Unit 09: Exceptions

Anthony Estey

CSC 115: Fundamentals of Programming II

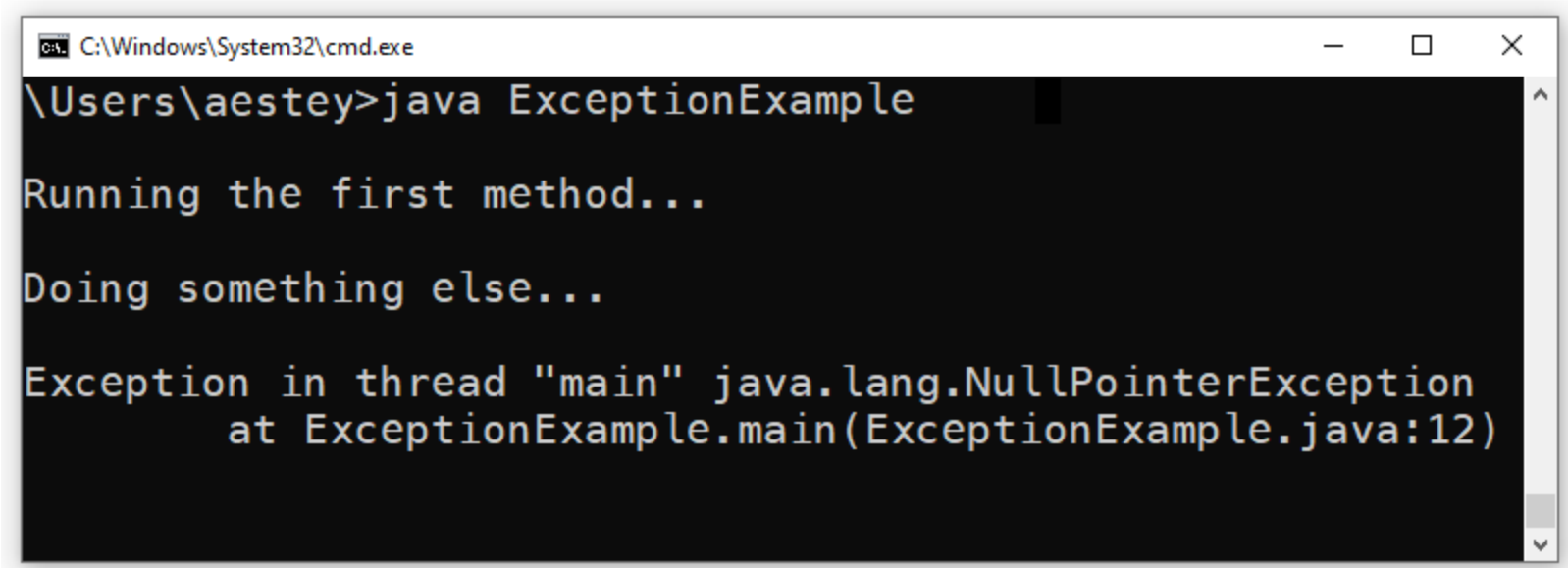
University of Victoria

Unit 09 Overview

- Learning Objectives: (You should be able to…)
  - describe the concept of an exception, and explains the reasons to use exceptions in Java programs
  - write code to catch common Java exceptions (like ArrayOutOfBoundsException)
  - write and use your own exceptions in Java programs

Exceptions - Motivation

- Sometimes when running our program an error occur
  - typically this results in the execution of our program being halted, and an error message is reported:



Exceptions - Motivation

- When using an electronic device, sometimes the operation we are trying to perform doesn't go as expected…
  - Often the program lets us know that there was a problem
  - But the whole system doesn't typically crash

Exceptions

- There are two types of exceptions that can occurs in a Java program:
  - **unchecked** exceptions and **checked** exceptions

- As the name indicates, the main difference between these two types of exceptions are that the Java compiler checks for one type during compilation, but not the other

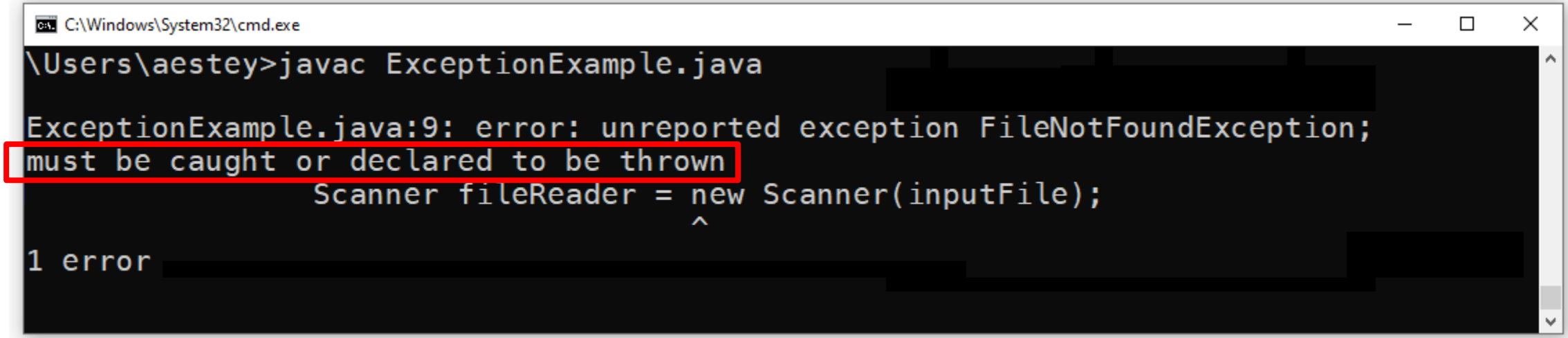
Unchecked Exceptions

- Unchecked exceptions occur when the program was able to compile without problem, but during execution a problem occurs
  - Examples: ArrayOutOfBoundsException, NullPointerException, StackOverflowException

- ArrayOutOfBoundsException
  - The compiler doesn't ensure that all array indexes stay within the bounds of the array
- Why not?
  - The index being accessed might be specified by user input or read in from a webpage or input file
  - it's impossible to know ahead of time when this might occur

Checked Exceptions

- Checked exception are exceptions that the Java compiler forces us to handle in our program in order for it to compile without issues
- For example:
  - When reading from a file, we need to specify what to do if the file cannot be found (FileNotFoundException) or the program will not compile
  - `Scanner fileReader = new Scanner(inputFile);`



Checked Exceptions

- Checked exception are exceptions that the Java compiler forces us to handle in our program in order for it to compile without issues

- FileNotFoundException
  - We must either state that we choose not to handle the exception (which would be accepting that the program will crash if the exception occurs)
  - Or declare that we will **catch** the exception, for which we write code that will be executed if the exception occurs (and hopefully handles the exception gracefully)

- Either way, we need to specify what to do if the exception occurs

Option 1: (declared to be thrown)

- The **throws** clause allows us to specify that we are aware an exception could occur within a block of code, and that we choose to ignore it
- Sometimes this is compared to a waiver of liability form:
  - *"I hereby agree that this method might throw an exception, and I accept the consequences (that the program will **crash**) if this happens"*

- Example:

```
public static void main(String[] args) throws FileNotFoundException {
```

We are aware the file we want to read may not be found, and it is okay for the program to crash

Option 2: Exception Handling

- **Exception handling** allows us to change the default behaviour of a program when something goes wrong, by specifying what should be done when a specific error occurs

- Exception handling typically has two parts:

1. A **try block**: where we try and do something that may cause an error
2. An **exception handler:** where we indicate what should happen if the error occurs

Exceptions

- General form:

```
try {
      statement1
      statement2
      …
} catch (someException e) {
      statement1
      statement2
      …
}
```

An exception is an error that occurs while the programming is running, causing the program to abruptly halt.

The `try/catch` statements are used to gracefully handle exceptions.

The code in the **`try`** block is executed, and if no errors occur, the code in the **`catch`** block is ***skipped***.

If a statement in the **`try`** block raises an exception, then the code in the **`catch`** block is immediately executed, and any remaining code in the **`try`** block is skipped.

Specific Example

```
int i = 0;
int[] arr = {6, 1, 3, 4};
try {
    System.out.println("Printing array contents:");
    while (i < 10) {
        System.out.println(arr[i]);
        i++;
    }
    System.out.println("finished printing 10 values");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Error with an array access!");
    System.out.println("index " + i + " is out of bounds");
}
```

The values in the array are output without an issue

Then the exception occurs, so this print statement is never executed. The code in the catch block is executed instead