Exception Handling

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
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 is on the order of 10 times slower compared to writing good code to avoid an exception

- 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 are zero or more `catch` blocks
  - 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 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 (or after the `try` block if there are no catch blocks)
  - The `finally` clause will be executed regardless of whether an exception was thrown in the `try` block or not, regardless whether it was handled by a `catch` block
  - Even if an thrown exception was not handled by a `catch` clause, the `finally` clause will still execute
Exception Handling Example

```java
public class Test {
    public void foo(int[] arr) {
        try {
            for (int i = 0; i < 5; i++) {
                arr[i] = i;
            }
        } catch (ArrayIndexOutOfBoundsException aioobe) {
            System.out.println("aioobe: " + aioobe.getMessage());
        } finally {
            System.out.println("in finally block");
        }
    }
}

public class Main {
    public void bar() {
        int myArr[] = new int[5];
        foo(myArr);
    }
}
```
Exception Handling Example

```java
1 public class Test {
2   public void foo(int [] arr) {
3     try {
4       for (int i=0; i < 5; i++) {
5         arr[i] = i;
6       }
7     } catch (ArrayIndexOutOfBoundsException aioobe) {
8       System.out.println("aioobe: " + aioobe.getMessage());
9     } finally {
10       System.out.println("in finally block");
11     }
12   }
13 }
14
15 public class Main {
16   public void bar() {
17     int myArr[] = new int[3];
18     foo(myArr);
19   }
20 }
```

What happens if the size of the array is 3?
Exception Handling Example

```java
public class Test {
    public void foo(int[] arr) {
        try {
            for (int i = 0; i < 5; i++) {
                arr[i] = i;
            }
        } catch (ArrayIndexOutOfBoundsException aioobe) {
            System.out.println("aioobe: " + aioobe.getMessage());
        } finally {
            System.out.println("in finally block");
        }
    }
}

public class Main {
    public void bar() {
        foo(null);
    }
}
```

What happens if the array passed in is null?
Java Exception Inheritance Hierarchy

- Throwable
  - Error
    - NoClassDefFoundError
    - OutOfMemoryError
  - Exception
    - RuntimeException
      - NullPointerException
    - SQLException
    - IOException
    - NoClassDefFoundError
    - ArithmeticException
    - FileNotFoundException
    - NumberFormatException
    - ArrayIndexOutOfBoundsException
    - ArithmeticException
    - FileNotFoundException
Java Exception Inheritance Hierarchy

Runnable Errors

Error

NoClassDefFoundError

OutOfMemoryError

Exception

RuntimeException

SQLException

IOException

IOException

NullPointerException

NumberFormatException

ArrayIndexOutOfBoundsException

NoClassDefFoundError

FileNotFoundException

NullPointerException

NumberFormatException

ArrayIndexOutOfBoundsException

IOException

NullPointerException

NumberFormatException

ArrayIndexOutOfBoundsException

IOException
Java Exception Inheritance Hierarchy

Unchecked Exceptions

- Throwable
  - Error
    - NoClassDefFoundError
    - OutOfMemoryError
  - Exception
    - RuntimeException
      - NullPointerException
      - NumberFormatException
      - ArrayIndexOutOfBoundsException
      - ArithmeticException
  - SQLException
  - IOException
  - NoClassDefFoundError
  - FileNotFoundException
  - ArithmeticException

USC CSCI 201L 10/19
Java Exception Inheritance Hierarchy

Checked Exceptions

- Throwable
  - Error
    - NoClassDefFoundError
    - OutOfMemoryError
    - RuntimeException
    - SQLException
    - IOException
  - Exception
    - NoClassDefFoundError
    - OutOfMemoryError
    - RuntimeException
    - SQLException
    - IOException

- NullPointerException
- NumberFormatException
- ArithmeticException
- FileNotFoundException
- ArrayIndexOutOfBoundsException
- FileNotFoundException
Checked vs Unchecked Exceptions

- **Unchecked** exceptions do not have to be handled in a **try-catch** block
  - Exceptions that inherit from `RuntimeException` are **unchecked** exceptions
  - These exceptions can usually be avoided by good coding practices

- **Checked** exceptions must be handled in a **try-catch** block
  - Exceptions that do not inherit from `RuntimeException` are **checked** exceptions

- Note: Classes that inherit from `Error` are not exceptions and cannot be handled through exception handling
FileNotFoundException and IOException

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Test {
  public static void main(String[] args) {
    FileReader fr;
    BufferedReader br;
    try {
      fr = new FileReader("Test.java");
      br = new BufferedReader(fr);
      String line = br.readLine();
      System.out.println("#1:" + line);
      if (line != null) {
        line = br.readLine();
        System.out.println("#2:" + line);
      }
    } catch (FileNotFoundException fnfe) {
      System.out.println(fnfe.getMessage());
    } catch (IOException ioe) {
      System.out.print(ioe.getMessage());
    } finally {
      if (br != null) {
        try {
          br.close();
        } catch (IOException ioe) {
          System.out.print(ioe.getMessage());
        }
      }
      if (fr != null) {
        try {
          fr.close();
        } catch (IOException ioe) {
          System.out.print(ioe.getMessage());
        }
      }
    } // ends finally
  } // ends main
} // ends Test
```
FileNotFoundException and IOException

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
public class Test {
  public static void main(String [] args) {
    FileReader fr;
    BufferedReader br;
    try {
      fr = new FileReader("Test.java");
      br = new BufferedReader(fr);
      String line = br.readLine();
      System.out.println("#1:" + line);
      if (line != null) {
        line = br.readLine();
        System.out.println("#2:" + line);
      }
    } catch (IOException ioe) {
      System.out.println(ioe.getMessage());
    } catch (FileNotFoundException fnfe) {
      System.out.println(fnfe.getMessage());
    } finally {
      if (br != null) {
        try {
          br.close();
        } catch (IOException ioe) {
          System.out.print(ioe.getMessage());
        }
      }
      if (fr != null) {
        try {
          fr.close();
        } catch (IOException ioe) {
          System.out.print(ioe.getMessage());
        }
      }
    } // ends finally
  } // ends main
} // ends Test
```

This won’t compile because IOException handles the FileNotFoundException, so the FileNotFoundException block is an unreachable block of code.
FileNotFoundException and IOException

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Test {
    public static void main(String[] args) {
        FileReader fr;
        BufferedReader br;
        try {
            fr = new FileReader("Test.java");
            br = new BufferedReader(fr);
            String line = br.readLine();
            System.out.println("#1:" + line);
            if (line != null) {
                line = br.readLine();
                System.out.println("#2:" + line);
            }
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        } finally {
            if (fr != null) {
                try {
                    fr.close();
                } catch (IOException ioe) {
                    System.out.println(ioe.getMessage());
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException ioe) {
                    System.out.println(ioe.getMessage());
                }
            }
        }
    }
}
```

This compiles but is not good programming practice because the IOException won’t give us as much information about the problem as the more specific FileNotFoundException.
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
public class Test {
   public static void main(String[] args) {
      FileReader fr;
      BufferedReader br;
      try {
         fr = new FileReader("Test.java");
         br = new BufferedReader(fr);
         String line = br.readLine();
         System.out.println("#1:" + line);
         if (line != null) {
            line = br.readLine();
            System.out.println("#2:" + line);
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
      } finally {
         if (br != null) {
            try {
               br.close();
            } catch (IOException ioe) {
               System.out.print(ioe.getMessage());
            }
         }
         if (fr != null) {
            try {
               fr.close();
            } catch (IOException ioe) {
               System.out.print(ioe.getMessage());
            }
         }
      }
   } // ends finally
} // ends Test

This compiles but also is not good programming practice because the Exception will just give a message that says “Exception occurred”
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 method.
  - If no `catch` block handles that exception, the program will crash.
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 (which is what was passed into the constructor of the `Exception`)
  - This method can be overridden to return a specialized error message if the `String` entered into the constructor is not sufficient
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
Creating a Custom Exception Example

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