Exception Handling

CSCI 201L

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
HTTP://WWW-SCF.USC.EDU/~CSCI201
Outline

- Exception Handling
- Program
Exception Handling

- An exception is an indication of a problem that occurs during a program’s execution
  - Exceptions should not be part of the normal execution of a program
- Exception handling allows programs to handle exceptions when they occur
- In Java, we can handle most errors that would cause our programs to terminate prematurely using exception handling
try-catch Blocks

▪ Any code that has the potential to throw an exception should be enclosed in a try block

▪ Immediately following a try block will be at least one catch block
  › In parentheses after the word “catch” will be the exception parameter that represents the type of exception the catch handler can process
  › The exception parameter will also include an the variable name representing the type of exception that is being handled

▪ An optional finally clause can be included after all of the catch blocks
  › The finally clause will be executed regardless of whether an exception was thrown in the try block or not
  › Even if an exception that was not handled by a catch clause was thrown, the finally clause will still execute
public class Test {
  public static void main(String[] args) {
    try {
      int arr[] = new int[10];
      for (int i = 0; i < 11; i++) {
        arr[i] = i;
      }
    } catch (ArrayIndexOutOfBoundsException aioobe) {
      System.out.println("aioobe: " + aioobe.getMessage());
    } finally {
      System.out.println("in finally block");
    }
  }
}
Java Exception Inheritance Hierarchy

- Throwable
  - Error
    - NoClassDefFoundError
    - OutOfMemoryError
    - RuntimeException
    - SQLException
    - NoClassDefFoundError
    - ArithmeticException
    - FileNotFoundException
  - Exception
    - ArithmeticException
    - FileNotFoundException
Checked vs Unchecked Exceptions

- Unchecked exceptions do not have to be handled in a try-catch block
  - RuntimeExceptions are unchecked exceptions
  - These exceptions can usually be avoided by good coding practices

- Checked exceptions must be handled in a try-catch block
  - Other exceptions are checked exceptions
Java Exception Class

- The Exception class is defined in the java.lang package, so you do not need to import that class
  - Other exceptions are in other packages, so they will need to be imported
- Exception defines a constructor that takes a String representing the error message
- There is also an inherited method getMessage() that returns an exception object’s error message
  - This method can be overridden to return a specialized error message if the String entered into the constructor is not sufficient
Throwing Exceptions

- Exceptions can be thrown inside of `try` blocks if you desire.
- If a `catch` block handles that exception, it will:
  - If it doesn’t, the exception will be thrown up to the calling function.
  - If no `catch` block handles that exception, the program will crash.
FileNotFoundException and IOException

```java
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.FileNotFoundException;
4  import java.io.IOException;
5  public class Test {
6    public static void main(String [] args) {
7      try {
8        FileReader fr = new FileReader("Test.java");
9        BufferedReader br = new BufferedReader(fr);
10       String line = br.readLine();
11       System.out.println("Line #1 = " + line);
12       if (line != null) {
13         line = br.readLine();
14         System.out.println("Line #2 = " + line);
15       }
16       br.close();
17       fr.close();
18     } catch (FileNotFoundException fnfe) {
19       System.out.println("FileNotFoundException: " + fnfe.getMessage());
20     } catch (IOException ioe) {
21       System.out.println("IOException: " + ioe.getMessage());
22     }
23   }
24 }
```
On the previous slide, can we remove the catch block handling the `FileNotFoundException`?

```java
18     } catch (FileNotFoundException fnfe) {
19       System.out.println("FileNotFoundException: " + fnfe.getMessage());
20     } catch (IOException ioe) {
21       System.out.println("IOException: " + ioe.getMessage());
```

Yes, since `FileNotFoundException` inherits from `IOException`.

- Though the `getMessage()` method will not be as descriptive from higher exception classes.

So, could we replace the above code with the following?

```java
18     } catch (Exception e) {
19       System.out.println("Exception: " + e.getMessage());
20     }
```
Creating a Custom Exception

- You are able to create a custom exception class by inheriting from an existing exception
  - You can inherit from `Exception` if you want the exception to be required to be checked
- The `String` passed into the constructor of the `Exception` class will be the `String` returned from the `getMessage()` method
  - The `getMessage()` method can be overridden as well if a custom message is to be returned
- Use the keyword `throw` to specify when an exception will be thrown
- Use the keyword `throws` to specify that a method has the ability to throw an exception
class DivideByZeroException extends Exception {
    public DivideByZeroException(String message) {
        super(message);
    }
}

public class Test {
    public double divide(double num, double den) throws DivideByZeroException {
        if (den == 0) {
            throw new DivideByZeroException("divide by zero");
        }
        return num / den;
    }
    public static void main(String [] args) {
        try {
            Test t = new Test();
            double quotient = t.divide(100, 0);
        } catch (DivideByZeroException dbze) {
            System.out.println("DivideByZeroException: " + dbze.getMessage());
        }
    }
}
Overriding `getMessage()` Example

```java
1  class DivideByZeroException extends Exception {
2    private double numer;
3    private double denom;
4    public DivideByZeroException(String message, double numer, double denom) {
5      super(message);
6      this.numer = numer;
7      this.denom = denom;
8    }
9    public String getMessage() {
10       return "Dividing " + numer + "/" + denom + " " + super.getMessage();
11    }
12 }
13 public class Test {
14   public double divide(double numer, double denom) throws DivideByZeroException {
15     if (denom == 0) {
16         throw new DivideByZeroException("divide by zero", numer, denom);
17     }
18     return numer / denom;
19   }
20   public static void main(String [] args) {
21     try {
22       Test t = new Test();
23       double quotient = t.divide(100, 0);
24     } catch (DivideByZeroException dbze) {
25         System.out.println("DivideByZeroException: " + dbze.getMessage());
26     }
27   }
28 }
```
Outline

- Exception Handling
- Program
Write a program that prompts the user to enter two numbers. If the first number is not less than the second number, throw a custom exception called NumberGreaterException. Handle that exception in the main method and display the value of getMessage() to the user to match the output below.

c:\> java csci201.NumberExceptions
Enter the first number: 100
Enter the second number: 50
NumberGreaterException: 50 is not greater than 100.
c:\>