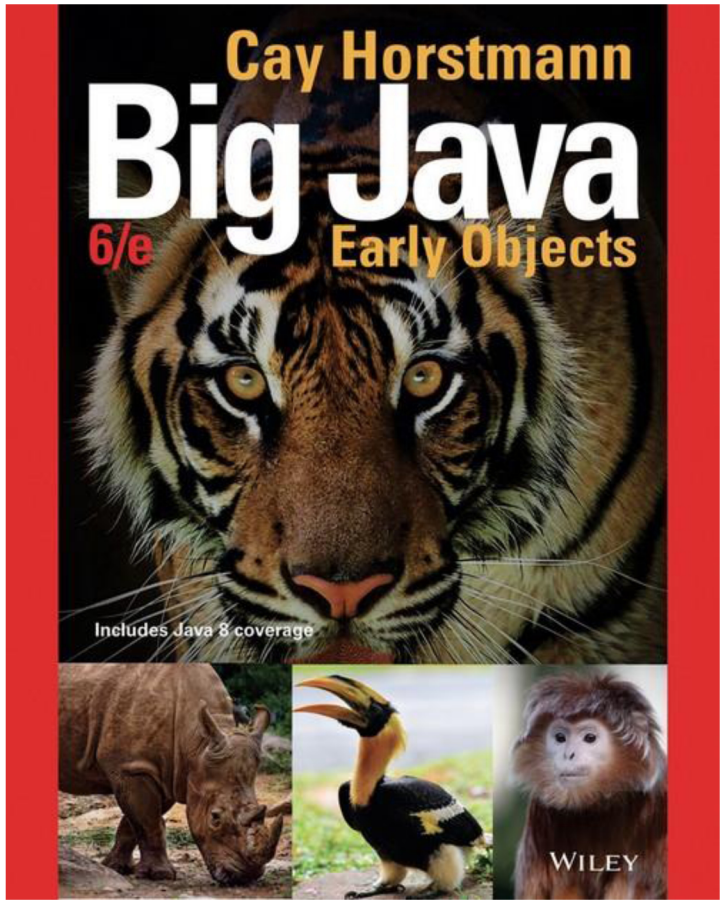

Chapter 11 – Input/Output & Exception Handling Part IV (Optional)

Application: Handling Input Errors

- Program asks user for name of file
  - File expected to contain data values
  - First line of file contains total number of values
  - Remaining lines contain the data
  - Typical input file:

```
3
1.45
-2.1
0.05
```

Case Study: A Complete Example

- What can go wrong?
  - File might not exist
  - File might have data in wrong format
- Who can detect the faults?
  - `Scanner` constructor will throw an exception when file does not exist
  - Methods that process input need to throw exception if they find error in data format
- What exceptions can be thrown?
  - `FileNotFoundException` can be thrown by `Scanner` constructor
  - `BadDataException`, a custom checked exception class for reporting wrong data format

Case Study: A Complete Example

- Who can remedy the faults that the exceptions report?
  - Only the `main` method of `DataAnalyzer` program interacts with user
    - Catches exceptions
    - Prints appropriate error messages
    - Gives user another chance to enter a correct file

section_5/DataAnalyzer.java

```
1  import java.io.FileNotFoundException;
2  import java.io.IOException;
3  import java.util.Scanner;
4
5  /**
6      This program reads a file containing numbers and analyzes its contents.
7      If the file doesn&amp;apos;t exist or contains strings that are not numbers, an
8      error message is displayed.
9  */
10 public class DataAnalyzer
11 {
12    public static void main(String[] args)
13    {
14       Scanner in = new Scanner(System.in);
15       DataSetReader reader = new DataSetReader();
16
```

Continued

section_5/DataAnalyzer.java

```
17          boolean done = false;
18          while (!done)
19          {
20             try
21             {
22                System.out.println("Please enter the file name: ");
23                String filename = in.next();
24
25                double[] data = reader.readFile(filename);
26                double sum = 0;
27                for (double d : data) { sum = sum + d; }
28                System.out.println("The sum is " + sum);
29                done = true;
30             }
31             catch (FileNotFoundException exception)
32             {
33                System.out.println("File not found.");
34             }
35             catch (BadDataException exception)
36             {
37                System.out.println("Bad data: " + exception.getMessage());
38             }
39             catch (IOException exception)
40             {
41                exception.printStackTrace();
42             }
43          }
44       }
45    }
```

The `readFile` Method of the `DataSetReader` Class

- Constructs `Scanner` object
- Calls `readData` method
- Completely unconcerned with any exceptions
- If there is a problem with input file, it simply passes the exception to caller:

```
public double[] readFile(String filename) throws IOException
{
   File inFile = new File(filename);
   Scanner in = new Scanner(inFile);
   try
   {
      readData(in);
      return data;
   }
   finally { in.close(); }
}
```

The readData Method of the DataSetReader Class

- Reads the number of values
- Constructs an array
- Calls readValue for each data value:

```
private void readData(Scanner in) throws BadDataException
{
   if (!in.hasNextInt())
   {
      throw new BadDataException("Length expected");
   }
   int numberOfValues = in.nextInt();
   data = new double[numberOfValues];
   for (int i = 0; i < numberOfValues; i++)
      readValue(in, i);
   if (in.hasNext())
      throw new BadDataException("End of file expected");
}
```

The `readData` Method of the `DataSetReader` Class

- Checks for two potential errors:
  1. File might not start with an integer
  2. File might have additional data after reading all values.
- Makes no attempt to catch any exceptions.

The readValue Method of the DataSetReader Class

```
private void readValue(Scanner in, int i) throws BadDataException
{
   if (!in.hasNextDouble())
      throw new BadDataException("Data value expected");
   data[i] = in.nextDouble();
}
```

Error Scenario

1. `DataAnalyzer.main` calls `DataSetReader.readFile`
2. `readFile` calls `readData`
3. `readData` calls `readValue`
4. `readValue` doesn't find expected value and throws `BadDataException`
5. `readValue` has no handler for exception and terminates
6. `readData` has no handler for exception and terminates
7. `readFile` has no handler for exception and terminates after executing `finally` clause and closing the `Scanner` object
8. `DataAnalyzer.main` has handler for `BadDataException`
   - Handler prints a message
   - User is given another chance to enter file name

section_5/DataSetReader.java

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.util.Scanner;
4
5  /**
6     Reads a data set from a file. The file must have the format
7     numberOfValues
8     value1
9     value2
10    . . .
11 */
12 public class DataSetReader
13 {
14    private double[] data;
15
```

Continued

section_5/DataSetReader.java

```
16      /**
17         Reads a data set.
18         @param filename the name of the file holding the data
19         @return the data in the file
20      */
21      public double[] readFile(String filename) throws IOException
22      {
23         File inFile = new File(filename);
24         Scanner in = new Scanner(inFile);
25         try
26         {
27            readData(in);
28            return data;
29         }
30         finally
31         {
32            in.close();
33         }
34      }
35
```

Continued

section_5/DataSetReader.java

```
36      /**
37         Reads all data.
38         @param in the scanner that scans the data
39      */
40      private void readData(Scanner in) throws BadDataException
41      {
42         if (!in.hasNextInt())
43         {
44            throw new BadDataException("Length expected");
45         }
46         int numberOfValues = in.nextInt();
47         data = new double[numberOfValues];
48
49         for (int i = 0; i < numberOfValues; i++)
50         {
51            readValue(in, i);
52         }
53
54         if (in.hasNext())
55         {
56            throw new BadDataException("End of file expected");
57         }
58      }
59
```

Continued

section_5/DataSetReader.java

```
60      /**
61         Reads one data value.
62         @param in the scanner that scans the data
63         @param i the position of the value to read
64      */
65      private void readValue(Scanner in, int i) throws BadDataException
66      {
67         if (!in.hasNextDouble())
68         {
69            throw new BadDataException("Data value expected");
70         }
71         data[i] = in.nextDouble();
72      }
73   }
```

section_5/BadDataException.java

```
1  import java.io.IOException;
2
3  /**
4     This class reports bad input data.
5  */
6  public class BadDataException extends IOException
7  {
8     public BadDataException() {}
9     public BadDataException(String message)
10    {
11       super(message);
12    }
13 }
```

Self Check 11.24

Why doesn't the `DataSetReader.readFile` method catch any exceptions?

Answer: It would not be able to do much with them. The `DataSetReader` class is a reusable class that may be used for systems with different languages and different user interfaces. Thus, it cannot engage in a dialog with the program user.

Self Check 11.25

Suppose the user specifies a file that exists and is empty. Trace the flow of execution.

Answer: `DataAnalyzer.main` calls `DataSetReader.readFile`, which calls `readData`. The call `in.hasNextInt()` returns `false`, and `readData` throws a `BadDataException`. The `readFile` method doesn't catch it, so it propagates back to main, where it is caught.

Self Check 11.26

If the `readValue` method had to throw a `NoSuchElementException` instead of a `BadDataException` when the next input isn't a floating-point number, how would the implementation change?

Answer: It could simply be

```
private void readValue(Scanner in, int i)
{
   data[i] = in.nextDouble();
}
```

The `nextDouble` method throws a `NoSuchElementException` or a `InputMismatchException` (which is a subclass of `NoSuchElementException`) when the next input isn't a floating-point number. That exception isn't a checked exception, so it need not be declared.

Self Check 11.27

Consider the `try/finally` statement in the `readFile` method. Why was the `in` variable declared outside the `try` block?

Answer: If it had been declared inside the `try` block, its scope would only have extended until the end of the `try` block, and it would not have been accessible in the `finally` clause.

Self Check 11.28

How can the program be simplified when you use the "automatic resource management" feature described in Special Topic 11.6?

Answer: The `try/finally` statement in the `readFile` method can be rewritten as

```
try (Scanner in = new Scanner(inFile))
{
   readData(in);
   return data;
}
```