

Story mode

Welcome to the **Story Mode** of this course. At this point, you must've wondered what is the practical application of the knowledge you have learned so far or you are about to learn. How would this **help** you **build** something? Where is it **applicable** and how do you connect the dots that represent the rest of your knowledge? Well this is exactly the kinds of questions we are going to try to answer while going through the story mode. We are calling it a story mode, because the idea behind all this is how we can use every single of the piece of knowledge we get from the lectures and apply it in order to make a whole product and all of this is done on small steps. And each one has its own purpose for the whole "story".

The project we are going to build is a **bash (command interpreter)**, however it has to be with some special functions, in order to be adapted for the needs of **SoftUni**.

This is why we are going to call it **BashSoft**.

Like we mentioned above, the process will essentially consist of many small steps which when put together will result in a real-world application. The very first step we will need to complete is find some way to work with the data currently stored on our computer. After all if we want a bash-like application to have any real use, we would want to give it some data (files) to work with. To provide that data we first need to retrieve it, i.e. reach all the files we would eventually need. How do we go about this? Even a journey of million miles begins with a single step, so let's dive in.

1. Stacks and Queues

First of all we will probably need a way for traversing the file system, and this is where the [Queues](#) from the first lecture "Stacks and Queues" come in handy. There we will learn about a very popular algorithm called [Breadth-first search \(or BFS\)](#), which in our case will give us the ability to traverse not only one folder but each folder in the current subfolders and then each subfolder in each subfolder of the first subfolder etc.

```
--C:\Program Files (x86)\Dropbox\Client
--C:\Program Files (x86)\Dropbox\CrashReports
Access denied!
--C:\Program Files (x86)\Dropbox\OldBinaries
--C:\Program Files (x86)\Dropbox\Update
--C:\Program Files (x86)\Evernote\Evernote
--C:\Program Files (x86)\Google\Chrome
--C:\Program Files (x86)\Google\CrashReports
Access denied!
--C:\Program Files (x86)\Google\Update
--C:\Program Files (x86)\GtkSharp\2.12
```

When we have the ability to traverse the file system, we have a lot of options what to do with the data. One way this can become handy is, when we have to deal with direct file manipulation at later stages of our **BashSoft**.

2. Sets and Dictionaries

Next little step we have to take is start thinking about **speed** of our app. You might think “But we have nothing yet, and we worry about speed?” While true, it’s a *very* good idea to use proper data structures as foundation of our bash in order to avoid rebuilding the whole thing at later stages.

Also, **no one** likes slow programs, so you need to make **fast operations** in your applications and have fast and efficient data structures that hold your data like **Sets and Dictionaries**. We will learn about those in our second lecture. The real strength of using structures like these is probably most obviously shown below

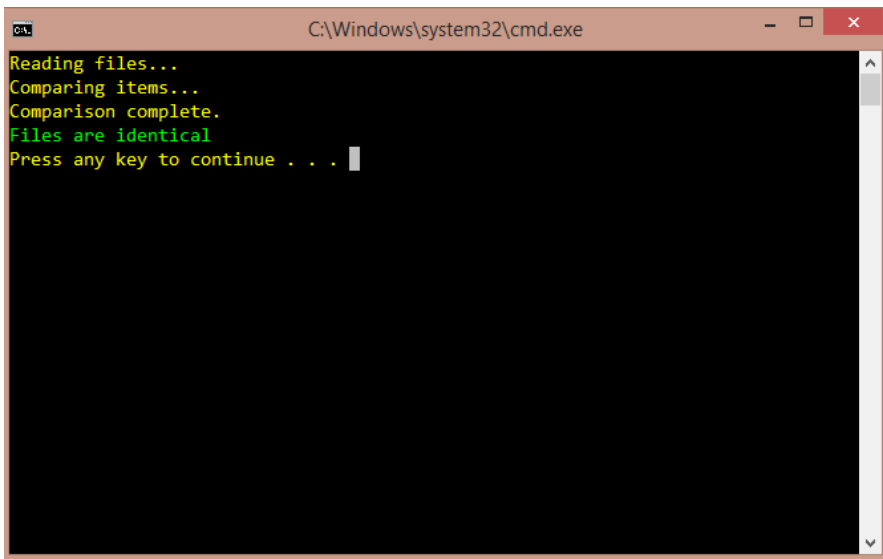
	A	B	C	D
1		seconds	milliseconds	microseconds
2	listTime	10	150	562
3	dictionaryTime	0	2	307

This picture represents what is the speed difference between searching in a List/Dictionary with 50000 values, 50000times. This is why you might have to figure out how these data structures work in order to write your application to run really fast, because otherwise the student using your software are going to have to wait.

3. Files and Directories

So far, we can traverse and **search/store** data. Pretty solid foundation for absolutely any real world application. Next let’s add some custom functionality.

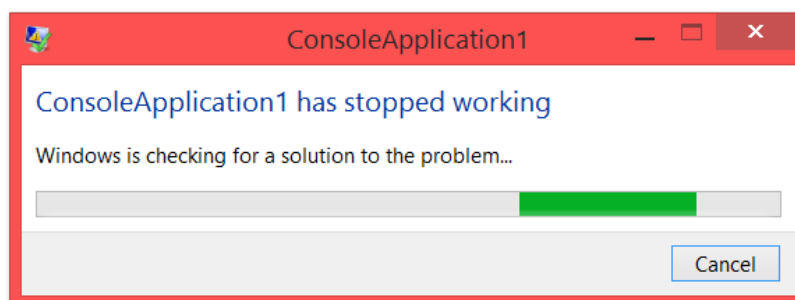
At our third lecture we will discuss the topic of **Files and Directories**. This topic has to do with manipulating files. Reading files, writing in files, creating and deleting files. Pretty important topic when we want to do *anything* with the data stored on our computer. Since we have learned how to retrieve and store data in the previous two lectures now we will start using that data. We will do that by implementing our very own **judge system**. Which will also be the very first actually functional part of our bash application.



4. Exception Handling

By the time we get to our next lecture, **Exception Handling**, we are going to see how many thing can go unexpectedly wrong. If you are writing a bigger application and so then would be the moment to start “**catching**” such **unexpected** events and **handling** them in such a way that the user is not disgusted of the user experience of our application. An example of an **unexpected** event is when we try to open and read the content of folders and files, for which we don’t have rights to.

After all, rarely do we feel good when we see this:



5. Strings And Text Processing

In our next lecture, we are going to learn how to apply the **string manipulation** methods” in order to make our command interpreter for the **BashSoft** so that we can actually make the previous implemented actions, using only commands like **ls** “**path**” for traversing given

folder, "**cmp file1 file2**", **filter Unity excellent take 20** and so on. We will see how important strings are in the connection with the user.

We are going to implement the following operations in our whole BashSoft and after that implement the ones that we currently have and implement later the ones that are yet to come.

BASH:

open file in current folder - **open fileName**

make directory in current folder - **mkdir directoryName**

traverse current directory in some depth - **ls depth**

compare content of two files given their absolute path - **cmp path1 path2**

change current directory given a relative path - **changeDirRel relative path** or **". ."** for returning to the previous folder

change current directory given an absolute path - **changeDir absolutePath**

read students data base from a given file in the current folder - **readDb fileName**

filter students from given course by some criteria and take some of them - **filter courseName excellent/average/poor take 2/10/42/all**

order students from given course ascending/descending and take some of them - **order courseName ascending/descending take 5/17/23/all**

download file - **download pathOfFile**

download file asynchronously - **downloadAsynch pathOfFile**

get help - **help**

6. Regular expressions

Here we are going to use the power of **regex** to easily **filter** the **input data** that is being filled in the data structure, so that we are sure that only entries that **match** a given **format** and therefore are **valid**, **will** be **inserted**. We will see that it is easier to be done with regex, that with strings and after this piece we will have a very strict pattern for the possible user names of students and also the possible names for courses.

7. Functional programming

By this time we should have quite a big project with a lot of functionality, but now it's time to write some helper **functions**, that help the person that calculates the results from given exam and we are going to need you to read the database and **filter** some students by give **criteria**, or sort them in **ascending** or **descending** order. In the end you should have functionality looking pretty much like the following:

```
E:\bojo\Labs\StoryMode\Executor\bin\Debug> order C#_Feb_15 ascending take all
The course you are trying to get does not exist in the data base!
E:\bojo\Labs\StoryMode\Executor\bin\Debug> order Java_May_2015 ascending take all
Ruja16_23 - 41
Ivo42_230 - 57
Ivan23_923 - 75, 43
Ivo12_2341 - 77, 78
Sten16_96 - 29, 100, 28
Kiko23_4144 - 4, 33, 30, 11, 45, 67
Desi12_2001 - 15, 56, 54, 72, 91, 96, 47, 4
Stan21_23 - 45, 56, 71, 61, 33, 76, 98, 15, 65, 91
E:\bojo\Labs\StoryMode\Executor\bin\Debug> _
```

8. LINQ/Streams Api:

In this lecture we are **not going to add** any **new functionality**, however the accent of the whole piece will be to show you how **LINQ/Streams API** **saves us from writing** our functions for **filtering and sorting** the **data structure** and instead we can use them. This way we have **less code**, it is probably **more understandable** and finally it is probably **easier to read and write**.

9. Asynchronous:

I think we have built a great project, but there is one thing we have not implemented and that is the download of files. We will see how we “parse” the HTML of a web page and **Download** all it's **images** and first we are going to do it the slow way and then using asynchronous programming, which we hope to work much faster in our case.

Connecting all the pieces at the end should give us a command interpreter applicable for the Judge server of SoftUni. Good luck on this great adventure.