| " +

linkedHashSet.iterator().next());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
[32, 0, 33, 2, 34, 35, 37, 5, 38, 8, 9, 11, 45, 14,
17, 23, 24, 27, 29]
[0, 2, 5, 8, 9, 11, 14, 17, 23, 24, 27, 29, 32, 33,
34, 35, 37, 38, 45]
[27, 35, 2, 33, 9, 29, 14, 34, 37, 32, 23, 45, 38,
24, 17, 11, 5, 8, 0]
Is 5 presented? true
Let's compare first elements: 32 | 0 | 27
```

Множества (`Set`)

Множества (**`Set`**) представляют собой контейнеры, в которых каждый элемент встречается ровно один раз. Основными методами контейнеров типа **`Set`** являются:

- **`add(Object obj)`** – осуществляет добавление объекта **`obj`**, переданного в качестве аргумента, в конец контейнера, если он отсутствует в контейнере;
- **`addAll(Collection collection)`** – осуществляет добавление элементов коллекции **`collection`** в конец контейнера, если они отсутствуют в контейнере;
- **`retainAll(Collection collection)`** – удаляет из контейнера элементы, не содержащиеся в коллекции **`collection`**;
- **`remove(Object obj)`** – удаление объекта **`obj`** из контейнера;
- **`removeAll(Collection collection)`** – удаление всех

элементов коллекции **collection** из контейнера;

- **size()** – получение числа элементов контейнера;
- **contains(Object obj)** – проверка наличия элемента **obj** в контейнере.

Тип **Collection** представляет группу элементов, например, список, множество и т.д.
Вызов методов происходит напрямую от объекта-множества с указанием имени метода через точку после имени объекта. С остальными методами можно ознакомиться в документации по ссылке:
https://docs.oracle.com/javase/7/docs/api/java/util/Set.html

Получение значений элементов множества возможно с использованием итератором для созданного контейнера.
К основным контейнерам типа **Set** относятся:

- **HashSet** – использует хеширование (преобразование входных данных в выходную строку в бинарном виде) для ускорения выборки элементов, вследствие чего последовательность вывода элементов при выборке незакономерна;
- **TreeSet** – осуществляет быструю сортировку элементов при выборке;
- **LinkedHashSet** – расширяет класс **HashSet**, позволяя получить значения контейнера в том порядке, в котором они добавлялись в контейнер.

8. Создайте класс **SetCheck**, определив в нем следующие методы:
   - **setFromString(..)** – принимает на вход строку, из которой необходимо составить контейнер, выделив в ней отдельные слова, и контейнер типа **Set**, который будет заполнен словами;
   - **printSet(..)**, принимающий в качестве аргумента контейнер типа **Set** и осуществляющий поэлементный вывод информации об объектах контейнера с использованием итератора;
   - **union(..)** – принимает на вход два множества (контейнера типа **Set**), осуществляет операцию объединения множеств с использованием стандартного метода контейнера типа **Set** и возвращает новое множество, полученное за счет объединения множеств;
   - **intersection(..)** – принимает на вход два множества (контейнера типа **Set**), осуществляет операцию пересечения

множеств с использованием стандартного метода контейнера типа **Set** и возвращает новое множество, полученное за счет пересечения множеств;

- **subtraction(..)** – принимает на вход два множества (контейнера типа **Set**), осуществляет операцию разности множеств с использованием стандартного метода контейнера типа **Set** и возвращает новое множество, полученное за счет разности множеств.

Для реализации операций **union**, **intersection** и **subtraction** используйте стандартные методы контейнера **Set**. Создайте для класса главный запускаемый метод, в котором осуществите проверку всех методов на контейнерах типов **HashSet<String>**, **TreeSet<String>** и **LinkedHashSet<String>**, созданных из указанной строки:

```
A computer's memory can be viewed as a list of cells into which
numbers can be placed or read. Each cell has a numbered address
and can store a single number. The computer can be instructed to
put the number 123 into the cell numbered 1357 or to add the
number that is in cell 1357 to the number that is in cell 2468
and put the answer into cell 1595.
```

9. Создайте файл **MapExample.java**, демонстрирующий работу с контейнерами типа **Map**:

```
import java.util.*;
public class MapExample {
    public static void main(String[] args) {
        Map<String,Integer> hashMap = new
HashMap<>();
        Map<String,Integer> treeMap = new
TreeMap<>();
        Map<String,Integer>linkedHashMap=new
LinkedHashMap<>();
        Random random = new Random();
        for (int i=0; i<20; i++) {
            int val = random.nextInt(50);
            hashMap.put("k" + i,val);
            treeMap.put("k" + i,val);
            linkedHashMap.put("k" + i,val);
        }
        // вывод контейнеров
        System.out.println(hashMap);
        System.out.println(treeMap);
```

```
        System.out.println(linkedHashMap);
        // получение значений по ключу
        System.out.println("Let's compare first
elements: " +
                hashMap.get("k0") + " | " +
                treeMap.get("k0") + " | " +
                linkedHashMap.get("k0"));
        // проверка наличия ключа
        System.out.println("Is key k30 presented? " +

hashMap.containsKey("k30"));
        // проверка наличия значения
        System.out.println("Is value 23 presented? "
+

treeMap.containsValue(23));
        // получить множество ключей контейнера
        System.out.println("Keys: " +
linkedHashMap.keySet());
        // получить множество значений контейнера
        System.out.println("Values: " +
hashMap.values());
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
{k0=9, k1=14, k2=44, k3=20, k4=36, k5=28, k11=24,
k6=4, k10=27, k7=6, k13=46, k8=27, k12=7, k9=46,
k15=29, k14=32, k17=12, k16=23, k19=27, k18=42}
{k0=9, k1=14, k10=27, k11=24, k12=7, k13=46, k14=32,
k15=29, k16=23, k17=12, k18=42, k19=27, k2=44, k3=20,
k4=36, k5=28, k6=4, k7=6, k8=27, k9=46}
{k0=9, k1=14, k2=44, k3=20, k4=36, k5=28, k6=4, k7=6,
k8=27, k9=46, k10=27, k11=24, k12=7, k13=46, k14=32,
k15=29, k16=23, k17=12, k18=42, k19=27}
Let's compare first elements: 9 | 9 | 9
Is key k30 presented? false
Is value 23 presented? true
Keys: [k0, k1, k2, k3, k4, k5, k6, k7, k8, k9, k10,
k11, k12, k13, k14, k15, k16, k17, k18, k19]
```

```
Values: [9, 14, 44, 20, 36, 28, 24, 4, 27, 6, 46, 27,
7, 46, 29, 32, 12, 23, 27, 42]
```

Карта (Map)

Карты (**Map**) представляют собой контейнеры с парами ключ-значения, также называемыми ассоциативными массивами. Основными методами контейнеров типа `Map` являются:

- **put(K key, V value)** – вставка пары ключ-значение (**key-value**) в контейнер;
- **entrySet()** – множество пар ключ-значение (тип **Map.Entry**) для контейнера;
- **get(Object key)** – получение значения по ключу **key**;
- **containsKey(Object key)** – проверка наличия ключа **key** в контейнере;
- **containsValue(Object value)** – проверка наличия значения **value** в контейнере;
- **remove(Object key)** – удаление пары ключ-значение по ключу **key**;
- **keySet()** – получение множества ключей контейнера;
- **values()** – получение коллекции значений контейнера;
- **size()** – получение числа элементов списка.

Вызов методов происходит напрямую от объекта-множества с указанием имени метода через точку после имени объекта. С остальными методами можно ознакомиться в документации по ссылке:
https://docs.oracle.com/javase/7/docs/api/java/util/Map.html

К основным контейнерам типа **Map** относятся:

- **HashMap** – использует хеширование для ускорения выборки элементов, вследствие чего последовательность вывода элементов при выборке незакономерна;
- **TreeMap** – осуществляет быструю сортировку пар ключ-значение по ключам при выборке;
- **LinkedHashMap** – расширяет класс **HashMap**, позволяя получить значения контейнера в том порядке, в котором они добавлялись в контейнер.

10. Создайте класс **Person**, со следующими полями (все имеют модификатор доступа **private**):

- **String lastname** – фамилия;
- **String firstname** – имя;
- **String middlename** – отчество;
- **Calendar birthday** – дата рождения.

Со спецификацией типа **Calendar** можно ознакомиться по ссылке: **http://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html**

Добавьте классу геттеры и сеттеры для каждого поля, а также метод **getFIO()**, возвращающий строковое значение информации о человеке в формате "**Фамилия И.О.**".

11. Создайте класс **MapCheck**, определив в нем следующие методы:

- **setMap(..)** – принимает на вход контейнер типа **List<Person>** и контейнер типа **Map** и осуществляет заполнение контейнера типа **Map** по следующему принципу: в качестве ключей выступают года рождения людей, в качестве значений – списки людей (**ArrayList<Person>**) с годом рождения, указанным в качестве ключа; метод возвращает заполненный контейнер типа **Map**;
- **compareMaps(..)** – принимает на вход три контейнера типов **HashMap<Integer, ArrayList<Person>>**, **TreeMap<Integer, ArrayList<Person>>** и **LinkedHashMap<Integer,ArrayList<Person>>** и осуществляет поэлементный вывод информации о парах ключ-значение, где значение выводится с помощью метода **getFIO()** класса **Person** в следующем виде, используя разделители, располагающиеся друг под другом (в виде таблицы):

```
HashMap   | TreeMap   | LinkedHashMap
Fam I.O.  | Fam I.O.  | Fam I.O.
```

Создайте для класса главный запускаемый метод, в котором осуществите проверку всех методов на контейнерах типов **HashMap<Integer, ArrayList<Person>>**, **TreeMap<Integer, ArrayList<Person>>** и **LinkedHashMap<Integer, ArrayList<Person>>**, созданных с использованием метода **setMap()** для списка из 9 разных объектов класса **Person** (по три объекта на каждый год рождения).

12. Создайте класс **Research**, в котором реализуйте проведение анализа эффективности работы методов по нижеуказанной программе исследований для различного количества элементов в контейнерах, равного **10**, **100**, **1000** и **10000**. Показателем эффективности является время выполнения метода. По каждому из указанных методов для каждого из контейнеров (8 представленных в программе исследований) необходимо выводить информацию в следующем виде:

<количество_элементов> elements (<тип_контейнера>): <метод_и_его_параметры>: <время_в миллисекундах> мс

Группируйте вывод по количеству элементов, оставляя пустую строку между группами.

Все контейнеры содержат внутри себя объекты следующего типа:

```
class Sample .. {
    private int id;
    /** Переопределяет метод Object.toString()
    * Возвращает информацию об объекте в формате id:..,
    * где вместо многоточия подставляется уникальный id
    * текущего объекта, в строковом виде.
    */
    public String toString(){
        . . .
    }
}
```

Для корректной работы сортировки в контейнерах типов **TreeSet** и **TreeMap** необходимо классом **Sample** реализовать интерфейс **Comparable** и реализовать метод **compareTo(..)** с целью проведения сравнения при сортировке по идентификатору объекта.

При заполнении контейнеров ключ для **Map** – "k..", где вместо многоточия указывается номер текущего элемента (пары ключ-значение), например k20.

Программа исследования:

Для контейнеров типов **ArrayList**, **LinkedList**:

- вставка элемента в конец списка;
- вставка элемента в середину списка;
- получение элемента с позиции в начале списка;
- получение элемента с позиции в середине списка;

- получение элемента с позиции в конце списка;
- проверка наличия элемента в списке;
- вывода содержимого контейнера на экран;
- удаление элемента (по значению) из списка;
- удаление коллекции элементов из списка (на примере использования **`Set`**);
- удаление элемента с позиции в начале списка;
- удаление элемента с позиции в середине списка;
- удаление элемента с позиции в конце списка.

Для контейнеров типов **`HashSet`**, **`TreeSet`**, **`LinkedHashSet`**:

- добавление элемента в множество;
- добавление значений коллекции в множество (на примере включения **`List`** в **`Set`**);
- проверка наличия элемента в множестве;
- удаление элемента (по значению) из множества;
- удаление коллекции элементов из списка (на примере использования **`List`**);
- вывода содержимого контейнера на экран.

Для контейнеров типов **`HashMap`**, **`TreeMap`**, **`LinkedHashMap`**:

- вставка пары ключ-значение;
- получение множества пар ключ-значение;
- получение значения по ключу;
- проверка наличия ключа в контейнере;
- проверка наличия значения в контейнере;
- удаление пары ключ-значение по ключу;
- получение множества ключей контейнера;
- получение множества значений контейнера.

Лабораторная работа № 10: Работа с файлами и многопоточность

1. Создайте файл **FilesExample.java**, демонстрирующий работу с файлами:

```
import java.io.*;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import                                                  static
java.nio.file.StandardOpenOption.APPEND;

public class FilesExample {

    private static void inputStreamCopy(String path,
String fileInName, String fileOutName){
        FileInputStream in;
        FileOutputStream out;
        // побайтовые чтение и запись
        try {
            in    =    new    FileInputStream(path    +
fileInName);
            out    =    new    FileOutputStream(path    +
fileOutName);
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
            System.out.println("File    copied    with
FileInputStream");
        }
        catch(IOException e){
            System.out.println("Exception caught:");
            // вывод полной информации об исключении
            e.printStackTrace();
        }
    }

    private   static   void   bufferedReaderCopy(String
path, String fileInName, String fileOutName) {
```

```
        // построчные чтение и запись
        try       (BufferedReader       reader       =
Files.newBufferedReader(Paths.get(path + fileInName),

Charset.defaultCharset())) {
            String line = null;
            try      (BufferedWriter      writer      =
Files.newBufferedWriter(Paths.get(path                +
fileOutName),

Charset.defaultCharset(), APPEND)) {
                while  ((line  =  reader.readLine())  !=
null) {
                    writer.write(line,              0,
line.length());
                    writer.write('\n');
                }
                System.out.println("File  copied  with
BufferedReader");
            } catch (IOException e) {
                System.out.println("Exception   caught
(write):");
                System.err.format("IOException:
%s%n", e);
            }
        } catch (IOException e) {
            System.out.println("Exception       caught
(read):");
            System.err.format("IOException:     %s%n",
e);
        }
    }

    public static void main(String[] args){
        String  path  =  System.getProperty("user.dir")
+ "/src/";
        BufferedReader inputBuf;
        List<String> students = new ArrayList<>();
        try {
            inputBuf    =    new    BufferedReader(new
InputStreamReader(System.in));   //   для   ввода   с
клавиатуры
            System.out.println("Enter   student   names
('0' - exit): ");
```

```
            while (true) {
                // ввод с клавиатуры
                String input = inputBuf.readLine();
                if ("0".equals(input)) break;
                else students.add(input);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(students);
        inputStreamCopy(path,
"fileIn.txt","fileOut_01.txt");
        bufferedReaderCopy(path,
"fileIn.txt","fileOut_02.txt");
    }
}
```

Также создайте пустые файлы **fileOut_01.txt** и **fileOut_02.txt** и файл **fileIn.txt** в папке **src** со следующим содержимым:

```
Information - knowledge that you get about someone or something:
facts or details about a subject.
Information - knowledge obtained from investigation, study, or
instruction.
```

Запустите программу и удостоверьтесь в ее работоспособности.
Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Enter student names ('0' - exit):
Student1
Student2
0
[Student1, Student2]
File copied with FileInputStream
File copied with BufferedReader
```

Содержимое файла **fileOut_01.txt**:

```
Information - knowledge that you get about someone or something:
facts or details about a subject.
Information - knowledge obtained from investigation, study, or
instruction.
```

Содержимое файла **fileOut_02.txt**:

```
Information - knowledge that you get about someone or something:
facts or details about a subject.
Information - knowledge obtained from investigation, study, or
instruction.
```

Работа с файлами

В Java существует несколько различных способов работы с файлами, основными из которых является использование **`FileInputStream`** и **`BufferedReader`**. Рассмотрим различные опции открытия файлов, хранящиеся в классе **`java.nio.file.StandardOpenOption`**, которые указываются при создании объектов работы с файлом:

- **`WRITE`** – запись в файл;
- **`APPEND`** – дополнение файла;
- **`TRUNCATE_EXISTING`** – удаление предыдущего содержимого;
- **`CREATE_NEW`** – создание нового файла, если файл с таким именем существует, то выдается исключение;
- **`CREATE`** – открытие файла, если он существует, или создание нового файла;
- **`DELETE_ON_CLOSE`** – удаление файла после закрытия (например, для временных файлов).

`FileInputStream` и `FileOutputStream`

Данные классы предназначены для побайтового считывания и записи в файл. **`FileInputStream`** служит для считывания данных из файла. Доступ к файлу устанавливается следующим образом:
`FileInputStream(File file)` – в качестве аргумента передается объект типа **`File`**.
`FileInputStream(String name)` – в качестве аргумента передается путь к файлу.

Основные функции:

- **`available()`** – проверка количества байт, доступных для считывания;
- **`read()`** – считывание байта из потока файла;
- **`close()`** – закрытие потока файла.

Для побайтового считывания данных из файла может быть использован объект класса **`FileOutputStream`**:
`FileOutputStream(. . .)` – в качестве аргумента передается объект

типа **File** или путь к файлу.
FileOutputStream(. . . , boolean append) – первый аргумент – объект типа `File` или путь к файлу, второй аргумент – флаг открытия файла на дополнение.

Основные функции:

- **write()** – запись байта в файл;
- **close()** – закрытие потока файла.

BufferedReader

BufferedReader используется совместно с **InputStreamReader** или **BufferedReader**, при этом отдельно для входного и выходного файлового потока создается свой объект класса **BufferedReader**. Для чтения данных из файла может быть использована следующая запись:
BufferedReader reader =
Files.newBufferedReader(Paths.get(name),
Charset.defaultCharset()), где:

- **Files** – класс, содержащий static-методы для работы с файлами и директориями.
- **newBufferedReader()** – метод класса **Files** для получения объекта типа **BufferedReader** для считывания данных. Аргументы - объекты типов **Path** (путь к файлу) и **Charset** (кодировка файла).
- **Paths.get(name)** – позволяет получить объект типа Path.
- **Charset.defaultCharset()** – получение стандартной кодировки в виде объекта типа **Charset**

Для считывания данных доступны следующие функции объекта типа **BufferedReader**:

- **read()** – считывание символа из файла.
- **readLine()** – считывание строки из файла.
- **skip()** – пропуск некоторого количества символов, принимает на вход количество символов для пропуска.
- **close()** – закрытие потока.

Для записи в файл используется следующий формат:
BufferedWriter writer =
Files.newBufferedWriter(Paths.get(name),
Charset.defaultCharset(), APPEND), где:

- **newBufferedWriter()** – метод класса **Files** для получения объекта типа **BufferedWriter** для записи данных. Аргументы - объекты типов **Path** (путь к файлу), **Charset** (кодировка файла) и опция открытия файла.

Основные функции **BufferedWriter**:

- **write(…)** – в зависимости от входных аргументов позволяет записать отдельный символ, массив символов или подстроку.
- **flush()** – запись всех данных, находившихся в буфере.
- **close()** – закрытие потока.

Обработка исключений (try-catch-finally)

Обработка исключений в Java осуществляется блоком **try-catch-finally**:

```
try{
    . . .
}
catch (<тип_исключения> <имя_переменной>) {
    . . .
}
. . .
catch (<тип_исключения> <имя_переменной>) {
    . . .
}
finally {
    . . .
}
```

В блоке **try** указываются операторы, при выполнении которых могут возникнуть исключения. Исключения указываются в блоке **catch** совместно с их типом. В Java Иерархия исключений представляется следующим образом:

- **Throwable**
  - **Error**
    - . . .
  - **Exception**
    - **RuntimeException**
    - . . .

Примеры исключений:

- **RuntimeException** – ошибки выполнения программы;
- **NullPointerException** – ошибка (не-)указания значения **null**;
- **ArrayIndexOutOfBoundsException** – выход за границы массива;
- **IOException** – ошибка доступа к файлу.

В блоке **finally** определяются операторы, которые должны быть выполнены в любом случае после выполнения блоков **try** и **catch**.

Если известно, что метод может выдать некоторое исключение, можно прописать операторы без **try-catch**, указав, что метод выбрасывает исключение с помощью ключевого слова **throws**:

```
public static void main(String[] args) throws
Throwable { . . . }
```

Для самостоятельного вызова исключения используется ключевое слово **throw**:

```
public static void main(String[] args) {
    throw new Error();
}
```

2. Создайте файл **osi.txt** со следующим содержимым (данные разделены точкой с запятой):

```
Level 7; application; example: HTTP
Level 6; presentation; example: JPEG
Level 5; session; example: RPC
Level 4; transport; example: TCP
Level 3; network; example: IPv6
Level 2; data link; example: IEEE 802.2
Level 1; physical; example: USB
```

3. Для созданного на предыдущем шаге файла **osi.txt** двумя способами (с использованием **FileInputStream-FileOutputStream** и **BufferedReader**) реализуйте считывание данных из файла **osi.txt** и запись обратно отсортированных данных (с 1 по 7 уровень модели OSI) в файл **osiReverse.txt**.

4. Создайте файл **Calculation.java**, демонстрирующий работу с реализацией интерфейса **Runnable** для работы с потоками:

```
public class Calculation implements Runnable {
    private int id;
    private double time;
    public Calculation(int id, double time) {
        this.id = id;
        this.time = time;
    }
    @Override
    public void run() {
        double passedTime = 0;
        while (true) {
            if (ThreadExample.currentCalculation ==
0) ThreadExample.currentCalculation = this.id;
            if (ThreadExample.currentCalculation ==
this.id) {
                passedTime += 0.5;
                System.out.print(".");
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            if (passedTime >= this.time) {
                ThreadExample.currentCalculation = 0;
                System.out.println("\nCalculation #"
+ this.id + " finished!");
                break;
            }
        }
    }
}
```

5. Создайте файл **ThreadExample.java**, демонстрирующий работу с потоками:

```
public class ThreadExample {
    volatile static int currentCalculation;
    public static void main(String[] args){
```

```
        Thread calc1 = new Thread(new
Calculation(1,3.5));
        calc1.start();
        Thread calc2 = new Thread(new
Calculation(2,5.5));
        calc2.start();
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
........
Calculation #1 finished!
..........
Calculation #2 finished!
```

Многопоточность

Многопоточность часто используется для ускорения процессов сложных вычислений, реализации взаимодействия пользователя с интерфейсом приложений и т.д. В Java многопоточность реализуется с использованием класса **Thread** и классов, реализующих интерфейс **Runnable**, предоставляющий метод работы с внешне запускаемыми фрагментами кода. Для классов, реализующих интерфейс **Runnable** обязательно должен быть определен метод **run()**, отвечающий за запуск потока.

При создании двух и более потоков может быть реализовано параллельное выполнение некоторых необходимых вычислений. Для этого два потока должны быть запущены один за другим. Класс **Thread** отвечает непосредственно за сами потоки. Таким образом, класс, отвечающий за выполнение определенных вычислений, должен либо наследовать класс **Thread**, либо реализовывать интерфейс **Runnable**.

Создание потоков осуществляется следующим образом:
Thread runner = new Thread(new Calculation(1,3.5));
Если некоторый класс **Calculation** реализует **Runnable**.

Или же:
Thread runner = new Calculation(1,3.5);
Если класс **Calculation** наследует **Thread**.

Запуск потока осуществляется с помощью метода **start()**. static-метод **Thread.sleep(..)** позволяет поставить выполнение программы на паузы на число миллисекунд, переданное в качестве аргумента метода.

Для контроля доступа нескольких потоков к одному и тому же ресурсу необходимо использовать модификатор **volatile**, разрешающий изменение ресурса нескольким потокам одновременно. Для запрета выполнения определенного метода, если он уже выполняется в текущий момент времени другим потоком используется ключевое слово **synchronized**:

```
public synchronized void f(){ . . . }
```

Блокировка объектов может быть реализована следующим образом:

```
public synchronized void f(){
    synchronized (this){
        . . .
     }
}
```

В скобках может быть указан любой объект, необходимый для ограничения к нему доступа.

6. Реализуйте анализ некоторого текстового документа (записанного в строку) двумя потоками. При этом, первый поток должен составить контейнер с частотами появления всех символов, содержащихся в тексте, и записать его в общедоступное поле типа **Map** главного запускаемого класса. После окончания вычислений, необходимо вывести информацию об окончании работы с документом. Второй поток ждет освобождения доступа к документу, пока первый поток не завершит работу, после чего он определяет три наиболее и три наименее встречающихся в тексте символа и выводит символ и частоту его появления в тексте на экран, после чего осуществляется вывод сообщения об окончании работы с документом. При реализации использовать **synchronized** и **volatile**. Документ должен содержать не менее 1000 символов.

7. Создайте файл **Server.java**, демонстрирующий работу с серверными сокетами:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Server extends Thread {
    private int port;
    public Server(int port) {
        this.port = port;
    }
    @Override
    public void run() {
        BufferedReader in = null;
        PrintWriter out= null;
        ServerSocket server = null;
        Socket fromClient = null;
        try {
            server = new ServerSocket(port);
            fromClient = server.accept();
            in = new BufferedReader(new
InputStreamReader(
                    fromClient.getInputStream()));
            out = new
PrintWriter(fromClient.getOutputStream(), true);
            String input;
            System.out.println("Client "
+in.readLine().split(":")[0]+ " connected");
            while ((input = in.readLine()) != null) {
                if (input.contains("exit")) {
                    System.out.println("Shutting
down...");
                    break;
                }
                else {
                    String[] msg = input.split(":");
                    out.println("received from " +
msg[0] + " : " + msg[1]);
                }

            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                out.close();
```

```
                in.close();
                fromClient.close();
                server.close();
            } catch (IOException ex) {}
        }
    }
}
```

8. Создайте файл **Client.java**, демонстрирующий работу с клиентскими сокетами:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class Client extends Thread {
    private String host;
    private int port;
    private int id;
    public Client(String host, int port, int id) {
        this.host = host;
        this.port = port;
        this.id = id;
    }
    private String formatMessage(String msg) {
        return "client#" + this.id + ":" + msg;
    }
    @Override
    public void run() {
        Socket fromServer = null;
        BufferedReader in = null;
        PrintWriter out = null;
        BufferedReader inputBuf = null;
        try {
            String clientMsg, serverMsg;
            System.out.println("Connecting to " +
host + ":" + port);
            fromServer = new Socket(host, port);
            in = new BufferedReader(new
InputStreamReader(fromServer.getInputStream()));
            out = new
PrintWriter(fromServer.getOutputStream(), true);
```

```
            inputBuf = new BufferedReader(new
InputStreamReader(System.in));
            // start message (to be sure client is
connected)
            out.println(formatMessage(""));
            // read message from console and send to
server
            while ((clientMsg = inputBuf.readLine())
!= null) {

out.println(formatMessage(clientMsg));
                serverMsg = in.readLine();
                System.out.println("Server:
"+serverMsg);
                if (clientMsg.contains("exit")) {
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                out.close();
                in.close();
                inputBuf.close();
                fromServer.close();
            } catch (IOException ex) {}
        }
    }
}
```

9. Создайте файл **SocketExample.java**, демонстрирующий работу с сокетами:

```
public class SocketExample {
    public static void main(String[] args) {
        String host = "0.0.0.0";
        int port = 32000;
        Thread server = new Server(port);
        // server thread
        server.start();
        Thread client = new Client(host, port, 1);
        // client thread
        client.start();
```

```
    }
}
```

Запустите программу и удостоверьтесь в ее работоспособности. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Connecting to 0.0.0.0:32000
Client client#1 connected
Hello
Server: received from client#1 : Hello
Have a nice day!
Server: received from client#1 : Have a nice day!
exit
Shutting down...
Server: null
```

Работа с сокетами

Сокеты являются низкоуровневыми сетевыми соединениями и позволяют осуществлять быстрый обмен данными, в частности при создании клиент-серверных приложений. Для каждого из клиентов приложения должны быть созданы клиентские сокеты (объекты типа **`Socket`**), а для серверов – серверные сокеты – объекты типа **`ServerSocket`**.

Взаимодействие происходит следующим образом – клиенту известен порт и адрес сервера. Изначально клиент запрашивает у сервера доступ к нему и устанавливается соединение. Клиент может отправлять некоторые сообщения серверу и ожидать от него ответа. Сервер принимает некоторые сообщения или запросы от клиента и отсылает ему ответы.

В примере рассмотрена реализация некоторого echo-сервера, который устанавливает соединение с клиентом и, получив от клиента сообщение, отсылает его ему обратно. **`PrintWriter`** позволяет реализовать потоковую отправку текстовых сообщений. Создание объекта типа **`PrintWriter`** происходит следующим образом:

```
PrintWriter out = new
PrintWriter(fromClient.getOutputStream(), true);
```

Первый аргумент представленного конструктора – поток для отправки (записи) сообщений клиенту, второй аргумент – признак автоматической отправки данных.

При реализации работы сокетного взаимодействия нескольких клиентов и

некоторого сервера необходимо для каждого сокета-клиента создавать серверный поток для осуществления попарного взаимодействия. Для этого необходимо связывать идентификационные данные клиента и сервера, чтобы не возникло коллизий при передаче сообщений.

10. Разработайте клиент-серверное приложение на основе сокетов, реализующее функции чата для двух клиентов. Клиенты могут отправлять друг другу сообщения. Сообщение от клиента попадает на сервер, сервер его обрабатывает и возвращает сообщение второму клиенту. Необходимо осуществить ввод сообщений с клавиатуры для обоих клиентов и вывод переданных сервером сообщений от другого клиента (вместе с именем-идентификатором второго клиента) в консоль. Вывод для каждого клиента осуществлять в его консоль в следующем формате:

```
Me:
. . . // сообщение введенное с клавиатуры текущим клиентом
Other: // идентификатор второго клиента
. . . // сообщение-ответ второго клиента
. . .
```

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА КОМПЬЮТЕРНЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Компьютерные образовательные технологии» (КОТ) была создана в 2001 году и осуществляла подготовку специалистов направления «Информационные системы» на факультете информационных технологий и программирования по ряду дисциплин (информатика, информационные технологии, дискретная математика, моделирование систем, веб-программирование и др.). В 2003 году кафедра КОТ перешла в статус выпускающей кафедры. В этот год был впервые открыт набор студентов на специальность «Информационные технологии в образовании». В 2011 году кафедра КОТ перешла в состав факультета компьютерные технологии и управление (КТиУ), а в 2015 – в состав факультета программной инженерии и компьютерной техники (ФПИиКТ) мегафакультета КТиУ.

Андрей Владимирович Лямин, Елена Николаевна Череповская

Объектно-ориентированное программирование. Компьютерный практикум

Компьютерный практикум

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49