File Handling

Info 2

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

File-based input-output

- Input-output is particularly error-prone because it involves interaction with the external environment.
- The **`java.io`** package supports input-output.
- **`java.io.IOException`** is a checked exception.

File and Path

- **`java.io.File`** provides information about files and folders/directories.
- **`java.nio.file.Path`** is the Java 7 alternative.
- **`File`** is a class; **`Path`** is an interface.
- The **`Files`** and **`Paths`** classes in **`java.nio.file`** are also used.

Readers, writers, streams

- Readers and writers deal with textual input.
  - Based around the **`char`** type.
- Streams deal with binary data.
  - Based around the **`byte`** type.
- The *address-book-io* project illustrates textual I/O.

File output

- The three stages of file output.
  - Open a file.
  - Write to the file.
  - Close the file (but see ARM in Java 7).
  - (ARM - Automatic Resource Management)
- Failure at any point results in an **`IOException`**.
- Use **`FileWriter`** for text files.

Text output to file

```
try {
    FileWriter writer = new FileWriter("name of file");
    while(there is more text to write) {
        ...
        writer.write(next piece of text);
        ...
    }
    writer.close();
}
catch(IOException e) {
    something went wrong with accessing the file
}
```

What might be problematic with this??

See the *weblog-analyzer* project

Try-with-resource – Java 7

- Used for ensuring 'resources' are closed after use.
- Removes need for explicit closure on both successful and failed control flows.
- Also known as 'automatic resource management' (ARM).

Try-with-resource

```
try(FileWriter writer = new FileWriter("name of file")) {
    while(there is more text to write) {
        ...
        writer.write(next piece of text);
        ...
    }
}
catch(IOException e) {
    something went wrong with accessing the file
}
```

No close() call required in either clause.
See the *weblog-analyzer-v7* project.

Text input from file

- Use the **`FileReader`** class.
- Augment with **`BufferedReader`** for line-based input.
  - Open a file.
  - Read from the file.
  - Close the file (but see ARM in Java 7).

Text input from file

```
try {
    BufferedReader reader =
        new BufferedReader(new FileReader("filename"));
    String line = reader.readLine();
    while(line != null) {
        do something with line
        line = reader.readLine();
    }
    reader.close();
}
catch(FileNotFoundException e) {
    the specified file could not be found
}
catch(IOException e) {
    something went wrong with reading or closing
}
```

See *tech-support-io*

Text input from file – Java 7

- **`BufferedReader`** created via static **`newBufferedReader`** method in the **`java.nio.file.Files`** class.
- Requires a **`Charset`** from **`java.nio.charset`**, e.g.:
  - **`"US-ASCII"`**
  - **`"ISO-8859-1"`**

Text input – Java 7

```
Charset charset =
        Charset.forName("US-ASCII");
Path path = Paths.get("file");
try(BufferedReader reader =
      Files.newBufferedReader(path, charset)) {
    use reader to process the file
}
catch(FileNotFoundException e) {
   deal with the exception
}
catch(IOException e) {
   deal with the exception
}
```

Text input from the terminal

- **`System.in`** maps to the terminal:
  - Its type is **`java.io.InputStream`**
- It is often wrapped in a **`java.util.Scanner`**.
- **`Scanner`** with **`File`** is an alternative to **`BufferedReader`** with **`FileReader`**.

Scanner: parsing input

- **`Scanner`** supports *parsing* of textual input.
  - **`nextInt`**, **`nextLine`**, etc.
- Its constructors support **`String`**, **`File`** and **`Path`** (Java 7) arguments.

Review

- Input/output is an area where errors cannot be avoided.
- The environment in which a program is run is often outside a programmer's control.
- Exceptions are typically *checked*.

Review

- Key classes for text input/output are **`FileReader`**, **`BufferedReader`**, **`FileWriter`** and **`Scanner`**.
- Binary input/output involves **`Stream`** classes.
- Java 7 introduces the **`Path`** interface as an alternative to **`File`**.
- Java 7 introduces try-with-resource.

Let's put it all together!

- Write a command line app that:
  - takes a file name as argument
  - takes an output file name or writes to std out
  - does some conversion or analytics in the file (collect ideas for that)
  - put a focus on error handling on different levels (File Name, single line?)
  - use Object-Oriented Design: Create Classes, Distribute Responsibilities
  - JUnit-Test your application!