CS133: Developing Programming Principles

Lecture 18

Text Files

Not the Book's Approach

We're doing things a little differently than the approach in the text.

Text files

A text file contains a number of lines, each consisting of a sequence of characters.

```
To: Carrie
From: Charlie
The quick brown
fox jumped over
the lazy dog.
```


Reading a Text File

1. Open the file, using a name known to the operating system:
   ```
   C:\My Stuff\CS133\Junk\test.txt
   /u/charlie/Courses/CS133/test.txt
   ```
2. Read and process each line of text.
3. Close the file.

Opening the File

```
import java.io.*;
import java.util.*;

FileReader in
      = new FileReader(filename);
Scanner myScanner
      = new Scanner(in);
```


Class FileReader

`FileReader` is a subclass of **`InputStream`** and can be used to create a **`Scanner`**.

```
FileReader in
  = new FileReader("C:\\test.txt");
```

Throws **`FileNotFoundException`** if the file does not exist or can't be opened.

Reading and Printing a File

```
try {
  FileReader in
      = new FileReader("C:\\test.txt");
  Scanner scanner = new Scanner(in);
  while (scanner.hasNext()) {
    System.out.println(scanner.nextLine());
  }
  scanner.close();
} catch (FileNotFoundException e) {
  System.exit(0);
}
```


Scanner Class

All methods of the **`Scanner`** class may be used to read from the file.

```
public boolean hasNext()
```

The **`hasNext()`** method may be used to determine when the end of the file is reached.

Counting Lines of Code

A commonly used software engineering metric is “lines of code”.

Is this a good way of measuring programmer productivity? Is it a good way of estimating effort?

How many lines did you write today?

Counting the number of lines in a Java file

```
int lines = 0;
try {
  FileReader in
      = new FileReader("C:\\X.java");
  Scanner scanner = new Scanner(in);
  while (scanner.hasNext()) {
    scanner.nextLine();
    lines++;
  }
  scanner.close();
} catch (FileNotFoundException e) {
}
```

Writing a Text File

1. Create and open the file, using a name known to the operating system:
   ```
   C:\My Stuff\CS133\Junk\test.txt
   /u/charlie/Courses/CS133/test.txt
   ```
2. Generate and write each line of text.
3. Close the file.

Creating/Opening the File

```
import java.io.*;

FileWriter out
      = new FileWriter(filename);
PrintWriter writer
      = new PrintWriter(out);
```

Class **`FileWriter`**

Creates and opens an empty file for writing. If the file exists, it's overwritten.

`FileWriter` throws an **`IOException`** if the file can't be created for writing.

Class **`PrintWriter`**

`PrintWriter` supports all the **`print()`** and **`println()`** methods supported by **`System.out`**.

(Although **`System.out`** is not itself a **`PrintWriter`**.)

Example

Write a class to maintain information about a user between executions of a program (e.g., name and age).

Store the information before exit.

Restore the information at the start of execution.

Example

```
import java.util.*;
import java.io.*;
public class User {
  private String name;
  private int age;

  public User() {
    this.name = "";  this.age = 0;
  }

  public User(String name, int age) {
    this.name = name; this.age = age;
  }
```

Example

```
public void setName(String name) {
  this.name = name;
}

public void setAge(int age) {
  this.age = age;
}

public String getName() { return name;}

public int getAge() { return age;}
```


Example – save method

```
public boolean save(String fileName) {
 PrintWriter writer;
 try {
   FileWriter out = new FileWriter(fileName);
   writer = new PrintWriter(out);
 } catch (IOException e) {
   return false;
 }
 writer.println(name);
 writer.println(age);
 writer.close();
 return true;
}
```

Example – `restore` method

```
public boolean restore(String fileName) {
 Scanner scanner;
 try {
   FileReader in = new FileReader(fileName);
   scanner = new Scanner(in);
 } catch (FileNotFoundException e) {
   return false;
 }
 name = scanner.nextLine();
 age = scanner.nextInt();
 scanner.close();
 return true;
}
```


Example - Saving

```
String fileName = "C:\\userInfo.txt";

User user = new User("Sidney Crosby",
                     18);

user.save(fileName);
```

Example – File Contents

C:\userInfo.txt

```
Sidney Crosby
18
```


Example - Restoring

```
String fileName = "C:\\userInfo.txt";

User user = new User();

user.restore(fileName);
```

Appending to a Text File

To add text to an existing file a different **`FileWriter`** constructor must be used:

```
public FileWriter(String fileName,
                  boolean append);
```

If **`append`** is **`true`**, text will be written to the end of the file.

Summary

– Reading from text files.

– Writing to text files.

– Appending to text files.

ERROR: undefined
OFFENDING COMMAND:

STACK: