Test Harness Overview

By Peter Mitchell

Contents

1 Area for Improvement

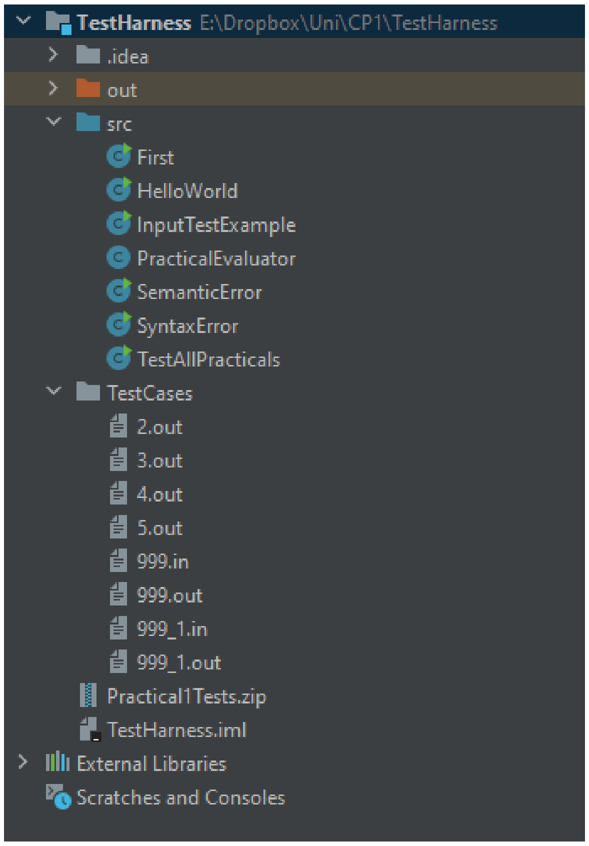
Currently the practicals lack a way to rapidly assert the output is correct. Students can sometimes be left guessing if their output is correct and demonstrators marking the work can need to wait for input and then spend time verifying it is correct.

2 Solution

Include with the practicals a test harness that can rapidly assess the completion of a students work. By comparing expected output for 1 or more test cases. Test cases can be paired with input so that the input can be injected. There are two possible variations of how this could be provided to students.

2.1 Option A

Include the test data all with the projects as a prepared folder of test cases. This is the simplest approach because it does not require additional work from the student's computer. This would include for example the PracticalEvalator.java, a TestAllPracticals.java file (or similar), and a folder called TestCases that contains files with a checkpoint number followed by ".in" and ".out". The ".in" are input files and ".out" are expected output files. Either from running the associated input or because of the program running as normal without needing input.



```
TestHarness E:\Dropbox\Uni\CP1\TestHarness
  .idea
  out
  src
    First
    HelloWorld
    InputTestExample
    PracticalEvaluator
    SemanticError
    SyntaxError
    TestAllPracticals
  TestCases
    2.out
    3.out
    4.out
    5.out
    999.in
    999.out
    999_1.in
    999_1.out
  Practical1Tests.zip
  TestHarness.iml
External Libraries
Scratches and Consoles
```

2.2 Option B

The same as above, but only the two java files are included. Then have the test cases populated automatically by the test harness by downloading a zip file for that practical then auto extracting to the folder. Currently this is working by downloading the Practical1Tests.zip shown in the image from my GitHub. Then if the file was successfully downloaded (or already existed), the file is extracted into TestCases. The advantage of this option is additional test cases can easily be added without changing the original practical. The zip file can be deleted, and it would repopulate with all additional test cases in the TestCases folder.

3 Example of Implementation

3.1 The TestAllPracticals File

```java
public class TestAllPracticals {
    public static void main(String[] args) {

        //deleteTestCases();
        //PracticalEvaluator.setupTestCases(1);
        //PracticalEvaluator.testAllCases(999, InputTestExample::main);

        //demoDownloadingAndTestingAllCases();
        //demoTestingWithJustInput();
        demoTestingWithInputAndExpectedOutput();
    }

    public static void demoDownloadingAndTestingAllCases() {
        // Downloads all TestCases for practical 1 and extracts them to TestCases
folder
        PracticalEvaluator.setupTestCases(1);
        //PracticalEvaluator.testAllCases(999, InputTestExample::main);

        // Example calling the main methods.
        PracticalEvaluator.testAllCases(2, HelloWorld::main);
        PracticalEvaluator.testAllCases(3, SyntaxError::main);
        PracticalEvaluator.testAllCases(4, SemanticError::main);
        PracticalEvaluator.testAllCases(5, First::main);
        PracticalEvaluator.testAllCases(999, InputTestExample::main);

        // Example calling a static method
        PracticalEvaluator.testAllCases(5, First::doCheckpoint5);

        // Example calling non-static method
        PracticalEvaluator.testAllCases(5, ()-> new First().doCheckpoint5RQObj());
    }

    public static void demoTestingWithJustInput() {
        PracticalEvaluator.testAllInputsNoValidation(999, InputTestExample::main,
false);

        PracticalEvaluator.testAllInputsNoValidation(2, HelloWorld::main, false);
    }

    public static void demoTestingWithInputAndExpectedOutput() {
        PracticalEvaluator.testAllInputsNoValidation(999, InputTestExample::main,
true);

        PracticalEvaluator.testAllInputsNoValidation(2, HelloWorld::main, true);
    }
}
```

3.2 Example of First.java (Checkpoint 5)

```java
public class First {
    public static void main(String[] args) {
        System.out.println(7 * 6 + " = 7 * 6 (answer 1)");
        System.out.println("3 + 5 = " + (3 + 5) + " (answer 2)");
        System.out.println("The ideas of \"state\" and \"sequence\"");
        System.out.println("are fundamental to most programming.");
    }

    public static void doCheckpoint5() {
        System.out.println(7 * 6 + " = 7 * 6 (answer 1)");
        System.out.println("3 + 5 = " + (3 + 5) + " (answer 2)");
        System.out.println("The ideas of \"state\" and \"sequence\"");
        System.out.println("are fundamental to most programming.");
    }

    public void doCheckpoint5RQObj() {
        System.out.println(7 * 6 + " = 7 * 6 (answer 1)");
        System.out.println("3 + 5 = " + (3 + 5) + " (answer 2)");
        System.out.println("The ideas of \"state\" and \"sequence\"");
        System.out.println("are fundamental to most programming.");
    }
}
```

3.2.1 Example of 5.out to match with First.java

```
42 = 7 * 6 (answer 1)
3 + 5 = 8 (answer 2)
The ideas of "state" and "sequence"
are fundamental to most programming.
```

3.2.2 Example Output Testing with 5.out (no download/extract)

```
Validating Checkpoint 5

Finished Validation of Checkpoint 5.

Correct Lines: 4, Incorrect: 0, Incorrect Extra: 0, Unnecessary Blanks: 0, Missing Lines: 0

Validation PASSED!



All Checkpoint 5 tests completed. 1 of 1 successfully passed.




Process finished with exit code 0
```

3.3 Example of Dealing with Input Multiple Cases

3.3.1 Sample Code

```
import java.util.Scanner;

public class InputTestExample {
    public static void main(String[] args) {
        System.out.println("Welcome!");
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter your age: ");
        int input = scan.nextInt();
        System.out.println("Output age is: " + (input+1));
    }
}
```

3.3.2 Contents of 999.in

```
22
```

3.3.3 Contents of 999.out

```
Welcome!
Enter your age: Output age is: 23
```

3.3.4 Contents of 999_1.in

```
87
```

3.3.5 Contents of 999_1.out

```
Welcome!
Enter your age: Output age is: 88
```

3.3.6 Test Code

```
public class TestAllPracticals {
    public static void main(String[] args) {
        PracticalEvaluator.setupTestCases(1);
        PracticalEvaluator.testAllCases(999, InputTestExample::main);
    }
}
```

3.3.7 Test Output

Tests Files Already Downloaded. Skipping Download.

Delete Practical1Tests.zip to re-download.

    Extracting file: TestCases/2.out

    Extracting file: TestCases/3.out

    Extracting file: TestCases/4.out

    Extracting file: TestCases/5.out

```
    Extracting file: TestCases/999.in
    Extracting file: TestCases/999.out
    Extracting file: TestCases/999_1.in
    Extracting file: TestCases/999_1.out
Validating Checkpoint 999
Finished Validation of Checkpoint 999.
Correct Lines: 2, Incorrect: 0, Incorrect Extra: 0, Unnecessary Blanks: 0, Missing Lines: 0
Validation PASSED!

Validating Checkpoint 999 Using Input Variation: 1
Finished Validation of Checkpoint 999.
Correct Lines: 2, Incorrect: 0, Incorrect Extra: 0, Unnecessary Blanks: 0, Missing Lines: 0
Validation PASSED!

All Checkpoint 999 tests completed. 2 of 2 successfully passed.



Process finished with exit code 0
```

3.3.8 Example of Error Output (Changing +1 to +2 in output code)

```
Tests Files Already Downloaded. Skipping Download.
Delete Practical1Tests.zip to re-download.
    Extracting file: TestCases/2.out
    Extracting file: TestCases/3.out
    Extracting file: TestCases/4.out
    Extracting file: TestCases/5.out
    Extracting file: TestCases/999.in
    Extracting file: TestCases/999.out
    Extracting file: TestCases/999_1.in
    Extracting file: TestCases/999_1.out
Validating Checkpoint 999
Line mismatch. Expected on line 3:
Enter your age: Output age is: 23
Got:
Enter your age: Output age is: 24
Finished Validation of Checkpoint 999.
Correct Lines: 1, Incorrect: 1, Incorrect Extra: 0, Unnecessary Blanks: 0, Missing Lines: 0
Validation did not pass all expectations.
Input for failed case was:
22
End of input.

Validating Checkpoint 999 Using Input Variation: 1
Line mismatch. Expected on line 3:
Enter your age: Output age is: 88
Got:
Enter your age: Output age is: 89
Finished Validation of Checkpoint 999.
Correct Lines: 1, Incorrect: 1, Incorrect Extra: 0, Unnecessary Blanks: 0, Missing Lines: 0
Validation did not pass all expectations.
Input for failed case was:
87
End of input.
```

```
All Checkpoint 999 tests completed. 0 of 2 successfully passed.



Process finished with exit code 0
```

3.4 Test Example with Input Injection and only showing Actual Output (No Validation)

3.4.1 Test Code

```java
public static void demoTestingWithJustInput() {
    PracticalEvaluator.testAllInputsNoValidation(999, InputTestExample::main, false);

    PracticalEvaluator.testAllInputsNoValidation(2, HelloWorld::main, false);
}
```

3.4.2 Test Output

```
Checkpoint 999

Input:

22

End of input.



Actual output:

Welcome!

Enter your age: Output age is: 23

End of Test Case


Checkpoint 999 Using Input Variation: 1

Input:

87

End of input.
```

Actual output:

Welcome!

Enter your age: Output age is: 88

End of Test Case

Checkpoint 2

Actual output:

Hello World

End of Test Case

Process finished with exit code 0

3.5 Test Example with Input Injection showing Expected+Actual Output (No Validation)

3.5.1 Test Code

```java
public static void demoTestingWithInputAndExpectedOutput() {
    PracticalEvaluator.testAllInputsNoValidation(999, InputTestExample::main, true);

    PracticalEvaluator.testAllInputsNoValidation(2, HelloWorld::main, true);
}
```

3.5.2 Test Output

Checkpoint 999

Input:

22

End of input.

Expected output:

Welcome!

Enter your age: Output age is: 23

End expected output.

Actual output:

Welcome!

Enter your age: Output age is: 23

End of Test Case

Checkpoint 999 Using Input Variation: 1

Input:

87

End of input.

Expected output:

Welcome!

Enter your age: Output age is: 88

End expected output.

Actual output:

Welcome!

Enter your age: Output age is: 88

End of Test Case

Checkpoint 2

```
Expected output:
Hello World
End expected output.


Actual output:
Hello World
End of Test Case



Process finished with exit code 0
```

4 Final Comments

The examples demonstrated are not final. In a final version I could modify the output to suit any requirements. I likely would add a coloured mode to make the output easier to read. To make the current code work for the practicals it would just require generating input/output files to match with each of the pracs (would not work for prac 10 due to GUI, there could be other ways to verify this). And then including the PracticalEvaluator.java file in each of the start files along with another java file that is prepared to call each of tests.

This system has only been tested on a personal Windows computer. It should work on Mac and the Uni computers, but I cannot verify this myself at this time.

Utilising this proposed system could have me provide:

- The PracticalEvaluator.java code that does all the work.
- Generation of test cases for every practical (*.in and *.out files) and packaging of test cases into ZIP files.
- Updating of the ZIP files provided to students for the start code to include the PracticalEvaluator.java file and a file ready to test each practical.
- Preparation of a brief guide explaining to students how to use the tool such that the PDFs for each practical should not need to be modified, this separate PDF could simply explain for all practicals.