Программирование. Занятие №0

Цель данного занятия - изучить процесс создания программы на языке Java и подготовиться к выполнению первой лабораторной работы.

Для этого нужно научиться:

- подключаться к рабочей среде на сервере helios;
- писать код программы в текстовом редакторе;
- компилировать программу из командной строки;
- понимать сообщения об ошибках и исправлять ошибки;
- запускать программу из командной строки;
- пользоваться jshell для изучения возможностей Java;
- создавать jar-архив, содержащий готовую программу.

Введение

Учебные материалы к курсу "Программирование" находятся на портале **https://se.ifmo.ru** в соответствующем разделе. Для доступа к некоторым разделам портала и рабочей среде для выполнения заданий **нужны имя пользователя и пароль**. Их можно получить на странице **https://se.ifmo.ru/passwd** (предварительно следует войти в ИСУ со своими учетными данными).

Обычно в заданиях к лабораторным работам указано, что демонстрация их выполнения должна производиться на сервере se.ifmo.ru. На сервере установлена операционная система FreeBSD и программное обеспечение для разработки на языке Java – **Java Development Kit 17**.

Подключиться к серверу для работы в командной строке (компиляции и запуска своей программы) можно по протоколу SSH, а скопировать файлы со своего компьютера на сервер или с сервера к себе – по протоколу SCP или SFTP.

Параметры подключения:

Host Name:	se.ifmo.ru
Port:	2222
Login as (User name):	можно получить на странице https://se.ifmo.ru/passwd
Password:	

Это можно сделать как из командной строки со своего компьютера или с компьютера в учебной аудитории, так и с помощью одного из многочисленных графических клиентов, часть из которых приведена в таблице.

Название	ОС	Сайт
PuTTY	Windows	https://www.chiark.greenend.org.uk/~sgtatham/putty/
WinSCP	Windows	https://winscp.net/
SmarTTY	Windows	https://sysprogs.com/SmarTTY/
Ásbrú CM	Linux	https://www.asbru-cm.net/
Muon	Linux	https://github.com/subhra74/snowflake
iTerm2	MacOS	https://iterm2.com/
Core Shell	MacOS	https://codinn.com/shell/
Hyper	Win/Lin/Mac	https://hyper.is/
Tabby	Win/Lin/Mac	https://tabby.sh/

Пример подключения из командной строки (Windows, Linux, MacOS) (вместо ***s000000*** нужно подставить свое имя пользователя):

```
C:\> ssh -l s000000 -p 2222 se.ifmo.ru
```

При первом подключении обычно выдается сообщение о том, что невозможно проверить аутентичность сервера, так как его публичный ключ отсутствует в базе известных ключей. Нужно ответить yes на предложение добавить его в базу известных хостов.

```
The authenticity of host '[se.ifmo.ru]:2222 ([77.234.196.4]:2222)' can't be established.
ECDSA key fingerprint is SHA256:gwyf8uH0khUL3O4+0cHO2YPRQD0d2nonu/GlQjrkJoo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[se.ifmo.ru]:2222' (ECDSA) to the list of known hosts.
```

Введите свой пароль на запрос Password: Обратите внимание, что при вводе пароля ничего не отображается. Это нормально.

Для копирования файлов (например файла MyProgram.java) на сервер helios можно использовать команду scp (вместо ***s000000*** нужно подставить свое имя пользователя):

```
C:\> scp -P 2222 MyProgram.java s000000@se.ifmo.ru:~
```

Первая программа

Напишем первую программу на языке Java, которая выводит в стандартный вывод приветствие Hello, world!

Создадим в текстовом редакторе файл с исходным кодом, обращая внимание на регистр символов:

```
1  public class Hello {
2    public static void main(String[] args) {
3      System.out.println("Hello, world!");
4    }
5  }
```

Java требует, чтобы весь код находился только внутри классов, поэтому в строке 1 объявляем общедоступный (public) класс с именем Hello. При запуске программы Java пытается найти и выполнить метод с именем main. Объявим этот метод в строке 2. Этот метод должен иметь модификаторы public и static, возвращать значение типа void (ничего не возвращать) и принимать массив строк (String[] args). В строке 3 находится основная часть программы - инструкция печати строки "Hello, world!" в стандартный поток вывода, завершающаяся точкой с запятой. System (обязательно с заглавной буквы) - это класс для работы с системой, out - стандартный поток вывода, println - метод для печати с переходом на следующую строку. Сроки 4 и 5 содержат фигурные скобки, закрывающие объявления метода и класса.

Сохраним исходный код программы в файле Hello.java. Обратите внимание, что файл должен иметь расширение .java и имя, совпадающее с названием класса Hello.

Запустим Java-машину (java), указав ей в качестве параметра файл с исходным кодом (Hello.java), и получим результат - строку "Hello, world!":

```
[_@helios ~]$ java Hello.java

Hello, world!
```

Из-за особенностей настройки Java на сервере helios при запуске команд JDK выводится сообщение про опции -XX:MaxHeapSize и -XX:MaxMetaspaceSize - его можно игнорировать.

Запускать программу таким образом (без предварительной компиляции) можно только в том случае, если в файле с исходным кодом находится всего один общедоступный класс. Поэтому данный способ запуска годится исключительно для коротких тестовых

программ. Для более сложных проектов следует использовать полноценный цикл - компиляция исходного кода в байт-код с помощью компилятора (javac), и дальнейшее исполнение байт-кода с помощью виртуальной Java-машины (java).

Компилируем исходный код программы с помощью компилятора `javac`. В качестве параметра указываем имя файла с исходным кодом. При компиляции либо выводятся сообщения об ошибке, либо компиляция выполняется молча. При успешной компиляции создается файл `Hello.class`, содержащий байт-код.

```
[_@helios ~]$ javac Hello.java
```

Запускаем программу с помощью интерпретатора `java`. В качестве параметра указываем имя основного класса программы, в котором есть метод `main`. В нашем случае это "Hello". При этом расширение не указывается. Программа запускается и выводит строку "Hello, world!".

```
[_@helios ~]$ java Hello

Hello, world!
```

Теперь самостоятельно попробуйте заменить строку 3 на следующие варианты и посмотреть, как изменяется поведение программы (цветом выделены отличия от предыдущего варианта):

```
1     System.out.print("Hello, world!");
2     System.out.print("Hello, world!");System.out.println();
3     System.out.print("Hello,\t\tworld!\n")
4     System.out.println("Hel\\lo, \"world!\"");
5     System.out.println("Hello", "world!");
6     System.out.println("Hello" + ", " + "world!");
7     System.out.println("\u0048ello, world!\u0009");
8     System.out.println("Привет, мир!");
9     System.err.println("Привет, мир!");
10    System.err.println("""
      Привет,
      мир""");
```

В некоторых случаях компилятор выдает ошибки. Исправьте их.

Числовые константы и выражения

Для выполнения данного этапа удобнее воспользоваться интерактивной оболочкой jshell. Она работает в режиме REPL (read-eval-print-loop), то есть читает введенную строку, выполняет код, печатает результат и повторяет эти действия. При этом не обязательно создавать полноценную программу, достаточно ввести арифместическое или другое выражение и нажать Enter. Не обязательно даже использовать метод println().

Запустим jshell:

```
[_@helios ~]$ jshell
|  Welcome to JShell -- Version 17.0.12
|  For an introduction type: /help intro

jshell>
```

Теперь можно ввести либо инструкцию, либо выражение, либо команду jshell. Команды начинаются с символа /, инструкции и выражения можно вводить как есть. Например, попробуем уже знакомую инструкцию System.out.println. Точку с запятой в конце ставить не обязательно, и при нажатии Enter сразу выводится ррезультат.

```
jshell> System.out.println("Hello, world!")
Hello, world!
```

Можно не вводить целиком инструкцию, а ограничиться только самой строкой - "Hello, world!". jshell вычисляет значение введенного выражения, присваивает его внутренней переменной $2 и выводит полученное значение.

```
jshell> "Hello, world!"
$2 ==> "Hello, world!"
```

Обратите внимание, что в этом случае строка выводится в кавычках, чтобы показать, что это именно значение, а не вывод результата выполнения.

Теперь попробуем поизучать как Java работает с целыми числами и простыми арифметическими выражениями. При этом в выражениях можно использовать результаты предыдущих вычисленных выражений, указывая их с символом доллара $.

```
jshell> 42
$3 ==> 42
```

```
jshell> 42 + $3
$4 ==> 84
jshell> 1 + 2 + 3
$5 ==> 6
jshell> $4 - $5
$6 ==> 78
```

Далее самостоятельно с помощью jshell попробуйте вычислить выражения с целыми константами, перечисленные в таблице. Изучите, какое значение имеет каждое выражение, попробуйте объяснить, почему оно именно такое.

1	`10 - 20`	11	`(3 + 5) * 2`
2	`10 - (5 + 6 + 7)`	12	`6 / 2 * (4 - 1) * 3 - 5`
3	`10 - 1`	13	`9 / 3`
4	`010 - 1`	14	`9 / 4`
5	`0x10 - 1`	15	`9 % 4`
6	`0b10 - 1`	16	`4 * 2 + 1`
7	`0xA + 0b1010 + 012 + 10`	17	`9 / 0`
8	`1_999_999_999 + 1`	18	`32767 * 32769`
9	`5 * 2 + 3`	19	`65536 * 65536`
10	`3 + 5 * 2`	20	`65536 * 65536L`

Теперь попробуйте поработать со значениями с плавающей точкой, а также математическими функциями. Они находятся в классе Math, и для их вызова нужно указывать имя класса, например Math.sin(0).

1	`9.0 / 4`	13	`65536.0 * 65536.0`
2	`9.0 / 0`	14	`Math.pow(5, 2)`
3	`-9.0 / 0`	15	`Math.sqrt(49)`
4	`0.0 / 0`	16	`Math.sqrt(-100)`
5	`1.2 + 1.2`	17	`Math.cbrt(27)`
6	`1.1 + 1.3`	18	`Math.exp(1)`
7	`1000.0 * 1000.0`	19	`Math.log(Math.E)`
8	`1000.0 * 10000.0`	20	`Math.log10(1000)`
9	`2.5E0`	21	`Math.PI ; Math.E`
10	`2.5E1`	22	`Math.sin(Math.PI / 2)`
11	`2.5E-1`	23	`Math.cos(0)`
12	`2.5E-3`	24	`Math.tan(Math.PI / 4)`

Логические и символьные значения

Для представления логических значений существует тип boolean, имеющий два значения - true и false. Попробуйте потренироваться использовать значения типа boolean в выражениях.

```
1     true & false
2     true | false
3     true ^ false
4     ! true
5     true & true
6     true | true
7     true ^ true
8     true < false
9     0 > 1
10    1 > 0
11    1 > 0 == true
12    2 * 2 == 4
13    2 * 2 != 4
14    1.1 + 1.3 == 2.4
15    0.0 == -0.0
16    false == 0
```

Для работы с символами используются символьные (заключаются в одинарные кавычки) и строковые (заключаются в двойные кавычки) константы.

```
1     'a'
2     'a' + 'a'
3     'a' + 0
4     'a' < 'b'
5     'a' * 'b'
6     'a' == 'b'
7     'a' == "a"
8     "a" < "b"
9     "Hello".length()
10    "Hello".charAt(0)
11    "Hell" + "o"
12    "Hell" + 'o'
13    "1" + "1" + "1"
14    1 + 1 + "1"
15    "1" + 1 + 1
16    "1" + (1 + 1)
```

Переменные и присваивание

Значения можно сохранять в переменных. Для использования переменной ее нужно сначала объявить, указав тип переменной. После объявления переменной можно присваивать значение и использовать ее в выражениях.

В языке Java тип переменной должен соответствовать типу значения, которое ей присваивается. Типы могут быть примитивными и ссылочными. Примитивных типов - восемь: `byte`, `short`, `int`, `long`, `float`, `double`, `char`. `boolean`. К ссылочными типам относятся массивы, классы, интерфейсы, перечисления, записи. Один из ссылочных типов уже встречался в заданиях - это класс `String`.

Попробуйте выполнить очередную порцию примеров, чтобы разобраться, как работать с переменными и как можно присваивать им значения.

```
1   x = 5
2   int x
3   x = 1
4   int y = x + 10
5   x = 5; y
6   y = x + 10
7   boolean q = x < y
8   q = 0
9   x = 1; y = ++x; x; y
10  x = 1; y = x++; x; y
11  y += x
12  x = y = 1 ; y
13  x += y += 2 ; x ; y
14  x = 1; y = ++x; x; y
15  x = 1; y = x++; x; y
16  y += x
17  x = y = 1 ; y
18  x += y += 2 ; x ; y
19  byte b = 127
20  x = b
21  b = x
22  b = (byte) x
23  char c = 'a'
24  c += 1
25  c = c + 1
26  c = (char) (c + 1)
27  c++
28  int i = 0; i += c
29  c = 0; c += c
30  String s = ""; s += c
```

Массивы

Для обработки однотипных данных удобнее использовать массивы. При объявлении массива указывается его тип, при создании массива указывается количество элементов в нем. Доступ к элементам производится по индексу, который указывается в квадратных скобках.

Выполните еще несколько заданий для знакомства с массивами.

```
1   int[] a;
2   a[0]
3   a = new int[5]; a[0]
4   int[b] = {1,2,3,4,5,6}
5   a.length ; b.length
6   b[0] ; b[5]
7   a[2] + b[3] * b[4]
8   b[6]
9   int[][] d
10  d = new int[2][2]
11  d[0]
12  d[0][0]
13  d[0] = new int[3] ; d
14  int[][] j = {{1,2},{3,4}}
15  j[1][1] = 5 ; j
16  j[a[2]][b[0]]++ ; j
```

Чтобы выйти из jshell, нужно выполнить команду `/exit`

Управляющие инструкции

В метод main передается один параметр - массив типа String, содержащий аргументы командной строки, которые были указаны после имени класса при запуске программы.

```
1  public class Hello {
2    public static void main(String[] args) {
3      System.out.println("Привет, " + args[0] + "!");
4    }
5  }
```

Элемент массива args[0] содержит первый аргумент командной строки после имени класса. Попробуем откомпилировать и запустить программу с дополнительным аргументом.

```
[_@helios ~]$ javac Hello.java
[_@helios ~]$ java Hello Вася
Привет, Вася!
```

Если при запуске программы не указать аргумент, то возникнет исключение при попытке обратиться к args[0], так как массив пустой.

```
[_@helios ~]$ java Hello

Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 0 out
of bounds for length 0
      at Hello.main(Hello.java:3)
```

Добавим проверку на наличие аргумента перед тем, как обращаться к нему.

```
1  public class Hello {
2    public static void main(String[] args) {
3      if (args.length > 0) {
4        System.out.println("Привет, " + args[0] + "!");
5      } else {
6        System.out.println("Привет, мир!");
7      }
8    }
9  }
```

После ключевого слова if в круглых скобках указывается условие, имеющее тип boolean, если оно истинно, то выполняется код между фигурными скобками после условия, если ложно, то выполняется код после ключевого слова else.

Теперь программа корректно обрабатывает отсутствие аргументов, но, если мы укажем 2 аргумента, второй игнорируется.

```
[_@helios ~]$ javac Hello.java
[_@helios ~]$ java Hello Вася Петя
Привет, Вася!
```

Добавим дополнительные проверки в программу. Самостоятельно откомпилируйте и проверьте программу с разными аргументами.

```
1   public class Hello {
2     public static void main(String[] args) {
3       if (args.length == 0) {
4         System.out.println("Привет, мир!");
5       }
6       if (args.length == 1) {
7         System.out.println("Привет, " + args[0] + "!");
8       }
9       if (args.length == 2) {
10        System.out.println("Привет, " +
11          args[0] + " и " + args[1] + "!");
12      }
13      if (args.length > 2) {
14        System.out.println("Привет, все!");
15      }
16    }
17  }
```

Чтобы исключить повторяющийся код, имеет смысл создать отдельный метод (функцию) для вывода приветствия с параметром типа String. Назовем его greet. Проверьте. что работа программы не изменилась.

```
1    public class Hello {
2      public static void greet(String name) {
3        System.out.println("Привет, " + name + "!");
4      }
5      public static void main(String[] args) {
6        if (args.length == 0) {
7          greet("мир");
8        }
9        if (args.length == 1) {
10         greet(args[0]);
11       }
12       if (args.length == 2) {
13         greet(args[0] + " и " + args[1]);
14       }
15       if (args.length > 2) {
16         greet("все");
17       }
18     }
19   }
```

Вместо множества блоков if можно использовать блок switch - код будет более читаемым. Проверьте работу программы.

```
1    public class Hello {
2      public static void greet(String name) {
3        System.out.println("Привет, " + name + "!");
4      }
5      public static void main(String[] args) {
6        switch (args.length) {
7          case 0: greet("мир"); break;
8          case 1: greet(args[0]); break;
9          case 2: greet(args[0] + " и " + args[1]); break;
10         default: greet("все"); break;
11       }
12     }
13   }
```

Давайте теперь в случае, когда у нас больше 2 аргументов, выведем приветствие для всех с помощью цикла for со счетчиком.

```
1   public class Hello {
2     public static void greet(String name) {
3       System.out.println("Привет, " + name + "!");
4     }
5     public static void main(String[] args) {
6       switch (args.length) {
7         case 0: greet("мир"); break;
8         case 1: greet(args[0]); break;
9         case 2: greet(args[0] + " и " + args[1]); break;
10        default:
11          for (var i = 0; i < args.length; i++) {
12            greet(args[i]);
13          }
14          break;
15      }
16    }
17  }
```

```
[_@helios ~]$ javac Hello.java
[_@helios ~]$ java Hello Вася Михалыч "Ваше Высокопреосвященство"
Привет, Вася!
Привет, Михалыч!
Привет, Ваше Высокопреосвященство!
```

Так как цикл в любом случае выводит отдельное приветствие для каждого, можно убрать проверку на один и два аргумента.

Так как в данной программе значения счетчика никак не используются, можно вместо стандартного цикла for использовать цикл for по элементам массива.

```
1   public class Hello {
2     public static void greet(String name) {
3       System.out.println("Привет, " + name + "!");
4     }
5     public static void main(String[] args) {
6       if (args.length > 0) {
7         for (String s : args) {
8           greet(s);
9         }
10      } else {
11        greet("мир");
12      }
13    }
14  }
```

В качестве самостоятельной практики попробуйте заменить цикл for циклами while и do-while.

Откомпилируем итоговую программу, проверим, что она работает и выдает ожидаемые результаты. Теперь создадим исполняемый jar-архив, чтобы программу удобно было передавать другим. В простейшем случае нам понадобятся опции -c для создания архива, -f для указания имени архива, -e для указания имени основного класса, который содержит метод main. В конце перечисляются файлы, которые должны попасть в архив. В нашем случае это Hello.class, если файлов с расширением .class много, можно задать шаблон "*.class".

```
[_@helios ~]$ jar -c -f hello.jar -e Hello Hello.class
```

Запустить упакованную программу можно следующим образом.

```
[_@helios ~]$ java -jar hello.jar Январий
Привет, Январий!
```