# Practical Project: Random Sentences Generator

This is additional practical project and **it is not mandatory and it is not included in the final score**. The main purpose is to use gained knowledge in different type of problems and to improve your portfolio and GitHub skills.

A picture containing text, outdoor, case, accessory

Description automatically generated

This **random sentence generator** is just for fun! These sentences can provide humour and be a cool way to surprise others by sharing a standout sentence on social media platforms and gathering your network's reaction.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

## Create GitHub Repository

Create a **new repository** from <https://github.com/new>. Choose a **meaningful name**, e. g. "RandomSentencesGeneratorByUsername", add a **short description,** and make your repo **public**. Also, **add a** README.md file and .gitignore **for Visual Studio**. Finally, **change the license** to "MIT" and click on the [Create] **button** to **create your repository**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Please choose **your original and unique name** for your project!  Your GitHub profile should be **unique**, not the same as your colleagues'.  You can follow this tutorial, but you can also **make changes** and **implement your project differ** from your colleagues. |

Now your **repository is created** and should look like this:

Graphical user interface, text, application, email

Description automatically generated

Now let's see how to **write the code** of our application.

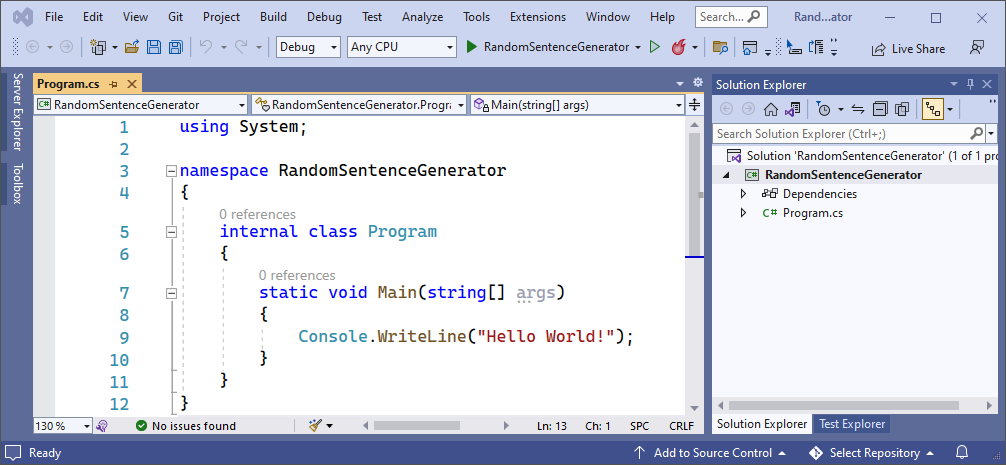
## Write the Sentences Generator Code

Let's create the application and play with it.

### Create a Visual Studio Project

First, we should **start Visual Studio** and **create a new C# console application**. Then, **choose an appropriate name** and a **place to save the project**. On the next screen, choose [.NET 6 (Long-term support)] and create the project.

Our project should be created and should look like this:



Before we continue, let's change the **name** of our main class to something more **meaningful**:

Graphical user interface, text, email

Description automatically generated

### Implement the Generator Logic

Now let's start working on our project.

#### Create the Sentence Model

To create our **sentences** we are going to need: **names**, **places**, **verbs**, **nouns**, **adverbs** and **details**. The **sentence** that we will create is based on the following **model**:

* One sentence needs [Who from where] [Action] [Detail] to be created.
  + "Who from where" example: [Name + from + Place] ("David from London").
  + "Action" example: [Adverb] + [Verb] + [Noun] ("calmly watched the sunset").
  + "Detail" example: "near the river", "at home", "in the park".

#### Add Words for the Sentences

Let's start by creating **arrays** with all the **words** that we are going to use to create a **random** **sentence**. **Arrays** are used to **store** **multiple** values in a **single** **variable**, instead of **declaring** **separate** **variables** for each **value**.

To **declare** an **array**, define its **variable** **type** with **square** **brackets**, do it as follow:



Now let's create our first **array** and call it "names". To fill the **array** we have to use **curly** **brackets**. Inside the **brackets**, write **names**, **separated** by a **comma**. These are some example names that you can use:

|  |
| --- |
| "Peter", "Michell", "Jane", "Steve" |

You array should look like this:



Now we need to create **arrays** with words for "**places**", "**verbs**", "**nouns**", "**adverbs**" and "**details**". Do this by yourself. Here are some **words** you can use:

* **Places:**

|  |
| --- |
| "Sofia", "Plovdiv", "Varna", "Burgas" |

* **Verbs:**

|  |
| --- |
| "eats", "holds", "sees", "plays with", "brings" |

* **Nouns:**

|  |
| --- |
| "stones", "cake", "apple", "laptop", "bikes" |

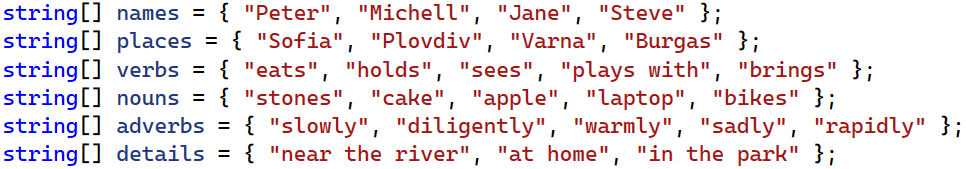
* **Adverbs:**

|  |
| --- |
| "slowly", "diligently", "warmly", "sadly", "rapidly" |

* **Details:**

|  |
| --- |
| "near the river", "at home", "in the park" |

Finally, arrays should look like this:



More information about **arrays**: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/>.

#### Create a Method for Getting a Random Word

Now we are going to create a **method**. Generally, **methods** are useful to **improve** code **reusability** by **reducing** code **duplication**. If we have the same **functionality** to perform in **multiple** **places**, then we can create one **method** with the required **functionality** and reuse it wherever it is **necessary** in the **application**. In our case, the **method** will help us choose **random** **words** every time.

To create a **method,** you need the following things:

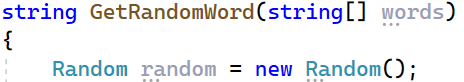
* First, our method should have a **return type** string.
* Second, we need a **name** for the **method**.
* Third, we should define **parameters** that the **method** will receive.

Do it as follow:

A picture containing text

Description automatically generated

Now let's write the method logic. First, we need to create a **variable** from the type Random – you already know how to do that:



Now we should use the Next() **method** of the Random class to **choose a random index**. However, the index should **not be greater than the length of the words array**, so do it like this:



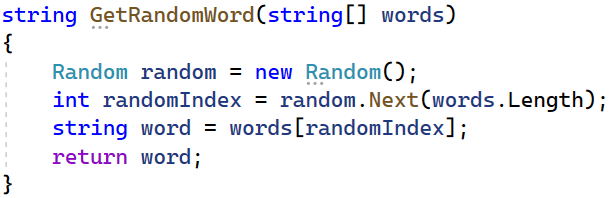
Next thing is to create a **variable** of type string for **our random generated word**. This word will be on the **randomly-generated index** from the words **array**:



The last thing we should do is to **return** our **random** generated **word** to the method:



Now our **method** GetRandomWord() is created and ready to use. It looks like this:



We will learn more about methods in the [Methods/Functions lesson](https://softuni.bg/trainings/3835/programming-fundamentals-september-2022#lesson-44415), but you can read more information about **methods** here: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods>.

It's time for the easy part – let's make the generator work.

#### Implement Generator Logic

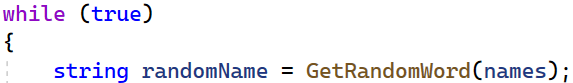
First, we should create an endless while loop. You already know how to do this:

Graphical user interface, text, application, chat or text message

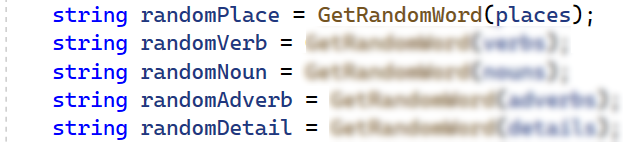
Description automatically generated

Now we should create **variables** for all different **random** **words**. To do this we will use our **method** GetRandomWord(), which will do all the work for us.

First, create a **variable** from the type string and name it "randomName". Make the **variable** keep the result from our GetRandomWord() method and **pass our** words **array** as an **argument** to the method. Do it as follow:



Now try to create **variables** for the other **words** yourself. They should all **pass the necessary arrays** and **keep the results** from the GetRandomWord() **method**. Finally, it should look like this:



Next thing is to **construct** our **random** **sentence**. Remember the **model** that we are working on – first, we need "**Who from where**", then "**Action**" and last "**Details**":

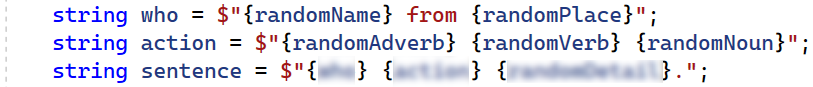
To construct "**Who from where**" we need **[name + from + place]**. Do it like this:



To construct "**Action**" we need **[adverb + verb + noun]**. Do it like this:



We already have our **details** ready so the last thing we should do is **to combine them** in a **sentence**. Use the **model** and try to do it yourself:

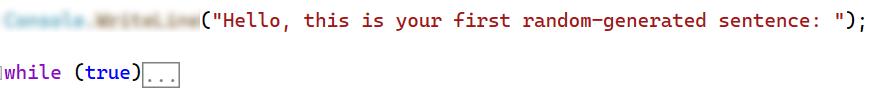


Now what is left is to **write** the **sentence** on the **console**. Next, write a **message** to the user to press [Enter] to **generate** a new **sentence** and **read** his **input**. You know how to do that:

Logo, company name

Description automatically generated

You can also **write** a **greeting** **message** before the while loop:



This is all it takes to **finish** our **project**, after you run it, the generator should look like this:

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Now let's upload it to GitHub.

## Upload Your Project to Github

We already know how to clone our repository by using **Git** **Bash** or **TortoiseGit**.

### Use TortoiseGit (Option 1)

Use **Git** **clone** for cloning with TortoiseGit. Go to the desired directory, **right-click** on a blank space anywhere in the folder and click [Git Clone**]**. Now go to our newly created **repository** and copy the repository's **URL** – you should already know how to do this. The last thing that we should do is to open our TortoiseGit to paste the **URL** and click [OK].

Graphical user interface, application

Description automatically generated

Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**RandomSentencesGeneratorByPeter**" in our case.

**Move your files** from your **old folder** to the **new repo** one. It should look like this:

Graphical user interface, application, PowerPoint

Description automatically generated

Now to **upload** our changes from our working project folder to GitHub.

We can use TortoiseGit's [GitCommit…]. Go to your project's folder, **right-click** on blank space anywhere in the folder and click [Git Commit -> "main"…].

Add an **appropriate** message and click [Add] so you don't miss any files, finally click [Commit].

Graphical user interface, application

Description automatically generated

After that click [Push] and then [OK]:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

This is all you need to **upload your project source code** to your **GitHub repository** using TortoiseGit.

### Use Git Bash (Option 2)

**Alternatively**, use **Git Bash** to **commit** and **push** your local changes to the **repo**.

Go to the desired **directory**, right click on blank space **anywhere** in the folder, select "**Git Bash Here**" to open the Git command line console. If the "**Git Bash Here**" menu is missing, you should first install Git. Type **"**gitclone**"** command followed by the link of your **repository**:

|  |
| --- |
| git clone |

This command is for cloning with **Git Bash**, paste your **repository** **URL** after the command.

Text

Description automatically generated

Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**RandomSentencesGeneratorByPeter**" in our case.

Next thing to do is to add your project files into your cloned repository folder. It should look like this:

Graphical user interface, application, PowerPoint

Description automatically generated

Now we are ready to upload our changes from "**Git Bash clone**". Go to the desired **folder**, right click on blank space anywhere in the folder, select "GitBashHere" and run the following **commands**.

Type the following command:

|  |
| --- |
| git status |

The **git status** command displays the state of the working directory and the **staging area**.

A screenshot of a computer

Description automatically generated

Now type:

|  |
| --- |
| git add . |

This command **adds** all modified files.

Next type:

|  |
| --- |
| git commit -m "Your message here" |

This command**commits** your changes. We also should **add** an appropriate **message**.

Second to the last type.

|  |
| --- |
| git pull |

This command **updates** your local **repository**.

Now the last thing that we should do is to **push** our changes by using the command:

|  |
| --- |
| git push |

This command **pushes** your changes to our local **repository**.

Text

Description automatically generated

This is all you need to **update** your **repository** withGit Bash.

## \* Modify the Code, Write Your Own Features

Now, it's time to **play with the code** and **modify it**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | This is your own project. **Be unique**. Don't be a copy-paster!   * Implement your **own features**. * **Implement the code yourself**, using your own coding style, code formatting, comments, etc. * Make the project **more interesting**. Learn by playing with the code and adding your own changes. |

Below are a few **ideas** of what you can implement or modify as an addition to your code.

### Add More Words

You can think of **more words to add** to make the sentences more interesting and fun.

### Try Different Sentence Structures

You can **change your sentence** and make it more complex:

* You can turn your **sentence to a question**: ["Who" question word/phrase] + [Verb] + [Subject] + [Main Verb] + [Object or Other Information].
* You can add more sentence parts on the right places or change the place of the current ones.
* You can think of more ways to change your sentence.

### Additional Ideas

* Consider a way to create a more **complex sentence generator**.
  + Example of a more complex generator: <http://lomacar.github.io/Random-Sentence-Generator>.
* You can add anything else in your code, based on your own ideas?

### Commit to GitHub

Now **commit and push your code changes** to your GitHub repo!

|  |  |
| --- | --- |
| Icon  Description automatically generated | It is very important to **commit frequently** your code to GitHub. This way you create a **rich commit history** for your project and your GitHub contribution graph is growing:  A picture containing chart  Description automatically generated |

## Write a README.md File

It's highly recommended to provide **documentation as part of your project in GitHub** to describe what the project is **doing**. So, let's make one for this **project**. Let's start by editing the README.md file from our repo at GitHub:

Graphical user interface, text, application, email

Description automatically generated

### Documentation Sections

Add **information** about your project in your README.md file: project goals, technologies used, screenshots, live demo, etc. Typically, you should have the following **sections**:

* **Project title** (should answer the question "What's inside this project?")
* **Project goals** (what problem we solve, e. g. we implement a certain game)
* **Solution** (should describe how we solve the problem 🡪 algorithms, technologies, libraries, frameworks, tools, etc.)
* **Source code link** (give a direct link to your source code)
* **Screenshots** (add screenshots from your project in different scenarios of its usage)
* **Live demo** (add a one-click live demo of your code)

### Use Markdown

Note that the GitHub README.md file is written in the **Markdown language**. Markdown combines text and special formatting tags to describe formatted text documents.

### Project Goals

Start your documentation by describing your **project goals**. What problem does your project solve?

### Sample Documentation

This is an **example** of how you can document your project. Don't copy and paste it!

Graphical user interface, text, application, email

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | **Write the project documentation yourself**. Don't copy and paste it!  This is your **unique GitHub profile** and your own unique project. **Be different** from others. |

Find an **appropriate** **image** and add it. You can add **images** as follows:



### Your Solution

Describe how you **solve the problem**: algorithms, technologies, libraries, frameworks, tools, etc:

Table

Description automatically generated with medium confidence

You can use the [**backtick**](https://en.wikipedia.org/wiki/Backtick)(`) at the **start** and **end** of the **word** to make it **grey**:



You can also use the **double-asterisk** (\*\*) at the **start** and **end** of the word to **bold** it:



### Link to the Source Code

Add a **link** to your **source code** as follows:



### Screenshots

Add **screenshots** of your project:

1. **Take a screenshot** with your favourite tool (e.g. the [Snipping Tool](https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44) in Windows).
2. **Paste** the screenshot in the GitHub Markdown editor, using [Ctrl+V]:

Example screenshots for the "Random Sentences Generator" game:

Text

Description automatically generated

## Upload Your App to Replit

You already should have a **Replit** profile. Now let's add our **project** there so we can share it with our **friends** and add it to our **GitHub** profile. You already should know how to do that.

Open the **menu** in the upper **left corner**. Click [Create], then select the **language** in which your project is **written**, select a name, and **create** the project. If your project is in C#, choose "**Mono C#**":

Graphical user interface, text, application, chat or text message

Description automatically generated

Add your code in "Main.cs" file.

Graphical user interface, text

Description automatically generated

Click [Run] and enjoy your console application.

A picture containing graphical user interface

Description automatically generated

You can now **share** your app with your friends.

## Add Replit Link to Your README.md

Now add a "**one-click live demo**" of your project from your **GitHub** project documentation. You can do it as follows:

Text

Description automatically generated

You can take a **screenshot** from Replit.com and **paste it** into the GitHub documentation editor directly with **[Ctrl+V]**.

This is what it should look like after the changes in your README.md documentation:

Graphical user interface, text, application

Description automatically generated

Now we have completed our **Random Sentences Generator** and we have a new **project** in our GitHub portfolio.