Name: Mohamed Omar
GitHub: https://github.com/MohamedMagdyOmar/Geotab-Challenge

Report:

I am going to divide my comments into 2 parts, the first part are the comments that make the solution works (fixing Bugs) with out any refactoring, then comments related to refactoring the code.

I created 2 branches, one that has bugs fixed only, the other branch has the refactoring.

Fixing Bugs:

1. URLs are not correct and need to be fixed.
2. Code is not handling wrong user input, and this includes the following:
    a. User enter wrong character, for example not "r" for random joke, or "c" for category joke.
    b. User enter number of jokes that is below 1 or above 9, or any letters.
    c. Users did not enter FirstName and Last Name Correctly.

Refactoring Code:

1. UI/UX

- The UI is not consistent, sometimes the user has to press Enter after writing his choice, and sometimes he does not need to press Enter. I changed all of them and forced the user to press Enter after the user finished writing his choice to acts as confirmation for his choice.
- There should be clarification for the user about what this console application does, and how to use it.
- When the user choose "Joke by Category", the list of available categories is not shown to the user, so that in this case the user has to guess the category, which is not correct, we have to give him the list of category that we support, and then the user has the choice to select which category.
    - For example, the previous message was "press c to get categories", this message is not clear because the user may ask himself what kind of categories? Is it food categories?

2. Design

- When we are going to write a code, we have to consider at least the following principles:
    - SOLID Principles
    - What Design Pattern May Help
    - Clean Code guide
- If we look at the following role in SOLID Principles:

Problem:

- S -> Single responsibility principle
    - This principle means that every class should have only one responsibility.

        - If we apply this principle on the “program.cs”, we will find that this class has a lot of the business logic, it can “GetRandomJokes”, it can “GetCategories” and so on, so it has many responsibilities, so it breaks the first role.
    - O -> Open Closed Principle
        - Which means that class should be open for extension and closed for modification, if we look at the “program.cs”, if we need to add any new functionality, like new kind of jokes, we have to modify this class, so we are breaking this role.
    - Liskov Principle
        - Which means that child and parent class can be exchanged without problem, but since in this code we do not have any inheritance, so we did not apply and can not see this principle.
    - Interface segregation, Dependency inversion
        - Also, since we did not have any interfaces, or design, so we did not touch these principles

Solution:

- I redesigned the solution and assumes we have 2 kind of jokes, “Random Jokes”, and “Category Jokes”.
- I created “abstract class” called “Jokes”, that will have methods that are common between these 2 kinds of jokes, and if there is kind of jokes that has extra functionality will be added to its corresponding class.
- So now each kind of joke will have only methods that this class can do, so we apply the first principle of “Solid Principles
- If there is new kind of jokes that has appeared in the future, it will inherit from this abstract class, and will expose its functionality, and so we applied the second principle in “SOLID Principles”.
- I did not use interfaces because the solution is simple, and so I did not use dependency injection.
- I used “Builder” pattern to generate steps required for each kind of joke, and “Factory” pattern to decide which kind of jokes the user will proceed with.
- I followed Clean Code rules, especially the one related to “Names” rules, and “Function” rules.
- I noticed the use of a lot of “static” variable, and this is dangerous, especially if we are working with multithreading application that uses Event driven design, because it becomes difficult which thread has made the changes to this variable.

3. Testing

- There was not any unit testing, so I added a sample of unit test that covers the function that has some kind of logic.
- I tried to follow as much as I can the rules of unit testing described in “the art of unit testing” book, like the tests should run quickly, can be executed repeatedly by anyone one the team, use correct naming convention for the test cases, the test should test one functionality at a time,…

4. Logs

- I only add debug logs in case of exception to keep the console clean.

5. Future Enhancements

- We can add “Please Wait” while retrieving the data using API.
- We can add xml file that has strings that are printed in the console, and this xml can be extended in the future to support different languages and use localization to support multiple languages for our application.
- I hardcode the URL, but it will be better to add “config” file and add all URL in the file.
- Add more unit test to make 100 % coverage if possible.
- Allow user to re-enter his input instead of taking default value.
- The commit should be for small chunk of code, not big as I did.