Method modifiers

1. Access modifiers

Before each method, programmers can specify so-called access modifiers. These include the following keywords: `public`, `protected`, `private`.

These access modifiers let you restrict other classes' access to a method.

For example, if you write the `private` keyword before a method declaration, then the method can only be called from the same class in which it is declared. The `public` keyword allows access to the marked method from any method of any class.

There are a total of 3 such modifiers, but there are 4 types of access to a method. This is because the absence of an access modifier also means something.

	Access from...			
Modifiers	Any class	Child class	Its package	Its class
`public`	Yes	Yes	Yes	Yes
`protected`	No	Yes	Yes	Yes
no modifier	No	No	Yes	Yes
`private`	No	No	No	Yes

1. `public` modifier

A method (or variable, or class) marked with the `public` modifier can be accessed from anywhere in the program. This is the highest degree of openness — there are no restrictions.

2. `private` modifier

A method (or variable, or class) marked with the `private` modifier can only be accessed from the same class where it is declared. For all other classes, the marked method (or variable) is invisible. It's as if it does not exist. This is the highest level of restriction — only its own class.

3. No modifier (default modifier)

If a method (or variable) is not marked with any modifier, then it is considered to have the 'default modifier'. Variables or methods with that modifier (i.e. with none at all) are visible to all classes in the package in which they are declared. And only to them. This modifier is also sometimes called `package-private`, hinting that access to variables and methods is open to the entire package in which their class is located.

4. `protected` modifier

If a method is marked with the `protected` modifier, then it can be accessed from the same class, the same package, and descendants (classes that inherit the class in which the method is declared). We will analyze this topic in more detail in the Java Core quest.

You can use the `public` modifier on all your methods (as well as all your classes and class variables) until you reach the end of the Java Syntax quest. You will need the other modifiers when we start actively learning OOP.

Why are access modifiers needed?

They become necessary for large projects written by tens and hundreds of programmers at the same time.

Sometimes there are situations when a programmer wants to split an excessively large method into parts and move part of the code into helper methods. But at the same time, he or she doesn't want other programmers to call these helper methods, because the corresponding code may not work correctly.

So they came up with these access modifiers. If you mark a helper method with the word **private**, then no code other than your class can see your helper method.

2. `static` keyword

The `static` keyword makes a method static. We'll look at what that means later. For now, just remember a couple of facts about static methods.

Fact 1. A static method is not attached to any object, but instead belongs to the class in which it is declared. To call a static method, you need to write:

```
ClassName.MethodName()
```

Examples of static methods:

	Class name	Static method name
`Thread.sleep()`	`Thread`	`sleep()`
`Math.abs()`	`Math`	`abs()`
`Arrays.sort()`	`Arrays`	`sort()`

The class name before the name of a static method can be omitted if you call the static method from within its class. This is why you don't need to write `Solution` before the names of each of the static methods that are called.

Fact 2. A static method cannot access the non-static methods of its own class. A static method can only access static methods. As a result, we declare all the methods that we want to call from the `main` method static.

Why? You will be able to answer this question yourself when you start learning OOP and understand how static methods work.

3. `throws` keyword

There is another keyword that you've probably seen in a method declaration — the `throws` keyword. Unlike access modifiers and the `static` keyword, this keyword is placed after the method parameters:

```
public static Type name(parameters) throws Exception
{
  method body
}
```

We will consider its precise meaning a little later when we study exceptions.

But to touch on it superficially, we can say that a method marked with the throws keyword can throw errors (exceptions), meaning instances of the `Exception` class (and classes that inherit it). If several different types of

errors can occur in a class, then you need to list each of them separated by commas.

---

4. `main` method

The line where a method is declared, which contains all the modifiers, will affect how this method is called from other classes and methods. It affects the type of result the method will return and indicates what errors are possible as it runs.

Such a line is called a method declaration and has the following general format:

```
access modifier static Type name(parameters) throws exceptions
```

General format of a method declaration

Where `access modifiers` is replaced by `public`, `protected`, `private`, or nothing at all;

if the method is static, then the `static` keyword appears (it is absent for non-static methods)

`Type` is the type of the return value (`void` if there is no result)

Now you probably understand what all the keywords mean in the declaration of the `main` method:

```
public static void main(String[] args) throws Exception
```

Declaring the `main` method

Access to the `main()` method is possible from any class, as indicated by the `public` keyword.

The method is static, so it can be called explicitly as `Solution.main()`.

The `main` method does not return any result. The return type is `void` (no type).

The `main` method takes arguments(!): an array of strings. And the parameter name `args` suggests 'arguments' to our minds. When the program starts, you can pass it arguments — an array of strings. They will be contained in the `args` array in the `main()` method.

Unhandled errors like `Exception` (or its descendants) can occur in the `main()` method.