

Agenda

- Project
- Continue Test Data Generation

The Testing Project

CSE 4321

Motivation

- Provides an opportunity to practice the basic concepts, principles, and techniques covered in this course
- Specifically, you will apply control flow testing to a program of moderate size
 - Printtokens.java: a Java class that implements a string tokenizer (available in Canvas).
 - about 500 lines of code with seeded faults

Project Team

- You could choose to work on the project by yourself or with a teammate.
 - If you work with a teammate, send our TA an email to identify your team members by March 13 (midnight).
 - Team members will divide up the work (and clearly state each member's contribution in the project report) and resolve any conflicts by yourselves.
 - Both members will receive the same grade.

Input and Output

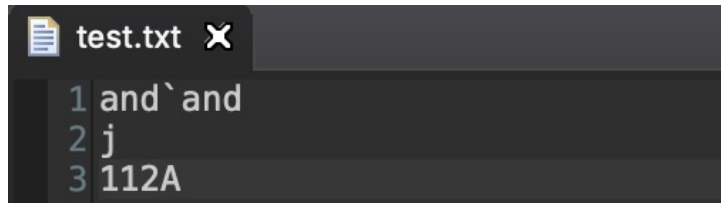
- Input
 - Location of a file (i.e., input to the main() method) that contains the tokens to be classified
 - If a location is not given, the user is expected to type the tokens to be classified on the console.
- Output
 - Type of each token, one per line

Type of Tokens

- **Keyword:** and, or, if, xor, lambda, =>
- **Special Symbol:** “(”, “)”, “[”, “]”, “””, “””, “,” (lparen, rparen, lsquare, rsquare, quote, bquote, comma)
- **Identifier:** begins with a letter and contains only letters and digits, e.g., a, aa, a1, a2.
- **Number Constant:** e.g., 123, 1, 321
- **String Constant:** e.g., “asd”, “123”
- **Character Constant:** e.g., #a, #b, #c, #d
- **Comment:** Anything starts with “;”
- **Error:** Everything else

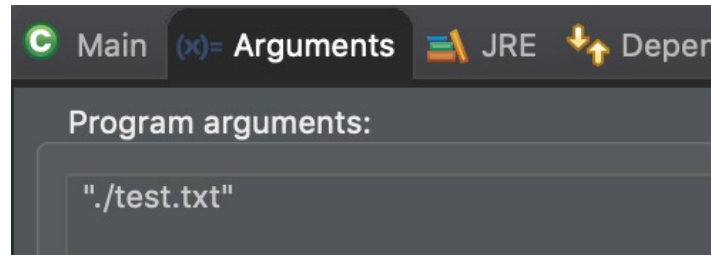
Example

Input:



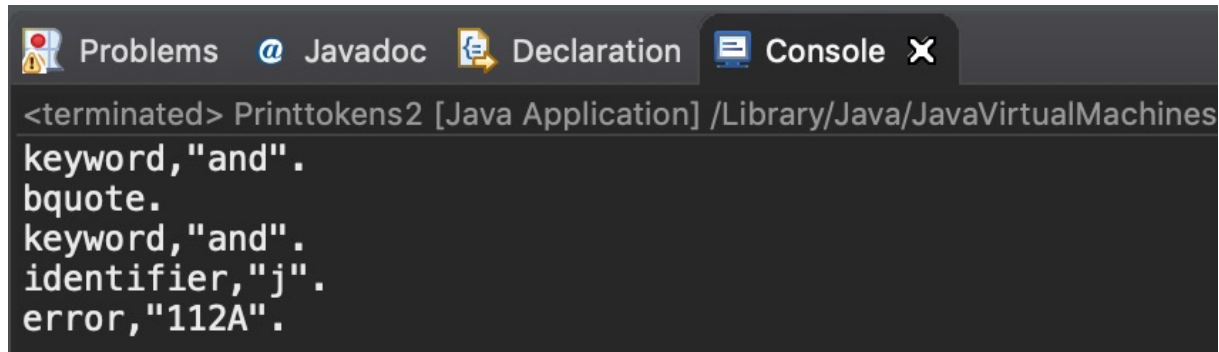
```
test.txt x
1 and`and
2 j
3 112A
```

Arguments:



```
Main (x)= Arguments JRE Depen
Program arguments:
"./test.txt"
```

Output:



```
Problems @ Javadoc Declaration Console x
<terminated> Printtokens2 [Java Application] /Library/Java/JavaVirtualMachines
keyword,"and".
bquote.
keyword,"and".
identifier,"j".
error,"112A".
```

Major Steps

- Create the Control Flow Graph (CFG) for each method
 - The code in the **catch** clauses can be skipped, i.e., these clauses do not need to be represented in the CFG
- **Unit Testing**: For each method other than the **main** method, select a set of test paths to achieve edge coverage, i.e., cover all the edges in the CFG of the method
- **Program Testing**: For the **main** method, select a set of test paths to achieve edge coverage for the entire program, i.e., cover all the edges in all the CFGs.
- Generate test cases, including test data and expected output, for each test path
 - If a test path you selected in the previous step is infeasible, a new test path needs to be selected
- Use JUnit to implement and execute the generated test cases.
- Provide edge coverage reports for tests executed.
 - One report for the unit tests of individual methods, and one report for the program-level tests of the main method

Deliverables

- **Due date (FIRM):** April 29 (midnight, last day of class)
 - A team can use slip days up to the maximum number of slip days remaining for any individual member.
- 20%: CFGs along with the corresponding source code, and the basic block table that identifies each basic block
- 30%: Test cases, including each test path, the corresponding test data to execute the test path, and the expected output
- 25%: JUnit source code
- 5%: Code coverage reports (one for unit testing of individual method, and the other for the program-level testing)
- 10%: Faults detected and corrections
- 5%: Summary and discussion.
- 5%: A ReadMe file that explains how to execute your JUnit code

Must-Have Files

- In addition to the source code, your project folder in GitHub must include a subfolder named “deliverables” that contains the following files:
 - CFG.pdf
 - Test Cases.pdf
 - Code Coverage Reports.zip
 - Faults and Corrections.pdf
 - Summary and Discussion.pdf
 - ReadMe.txt
- Typed reports are strongly recommended; if you choose to handwrite and scan the reports, it is your responsibility to ensure legibility.

- Faults detected and code coverage are not the main focus of our evaluation.
- If you follow the approach correctly, you could expect to detect at least 10 bugs.
- Focus on the “approach”, instead of the “faults”

Recommendations

- IDE: Eclipse; Code Coverage: JaCoCo/EclEmma
 - Other tools could be used, but you may be asked to give a demo for the TA
- When you find a fault, you may fix it before you continue testing. This could help to discover more faults.
- If fixing a fault alters the control flow, you are not required to construct a new CFG.
- Do not try to use CFG generation tools for Java programs; otherwise, you will receive no points on the project.
 - Many tools generate CFGs based on byte code, not source code. The output could be incorrect w.r.t the project specification.