Methods

+2

Published Oct 21, 2021•**Updated May 22, 2023**

Contribute to Docs

Methods are reusable pieces of code in classes. The difference between a method and a function is that methods are always related to a class or an object. Since in Java there is no possibility of defining logic outside of a class, there are no real functions given this definition. In that case, static methods can be used to have reusable logic without an object instance.

Methods consist of at least the following elements:

- Return type: The type of the value that is returned from the method
- Name: The name of the method

Additionally, they optionally include:

- Parameters: Methods can have no, one, or multiple parameters which consist of a data type and a name. Parameters make it possible to provide values to the method which can be used inside the body of a method.
- Modifier: Modifiers define in which way and from which context a method is callable.
- Exceptions: Exceptions can be thrown by method to make the caller react to unexpected situations (i.e. a non-existing file).
- Body: The body of a method contains all statements the method should execute when being called. The body is only optional in interfaces and abstract classes.

A Minimal Signature

Creating a method called .getOneNumber():

```
int getOneNumber() {
 return 1;
};
```

- Return type: int
- Name: getOneNumber
- Modifier: package private (the default)

Return Type

Methods can return any type listed under data types. Additionally, they can return instances of any class. If a method shouldn't return a value, it has to be defined with the return type void.

Name

The name of a method should describe as much as possible what the method is doing. Ideally, the programmer calling the method knows what the method does without needing to look at the body. By following the rule that a method should only do one thing, it's also easier to give it a concise name.

A valid name must follow the rules:

- It has to start with a letter or an underscore (_) and can contain digits.
- It could also start with a $ but the specification of the language says that this shouldn't be done
- It can't have a name of keyword (i.e. return or while) however keywords can be within the name

Examples for valid method names:

- getObject
- setNumber
- countUntil300
- _isValidCharacter
- GET_MAX_VALUE

Examples for invalid method names:

- 123number (method names mustn’t start with a number)
- get-object (hyphens are not allowed inside a method name)
- sum_number1&number2 (ampersands are not allowed inside a method name)
- assert (method names mustn’t have the name of a keyword)

By convention, method names start with a verb, and each word after the first word starts with a capitalized letter.

Parameters

A parameter is described by a data type and a name. With that name, the parameter can be used to access the value inside the method body. A method without any parameters must have empty parenthesis () after the method name. Multiple parameters have to be separated by a comma ,.

Parameters are the definitions inside the parenthesis of a method while *arguments* are the values provided, when the method is called. The values of the arguments are made available via the parameter names inside the method body.

By convention, a method should have a maximum of three parameters. If it’s necessary to have more than three it makes sense to create an object which is passed and contains the data.

Modifier

Modifiers can change how a method is allowed to be called (public, protected, private, package private), if the method is working on object state or should be executable without creating an object out of a class (static), or if the method is allowed to be replaced by inherited classes (final).

```
public int sum(int number1, int number2) {
 return number1 + number2;
}
```

- Modifier: public
- Return type: int
- Name: sum
- Parameters: int number1, int number2
- Body: return number1 + number2;

Visibility Modifiers

- public: Methods declared as public can be called from everywhere, inside and outside of the object or the class.
- private: Methods declared as private can only be called from inside the object or the class.
- package private: Methods declared as package private can only be called from classes within the same package.
- protected: Methods declared as protected can only be called from inside the class or from inside classes inherited from that class.

Additional Modifiers

`static`

Static methods can be called in classes without creating an object instance out of that class. That's why the object state in those methods can't be accessed with this. The static modifier can be combined with visibility modifiers.

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

- Modifiers: public static
- Return type: void
- Name: main
- Parameters: String[] args
- Body: System.out.println("Hello world");

`final`

The final keyword prevents methods from being overwritten in inherited classes. It can be combined with other modifiers.

Exceptions

Methods can throw Exceptions. An example of this is trying to access a file that doesn't exist. When calling a method, which throws an exception, the calling method has to take care of that exception or has to throw an Exception as well.

Here is an example of how Java throws an exception in a method that could execute a division by 0 and therefore the method has to throw an exception.

```
public float divide(float dividend, float divisor) throws ArithmeticException {
  return dividend / divisor;
}
```

- Modifiers: `public`
- Return type: `float`
- Name: `divide`
- Parameters: `float dividend, float divisor`
- Thrown exception: `throws ArithmeticException`
- Body: `return dividend / divisor`

Body

Everything between `{` and `}` is called the body of the method. The body contains the actual code, statements, or logic that is executed when the method is called.

If the method has a return type there needs to be at least one line in the body which returns a value of that type (i.e. `return a+b;`).

Methods also have access to values of the instance of a class when using the word `this`. With `this` other methods of that same class can be called or instance fields of the class can be accessed which hold values that are accessible through all methods of the class.