Reading and Writing Text Files Exceptions

15-121 Fall 2010

Margaret Reid-Miller

Reading a Text File

- A Scanner object can be connected to many input sources: keyboard, file, network, string
- To read a text file, we create a `Scanner` object passing a `File` object instead of `System.in` as the argument:

creates a Java `File` object (does not create a file on your disk)

```
// Read from data.txt file
Scanner inFile =
      new Scanner(new File("data.txt"));
int numValues = inFile.nextInt();
```


Testing for more input

- Scanner has methods to check for more input:

  `boolean hasNextLine()`
  Return `true` if the scanner object has another line in its input.

  `boolean hasNext()`
  Return `true` if the scanner object has any more tokens in its input.

  `boolean hasNextInt()`
  Return `true` if the scanner object has another token in its input and that token can be read as an `int`.

  `boolean hasNextDouble()`
  Return `true` if the scanner object has another token in its input and that token can be read as a `double`.

- These methods do not consume input; They just say whether and what kind of input is waiting.

Example: Count Words

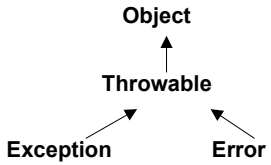
```
public static void main(String[] args)
                        throws IOException {
   Scanner inFile = new Scanner
         (new File("essay.txt"));
   int count = 0;
   while (inFile.hasNext()) {
      String word = inFile.next();
      count++;
   }

   System.out.println("Number of words is "
                  + count);
}
```

Exceptions

- Exceptions are unusual or erroneous situations from which a program may be able to recover.
- Often the problem is out of the control of the program, such as bad user input.
- When a exception occurs the Java runtime system creates an Exception object that holds information about the problem.
- An exception will cause a program to halt unless it is caught and handled with special code.

Errors

- Errors are problems that are so severe that it is not possible to recover from them, e.g., hardware error.
- Errors are objects of the class Error.
  - Example: `OutOfMemoryError`
- Programs can recover from Exceptions but must stop running for Errors.



Two Types of Exceptions

- **Checked** (compile-time) - Generally indicates invalid conditions outside of the program.
  - The compiler **requires** that you handle these exceptions explicitly.
  - Examples: `IOException`
    `FileNotFoundException`
- **Unchecked** (runtime)- Generally indicates error in the program's logic.
  - Examples: `ArrayIndexOutOfBoundsException`
    `NullPointerException`
- Any exception that inherits from **`RuntimeException`** is unchecked, otherwise it is checked.

Common Exceptions with Files

`FileNotFoundException:` (compile-time)

Could not find the file: The file should be in the folder from which you invoke the program.

`NoSuchElementException:` (runtime)

Attempted to read passed the end of the file. (E.g., A loop reads 8 integers when there are only 7 integers.)

`InputMismatchException:` (runtime)

Attempted to read one type of token, but the next token was a different type. (E.g., Used `nextInt` when the next token contained letters.)

Handling Exceptions

When an exception occurs a method can either

- ***catch*** the exception and execute some code to "handle" it, or
- ***throw*** the exception back to the method that called this method so that it may handle the exception.
  - If the exception is a (compile-time) checked exception, the method must declare that it throws the exception.

Throwing an Exception Explicitly

```
import java.util.Scanner;
import java.io.*;  // for File

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

      Scanner fileInput = new Scanner(
                        new File("data.txt"));
```

If there is a problem with opening the file for reading, Scanner will throw an exception.

If Scanner throws an exception, then main will throw it also (instead of catching it).

Example

- A ***robust*** program handles bad input gracefully, such as asking the user to reenter the name of a file when the file is not found.

```
Scanner console = new Scanner(System.in);
System.out.print("Enter file name: ")
String fileName = console.nextLine(fileName);

Scanner fileIn = new Scanner(
             new File(fileName));
```

If the user mistypes the file name, Scanner throws a **`FileNotFoundExcpetion`**.

Catching the Exception

```
String fileName = null;
do {
   System.out.print("Enter file name: ")
   String fileName = console.nextLine(fileName);
   try {
      Scanner in = new Scanner(new File(fileName));
   }
   catch (FileNotFoundException ex) {
      System.out.println("Error: File not found");
      fileName = null;
   }
}while (fileName == null);
```

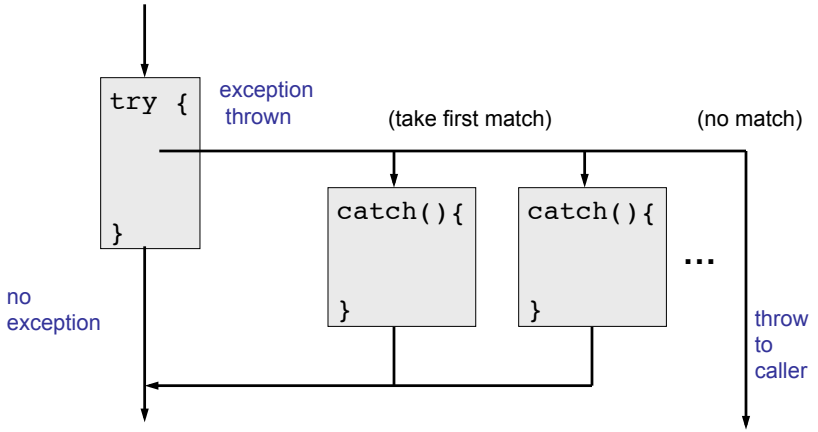
`ex` is a reference to the exception object.

The catch block can call methods on `ex` to find out more details about the exception.

Using a try-catch Statement

- Put code that might throw an exception in a **`try`** block.
- Put code that handles the exception in a **`catch`** block immediately following the `try` block.
- When the code inside the try block is executed:
  - If there are no exceptions, execution skips the catch block and continues after the catch block.
  - If there is an exception, execution goes **immediately** to the catch block and then continues after the catch block.

Flow of Control



Writing a Text File

Class requires
`import java.io.*;`

- Writing to a text file is similar to writing to `System.out` (console):

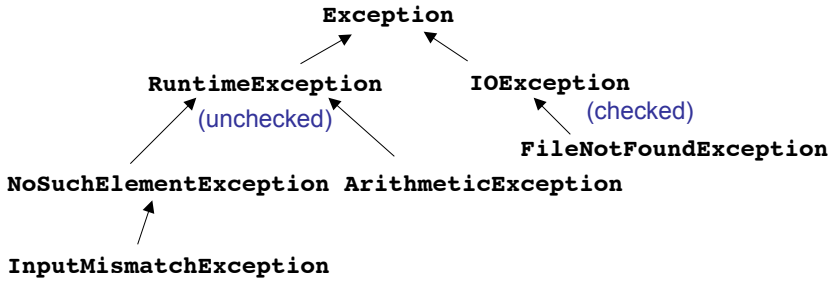
```
public static void main(String[] args)
                        throws IOException {
…
PrintWriter outFile = new PrintWriter(
                        "results.txt");
outFile.println("ANALYSIS for " + infileName);
outFile.print("Number of samples");
…
outFile.close();
```

name of file to create

Must close output file to ensure all the data are written to the file before the program terminates.

Exception Hierarchy

By looking at the Java API, you can determine whether an exception inherits from the **`RuntimeException`** class. If not, the method must catch or specify it throws the exception.



Using `throws` vs `try-catch`

- Methods can either catch or throw exceptions that occur during its execution.
- If the method throws it to its caller, the caller similarly can either catch or throw the exception.
- If the exception gets thrown all the way back to the main method and the main method also throws it, the runtime system stops the program and prints the exception with a “call stack trace.”
- Deciding which method should handle an exception is a software design decision, as is defining new exceptions for special types of errors.