Creating and Running Your First Java Application

To get an impression of how IntelliJ IDEA can help you develop and run Java applications, you can start by creating, building and running the age-old "Hello, World" example. By doing so, you'll be able to learn about the IDE features without the need to concentrate on coding.

Before you start

To develop Java applications, you need a Java Development Kit (JDK). So, the first thing to do is to make sure that you have a JDK installed.

If necessary, you can download an Oracle JDK from the Java SE Downloads page. Installation instructions can be found there as well.

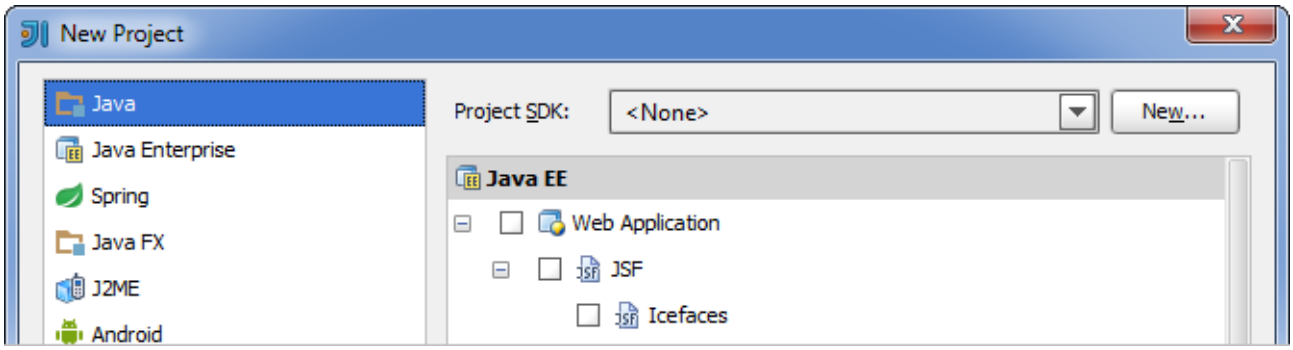
Creating a project

Any new development in IntelliJ IDEA starts with creating a project. So now, we are going to create a project with a name `HelloWorld`.

1. If no project is currently open in IntelliJ IDEA, click **Create New Project** on the Welcome screen. Otherwise, select **New Project** from the **File** menu.

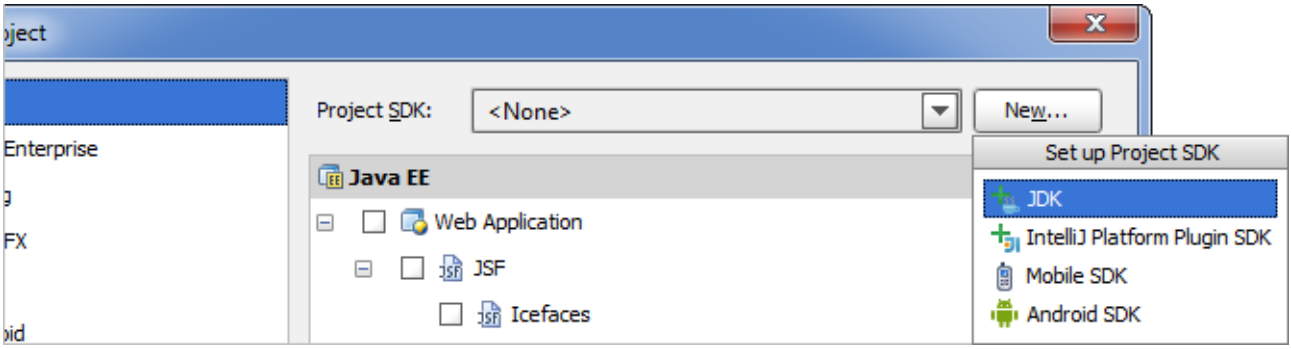
   As a result, the **New Project** wizard opens.

2. In the left-hand pane, select **Java**. (We want a Java-enabled project to be created, or, to be more exact, a project with a Java module.)

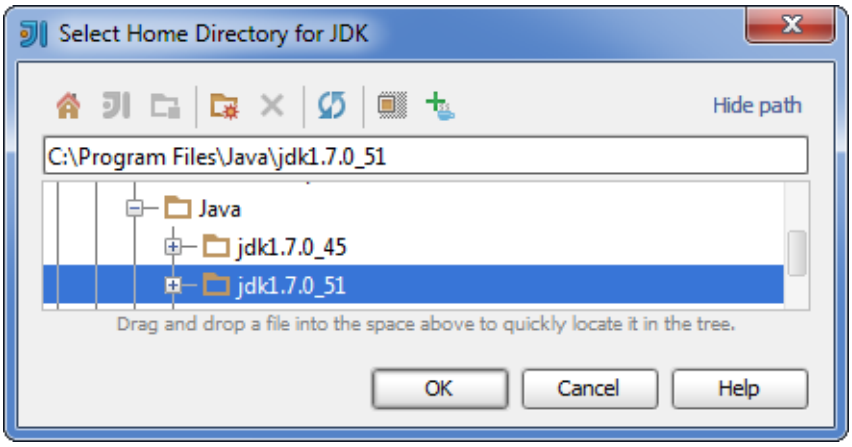


3. If you haven't defined a JDK in IntelliJ IDEA yet (in such a case **<None>** is shown the **Project SDK** field), you should do it now.

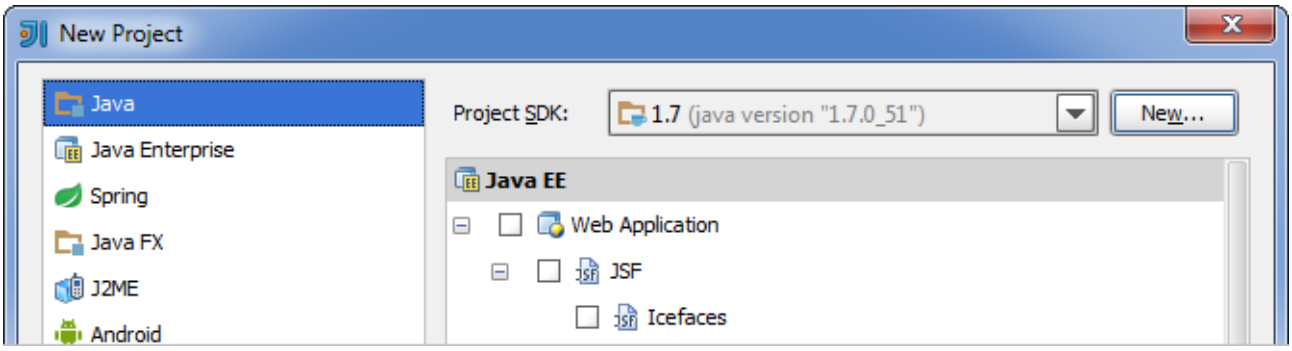
   Click **New** and select **JDK**.



   In the **Select Home Directory for JDK** dialog that opens, select the directory in which you have the desired JDK installed, and click **OK**.



   The JDK you have specified is shown in the **Project SDK** field.



   Note that the specified JDK will be associated by default with all projects and Java modules that you will create later.

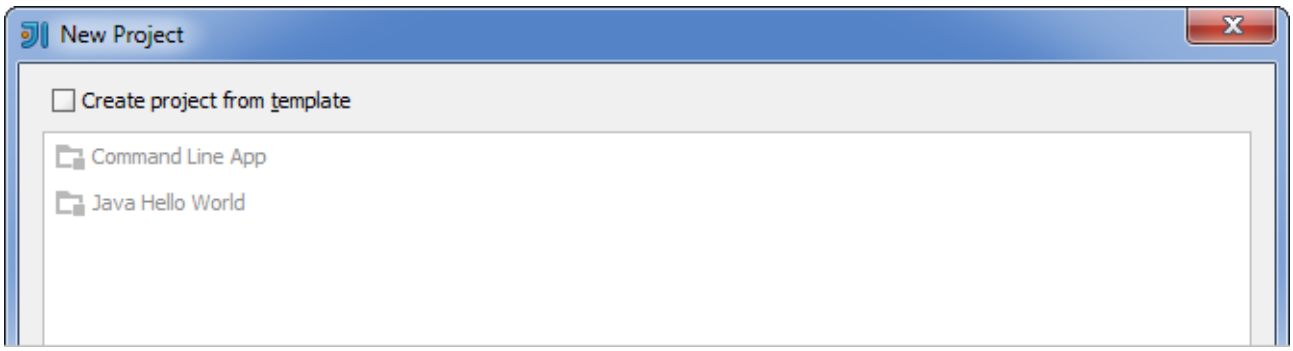
   Because our application is going to be a "plain old Java application", we don't need any "additional" technologies to be supported. So, don't select any of the options under **Java EE**.

   Click **Next**.

4. The options on the next page have to do with creating a Java class with a `main()` method.

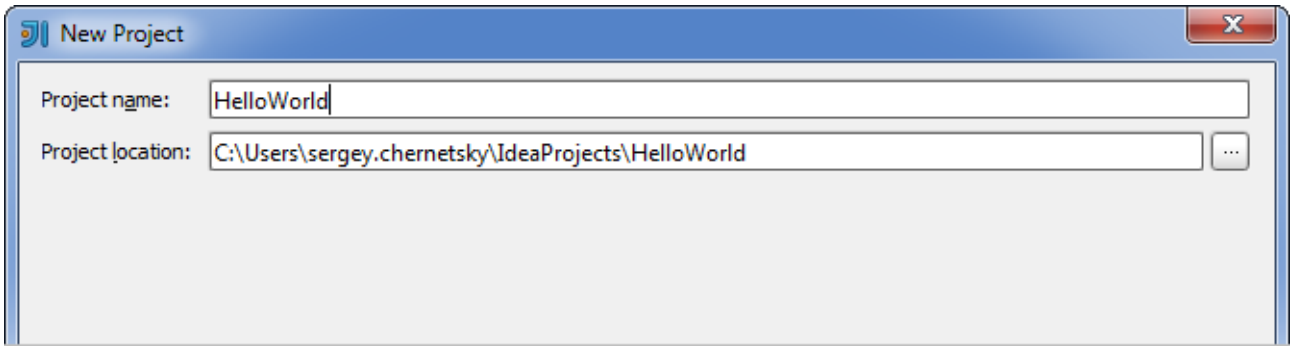
   Since we are going to study the very basics of IntelliJ IDEA, and do everything "from scratch", we don't need these options at the moment. In other circumstances, they may, of course, be very useful.

   So, don't select any of the options.



   Click **Next**.

5. On the next page, in the **Project name** field, type `HelloWorld`.

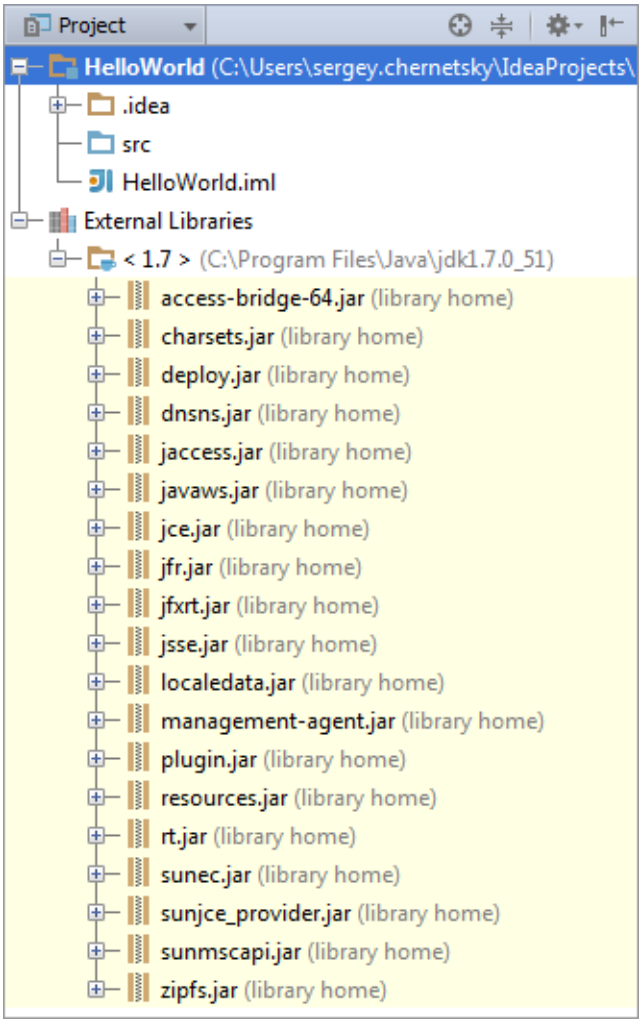


   Click **Finish**.

   Wait while IntelliJ IDEA is creating the project. When this process is complete, the structure of your new project is shown in the **Project** tool window.

Exploring the project structure

Let's take a quick look at the project structure.



There are two top-level nodes:

- **HelloWorld.** This node represents your Java module. The `.idea` folder and the file `HelloWorld.iml` are used to store configuration data for your project and module respectively. The folder `src` is for your source code.
- **External Libraries.** This is a category that represents all the "external" resources necessary for your development work. Currently in this category are the `.jar` files that make up your JDK.

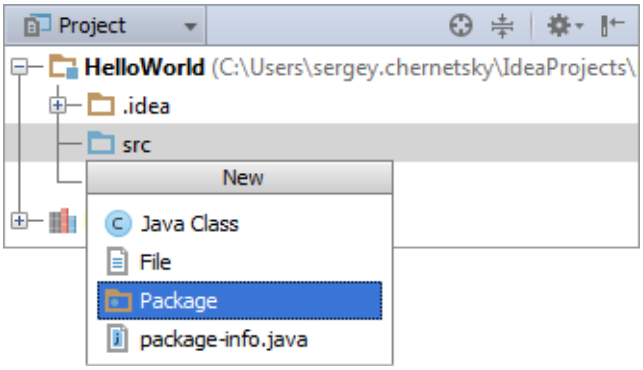
(For more information on tool windows in general and the **Project** tool window in particular, see IntelliJ IDEA Tool Windows and Project Tool Window.)

Creating a package

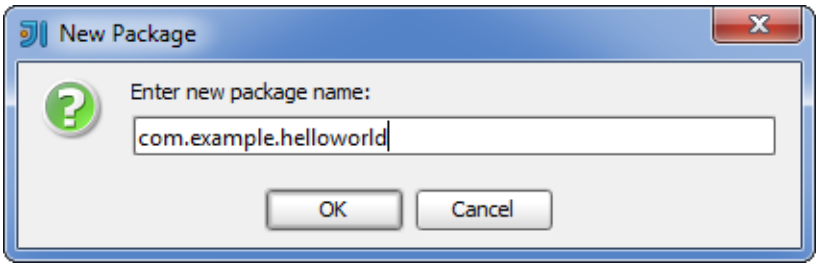
Now we are going to create a package for the `HelloWorld` class. Let the package name be `com.example.helloworld`.

1. In the **Project** tool window, select the `src` folder and press **`Alt+Insert`**. (Alternatively, you can select **File | New**, or **New** from the context menu for the folder `src`.)

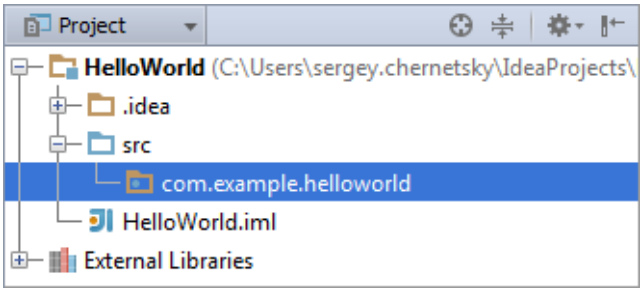
2. In the **New** menu, select **Package**. (Use the `Up` and `Down` arrow keys for moving within the menu, `Enter` for selecting a highlighted element.)



3. In the **New Package** dialog that opens, type the package name (`com.example.helloworld`). Click **OK** (alternatively, press `Enter`).

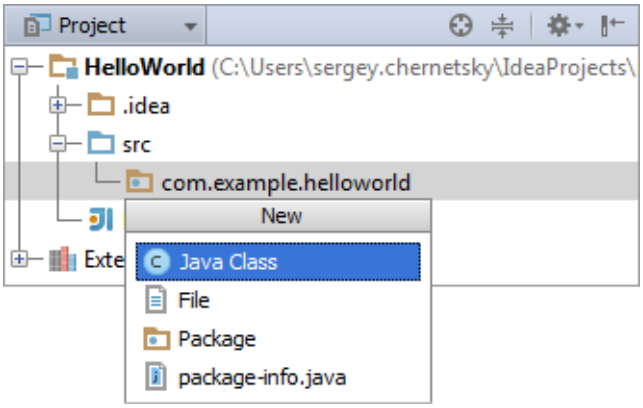


The new package is shown and selected in the **Project** tool window.

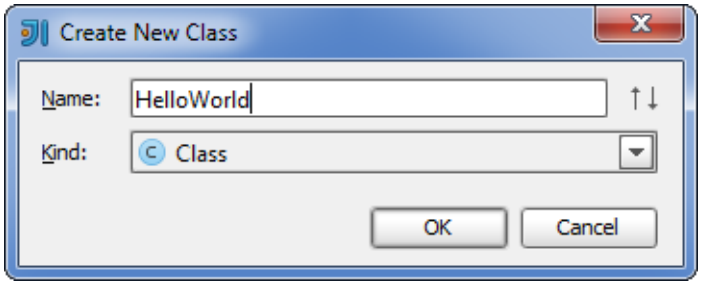


Creating a class

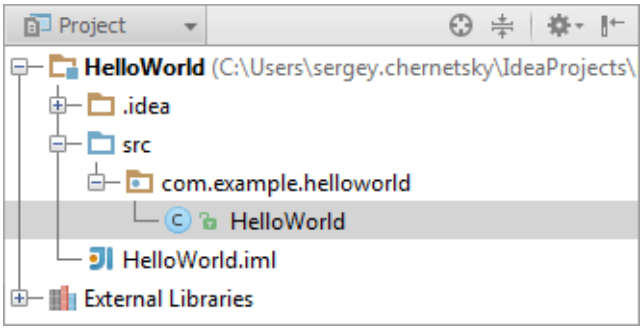
1. Press `Alt+Insert`. The **New** menu is shown; the **Java Class** option is currently highlighted. Press `Enter` to select it.



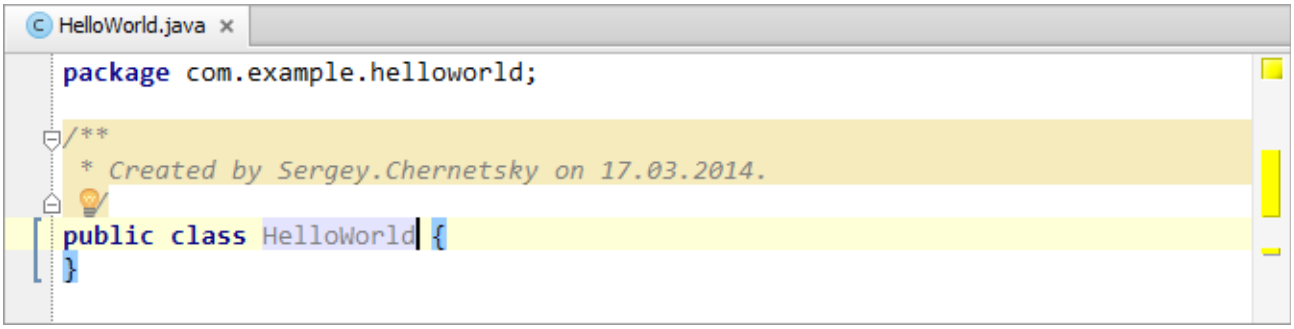
2. In the **Create New Class** dialog that opens, type `HelloWorld` in the **Name** field. The **Class** option selected in the **Kind** list is OK for creating a class. Press `Enter` to create the class and close the dialog.



The `HelloWorld` class is shown in the **Project** tool window.

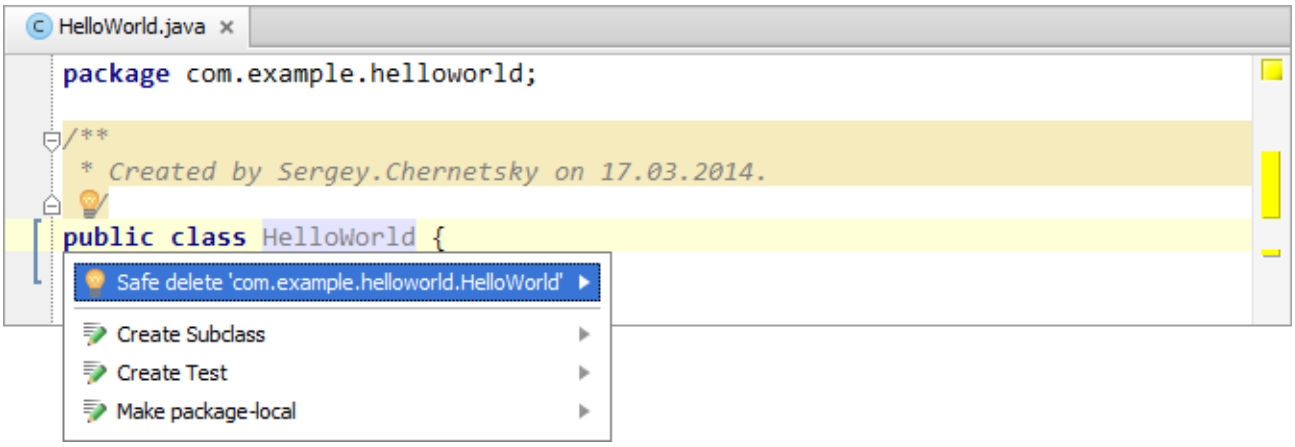


At the same time, the file `HelloWorld.java` (corresponding to this class) is opened in the editor.



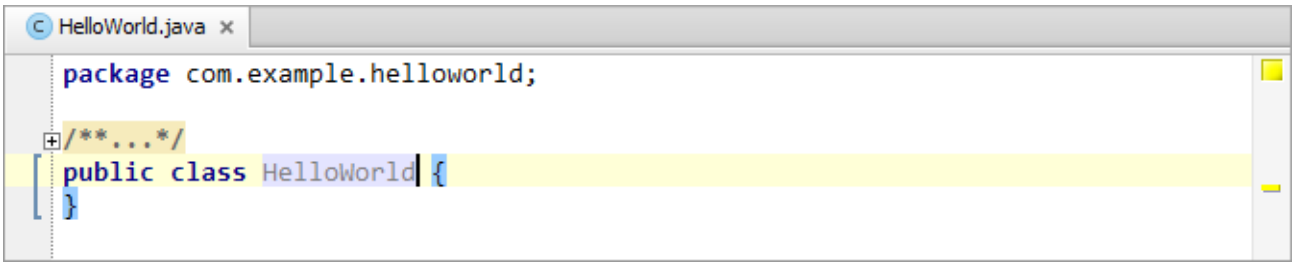
Note the package statement at the beginning of the file and also the class declaration. When creating the class, IntelliJ IDEA used a file template for a Java class. (IntelliJ IDEA provides a number of predefined file templates for creating various file types. For more information, see File and Code Templates.)

Also note a yellow light bulb. This bulb indicates that IntelliJ IDEA has suggestions for the current context. Click the light bulb, or press `Alt+Enter` to see the suggestion list.



At the moment, we are not going to perform any of the actions suggested by IntelliJ IDEA (such actions are called intention actions.) Note, however, that this IntelliJ IDEA feature may sometimes be very useful.

Finally, there are code folding markers next to the commented code block (⊟⊞). Click one of them to collapse the corresponding block. (You can as well place the cursor within the code block, and then use **Ctrl+NumPad -** or **Ctrl+Minus**, or **Ctrl+NumPad+** or **Ctrl+Equals** to collapse or expand the block. For more information, see Code Folding.)



Writing code for the HelloWorld class

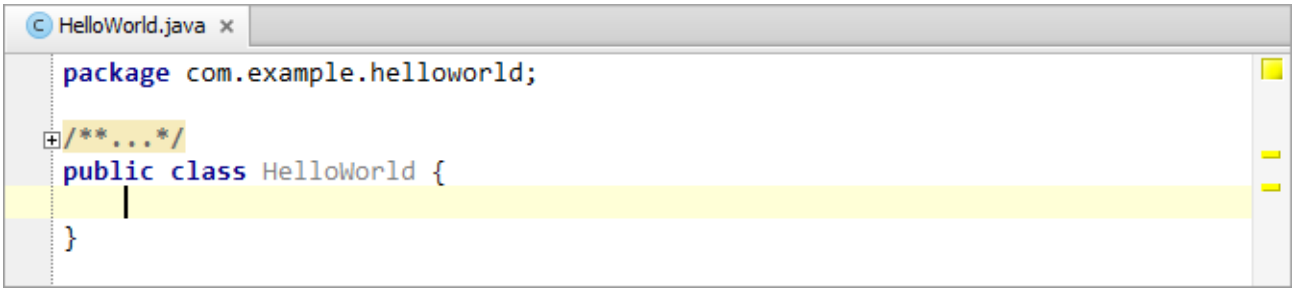
The code in its final state (as you probably know) will look this way:

```
package com.example.helloworld;

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

The package statement and the class declaration are already there. Now we are going to add the missing couple of lines.

Place the cursor at the end of the current line, after { and press **Enter** to start a new line. (To start a new line, in fact, you don't even need to go to a line end. Irrespective of the cursor position, pressing **Shift+Enter** starts a new line keeping the previous line intact.)

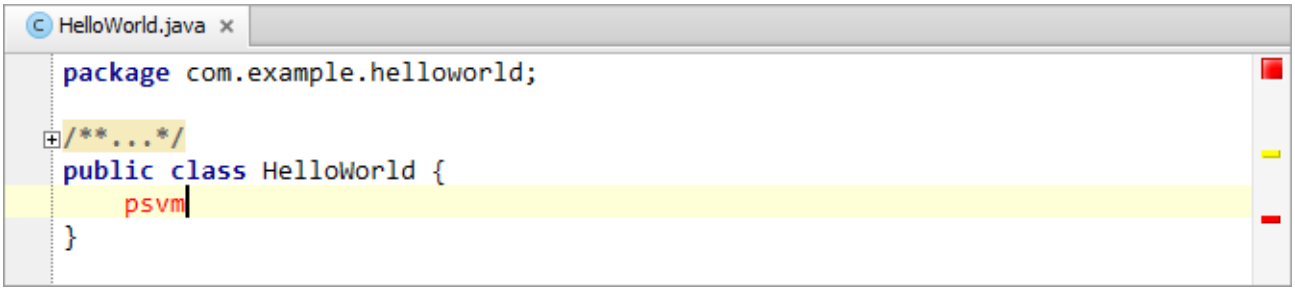


Using a live template for the main() method

The line

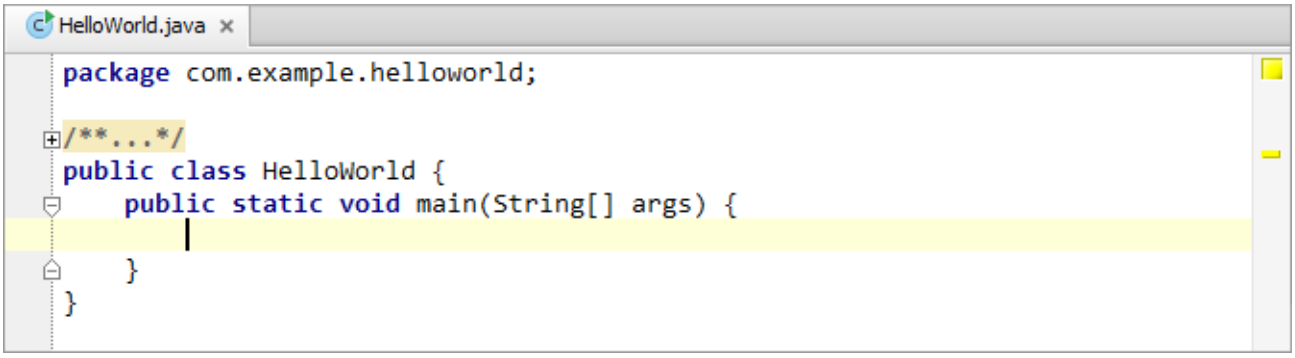
```
public static void main(String[] args) {}
```

may well be typed. However, we suggest that you use a different method. Type `psvm`



and press **Tab**.

Here is the result:



What we have used is a live template-based code generation facility. Live templates used by the corresponding mechanism are code snippets that can be quickly inserted into your code.

A live template has an abbreviation, a string that identifies the template (`psvm` in this example) and an expansion key, the keyboard key to be pressed to insert the snippet into code (`Tab` in this case). For more information, see Live Templates.

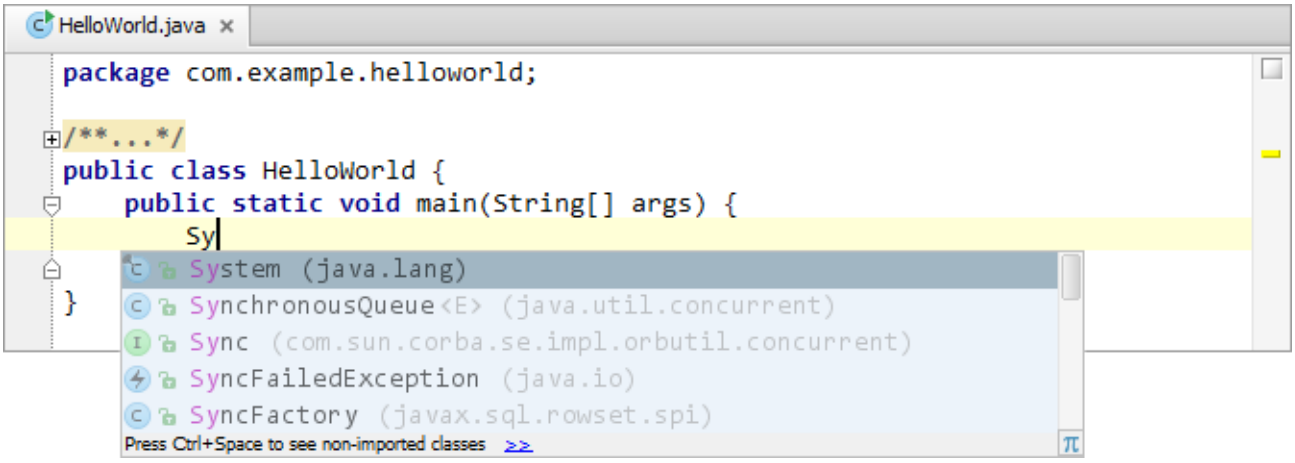
Using code completion

Now, it's time to add the remaining line of code

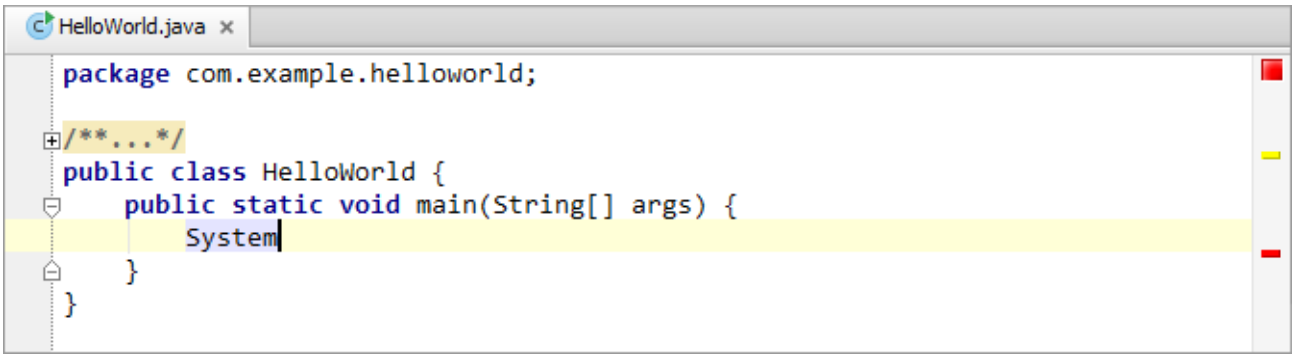
```
System.out.println("Hello, World!");
```

We'll do that using the IntelliJ IDEA code completion. Type `Sy`

The code completion suggestion list is shown.

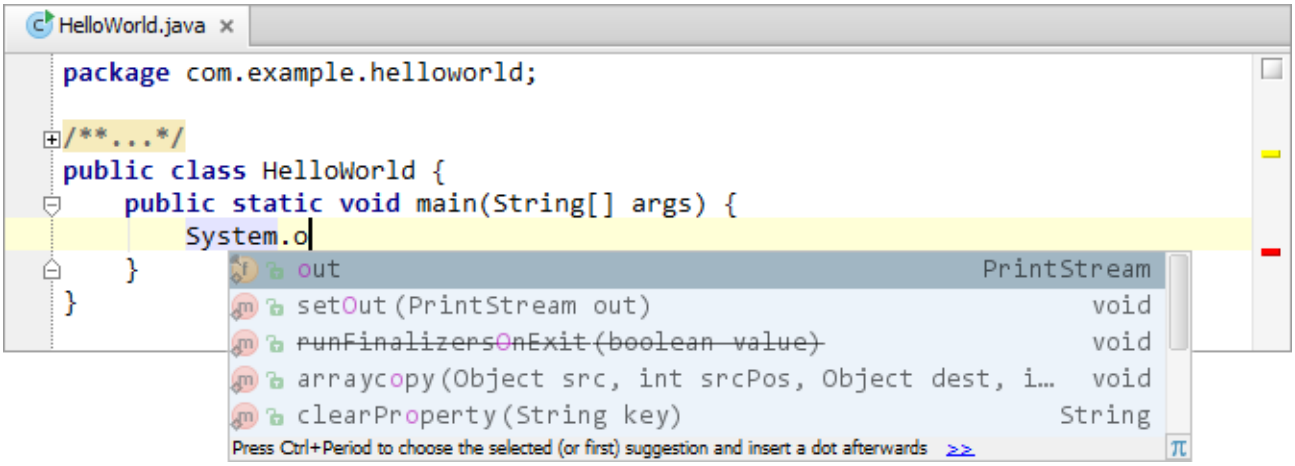


System (java.lang) is highlighted. This is the item that we want. Press `Enter` to select it.

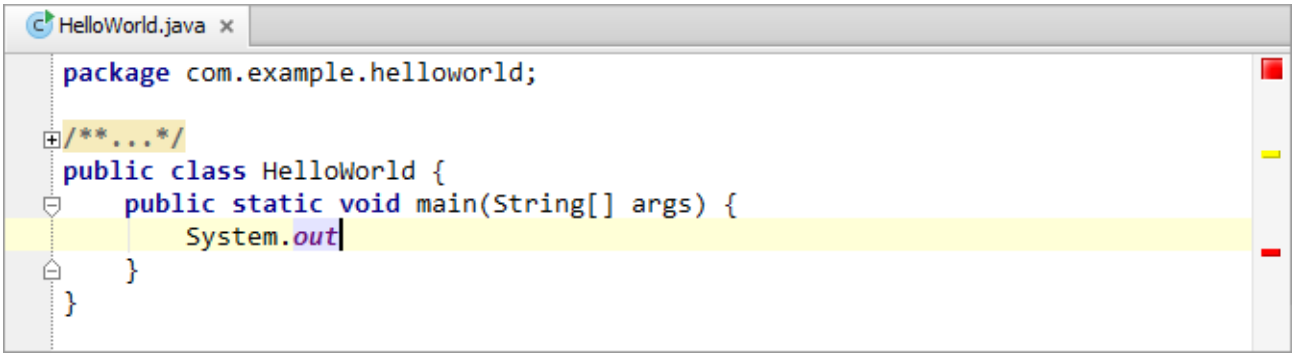


Type `.o`

The suggestion list is shown again.

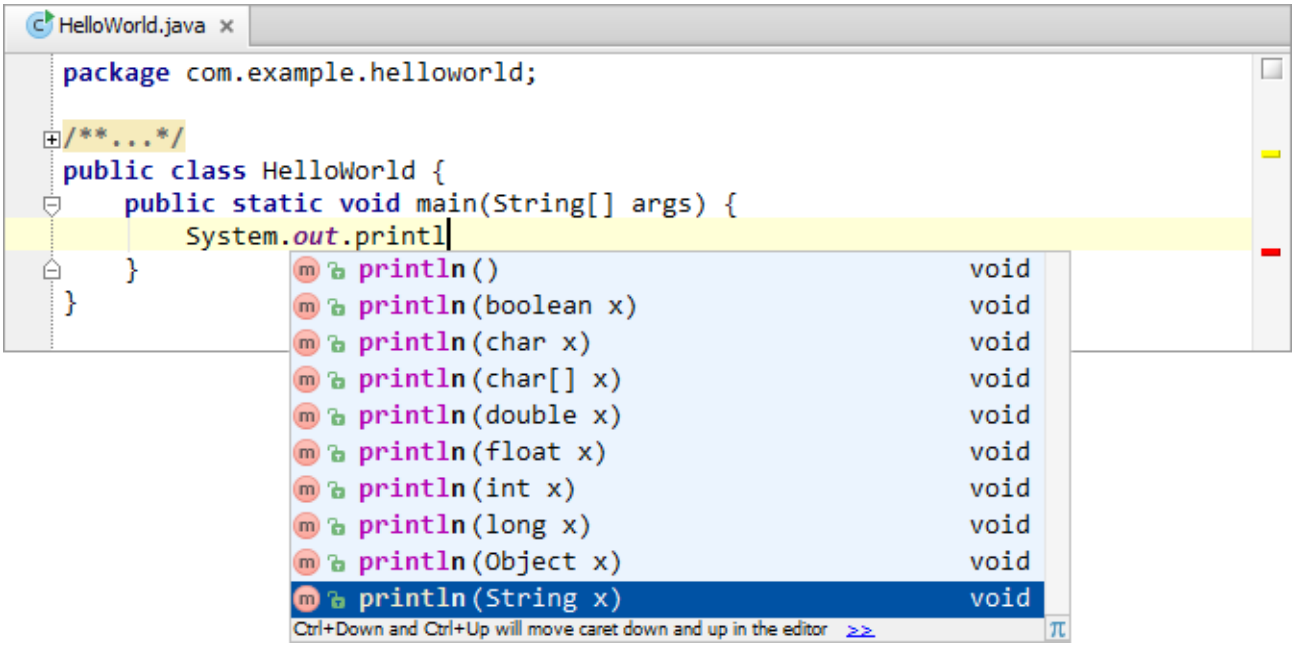


Press `Enter` to select **out**.

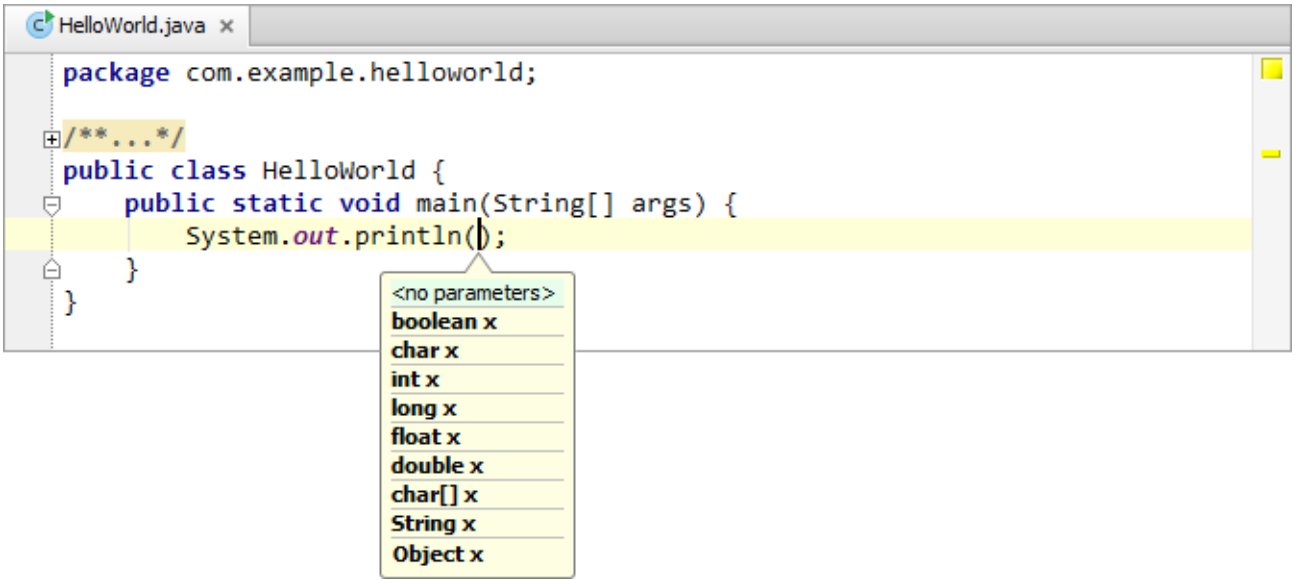


Type `.println`

Note how the suggestion list changes as you type. The method we are looking for is `println(String x)`.



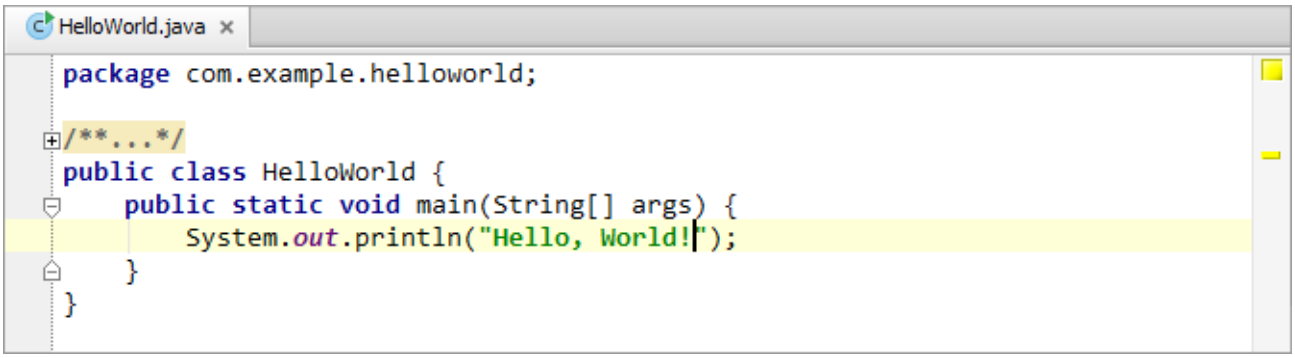
Select **println(String x)**.



IntelliJ IDEA prompts you which parameter types can be used in the current context.

Type "

The second quotation mark is inserted automatically and the cursor is placed between the quotation marks. Type `Hello, World!`



The code at this step is ready.

Using a live template for println()

As a side note, we could have inserted the call to `println()` by using a live template. The abbreviation for the corresponding template is `sout` and the expansion key is **Tab**. You can try using this template as an additional exercise. (If you think that it's enough for live templates, proceed to building the project).

Delete

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

Type `sout` and press **Tab**.

The line

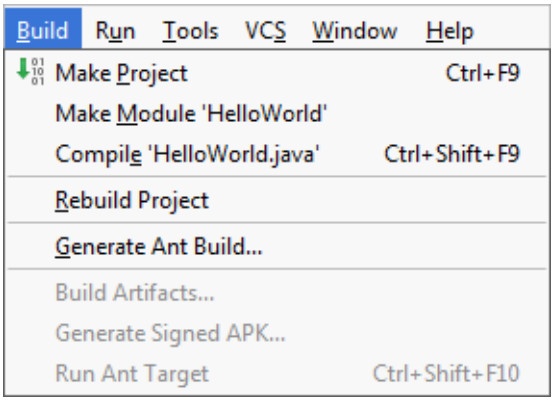
```
System.out.println();
```

is added and the cursor is placed between `(` and `)`.

Type " and then type `Hello, World!`

Building the project

The options for building a project or its parts are in the **Build** menu.



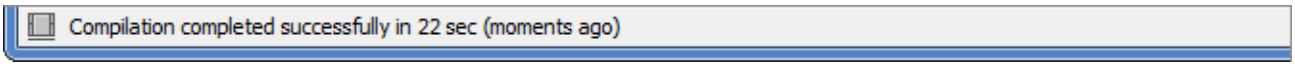
Many of these options are also available as context menu commands in the **Project** tool window, and in the editor for `HelloWorld.java`.

Finally, there is an icon in the upper-right part of the workspace that corresponds to the **Make Project** command ( ).

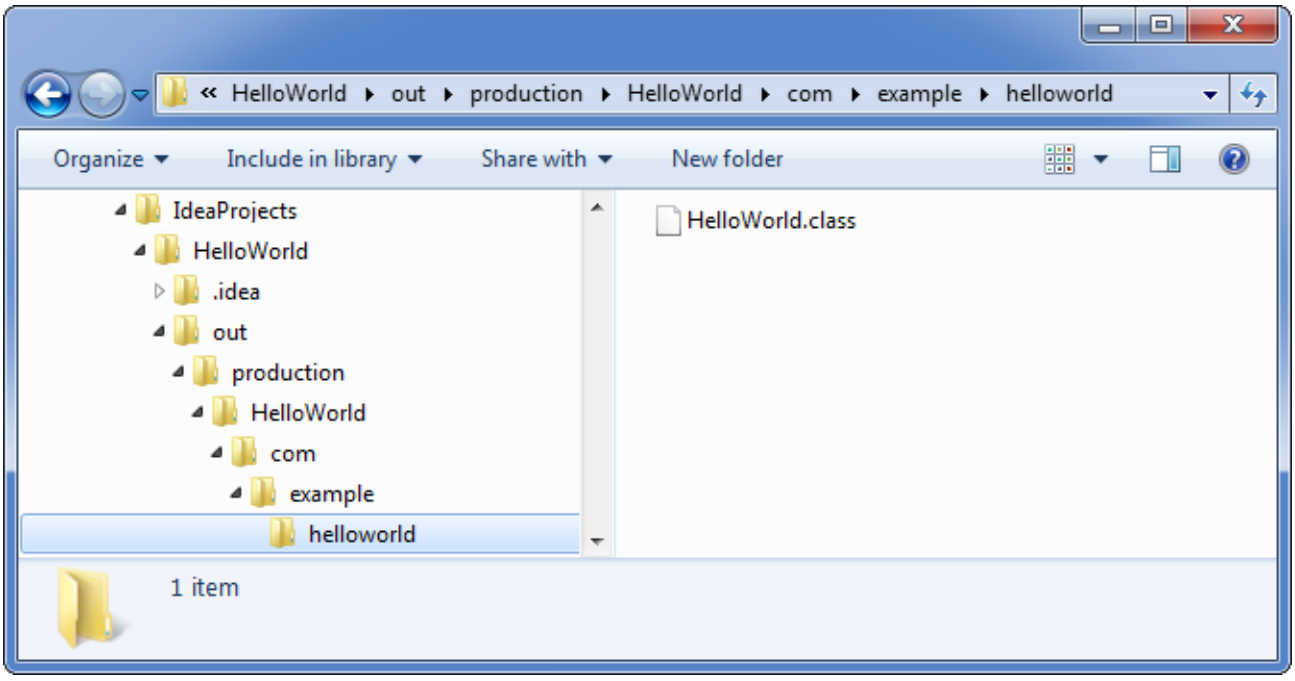
Now, let's build the project. (Building in this particular case is just compiling the Java source file into a class file.)

Select **Build | Make Project**. (The keyboard equivalent for this command is `Ctrl+F9`. Note that this shortcut is shown right in the menu as a useful hint.)

Wait while IntelliJ IDEA is compiling the class. When this process is complete, the corresponding information is shown on the Status bar.



Now, if you look at the module output folder, you'll find the folder structure for the package `com.example.helloworld` there and the file `HelloWorld.class` in the `helloworld` folder. (By default, the module output folder is `<project folder>\out\production\<module name>`; in our case, `<project folder>` and `<module name>` are both `HelloWorld`.)



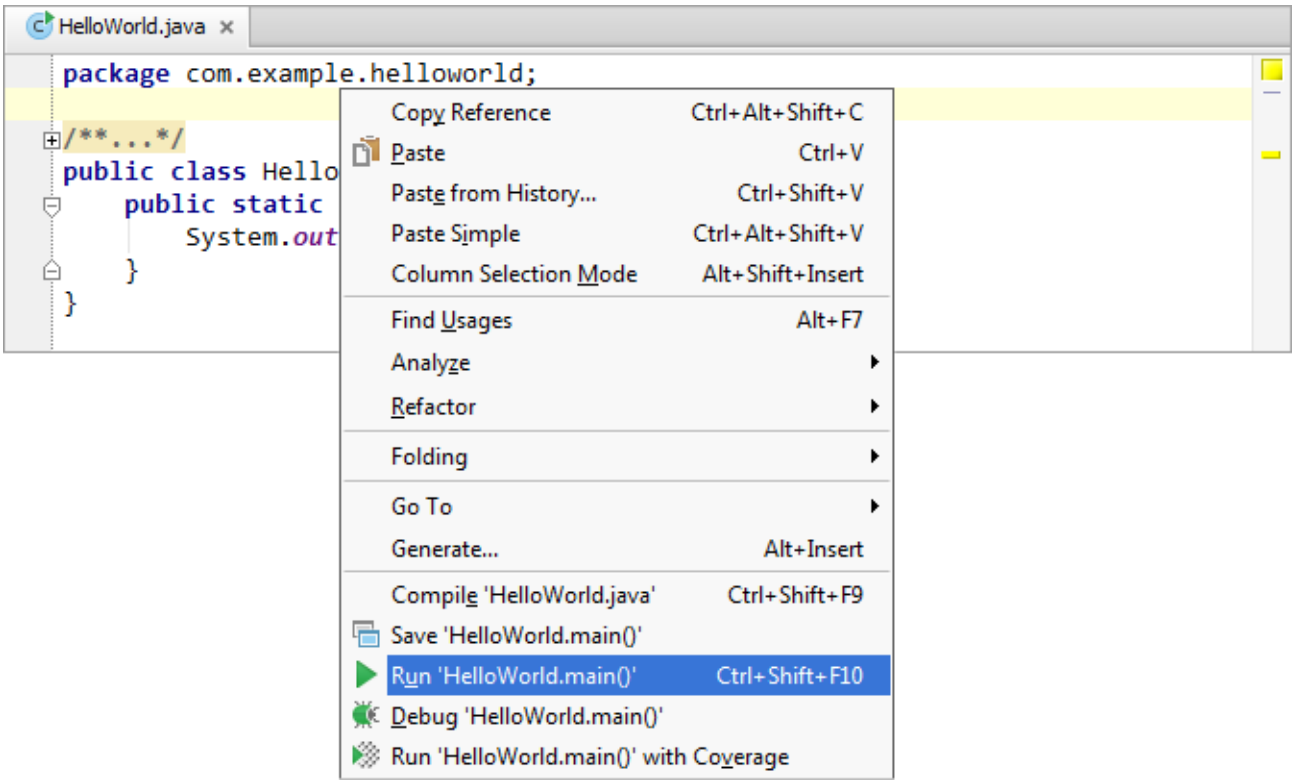
To find out more about building applications, see Build Process, Compilation Types, Configuring Module Compiler Output and Configuring Project Compiler Output.

Running the application

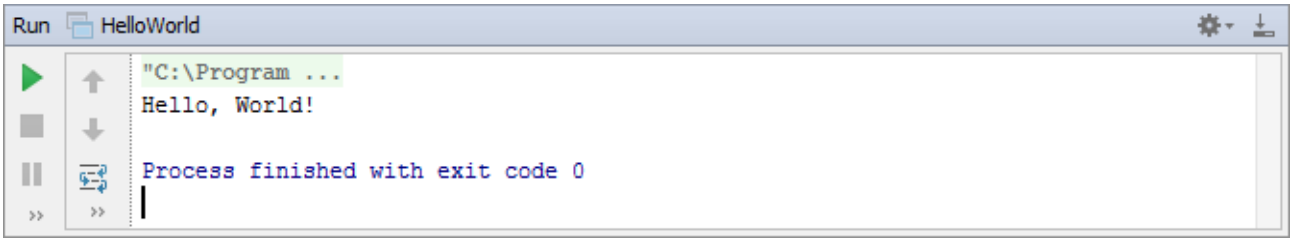
Applications in IntelliJ IDEA are run according to what is called run/debug configurations. Such configurations, generally, should be created prior to running an application. (For more information, see Running, Debugging and Testing.)

In the case of the `HelloWorld` class, there is no need to create a run/debug configuration in advance. The class contains a `main()` method which marks it as a command-line executable. Such classes can be run straight away, right from the editor. For this purpose, the **Run '<ClassName>.main()'** command is provided in the context menu for the class.

So, to run the class, right-click somewhere in the editing area and select **Run 'HelloWorld.main()'**.



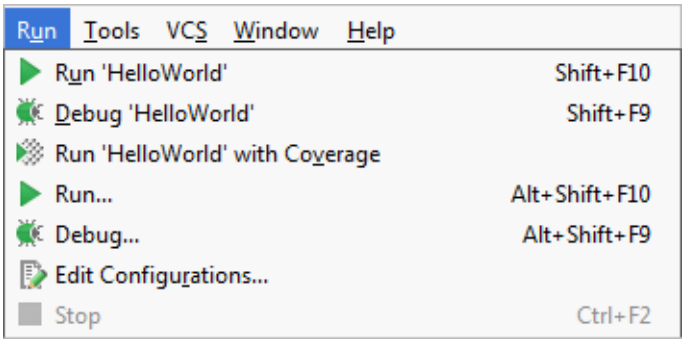
As a result, the **Run** tool window opens at the bottom of the screen. This tool window is responsible for displaying all the output from executed run configurations. (For more information, see Run Tool Window.)



On the first line, a fragment of the command that IntelliJ IDEA used to run the class is shown. (Click the fragment to see the whole command line including all options and arguments.) The last line shows that the process has exited normally, and no infinite loops occurred. And, finally, you see the program output `Hello, World!` between these two lines.

This is, basically, it. However, there are final remarks worth making related to running applications in IntelliJ IDEA:

- The options for running applications can be found in the **Run** menu.

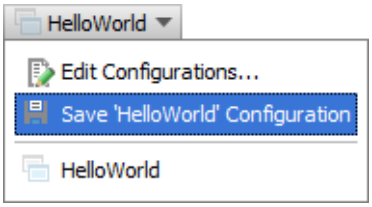


Most of the command names in this menu are self-explanatory. The **Edit Configurations** option provides access to a dedicated dialog for managing run configurations. Also note that keyboard shortcuts (shown right in the menu) are available for most of the commands.

- In the upper-right part of the workspace, there are controls related to running applications. These include the run/debug configuration selector and icons for running applications in various modes.

The configuration selector, obviously, lets you select a run/debug configuration that you want to be used. It also lets you access the run/debug configuration settings (**Edit Configurations**) and perform other tasks related to working with run/debug configurations.

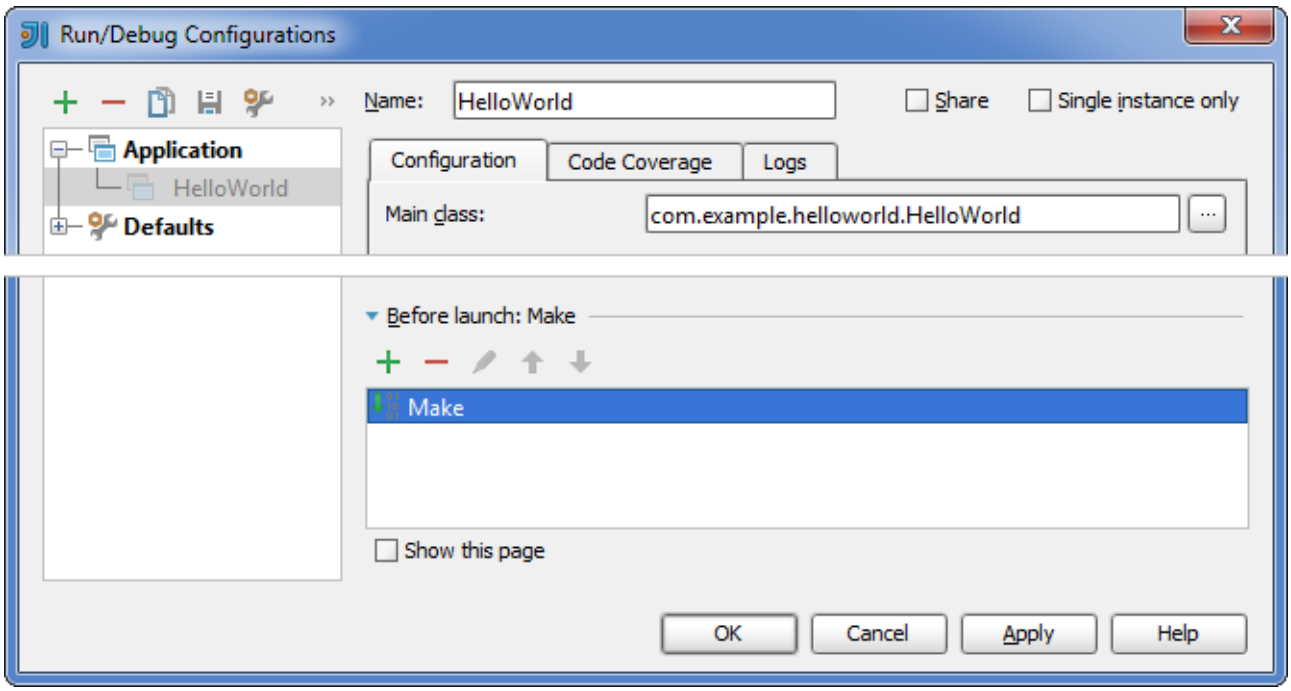
(As a result of running the `HelloWorld` class, a run/debug configuration `HelloWorld` was created as a temporary configuration. You can now save this run configuration to turn it into a permanent one (**Save 'HelloWorld' Configuration**).)



- The options for running applications and for working with run/debug configurations, if appropriate, are also present as context menu commands in the **Project** tool window and the editor.

- Run/debug configurations can do a lot more than just run applications. They can also build applications and perform other useful tasks.

  If you look at the settings for the run configuration that we have used (**Run | Edit Configurations**), you'll see that the **Make** option is included by default in the **Before launch** task list.



  So, you didn't have to perform the compilation as a separate task (**Build | Make Project**). You could have skipped that step and executed the run configuration as soon as the code was ready.