F# Task

Get started with F#

If you need help to get started with F# have a quick look at following:
https://dotnet.microsoft.com/en-us/learn/languages/fsharp-hello-world-tutorial/create

Your Task

Task

Create a simple command-line banking application in F# that allows users to manage their account balance, make deposits, and withdraw funds.

Language specific features

- Pure Functions: Functions that do not cause side effects and return the same result given the same input.

- Immutable Data: Data that cannot be changed once created, promoting safer and more predictable code.

For more Information about Pure Functions and Immutable Data in F# have a look at the Documentation:
https://learn.microsoft.com/en-us/dotnet/fsharp/tutorials/functional-programming-concepts

Criteria

- Pure Functions: There are separate pure functions for depositing, withdrawing, and checking balance
- Immutable Data: Use Immutable Data to store and manipulate account balances. -> Do not override the same variable again.
- Functionality and User Interaction: The application must continuously prompt the user to deposit, withdraw, check balance, or exit, and it should handle invalid inputs gracefully.
- File Persistence: The balance is saved to a external file and read again if the program is launched again. If there is no such file a new one is created with a balance of 1000.
- Error Handling: The application should handle invalid inputs gracefully and provide appropriate feedback to the user.

Specific Functionality

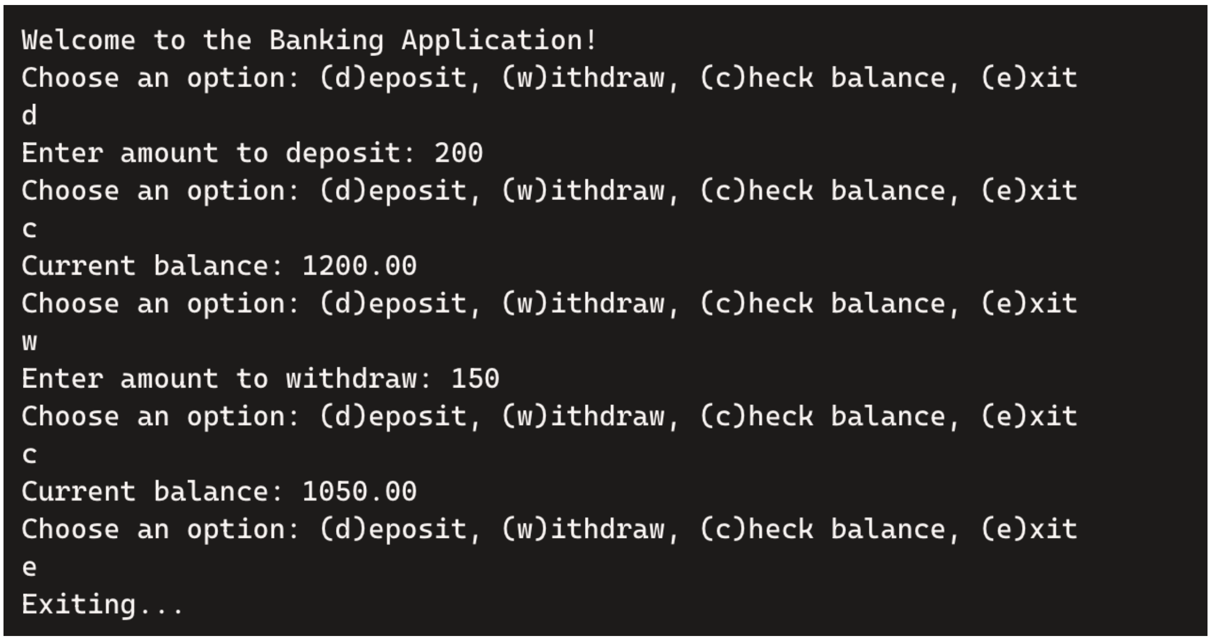
How the Application is Started:
- The application is started from the command line using **dotnet run** after navigating to the project directory.

Amount of Bank Accounts:
- Currently, the application does not support multiple user accounts. It maintains a

single account balance stored in an external file. If you wish to have multiple please explain the extended functionality in a README.md file.

Persistence of Data:
- The balance is saved to an external file, ensuring it is not lost between sessions. If the application stops, it can resume with the previous balance upon restart.

Example execution:



```
Welcome to the Banking Application!
Choose an option: (d)eposit, (w)ithdraw, (c)heck balance, (e)xit
d
Enter amount to deposit: 200
Choose an option: (d)eposit, (w)ithdraw, (c)heck balance, (e)xit
c
Current balance: 1200.00
Choose an option: (d)eposit, (w)ithdraw, (c)heck balance, (e)xit
w
Enter amount to withdraw: 150
Choose an option: (d)eposit, (w)ithdraw, (c)heck balance, (e)xit
c
Current balance: 1050.00
Choose an option: (d)eposit, (w)ithdraw, (c)heck balance, (e)xit
e
Exiting...
```

Deliverables

In a short readme.md file explain:
- How to execute your solution
- Decisions you needed to make for the task (for example why you handled all errors with only one error message and not more specific error messages for each case)
- Provide information about your operating system (MacOS, Linux, Windows..) This way I can test it on the same OS and hopefully don't run into issues because of this...

Put the folder of your command-line application into a ZIP file and hand this in.