SYSTEM UNDER TEST: NOTEPAD APPLICATION

Part 1. Design of properties

Q1. Which parts did you select to write properties for and why are they more likely to be good targets for property-based testing?

- Start Notepad
- Create text file
- Open text file
- Save text file
- Edit the text file
- Close file post edit and save.
- Close file without edit and save.
- Exit Notepad application.

These are the properties that relates to the code functionality written for the Notepad model created and tested.

Q2. Describe all of the properties you wrote. Include the purpose with each property, the code/implementation of the property as well as specific generators or setup code you needed.

- Start Notepad - Notepad program should be opened other than any other text editor program, when the user tries to start the Notepad Application.

- Create text file - Only a new empty file should be opened and user allowed to type other than an existing file. A temporary file should be created with a.txt file format before the user wants to save/exit.

- Open text file - Upon this action, the correct file should be opened and its contents loaded from the storage location.

- Edit text file - Append the file with only appropriate contents. No copy of graphics should be allowed to paste in the file.

  - All the commands under Edit option should be grayed out unless the text is selected to edit.

- Save text file - After the file creation, this property is checked to ensure whether the saved file is in .txt format in the given location.
  - Duplicate check with the same file is performed here.

- Close file post edit and save - This property enables us to check if there has been any edit action performed on the existing file and its contents has been loaded properly after append and saved in .txt format with the same file name in the given storage location.

- Close file without edit and save - This is checked so that any file that is left in open state just after creation and edit/save action is not performed, the temporary file should be deleted from the storage location.

- Exit Notepad application - This property is verified only when the user clicks on the "X" button at the top left corner of the application and the application is not in running state.
  - Any files in open state which has been new should be removed and existing files with any new contents appended will not be added to the existing file but saved under the same location.

Part 2. Manual mutation testing

Q3. Describe the mutations that you had to do and which property(ies) each mutation triggered as well as the output of the tool

Code snippets shown for each property.

Create File - This property has been triggered to create only a new file with empty contents and temporarily stored in the given file location.	@RunWith(JUnitQuickcheck.class)
	public class NotepadProperties {
	
	@ Property (trials = 5)
	public void createNewFile() {
	System.out.println("User creating File: ");
	NotePadModel notePadModel = new
	NotePadModel();
	notePadModel.e_Create();
	assertEquals(FILE_NAME,
	Files.exists(Paths.get(fileName))); }

Open and Edit File - These two properties have been combined such that only an existing file is opened and while editing unless the text is selected the commands under edit will not be highlighted.	@Property(trials = 5)
	public void EditFileProperty() {
	System.out.println("Checking for edit and save
	operation: ");
	NotePadModel notePadModel = new
	NotePadModel();
	notePadModel.e_SaveEdit();
	assertEquals(textToAppend,
	Files.exists(Paths.get(fileName))); }

Save File - Code has been enhanced to check if the file is saved in proper format	@Property(trials = 5) public void SaveFileProperty() {

after each action and to check duplicate also.
 	
This will be called even for Postedit and save operation also.
	System.out.println("Checking for File save: ");
	NotePadModel notePadModel = new NotePadModel();
	notePadModel.e_Save();
	assertEquals(FILE_NAME, notepadAdapter.readFile("FILE_NAME"));}
	
	check if file exists
	
	@Property(trials = 5)
	public static boolean isFilenameValid(String file) {
	File f = new File(file);
	try {
	f.getCanonicalPath();
	return true;
	} catch (IOException e) {
	return false; }

Exit Application - All temporary unsaved files are deleted and cleared from the location.	@Property(trials = 5)
	public void CloseFileProperty() {
	System.out.println("Checking for Close Application: ");
	NotePadModel notePadModel = new NotePadModel();
	notePadModel. e_Postclose();
	File temp = File.createTempFile("myTempFile", ".txt");
	temp.deleteOnExit(); }

Q4. For at least 3 properties/mutations, analyse to what extent the output from the PBT tool would have helped you find the problems and debug it.

Create file, Edit file and Save file - These properties have been chosen because these are the minimum basic actions that can be performed on a file. Any one error scenario would help us to analyze the code and refactor.

No failure cases were generated on 5 trials when the property based test case was run against the functionality, had there been a model which involved more of mathematical calculations, lot of test cases would have been randomly generated.

Property Based Testing has helped me to think over the code multiple times with the correctness of functionality and robustness. If any possible input is not supported, that has to be considered and included into the test code.

The framework will make a point of testing edge cases, and then it'll randomly generate a hundred possibilities. This has helped to save from writing all edge-case tests, which saves repetition in the test code without repetition but with extreme thoroughness.I felt Property-based tests are best combined with example-based tests.