SIT771 Object-Oriented Development

Pass Task 1.1: Hello World

Focus

Make the most of this task by focusing on the following:

- **Process:**

  Focus on how to use the software tools to create and run a dotnet program. You want to be able to create and run programs yourself when you complete this task.

Overview

As your first step, create the classic “Hello World” program. This will help ensure that you have all of the software installed correctly and are ready to move on with creating other programs.

There is a walkthrough of this task in Activity 2: "Your First Program" in Week 1 of Course 1. This includes videos that will show you what you need to do in order to get things working.

As this is your first program, the guidance in this task is very detailed. It includes information that you can use now to understand how to work with the terminal, and you can refer back to this if needed in the future. For this task you can follow along with the videos, as these show you what you need to do. It would be good to read over these details, as they help extend your understanding and give you tools to succeed with later tasks. While future tasks will have relevant details, most future tasks won't have this level of detail.

Submission Details

For this task, you need to make the class Hello World program and submit its code along with a screenshot showing that you got it working.

Submit the following files to OnTrack:

- Hello World source code (the *Program.cs* file)
- A screenshot of your program running.

Check the following things before submitting:

- Your code layout matches the expected format.
- You are using the write case for Classes and Methods (both PascalCase) and variables (camelCase).
- You can demonstrate how to compile and run from the terminal.

Instructions

The first task includes the steps needed for you to install the tools you will need in this unit. You will then use these tools to create the classic "Hello World" program.

Setting up the Project

1. Install the tools you need to get started.

   Check the install instructions for your operating system:

   - Install build tools, VS Code, and Dotnet core for Linux
   - Install xcode tools, VS Code, and Dotnet core for macOS
   - Install MSYS2, VS Code, and Dotnet core for Windows

   > **Note:**
   >
   > You can skip these steps on Deakin lab computers as the software is installed.
   >
   > Remember on Windows that you need to use `mingw64.exe` provided by MSYS2 to access your terminal.

2. If you don't already have one, make a directory (i.e., a *folder*) to store your code (e.g., Documents/Code).

   On a Deakin computer you will need to use one of the following locations: `/c/SIT/Scratch` or `/h/My\ Documents/Code`. (Note the backslash `\` before the space between `My` and `Documents`).

   - Navigate to your Documents directory in *Finder* or *File Explorer*
   - Right-click in the Documents directory and select **New Folder**, name it **Code**

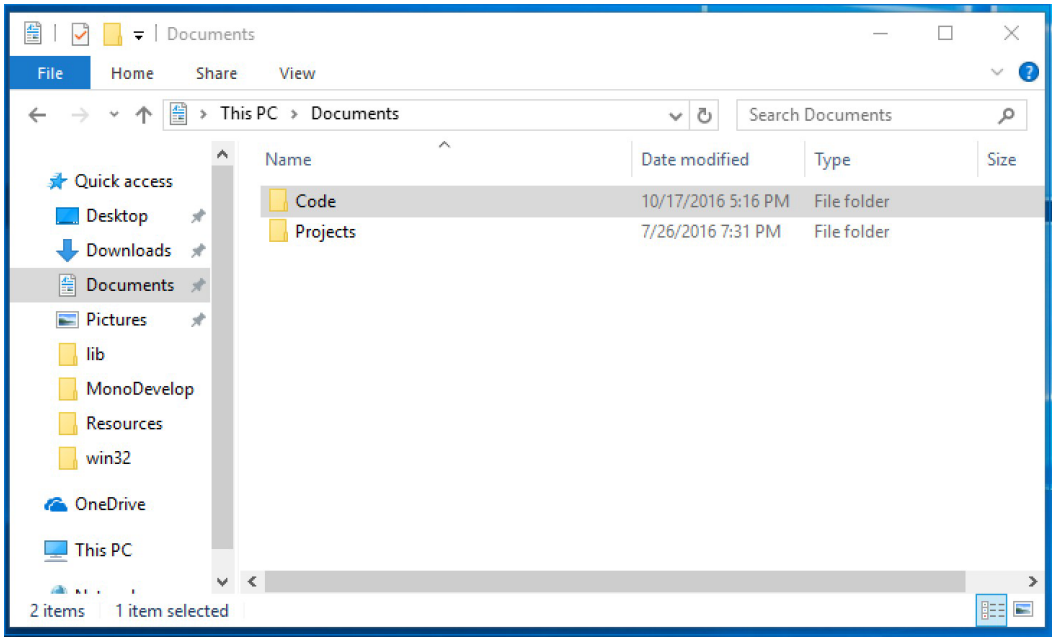


Figure: Windows explorer showing code folder in Documents

Feel free to place this somewhere else on your computer if you want, but please avoid using

spaces in the names of any of the folders. Spaces in names will make it hard to interact with from the Terminal as the terminal uses spaces to separate different parts of its commands.

3. Open your **Terminal** (Using the Mingw app for Windows)

Now we have a folder in place we need to switch to the Terminal to proceed. The Terminal is a program that gives you a command line interface to the computer. You type commands in, press enter, and the Terminal's shell interprets the text you type and performs the actions you requested. Using the terminal and writing programs have many similarities, which makes the Terminal very useful for software developers. As a result there are many advanced programming tools that you can use from the Terminal. For the moment we will stick to the basics, but once you get started there is so much more you can do with this tool.

4. Navigate to your new folder using the **cd** command.

   Within the terminal, your actions are centred upon a **working directory**. This then gives you easy and ready access to the files and other folders/directories that are located within the *working directory*. So, typically the first task you need to do is change the working directory. In this case we need to change into the *Code* directory you created in your *Documents* folder.

   For example, on Windows this would be something like this, to cd into a folder on my C: drive in `Users/andrew/Documents`.

   ```
   cd  /c/Users/andrew/Documents/Code
   ```

   For Mac and Linux, its a little easier as it includes a `~` shortcut to get to your home directory:

   ```
   cd ~/Documents/Code
   ```

   Your terminal is now using this as your working directory. You can check this using the `pwd` command which asks for the "present working directory".

   ```
   pwd
   ```

   Once you are in the right directory we can create a folder for the project and then initialise this with a new dotnet project.

   > File and folder names are case sensitive, so make sure you type this in carefully: `Code` and `code` are two different names!

1. Create a folder and initialise your Hello World program:

   You can create a directory from the command line using the `mkdir` command.

   To create a *HelloWorld* directory/folder within the *Code* folder you can just run `mkdir HelloWorld` as the terminal is currently in the `Code` folder.

   ```
   mkdir HelloWorld
   ```

   > If this doesn't work then check your installation steps, and ask for help on the discussion board. You need to get things working as quickly as possible, so get on to this ASAP.

2. Now change into that directory by typing `cd He` then hit the *tab* key to auto-complete the folder name. Making use of this awesome feature will help save typing and make you more productive. The command will be `cd HelloWorld`. Hit enter to run the command.

   Like with the `mkdir` command above, this is relative to the current working directory. So this will move into the *HelloWorld* folder in the *Code* folder, etc. File and folder names are relative to the current directory if they do not start with a tilde (~) or a forward slash (/). In these contexts, `~` represents your home directory and `/` represents the root (start) of the file system.

   Both `.` and `..` are also special identifiers, `.` represents the current directory and `..` represents the parent of the current directory. So if you ran `cd ..` when you are in the *Code* directory, it would take you back to the *Documents*.

   Run the following command to move into the HelloWorld folder, if you haven't done so already.

   ```
   cd HelloWorld
   ```

3. To setup the project run the following commands in the terminal:

   ```
   skm dotnet new console
   skm dotnet restore
   ```

   This will initialise the project, and restore the project settings. You use `new` to create a project, and `restore` if you have the project but need to restore its settings (such as when you create a new project or copy a project from someone else).

   Run the following command to see the list of files that these commands created:

   ```
   ls -lha
   ```

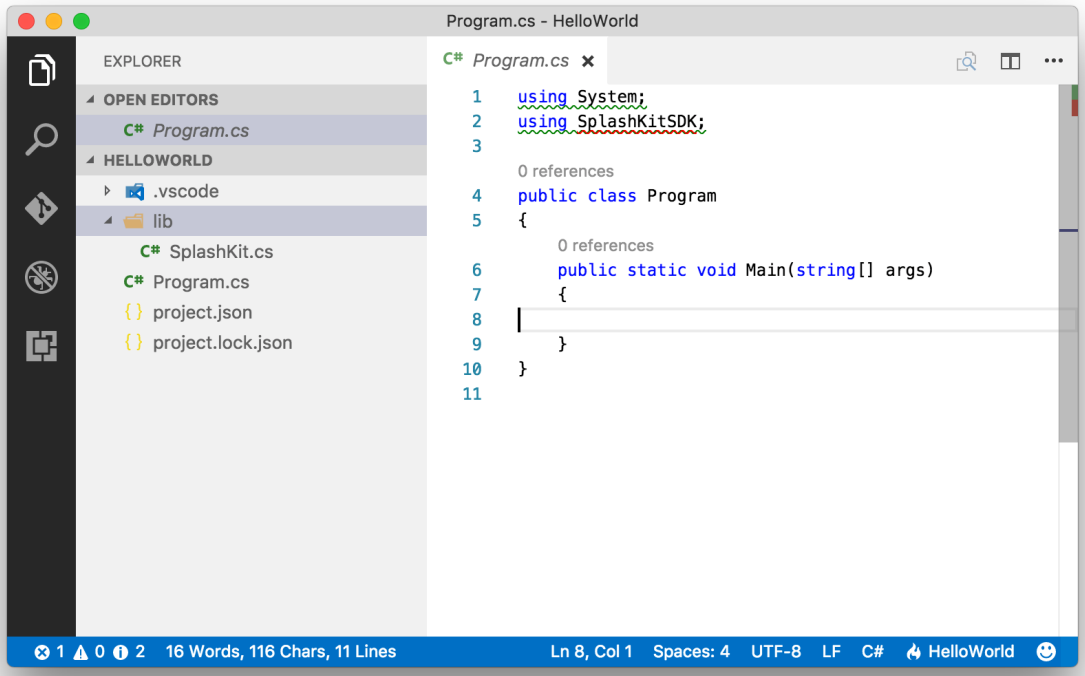
   The `-lha` tells the `ls` (list) command to print the names in a list (`-l`) in human readable format (`-h`) and to show all files (`-a`). You should see a `lib` folder, a `Program.cs` file

as well as the project files.

Writing the Code

1. Open **Visual Studio Code** and within it open your `HelloWorld` folder. The `File` menu should allow you to `Open Folder...` on Windows, or just `Open` on Mac and Linux. You can then use a standard dialog to navigate to and select your `HelloWorld` folder.

   You should see something like the following when this works. You can open the *Program.cs* file by double clicking it in the list on the left.



   On Mac and Linux you should just be able to run "`code .`" and VS Code will open with the current folder (which is "`.`" in the terminal).
2. Update the code to include the statements that will write hello world to the terminal. The code is shown below.

```
using System;
using SplashKitSDK;

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hello World");
    }
}
```

   Like the file and folder names in the Terminal, C# is case sensitive. So take care when typing this in. For example, if you call type `main` instead of `Main` it wont work. This can be a bit of a pain when you get stared, but with care you will be fine.

> **Beware**
>
> Watch out! Copying code from a PDF and pasting it into your own program may result in invalid characters that wont be processed when you upload to OnTrack. Please type these in yourself. This will help you learn, as you want to know how to do this yourself.

3. Save the *Program.cs* file.

   That's it. If you have typed this in correctly you have the code for your first program!

   Notice the color highlighting in the editor, this is known as *syntax highlighting*. Code editors use knowledge of the rules of the programming language (their syntax) to color different words based on their meaning in the language. These highlights help you visualise the structure of your program, and make sure that you don't have small typos in key words.

Compiling the Program

Now that you have the code, you need to get it into a format that the computer can use. There are many different tools that can be used to achieve this, but they can be generally categorised as either *interpreters* or *compilers*. An **interpreter** will read the programs code and then give the computer the instructions it needs to carry out the requested action as it goes. A **compiler** will read all of the program's code and then save the instructions for the computer into a separate *executable* file. When comparing compilers and interpreters, each has their own advantages and disadvantages. In general interpreters offer greater flexibility and can simplify the programming process, but are slower as they need to interpret the code as the program runs. In contrast, compilers are able to generate efficient code but are generally less dynamic.

The C# programming language uses a combined approach with both a compiler and interpreter (known as a runtime in this case). The C# compiler reads your code and produces intermediate code (IL) that it stores in a separate file. This file is then executable by the .NET runtime.

1. Switch back to your Terminal, or open it again and `cd` back into your project's folder.
2. Check you are in the right current directory. You can use the following commands:
    - List the files in this directory using the **ls** command. This will print out the list of files and folders in the directory.

      ```
      ls
      ```

    - Print the working directory using the **pwd** command

      ```
      pwd
      ```

3. Build your program code using the `dotnet` command line tool:

   ```
   skm dotnet build
   ```

4. Run the program using both `skm` and `dotnet`:

   ```
   skm dotnet run
   ```

   Your program should run, and you will see the message "*Hello World*" written to the Terminal.

   > If this all works you should feel confident that things are setup correctly. If you have any issues use the discussion board to get help, as it is important to get these sorted quickly.

5. Now lets try testing out that SplashKit is working correctly. Add in the extra code shown below. This will open a window, draw some text to it, display that text, and delay.

```
using System;
using  SplashKitSDK;

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hello  World");

        Window w = new Window("My First Program", 200, 100);
        w.DrawText("Hello World", Color.Black, 10, 45);
        w.Refresh(60);
        SplashKit.Delay(5000);
    }
}
```

> If this all works you have everything setup correctly! Use the discussion board if you have any issues, as it is important to get these sorted quickly.

Congratulations! You have written your first program. We will look at what all this code means soon, but for the moment let's take a look at how you can submit your work for feedback.

Submitting your work

Check that you are happy with your code, and then you are ready to submit it to OnTrack for feedback.

1. Login, and go back into Tasks 1.1P.
2. Change the status of the task from *Working on It* to **Ready for Feedback**
3. Check what files you will need to upload. In this case, it includes the program's code, as well as a screenshot of the running program.
4. To get a screenshot, you will need to prepare your code and windows. Extend the time for the delay to make sure you have time to get a screenshot. Then make sure to lay out the windows sothat it will be easy to grab the shot.

   Once ready, run your program and use the appropriate tools to get a screenshot.

   - In Windows you can use the Snipping Tool
   - In macOS you can use cmd+shift+4 to select an area for a screenshot.
   - In Ubuntu Linux, you can use Dash
5. Switch back to OnTrack and upload the code and screen shot.
6. Add a comment to your tutor, let them know if there are any things you want them to focus on.
7. Then finally, upload...

The files will be uploaded to the server, which will convert them into a single PDF file that can be included in your portfolio. This file will be shown to your tutor who will review it and get back to you shortly with some feedback.

Well done. You have finished this first task!