

Структура программы

Что делает текст кодом?

Содержание

1. Ключевые слова первой программы
2. Исполняемая функция
3. Код вывода в консоль

1. Ключевые слова первой программы



```
→ ↻ 🔒 https://www.onlinegdb.com/online_java_compiler
[File Icon] [Upload Icon] [Run] [Debug] [Stop] [Share] [Save]
Main.java
1 public class Main
2 {
3     public static void main(String[] args) {
4         System.out.println("Hello World");
5     }
6 }
```

В предыдущей лекции мы рассмотрели первую программу. Она очень простая и выводит в консоль (на экран) 1 строку – Hello World. По факту это код на линии 4. Но что же тогда делают все остальные линии? Кстати да, мы называем и нумеруем линии (по факту это делает ваша среда разработки – в данном случае веб-сайт) LoC – line of code. Это очень удобно и нет, не имеет значения сколько линий кода в вашей программе.

Мы сравнивали код со стихотворением и конечно же как и в стихе в коде тоже должен быть заголовок, название. Давайте посмотрим внимательно на файл, в котором написан код. Вы можете заметить что он называется Main.java. Если бы вы создали текстовый файл в блокноте и сохранили его, то могли бы назвать его как угодно, например Main.txt. Но так как мы пишем на языке Java то и расширение файла будет соответствующим. Очень легко понять, что название класса должно совпадать с именем файла. А давайте для примера изменим имя класса. Когда нажимаем на Run то компилятор (тот, что переводит ваш код на языке джава на тот язык, который понимает машина, т.е. машинный) выдает ошибку о том, что класс Other должен быть размещен в файле с именем Other.java. И это первая вещь которую вам нужно будет запомнить – мы создаем файл с расширением .java и имя этого файла должно совпадать с именем класса который в нем написан. Так что же это за класс?

```
Main.java
1 public class Other
2 {
3     public static void main(String[] args) {
4         System.out.println("Hello World");
5     }
6 }
7
```

input stderr

Compilation failed due to following error(s).

```
Main.java:1: error: class Other is public, should be declared in a file named Other.java
public class Other
      ^
1 error
```

Пока что мы условимся о том, что это заголовок, его имя ни на что не влияет. Но постарайтесь запомнить что есть договоренность, так называемый код-стайл и мы пишем имя класса с заглавной буквы (именно буквы, не с цифры или символа и не на кириллице, хотя это возможно, можете попробовать позже сами). Итак, вроде разобрались, что первой линией кода должно быть объявление класса – условимся говорить что это просто название нашего куска кода. Позже мы подробно рассмотрим что такое класс и что с ним можно делать. А пока сконцентрируемся на синтаксисе – т.е. на ключевых словах, которые собственно и делают наш код кодом. Чтобы в этом убедиться попробуем допустить ошибку в слове class, например написать его с заглавной буквы и заодно удалим первое слово public.

```
Main.java
1 Class Main
2 {
3     public static void main(String[] args) {
4         System.out.println("Hello World");
5     }
6 }
7
```

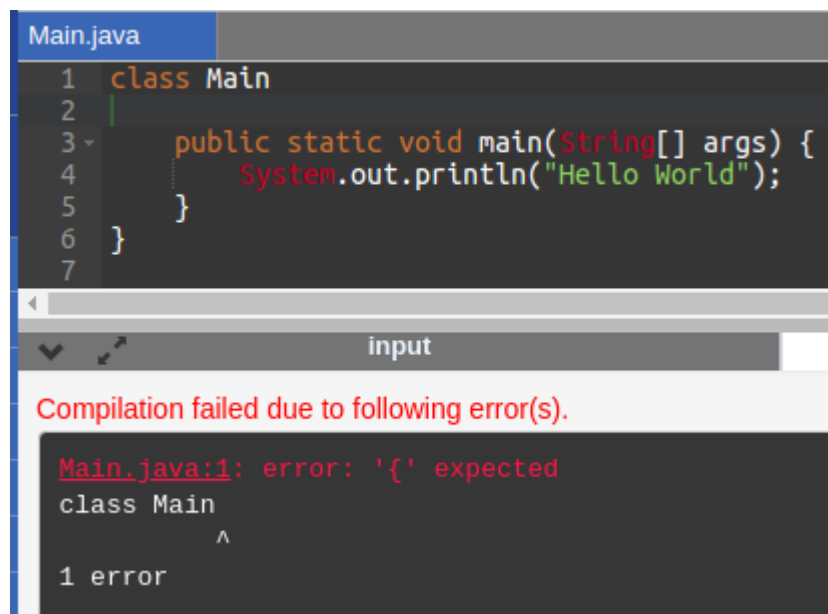
input

Compilation failed due to following error(s).

```
Main.java:1: error: class, interface, or enum expected
Class Main
^
Main.java:3: error: class, interface, or enum expected
    public static void main(String[] args) {
        ^
Main.java:5: error: class, interface, or enum expected
    }
    ^
3 errors
```

И мы получаем аж 3 ошибки! Но поверьте, удаление слова `public` ни на что не повлияло. Вы можете мне не верить на слово и вернуть написание слова `class` и снова запустить программу. Так о чем же наши ошибки? А все о том же – в файле под названием `Main.java` ожидается первой линией кода объявление `class Main`. Об этом была первая ошибка, после мы видим сообщение об ошибке на линии номер 3. Дело в том, что не найдя объявление класса компилятор пошел дальше и попытался найти его где-то после непонятного кода (неожиданно, но компилятор как и человек читает код сверху вниз, слева направо).

Забегая вперед скажем что ключевые слова в языке джава зарезервированы и именно по ним компилятор понимает собственно где начинается наш код и где он кончается. После объявления класса идет фигурная открывающаяся скобка (см. Рис 1). Можете сами удалить открывающуюся скобку на линии номер 2 и запустить код.



```
Main.java
1  class Main
2  {
3      public static void main(String[] args) {
4          System.out.println("Hello World");
5      }
6  }
7

input

Compilation failed due to following error(s).

Main.java:1: error: '{' expected
class Main
      ^
1 error
```

Кстати, вы заметили что компилятор подсказывает нам, что он ожидает открывающуюся скобку на линии номер 1, а не номер 2? На самом деле наше разделение на линии условное. Мы можем написать весь наш код на одной линии, но не особо удобно читать. Поэтому мы и переносим код на другую линию.

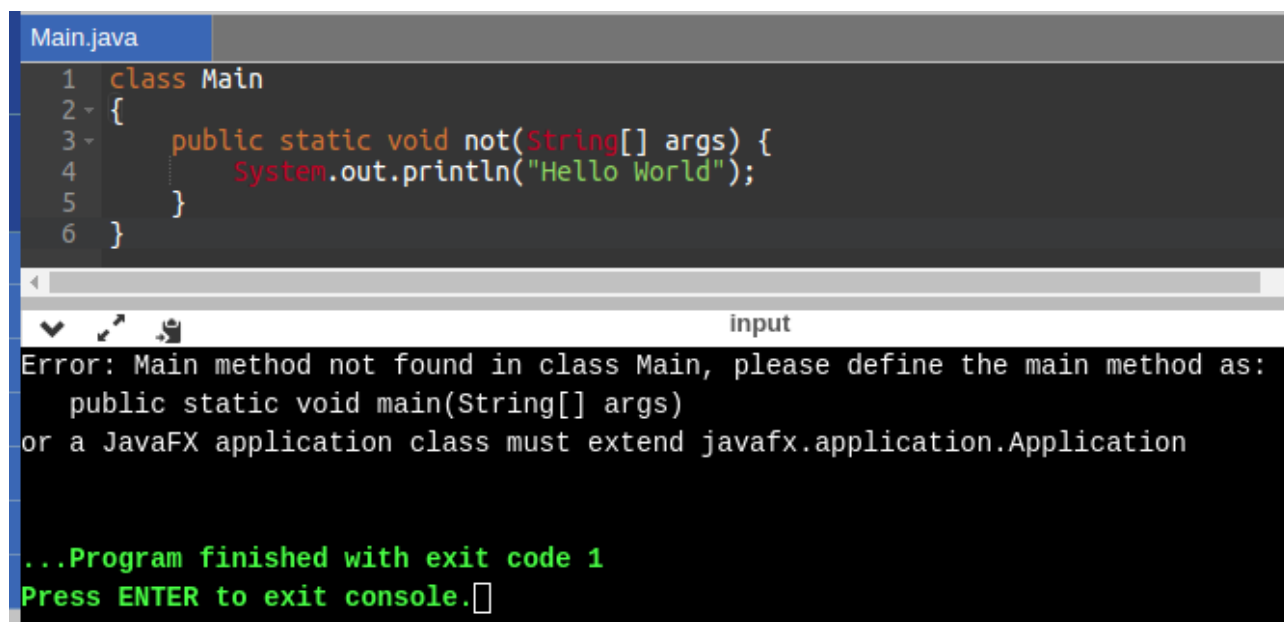
То же самое если сделать и с линией номер 6 мы увидим тот же результат – компилятор ожидает пару фигурных скобок. Открывающуюся после объявления класса и в самом конце, когда мы закончили писать код. Еще раз, обратите внимание, что порой для скобки на линии номер 2 является закрывающаяся скобка на линии номер 6. Если бы мы писали наш код в среде разработки, то она бы подсвечивала бы пары скобок. Самой частой ошибкой новичков является забывание закрывающей скобки, так что постарайтесь запомнить – скобки должны быть парными. Если вы открыли скобку, то и должны ее закрыть рано или поздно. Зачем же они нужны? Они обозначают начало и конец. В данном случае скобка на линии номер 2 обозначает начало класса, а скобка на линии 6 обозначает завершение кода.

2. Исполняемая функция

Идем дальше и рассмотрим что мы написали после открывающейся фигурной скобки и до закрывающей.

```
3.         public static void main(String[] args) {  
4.             System.out.println("Hello World");  
5.         }
```

Это называется исполняемая функция. Ведь когда мы нажимаем на кнопку Run компилятор ищет в нашем файле Main.java именно этот код, т.е. линию 3. Давайте проверим это и для примера тоже поменяем имя функции main на что-то другое.



The screenshot shows an IDE window titled 'Main.java' with the following code:

```
1 class Main  
2 {  
3     public static void not(String[] args) {  
4         System.out.println("Hello World");  
5     }  
6 }
```

Below the code editor, the console output shows an error message:

```
input  
Error: Main method not found in class Main, please define the main method as:  
    public static void main(String[] args)  
or a JavaFX application class must extend javafx.application.Application  
  
...Program finished with exit code 1  
Press ENTER to exit console.□
```

Кстати, вы заметили? Program finished with exit code 1. Ранее, когда в нашей программе все было корректно, то компилятор выдавал нам exit code 0. Это означает успешный итог выполнения кода. Итак, в чем же ошибка? Мейн метод не был найден в классе Мейн (согласен, здесь было бы лучше назвать наш класс иначе, для этого нужно создать файл например Example.java и в нем написать public class Example), пожалуйста, определите его как public static void main(String[] args). Т.е. сначала компилятор ищет класс с тем же названием что и имя файла, после чего ищет мейн метод (метод и функция это одно и то же и означает код, который что-то реально делает). Ну и сразу после объявления метода идет открывающаяся фигурная скобка на линии 3 и угадайте сразу – закрывающаяся на линии 5. Да, точно так же как и класс имеет начало и конец, так же и методы/функции имеют начало и конец. И они обозначаются фигурными скобками. Можете опять удалить фигурную и запустить и вы получите ожидаемую ошибку.

В вашем файле может быть больше 1 класса и больше 1 функции/метода, но об этом мы поговорим чуть позже. В наших лекциях мы будем стараться дозированно усваивать информацию. Так что пока не обращайтесь внимания на ключевые слова public static void, о них мы поговорим подробнее попозже.

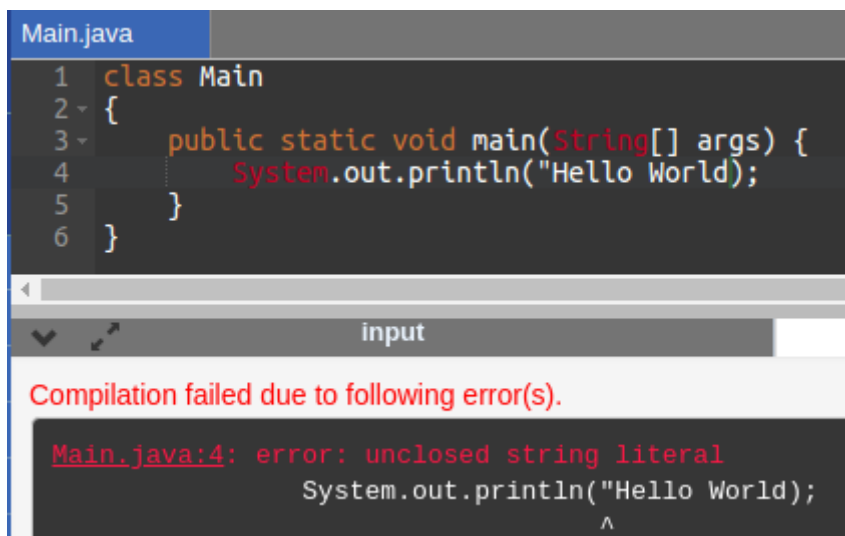
3. Код вывода в консоль

И наконец мы посмотрим внимательно на код на линии 4

```
System.out.println("Hello World");
```

Первое слово `System`. Заметьте, что оно написано с заглавной буквы. И как мы уже сегодня поняли – с заглавной буквы называются классы. Значит, в языке `java` есть некий класс с именем `System`. После идет точка. Что же она делает? Она дает доступ к другим классам. Сложно? Вовсе нет. Мы говорили, что в файле может быть больше 1 класса. В данном случае где-то внутри джавы есть классы, которые дают возможность выводить в консоль(на экран) что-то. Итак, после идет `out`. Чисто логически можно догадаться, что это нечто, что дает конкретный доступ к выводу. После уже идет сама функция/метод `println`. Мы уже говорили про исполняемую функцию. Ее отличие от всех других функций в том, что она выполняется когда мы жмем на кнопку `Run` и хотим выполнить наш код. т.е. где-то там в языке есть метод, который выводит на экран строку и называется `println`. После идут круглые скобки. Посмотрите на нашу `main` функцию. Там то же самое – имя функции и после идут круглые скобки. Все что внутри этих круглых скобок называется аргумент. В нашем случае, у функции `println` аргументом является строка. Точно так же как и с фигурными парными скобками, у нас должны быть парные круглые скобки и конечно же двойные кавычки.

Можем это проверить очень легко и просто. По очереди попробуйте удалить то одну, то другую скобку или кавычки.



Нам повезло. У нас достаточно умный компилятор джавы и он сразу понимает в чем ошибка. Он видит начало строки с двойной кавычки открывающейся и ищет закрывающуюся. Если не находит – говорит нам об этом. Хорошо. А что насчет ; в конце линии? Она тоже нужна? Давайте удалим и посмотрим что будет.

```
Main.java
1 class Main
2 {
3     public static void main(String[] args) {
4         System.out.println("Hello World")
5     }
6 }
```

input

Compilation failed due to following error(s).

Main.java:4: error: ';' expected
System.out.println("Hello World")
^

1 error

Вполне ожидаемо – мы получили ошибку компиляции. Но вы можете возразить – раньше нам не нужны были ;. Да, просто раньше мы не писали исполняемый код. Сначала у нас было объявление класса, которое заканчивалось фигурной скобкой, после у нас было объявление исполняемой функции и только после уже, внутри исполняемой функции мейн сам код вывода в консоль. Мы бы могли написать не Hello World, а нечто подлиннее и тогда бы оно не поместилось бы на одной строке. Как же тогда компилятор поймет где кончается наш код? По точке с запятой. Линией кода является на самом деле то, что в конечном счете заканчивается правильным символом - ;. Чтобы это проверить вам понадобится установка продвинутого текстового редактора. Мы займемся этим позже. А пока давайте поменяем текст, который выводится на экран.

```
Main.java
1 class Main
2 {
3     public static void main(String[] args) {
4         System.out.println("Hello World. Это моя первая программа на Джава!");
5     }
6 }
```

input

Hello World. Это моя первая программа на Джава!

...Program finished with exit code 0
Press ENTER to exit console.□

Т.е. мы должны понять то, что все, что мы напишем между парой двойных кавычек будет выведено на экран.

Итак, подытожим на том, что запомним структуру кода – класс, внутри которого исполняемый мейн метод, внутри которого можно писать все что угодно и этот код будет исполнен.