The Method

Java's documentation states that:

A *method* declares executable code that can be invoked, passing a fixed number of values as arguments.

Is the method a statement or an expression?

Like some of the abbreviated operators we learned about, a method can be a statement or an expression in some instances.

Any method can be executed as a statement.

A method that returns a value can be used as an expression, or as part of any expression.

What are functions and procedures?

Some programming languages will call a method that returns a value, a function, and a method that doesn't return a value, a procedure.

You'll often hear function and method used interchangeably in Java.

The term procedure is somewhat less common, when applied to Java methods, but you may still hear a method with a void return type, called procedure.

Declaring the Method

So there are quite a few declarations that need to occur as we create a method.

This consists of:

- Declaring Modifiers. These are keywords in Java with special meanings, we've seen `public` and `static` as examples, but there are others.
- Declaring the return type.
  - `void` is a Java keyword meaning no data is returned from a method.
  - Alternatively, the return type can be any primitive data type or class.
  - If a return type is defined, the code block must use at least one return statement, returning a value, of the declared type or comparable type.

Declaring the Method

- Declaring the method name. Lower camel case is recommended for method names.
- Declaring the method parameters in parentheses. A method is not required to have parameters, so a set of empty parentheses would be declared in that case.
- Declaring the method block with opening and closing curly braces. This is also called the method body.

Declaring the Parameters

Parameters are declared as a list of comma-separated specifiers, each of which has a parameter type and a parameter name (or identifier).

Parameter order is important when calling the method.

The calling code must pass arguments to the method, with the same or comparable type, and in the same order, as the declaration.

The calling code must pass the same number of arguments, as the number of parameters declared.

Declaring the Return Type

When declaring a return type:

`void` is a valid return type, and means no data is returned.

Any other return type requires a return statement, in the method code block.

The Return Statement for methods that have a return type

If a method declares a return type, meaning it's not void, then a return type is required at any exit point from the method block.

Consider the method block shown here:

```
public static boolean isTooYoung(int age) {
    if (age < 21 ) {
        return true;
    }
}
```

The Return Statement for methods that have a return type

So in the case of using a return statement in nested code blocks in a method, all possible code segments must result in a value being returned.

The following code demonstrates one way to do this:

```
public static boolean isTooYoung(int age) {
    if (age < 21 ) {
        return true;
    }
    return false;
}
```

The Return Statement for methods that have a return type

One common practice is to declare a default return value at the start of a method, and only have a single return statement from a method, returning that variable, as shown in this example method:

```
public static boolean isTooYoung(int age) {
    boolean result = false;
    if (age < 21 ) {
        result = true;
    }
    return result;
}
```

The Return Statement for methods that have void as the return type

The return statement can return with no value from a method, which is declared with a `void` return type.

In this case, the return statement is optional, but it may be used to terminate execution of the method at some earlier point than the end of the method block, as shown here:

```java
public static void methodDoesSomething(int age) {
    if (age > 21) {
        return;
    }
    // Do more stuff here

}
```

The Method Signature

A method is uniquely defined in a class by its name, and the number and type of parameters that are declared for it.

This is called the method signature.

You can have multiple methods with the same method name, as long as the method signature (meaning the parameters declared) are different.

This will become important later in this section, when we cover overloaded methods.

Default values for parameters

In many languages, methods can be defined with default values, and you can omit passing values for these when calling the method.

But Java doesn't support default values for parameters.

There are work-arounds for this limitation, and we'll be reviewing those at a later date.

But it's important to state again, in Java, the number of arguments you pass, and their type, must match the parameters in the method declaration exactly.

Revisiting the main method

Now, that we're armed with knowledge about methods, we can revisit the main method, and examine it again.

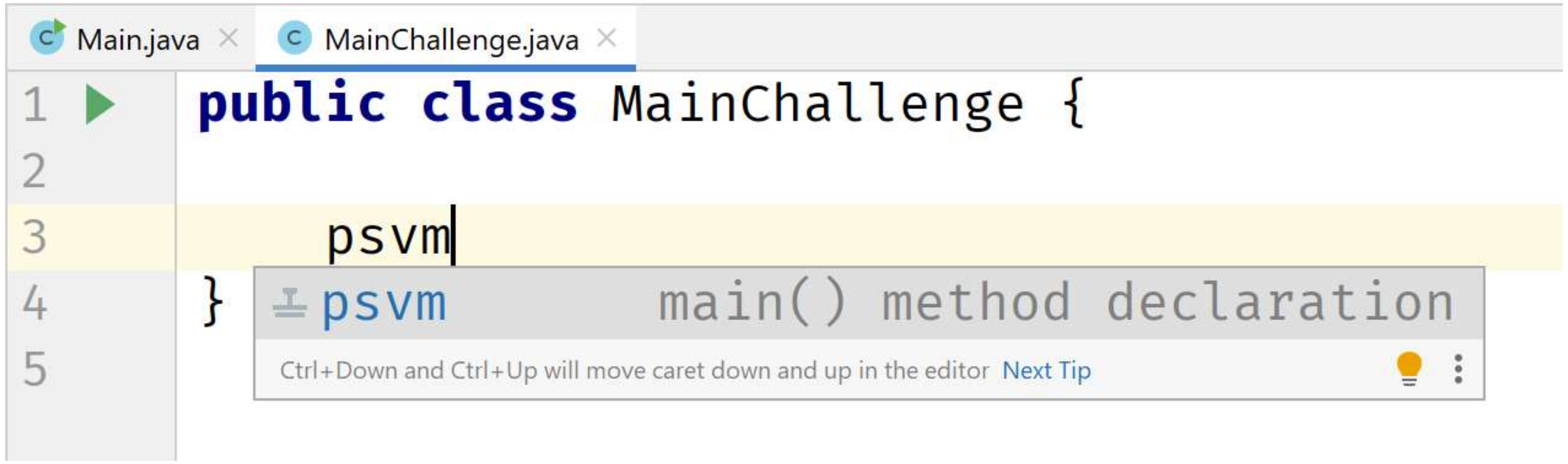
The main method is special in Java, because Java's virtual machine (JVM) looks for the method, with this particular signature, and uses it as the entry point for execution of code.

```
public static void main(String[] args) {
    // code in here
}
```

IntelliJ hint

Finally, in IntelliJ, if you type psvm and hit enter, IntelliJ will insert the main method signature as we show here.

The only reason to memorize this signature, would be if you were taking a certification exam.


Main.java
MainChallenge.java

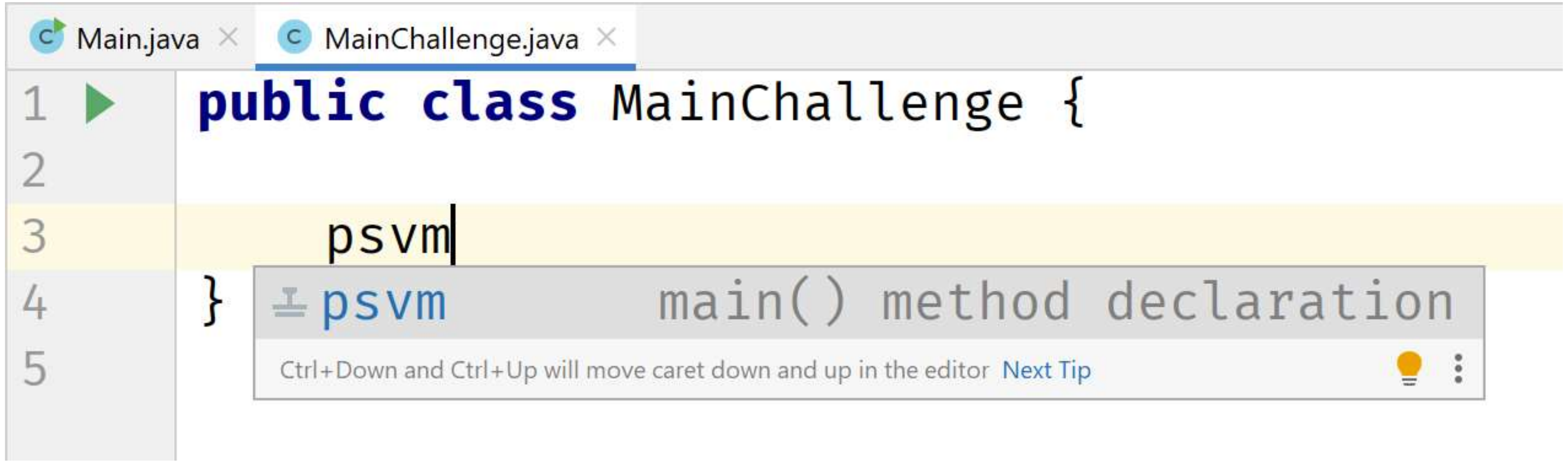
```
public class MainChallenge {

    psvm
}
```

psvm   main() method declaration
Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip

IntelliJ hint

Finally, in IntelliJ, if you type psvm and hit enter, IntelliJ will insert the main method signature as we show here.

The only reason to memorize this signature, would be if you were taking a certification exam.


Main.java
MainChallenge.java

```
public class MainChallenge {

    psvm
}
```

psvm main() method declaration
Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip