

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ**

Ордена Трудового Красного Знамени

**Федеральное государственное бюджетное образовательное
учреждение высшего образования**

«Московский технический университет связи и информатики»

Лабораторная работа № 1

Выполнила: Студентка группы БВТ2306

Максимова Мария

Москва 2024

Ссылка на Git-hub: <https://github.com/Mashaaaaa7/Lab1>

Для выполнения лабораторной работы №1 нам понадобится выполнить 2 задания:

Задание 1. Создайте программу, которая находит и выводит все простые числа меньше 100.

Задание 2. Создайте программу, которая определяет, является ли введенная строка палиндромом.

Для начала, запустим приложение IntelliJ Idea и создадим проект. Назовём его JavaHelloWorldProgram.java и напишем код для выполнения команды «Hello, word!» и сохраним программу под именем JavaHelloWorldProgram.java в каталоге для первой лабораторной работы. (См. рисунок 1)

A screenshot of the IntelliJ IDEA code editor. The code is written in Java and defines a class named JavaHelloWorldProgram with a main method that prints "Hello, world!". The code is as follows:

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

Рисунок 1

Далее, запускаю код:

A screenshot of the IntelliJ IDEA Run console. The console shows the execution of the JavaHelloWorldProgram. The output is: "C:\Program Files\bin\java.exe" -ja Hello, world! Process finished with exit code 0. The console window has a title bar that says "Run: JavaHelloWorldProgram".

```
Run: JavaHelloWorldProgram  
"C:\Program Files\bin\java.exe" -ja  
Hello, world!  
Process finished with exit code 0
```

Рисунок 2

Открою командную строку PowerShell и перейду в каталог, в котором сохранен файл программы. Для начала, мы можем набрать следующие команды чтобы скомпилировать и запустить программу:

```
PS C:\Users\Maria\Desktop\lab1\out\production\lab1> java.exe
Usage: java [options] <mainclass> [args...]
        (to execute a class)
or  java [options] -jar <jarfile> [args...]
        (to execute a jar file)
or  java [options] -m <module>[/<mainclass>] [args...]
    java [options] --module <module>[/<mainclass>] [args...]
        (to execute the main class in a module)
or  java [options] <sourcefile> [args]
        (to execute a source-file program)

Arguments following the main class, source file, -jar <jarfile>,
-m or --module <module>/<mainclass> are passed as the arguments to
main class.

where options include:

-cp <class search path of directories and zip/jar files>
-classpath <class search path of directories and zip/jar files>
--class-path <class search path of directories and zip/jar files>
            A ; separated list of directories, JAR archives,
            and ZIP archives to search for class files.
-p <module path>
--module-path <module path>...
            A ; separated list of elements, each element is a file path
            to a module or a directory containing modules. Each module is either
            a modular JAR or an exploded-module directory.
--upgrade-module-path <module path>...
            A ; separated list of elements, each element is a file path
            to a module or a directory containing modules to replace
            upgradeable modules in the runtime image. Each module is either
            a modular JAR or an exploded-module directory.
```

Рисунок 3

```

PS C:\Users\Maria\Desktop\lab1\out\production\lab1> javac.exe
Usage: javac <options> <source files>
where possible options include:
  @<filename>           Read options and filenames from file
  -Akey[=value]         Options to pass to annotation processors
  --add-modules <module>(,<module>)*
                        Root modules to resolve in addition to the initial modules,
                        or all modules on the module path if <module> is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
                        Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
                        Specify where to find user class files and annotation processors
  -d <directory>        Specify where to place generated class files
  -deprecation           Output source locations where deprecated APIs are used
  --enable-preview       Enable preview language features.
                        To be used in conjunction with either -source or --release.
  -encoding <encoding>   Specify character encoding used by source file
  -endorseddirs <dirs>   Override location of endorsed standards path
  -extdirs <dirs>        Override location of installed extensions
  -g                    Generate all debugging info
  -g:{lines,vars,source} Generate only some debugging info
  -g:none               Generate no debugging info
  -h <directory>        Specify where to place generated native header files
  --help, -help, -?     Print this help message
  -help-compiler, -X   Print help on compiler options

```

Рисунок 4

В первом случае будет вызван компилятор `javac.exe`, а во втором случае – запускатка `java.exe`, которая стартует нашу программу. Эти файлы лежат в папке `bin` нашего JDK

Скомпилируем код:

```

PS C:\Users\Maria\Desktop\lab1\out\production\lab1> cd C:\Users\Maria\Desktop\lab1\src
PS C:\Users\Maria\Desktop\lab1\src> java JavaHelloWorldProgram.java
Hello, word!
PS C:\Users\Maria\Desktop\lab1\src>

```

Рисунок 5

```

\Maria> javac JavaHelloWorldProgram.java

```

Рисунок 6

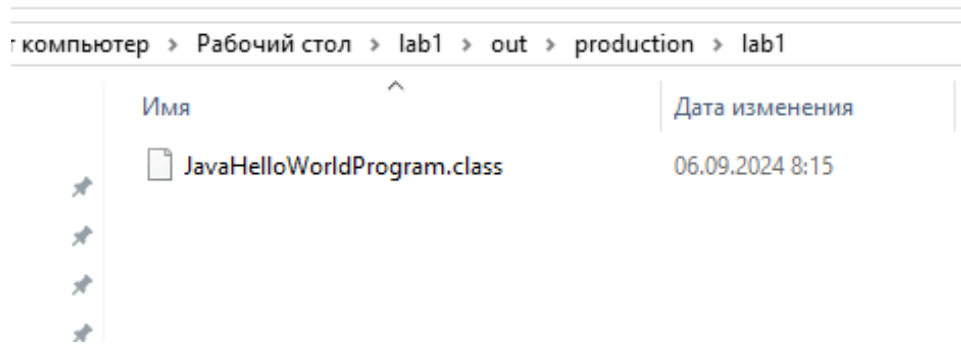


Рисунок 7

Эта команда вызовет компилятор, который создаст файл **JavaHelloWorldProgram.class**, содержащий скомпилированный код нашей java программы.

Чтобы запустить ее, нужно ввести команду **java** с именем класса в качестве параметра:

```
\Maria> java HelloWorldProgram
```

Рисунок 8

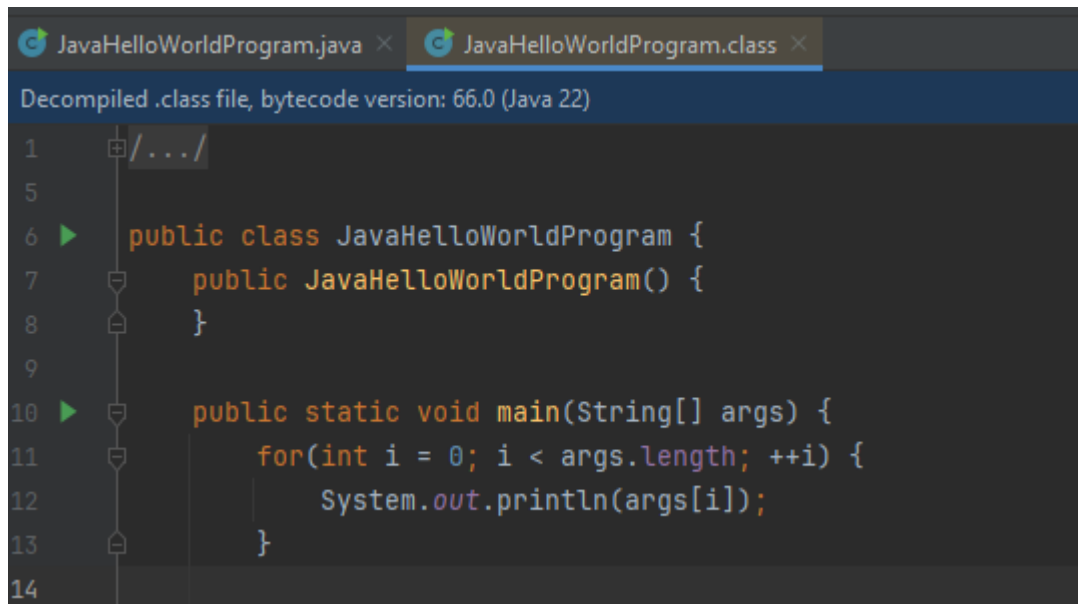
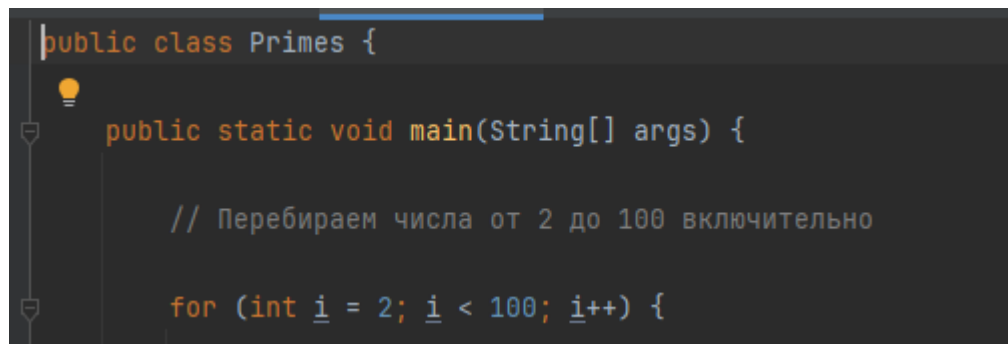


Рисунок 9

Далее, приступим к выполнению заданий. Начнём с задания 1. Нам нужно создать программу, которая находит и выводит все простые числа меньше 100. До 100 встречается 25 простых чисел:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

Начнём с понятия простого числа. Простое число — это натуральное число больше 1, у которого есть всего два делителя: единица и само число. Таким образом, нам нужно перебирать простые числа от 2 до 100 включительно. Создадим файл с именем Primes.java и опишем класс:



```
public class Primes {  
  
    public static void main(String[] args) {  
  
        // Перебираем числа от 2 до 100 включительно  
  
        for (int i = 2; i < 100; i++) {
```

Рисунок 10

Разберём подробнее:

1. **public class Primes:**

- Здесь мы объявляем класс с именем Primes. В Java все кодовые структуры организованы в классы, и каждая программа должна содержать хотя бы один класс. Класс является шаблоном или чертежом для создания объектов.

2. **public static void main(String[] args):**

- Это объявление метода main, который является точкой входа для любой Java-программы.
- **public:** Это модификатор доступа, который указывает, что метод main доступен из любого другого класса.
- **static:** Этот модификатор указывает, что метод принадлежит классу Primes, а не объекту класса. Это означает, что метод может быть вызван без создания экземпляра класса.
- **void:** Это возвращаемый тип метода, указывающий, что метод не возвращает никакого значения.
- **String[] args:** Это параметр метода, представляющий массив строк. Он используется для передачи аргументов командной строки в программу. Когда вы запускаете программу, вы можете передать в

нее параметры через командную строку, и они будут доступны в этом массиве.

3. for (int i = 2; i < 100; i++) { :

объявить переменную *i* целого типа (int) и присвоить ей начальное значение 2;

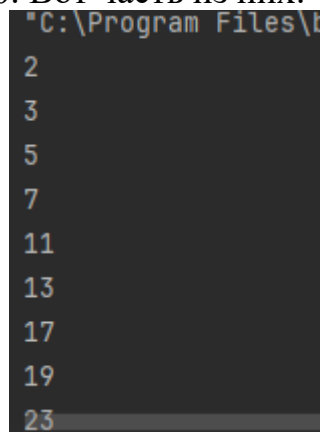
пока *i* меньше 100, выполнять код в фигурных скобках {...} после оператора for;

увеличивать значение переменной *i* на единицу после каждого выполнения кода в фигурных скобках.

4. if (isPrime(i)) { // Проверяем, является ли число простым

System.out.println(i); // Выводим простое число

После запуска данной программы, на выходе будут представлены 25 простых чисел от 2 до 100. Вот часть из них:



```
"C:\Program Files\B
2
3
5
7
11
13
17
19
23
```

Рисунок 11

Для второго задания надо сделать код, который скажет, палиндром слово или нет

Пишем код и запускаем программу:

```
public class Palindrome {  
  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            String s = args[i];  
            if(isPalindrome(s)){  
                System.out.println(s + " is a palindrome");  
            } else {  
                System.out.println(s + " is not a palindrome");  
            }  
        }  
    }  
}
```

Рисунок 12

```
public static String reverseString(String s) { 1 usage  
    String reversed = "";  
    for (int i = s.length() - 1; i >= 0; i--) {  
        reversed += s.charAt(i);  
    }  
    return reversed;  
}  
  
public static boolean isPalindrome(String s) { 1 usage  
    String reversed = reverseString(s);  
    return s.equals(reversed);  
}  
}
```

Рисунок 13

```
Palindrome x  
"C:\Program Files\bin\java.exe" "-javaagent:C:\Program File  
  
Process finished with exit code 0
```

Рисунок 14

Разберём код подробно:

1. `public class Palindrome` — это класс с именем `Palindrome`, который является публичным (доступным для использования другими классами).

2. Метод `main` — это точка входа в программу. Метод принимает массив строк `args` и выполняет цикл `for` для каждой строки в этом массиве.

3. Внутри цикла `for`:

`String s = args[i]` — строка `s` присваивается значению из массива `args` по индексу `i`.

`isPalindrome(s)` — вызывается метод `isPalindrome` со строкой `s` в качестве аргумента. Этот метод возвращает `true`, если строка является палиндромом, и `false` в противном случае.

Если строка является палиндромом, то выводится сообщение «`s is a palindrome`».

Иначе выводится сообщение «`s is not a palindrome`».

4. Метод `reverseString` — этот метод принимает строку `s` и возвращает её в обратном порядке. Он использует цикл `for` и добавляет символы строки `s` в обратном порядке к строке `reversed`.

5. Метод `isPalindrome` — этот статический метод проверяет, является ли строка `s` палиндромом. Для этого он вызывает метод `reverseString`, чтобы получить обратную строку, а затем сравнивает исходную строку с обратной. Если они равны, то строка является палиндромом. В противном случае она не является палиндромом.

Этот код позволяет проверить, является ли каждая строка в массиве `args` палиндромом или нет.

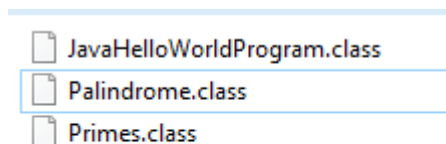


Рисунок 15

Для окончания лабораторной работы, нам понадобится программа `PowerShell`. Мы компилируем и тестируем программу, как указано в задании. На выходе, у нас выводятся наши слова и определяется, палиндром слово или нет.

```
PS C:\Users\Maria> cd C:\Users\Maria\Desktop\lab1\src
PS C:\Users\Maria\Desktop\lab1\src> javac Palindrome.java
PS C:\Users\Maria\Desktop\lab1\src> java Palindrome madam racecar apple kayak song noon
madam is a palindrome
racecar is a palindrome
apple is not a palindrome
kayak is a palindrome
song is not a palindrome
noon is a palindrome
PS C:\Users\Maria\Desktop\lab1\src> ■
```

Рисунок 16