Task

In this assignment, you'll get some practice at generating tests directly from user stories using Cucumber on the coffee maker example. The coffee maker allows you to set up recipes, add ingredients, and to make one of several beverages, provided that you insert enough money. It is an often-used pedagogical example from our colleague Laurie Williams at NC State university. A list of the user stories (requirements) and use cases is available in the .zip/.tgz below.

An initial project is available in the .zip / .tar.gz files below. This project contains a build.gradle file with all of the dependencies necessary to use JaCoCo (used in the next part of the assignment), along with the slightly buggy CoffeeMaker example.

CoffeeMaker_Cucumber_Assign
TGZ File

CoffeeMaker_Cucumber_Assign
ZIP File

Because Cucumber tests are supposed to be integration tests, we have included a CoffeeMakerUI.java file in the edu.ncsu.csc326.coffeemaker package and a set of UI commands in a child package called UI_Cmd that allow user-level interaction with the CoffeeMakerMain. The CoffeeMakerMain class is modal, and responds to different commands depending on the mode of the system. The coffee maker starts in the WAITING mode, and transitions to one of the other modes depending on the user action: {ADD_RECIPE, DELETE_RECIPE, EDIT_RECIPE, ADD_INVENTORY, CHECK_INVENTORY, PURCHASE_BEVERAGE}. These are, for the most part, self-explanatory.

The user interacts with the system using the following set of UI commands (each one is a class in the UI_Cmd directory):

ChooseService: this command allows the user to choose which service of the coffee maker to use. It changes the system mode.

DescribeRecipe: this command allows the user to describe a recipe (it is used in the EditRecipe and AddRecipe modes).

ChooseRecipe: this command allows a user to choose a recipe (it is used in the ChooseRecipe and DeleteRecipe mode).

InsertMoney: this command allows the user to insert money into the machine

Reset: this command aborts the current command and returns to the waiting state.

CheckInventory: this command stores the current inventory as a string that is accessible from the command

AddInventory: this command allows the user to add inventory to the system.

TakeMoneyFromTray: this (somewhat fictitious) command allows a user to retrieve money from the coin return tray, where extra money is deposited after a purchase.

With the CoffeeMakerUI and command classes, we can describe user interactions at the level of the user story tests. In order to see how the commands work, we have provided a simple command line interface in the Main.java file. You can run this application directly in Eclipse. Alternately, after building the project using gradle, you can use the run_me.bat (Windows) or run_me.sh (*nix and Mac) files to run the command line interface to see how the commands interact with the system.

Your job is to write cucumber test scripts that can describe the user story tests. Each test step will likely involve either creating a UI Command and evaluating it, or querying the state of the CoffeeMakerUI class. We have taken the user stories and provided them in an initial CoffeeMaker.feature file that is available in the src/test/resources directory to help you get started.

Download and expand the .zip / .tar.gz file into a directory of your choice, then to import the project into Eclipse, follow the directions to import gradle projects into Eclipse:

Loading a Gradle project into Eclipse
PDF File

If you want to do things from the command line or from another IDE, you can use gradle directly starting from the build.gradle and settings.gradle file. To install gradle, follow the download and install directions from https://gradle.org/ that are appropriate for your platform. However, in this case you are on your own.

Deliverable

Your task is to fill in two files: CoffeeMaker.feature and TestSteps.java, which together test the CoffeeMaker application to ensure it is working properly.

Inside the project, you will find the functional code, a couple of test steps to get you started. The goal is to construct a sufficient number of scenarios to find most of the bugs in the "buggy" version of the coffee maker that is included. You should be able to detect at least 5 bugs in the code using your Cucumber tests.

There should be at least one Cucumber test for each user story in the Coffee Maker web page.

To submit for grading, Create a .zip or .tgz file containing these two files (at the top level - do not add any directories)

How will this be graded?

In order to measure the adequacy of your tests, we will run your tests against each of the bugs in the original model, and a fixed version of the program. The tests should pass on the fixed version and detect the bugs in the buggy version.

You will get half credit for submitting tests which accurately report a working correct implementation. Then, you get incremental credit for each buggy version your tests successfully identify as not correct. There are 10 bugs in the "buggy" program; you get full credit for finding 5 bugs.