- Use case: Creating the .java files
  1. User provides preference input.
     a. Command-line arguments.
     b. Via prompts
  2. The arguments say to run apptest.json as an espresso test case. The program outputs espresso code into TestFile.java. The espresso goes onto his app gmdice.

- Use case: Translate UI events in a JSON file into an executable JAVA file for a specific testing framework.
  - The user inputs a JSON file containing events, which then is parsed by the program. The events then are turned into commands, which then gets written onto an executable java file for a desired testing framework. Sub-cases can be described for each specific testing framework. However, since they are all very similar to each other, these sub-cases can be understood simply by listing them out below:

    (a) Translate JSON file into JAVA file for Appium
    (b) Translate JSON file into JAVA file for Espresso
    (c) Translate JSON file into JAVA file for Robolectric
    (d) Translate JSON file into JAVA file for Robotium
    (e) Translate JSON file into JAVA file for UIAutomator

- Use case: Input pre-made JAVA file and run it on Appium server
  - The user inputs a JSON file of UI events, which then is parsed. The program then runs an Appium server in the background. The events are then run instrumented on-the-fly through this server, and the user gets visual feedback of the UI events as they are executed on a connected Android device or emulator.

- Edge case: Broken JSON file
  - The user inputs a JSON file that cannot be parsed by the program. In this case, an IOException is thrown from the relevant parse method to the main method, where it will be printed out to the terminal and the program will be halted.

- Edge case: Nonexistent JSON file
  - The user specifies a JSON file that does not exist in the file system. In this case, an IOException is thrown from the parse method (where the file path is called) to the Main method, where it will be printed out the terminal and the program will be halted.

- Edge case: Nonexistent App
  - (a) Nonexistent app for Robotium, Robolectric,
    - The program does not know if inputs are from an existing app. Therefore, it cannot understand if information for a nonexistent app is given.
  - (b) Nonexistent app for Appium

        - The user does not specify a valid app to run along with the Appium server. In this case, the program should throw an IOException with a message indicating that the app cannot be found.

- Edge case: Unknown IDs inside of app
    - The user specifies an ID that cannot be found within the app, so the testing framework cannot perform the needed action in order to test the app. Therefore, the framework will not be able to process the UI event. In this case, we must protect our code by writing “Cannot process action” on the .java file, surrounding our code with try-catch blocks, or use exceptions.
- Edge case: Invalid choice stuff
    - In case the user should choose an invalid choice, then app should let the user know that they have chosen the wrong options, give correct and valid options, and then halt. An alternative would be to use a while loop until the user provides the correct option.