CSCI 1103: File I/O, Scanner, PrintWriter

Chris Kauffman

Last Updated:
Mon Dec 4 10:03:11 CST 2017

Logistics

Reading from Eck

- Ch 2.1 on Input, File I/O
- Ch 11.1-2 on File I/O

Goals

- Scanner for input
- Input from different sources
- File output with PrintWriter

Lab12: Scanner

Exercises on reading input from different sources

Project 5

- Will post in next two days
- Due last few days of class

Input / Output (I/O) so far

Input

- So far have used the `TextIO` to get input from users
- `TextIO` was written by Eck (text author) for beginners: *not* part of standard Java
- `Scanner` is a similar class which *is* part of the standard Java library
- Will study `Scanner` to read from user, `String`, and `File`

Output

- So far only `System.out.println() / printf()` to put stuff on the screen
- Can also put stuff in files for permanent storage
- Will use `PrintWriter` for that

The Scanner Class

- On object which reads input from some source
- Consider demo program below: similar to `TextIO.java`

```
 1  // Short demo of the scanner class which reads an integer, double,
 2  // string from the user
 3
 4  import java.util.Scanner;    // notify compiler of Scanner use
 5
 6  public class ScannerDemo{
 7    public static void main(String args[]){
 8      System.out.println("Enter int double string:");
 9      Scanner input = new Scanner(System.in); // scan from keyboard
10      int i = input.nextInt();                // read an int
11      double d = input.nextDouble();          // read a double
12      String s = input.next();                // read a string
13      String s2 = input.next();                // read a string
14      System.out.printf("%d %f %s %s\n",i,d,s,s2);
15    }
16  }
```

Where could one find out about all the methods of Scanner?

Imports and Packages

- Java's library has MANY classes divided into packages
- Essential classes like `String`, `System`, `Math` live in the `java.lang` package which is always imported by default
- Less essential classes like `Scanner` live elsewhere such as in the `java.util` package
- Other packages must be imported with lines like

```
import java.util.Scanner;   // notify compiler of Scanner use
import java.io.File;        // notify compiler of File use
import java.util.ArrayList; // notify compiler ArrayList use

import java.util.*;         // using multiple classes from package
```

- Packages are a complex topic for another day

Detecting the End of Input

```
class Scanner{
  public String next()
  // Finds and returns the next complete token from
  // this scanner.

  public boolean hasNext()
  // Returns true if this scanner has another token
  // in its input. CK: safe to call next()
```

- In input loops, make use of `hasNext()` to determine if more input is present
- For interactive input, special characters can be used to signal end of input
  - Mac/Unix: press Ctrl-D to signal end of input
  - Windows: press Ctrl-Z to signal end of input

Sample of Detecting End of Input

```
 1  // Demonstrate the hasNext() method of Scanner
 2  import java.util.Scanner;
 3
 4  public class EndOfInput{
 5    public static void main(String args[]){
 6      System.out.println("To end input");
 7      System.out.println("- Mac/Unix: press Ctrl-D");
 8      System.out.println("- Windows:  press Ctrl-Z");
 9      Scanner input = new Scanner(System.in);
10      int count = 0;
11      String allStrings = "";
12
13  //    System.out.println("Enter a string:");
14      while(input.hasNext()){           // loop while input
15        System.out.println("Enter a string:");
16        String str = input.next();      // get a string
17        count++;                        // increment
18        allStrings += str + "\n";       // tack on to all
19        //System.out.println("Enter a string:");
20      }
21
22      System.out.printf("End of input reached\n");
23      System.out.printf("%d total strings typed\n",count);
24      System.out.printf("All strings:\n%s\n",allStrings);
25    }
26  }
```

Exercise: Sum Integers

- Repeatedly prompt for input
- Use `nextInt()` to read an int
- Add on to a total
- Go until end of input is indicated
- Use `hasNext()` for this

```
> javac SumIntegers.java
> java SumIntegers
Stop with Ctrl-D (Mac/Unix) or Ctrl-Z
Enter an integer:
12
Enter an integer:
14
Enter an integer:
1
Enter an integer:
9
Enter an integer:
End of input
Total: 36
```

Answer: Sum Integers

```
 1  // Exercise to sum numbers
 2  import java.util.Scanner;
 3
 4  public class SumIntegers{
 5    public static void main(String args[]){
 6      System.out.println("Stop with Ctrl-D (Mac/Unix) or Ctrl-Z (Windows)");
 7
 8      Scanner input = new Scanner(System.in);
 9      int total = 0;
10      System.out.println("Enter an integer:");
11      while(input.hasNext()){           // loop while input
12        int num = input.nextInt();      // get a number
13        total += num;                   // increment
14        System.out.println("Enter an integer:");
15      }
16      System.out.printf("End of input\n");
17      System.out.printf("Total: %d\n",total);
18    }
19  }
```

Constructors for Scanner

- Scanners are initialized with a source from which they read
- Mostly behave identically irrespective of a source: useful

```
// READ FROM THE TERMINAL
// Constructs a new Scanner that produces values
// scanned from the specified input stream.
Scanner in = new Scanner(System.in);

// READ FROM A STRING
// Constructs a new Scanner that produces values
// scanned from the specified string.
Scanner in = new Scanner("Give me a 1 Give me a 2");

// READ FROM A FILE
// Constructs a new Scanner that produces values
// scanned from the specified File.
Scanner in = new Scanner(new File("stuff.txt"));
// !! Oh - that IS interesting
```

Sum integers from Strings and Files

```
1  // Demo to sum integers from
2  // different strings
3  import java.util.Scanner;
4
5  public class SumStringInts{
6    public static void main(String args[]){
7      Scanner input;
8      int total;
9
10     String nums1 = "1 2 3 4";
11     input = new Scanner(nums1);
12     total = 0;
13     while(input.hasNext()){
14       int num = input.nextInt();
15       total += num;
16     }
17     System.out.printf("Total: %d\n",
18                       total);
19
20     String nums2 = "10 30 50 70 90";
21     input = new Scanner(nums2);
22     total = 0;
23     while(input.hasNext()){
24       int num = input.nextInt();
25       total += num;
26     }
27     System.out.printf("Total: %d\n",
28                       total);
29   }
30 }
```

```
1  // Demo to sum integers from a file
2  import java.util.Scanner;
3  import java.io.File;  // using File class
4
5  public class SumFileInts{
6    public static void main(String args[]){
7      try{
8        String filename = "numbers.txt";
9        Scanner input =
10         new Scanner(new File(filename));
11       int total = 0;
12       while(input.hasNext()){
13         int num = input.nextInt();
14         total += num;
15       }
16       System.out.printf("Total: %d\n",
17                         total);
18       input.close(); // for file scanning
19     }
20     catch(Exception e){
21       throw new RuntimeException(e);
22     }
23   }
24 }
```

What looks ugly and interesting in these examples?

Exceptions when Reading from Files

- Things can go wrong with `File` operations
- Typically throws a checked exception: one that must be caught or declared
- Will see methods mentioning checked exceptions they throw

```
public void aMethod(...) throws KersplosionException
                                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

- This is more complex than needed for our cases: don't want to catch OR declare so we will often take the following unsavory approach

```
public void someMethod(...){
  try {
    ...
    Scanner input = new Scanner(new File(filename));
    ...
  }
  catch(Exception e){                 // if an exception results
    throw new RuntimeException(e); // re-throw as a RuntimeException
  }
}
```

Common Pattern

- Lots of repeated code to loop and sum
- Really want one spot for this: a method like

```
public static int sumAll(Scanner input)
// read all integers in input and return their sum
```

- Could then pass in an open scanner as in

```
Scanner input;
input = new Scanner("1 2 3");
int sum1 = sumAll(input);
input = new Scanner("4 5 6 7");
int sum2 = sumAll(input);
input = new Scanner(new File(fname));
int sum3 = sumAll(input);
input.close();
```

Exercise: Pick a File to Sum

- Have several files like `numbers.txt`, `scores.txt`, `scores2.txt`
- Want to pick which one to sum up
- How could this be done using `Scanner`?
- *Hint: Using multiple Scanners with different sources may help*

Answer: Pick a File to Sum

See `SumPickFile.java`. Central technique: read from two places

```
Scanner userInput = new Scanner(System.in); // to read typed input
System.out.println("Enter a filename:");    // prompt
String filename = userInput.next();         // get filename

// to read file input
Scanner fileInput = new Scanner(new File(filename));

int total = sumAll(fileInput);              // method to sum all
fileInput.close();
System.out.printf("Total: %d\n", total);
```

- The pattern of specifying a file for a program to operate on is *very* common
- It is much more common to give this as a command line argument so it appears in `main(String args[])`
- Command line args be discussed soon

Scanning Errors

```
 1  // Demo of scanning errors
 2  import java.util.Scanner;
 3
 4  public class ScanError{
 5    public static void main(String args[]){
 6      String nums = "5 4 three 2 1";
 7      Scanner input = new Scanner(nums);
 8      int total = 0;
 9      while(input.hasNext()){
10        int num = input.nextInt();
11        System.out.printf("read %d\n",num);
12        total += num;
13      }
14      System.out.printf("Total: %d\n", total);
15    }
16  }
```

- Scanners read what they are told and throw exceptions if problems arise
- Next input can be checked with methods like `hasNextInt()` and `hasNextDouble()`
- Careful with these as loop conditions

```
> javac ScanError.java
> java ScanError
read 5
read 4
Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:864)
  at java.util.Scanner.next(Scanner.java:1485)
  at java.util.Scanner.nextInt(Scanner.java:2117)
  at java.util.Scanner.nextInt(Scanner.java:2076)
  at ScanError.main(ScanError.java:10)
```

Scanner Input Method Summary

Method	Purpose
`new Scanner(System.in)`	scan from terminal input
`new Scanner("hi 1 2.3")`	scan from string
`new Scanner(new File(fname))`	scan from file
`close()`	when reading files
`String next()`	get next String
`int nextInt()`	get next integer
`double nextDouble()`	get next double
`String nextLine()`	read a whole line
`boolean hasNext()`	`true` if more to read
`boolean hasNextInt()`	`true` if next is an int
`boolean hasNextDouble()`	`true` if next is a double
`boolean hasNextLine()`	`true` there is another line

PrintWriter for File Output

- A class that allows printing to the screen or to a file
- Constructing a `PrintWriter` with a `File` argument allows printing into that named file
- Similar to constructing a `Scanner` from a `File` argument reads from the file
- Methods of `PrintWriter` are familiar: `println()` / `printf()` along with `close()`

```
PrintWriter fileOut =
  new PrintWriter(new File("myfile.txt"));

fileOut.println("Here is text");
fileOut.printf("Numbers: %d %f\nStrings: %s\n",
               123, 4.56, "hello!");

fileOut.close();
```

Have a look at the PrintWriter Java Doc.

File Output Example

```
 1  // Show PrintWriter use to create a text file with a table
 2  import java.util.Scanner;
 3  import java.io.PrintWriter;
 4  import java.io.File;
 5
 6  public class PrintTable{
 7    public static void main(String args[]) throws Exception{
 8      Scanner input = new Scanner(System.in);
 9      System.out.println("File to save in (ex: stuff.txt):");
10      String fileName = input.next();
11      System.out.println("Height and Width of table (ex: 4 6):");
12      int height = input.nextInt();
13      int width  = input.nextInt();
14
15      PrintWriter fileOut = new PrintWriter(new File(fileName));
16      for(int row=1; row<=height; row++){
17        for(int col=1; col<=width; col++){
18          fileOut.printf("%3d ",row * col);
19        }
20        fileOut.println("");
21      }
22      fileOut.close();
23
24      System.out.printf("%d by %d table saved in %s\n",
25                        height,width,fileName);
26    }
27  }
```

Exercise: File Output Example

Examine `PrintTable.java`

1. Identify where a `PrintWriter` is created and how
2. Explain what methods are used to print output into the file and whether they are familiar or unfamiliar
3. Which method terminates output to that file?
4. How is the name of the output file and table size determined?
5. Explain a different way the filename/table size could be obtained at startup time based on last Friday's lecture
   Hint: you may need to use the Integer.parseInt(i) method to convert a String to an int

Answers: File Output Example

1. Identify where a `PrintWriter` is created and how
   ```
   PrintWriter fileOut =
     new PrintWriter(new File(fileName));
   ```
2. Explain what methods are used to print output into the file and whether they are familiar or unfamiliar
   ```
   fileOut.printf("%3d ",row * col);
   fileOut.println("");
   ```
   Both familiar from use with `System.out`
3. Which method terminates output to that file?
   ```
   fileOut.close();
   ```
4. How is the name of the output file and table size determined?
   A: A Scanner is used to read them from the user
5. Explain a different way the filename/table size could be obtained at startup time based on last Friday's lecture
   A: They could be specified on the command line and grabbed from the `args[]` array in `main()`
   ```
   public static void main(String args[]) throws Exception{
     String fileName = args[0];
     int height = Integer.parseInt(args[1]);
     int width  = Integer.parseInt(args[2]);
   ```
   See `PrintTableCmdArg.java`