Jediterm Terminal Widget

Members: Oscar Ossowski, Ivan Systerov, Mahir Faisal, Rehan Ahmed

April 6, 2023

Introduction: Jediterm Information

- Open source terminal widget for Java IDEs
- Main purpose of this application is to help developers test and debug their programs
- Provide a terminal widget for testing/debugging
- Compatible with Jetbrains IDEs

Level in overall sequence

- Unit testing
- Each feature will be tested separately for correctness
- Level 1 testing as we are checking for correctness
- Project maturity level of 1 due to limited comments and documentation present

Testing Criteria

- Testing focuses on accuracy and correctness
- Verify correctness by comparing to expected result
- Verify accuracy with multiple test cases
- Accuracy is verified by checking if there is minimal output differences among test cases

Testing Approaches Used

- Input Space Partitioning
- Control Flow Coverage
- Data Flow Coverage
- Logic Flow

Features to be tested

- Xterm emulation – Oscar
- Xterm 256 colours – Mahir
- Terminal tabs – Mahir
- Scrolling – Ivan
- Copy/Paste – Ivan
- Mouse support – Rehan
- Terminal resizing – Client side only – Rehan

ISP

Class Tested: ColorPaletteImpl

Methods Tested: getForegroundColorByIndex and getBackground

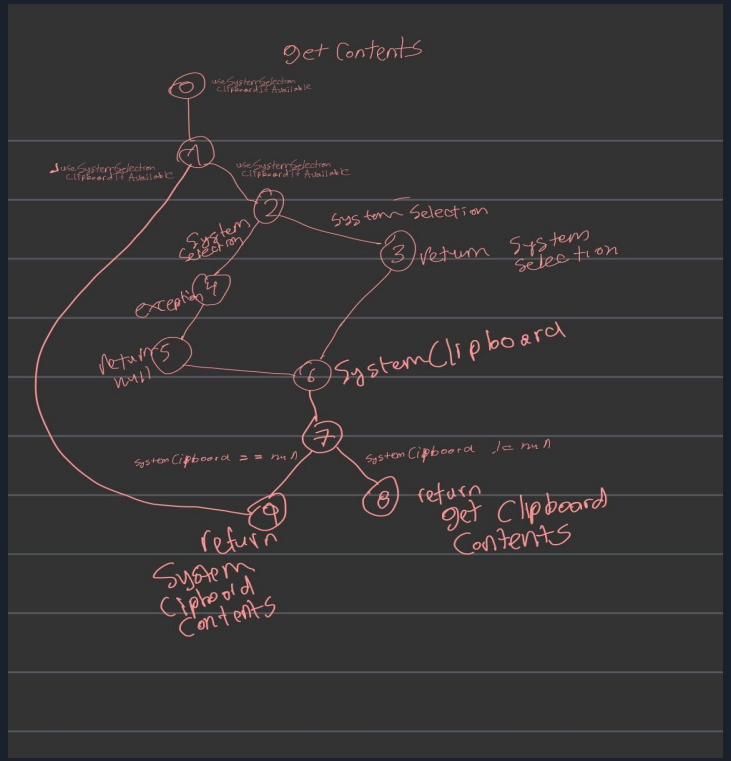
ISP Table

Characteristic	b1 (index < 0)	b2 (0 <= index <= 15)	b1 (index > 0)
Colors list index	-1	0, 1, 14, 15, 7	16

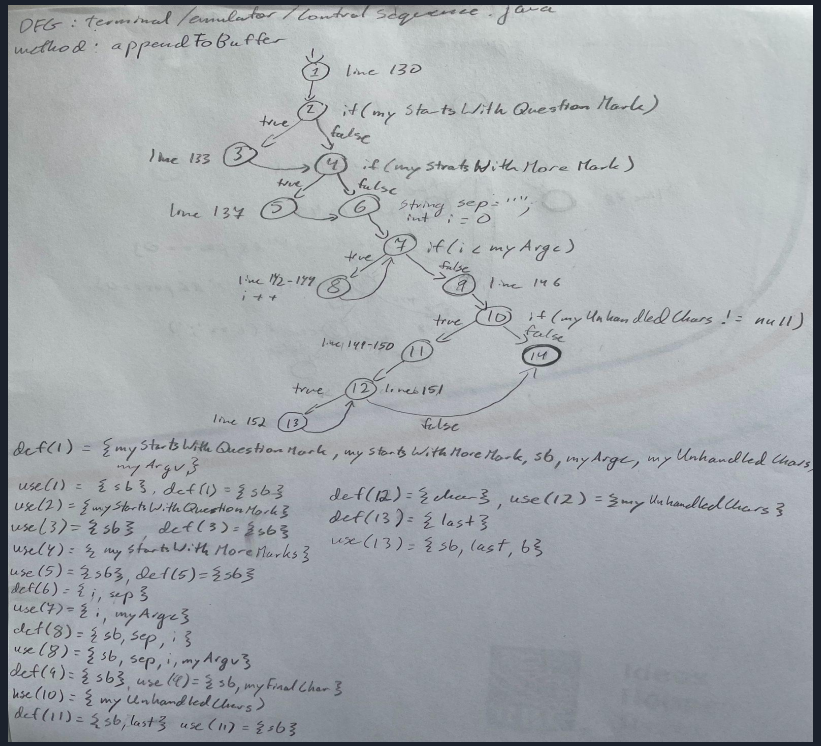
ISP

- Class purpose: Assigns a specific color palette to terminal foreground or terminal application window
- ISP is chosen because it verifies if both methods can handle invalid color object index inputs as well as valid color object index inputs and accurately return the correct outputs
- Method satisfies the RIPR model
- The two methods themselves return a valid color object from the assigned color palette based of the input index that is provided to the methods

CFG


getContents
0
useSystemSelection
Clipboard.IfAvailable
1
useSystemSelection
Clipboard.IfAvailable
useSystemSelection
Clipboard.IfAvailable
2
SystemSelection
SystemSelection
3
return SystemSelection
Exception?
4
return null
5
6
SystemClipboard
7
SystemClipboard == null
SystemClipboard != null
8
return getClipboardContents
9
return SystemClipboardContents

DFG



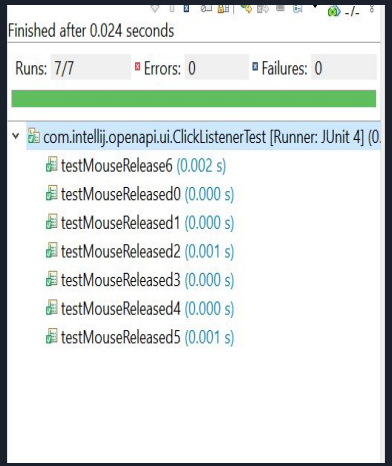
def(1) = {myStartsWithQuestionMark, myStartsWithMoreMark, sb, myArgc, myUnhandledChars, myArgv}

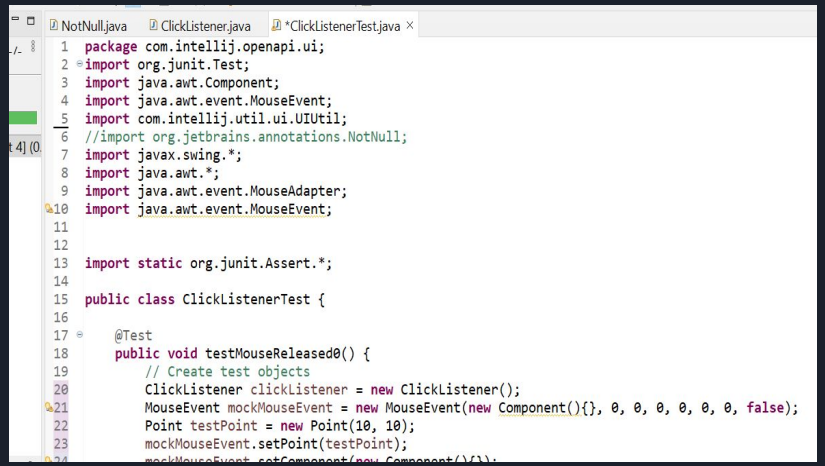
use(1) = {sb}, def(1) = {sb}
use(2) = {myStartsWithQuestionMark}
use(3) = {sb}, def(3) = {sb}
use(4) = {myStartsWithMoreMarks}
use(5) = {sb}, def(5) = {sb}
def(6) = {i, sep}
use(7) = {i, myArgc}
def(8) = {sb, sep, i}
use(8) = {sb, sep, i, myArgv}
def(9) = {sb}, use(9) = {sb, myFinalChar}
use(10) = {myUnhandledChars}
def(11) = {sb, last} use(11) = {sb}

def(12) = {char}, use(12) = {myUnhandledChars}
def(13) = {last}
use(13) = {sb, last, b}

Logic coverage

Logic Coverage was used for the mouseReleased method as it is a vital feature that requires each testing path to be functioning correctly. More specifically, the mouseReleased method allows the JediTerm software to be able to distinguish whether the mouse has a button clicked and without this functionality the software would stop listening to mouse input. Logic Coverage was chosen because the logical statement to determine whether the mouse has clicked a certain part of the screen included multiple boolean operators such as || and && which were used to determine the predicates for this formula efficiently.





Bugs found

- Unused function inputs
- Repetition of code segments unnecessarily
- Unreachable code segments
- Improper documentation of accepted inputs
- Mutually exclusive if statements that are not combined into one

Issues faced

- Environment setup
- Many methods having access control permissions set
- Lack of documentation/comments
- Java version mismatch with junit framework

Lessons learned

That the real testing abstraction is the friends we made along the way!

Lessons learned

- How to better trace requirements throughout many classes to ensure test propagation
- Building test cases and inputs based of off the testing paths, but in a practical setting
- How to decide which testing abstraction to use on each method to best cover it
- The effect of loops and just how much damage they actually cause in any graph based coverage approach

Questions?

Thank you!