CPSC 410 Check in 3

Group Number: 12

Members: Matthew Wu, Calvin Zhang, Ray Huang, Samantha Tseng, Shaun Gao

Mockup

Here is a list of the functionality we want to implement

Description	Example
Identify empty catch exception blocks	try{...} catch(Error e){} Highlight this and notify the user that there is an empty catch block

Preventing expensive exception-based control flow	for (...) {
	try{...}catch{...}
	}
	
	Highlight this, and notify the user that try catch blocks in loops are expensive. Suggest moving the try catch block to outside of the loop

Assign risk scores to exception handling code based on the frequency of errors	Issues that may cause the app to crash will be assigned a risk score of “high” Issues that may cause significant performance issues are assigned a risk score of “medium” Issues that will not affect execution much, but is still considered bad practice, will be assigned a risk score of “low”

Identify any open resource that is not closed by the catch block	Unclosed resources in catch blocks:
	FileInputStream fis = new FileInputStream(//);
	try {//}
	catch (Exception e) {//}
	finally {//} // No code to close the FileInputStream
	
	Highlight this and warn user to close the FileInputStream

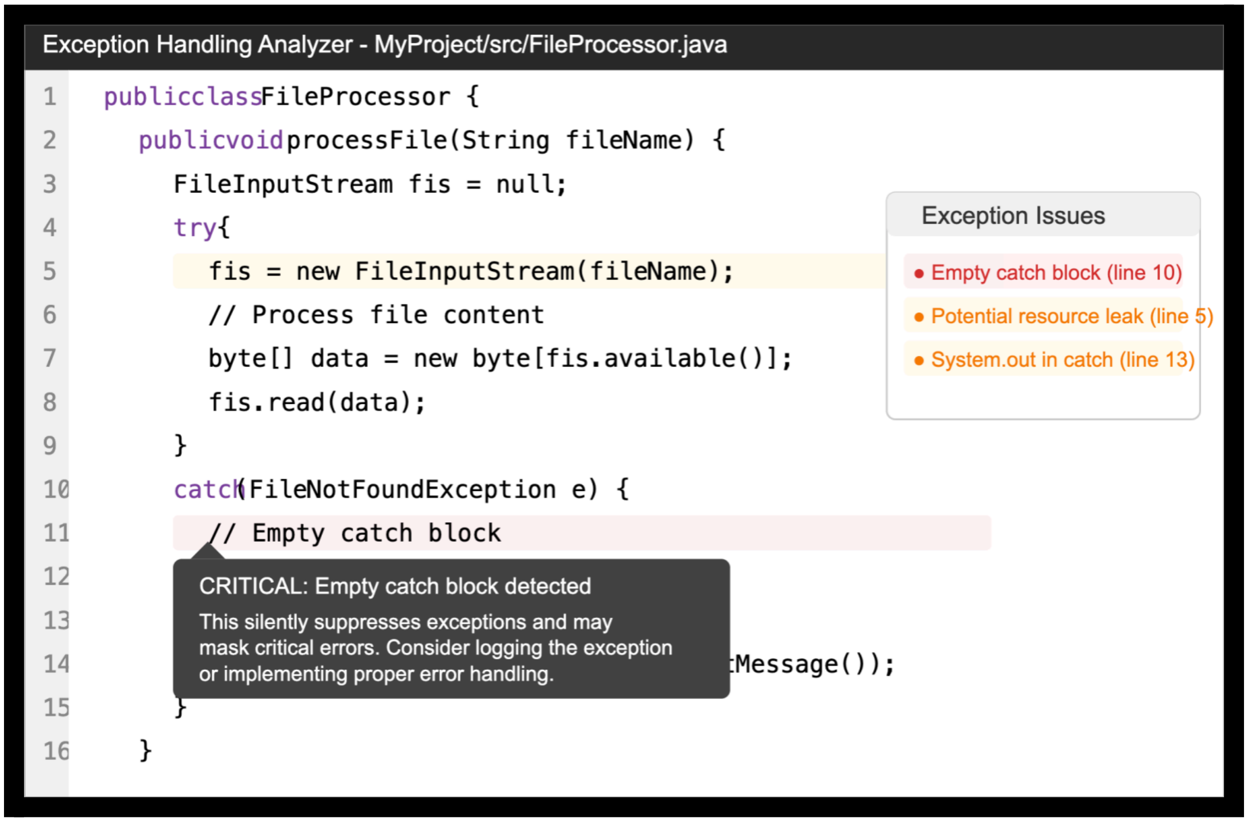
Suggest adding a try-catch block to methods that	public void readFile(String path) {
	   FileReader reader = new FileReader(path);

potentially generate errors if the user didn't include it	int data = reader.read();
	reader.close();
	}
	Our analyzer would highlight the 3 lines and suggest that the user add a try-catch block as the code might throw exceptions (e.g. IOException) not declared in the method signature.

Identify if the thrown error is not declared in the method	public void foo(String[] list) {
	
	If (list.size() == 0 {
	throw new InvalidInputException();
	}
	
	}
	
	Highlight this method because it doesn't declare the error that can be thrown

Identify if a catch block is still printing to System error	try{...}catch(Error e){System.err.print(e)} We will highlight this and warn the user that this is potentially misleading, as the system output still looks like an error is being thrown.
Identify any catch blocks that are always triggered	try{...}catch(Error e) {...} Highlight it and warn the user this catch block is always triggered.

Here is an example of what our visualization might look like:


Exception Handling Analyzer - MyProject/src/FileProcessor.java

```
publicclassFileProcessor {
  publicvoidprocessFile(String fileName) {
    FileInputStream fis = null;
    try{
      fis = new FileInputStream(fileName);
      // Process file content
      byte[] data = new byte[fis.available()];
      fis.read(data);
    }
    catch(FileNotFoundException e) {
      // Empty catch block


                                    Message());
    }
  }
```

Exception Issues

- ● Empty catch block (line 10)
- ● Potential resource leak (line 5)
- ● System.out in catch (line 13)

CRITICAL: Empty catch block detected

This silently suppresses exceptions and may mask critical errors. Consider logging the exception or implementing proper error handling.

First User Study

We will present our list of features and code examples to our user, and ask them for feedback on what they like and don't like

User 1:

- It's relatively hard to judge if a certain error would cause an app to crash, and assigning a score is subjective. It would be nice to allow developers to adjust scores themselves.
- Perhaps it would be helpful to see a summary of the most common exception handling issues in the codebase.
- Everything is good, but it feels like many functionalities overlap with what my IDE already provides.

User 2:

- The "always trigger" feature seems challenging — it might be difficult to determine if a block will always be executed.

- It would be nice to have some suggestions on how to handle a mishandled exception. For example, can you suggest appropriate ways of handling an exception based on context?
- I find the resource leak checking and exception in for loops helpful, these are critical things to check when writing a program.
- The risk scoring system is helpful but can be subjective. What criteria/method do you use for risk assessment?

Proposed Changes

1. TODO

Schedule

Mar 14 (Friday):

- Completed first set of user studies
- Explore Java parser framework
- Explore tools for visualization

Mar 21 (Friday):

- Added Unit tests
- 40% done implementation

Mar 28 (Friday):

- Added integration tests
- 80% done implementation
- Bug fixing
- Completed second set of user studies

April 4 (Friday):

- Project deadline