Lesson 7: Improving the User Interface

Updated for Java 1.5,
(with additions and modifications by)
Mr. Dave Clausen

Lesson 7: Improving the User Interface

- Construct a query-driven terminal interface.
- Construct a menu-driven terminal interface.
- Construct a graphical user interface.
- Format text, including numbers, for output.
- Handle number format exceptions during input.

Lesson 7: Improving the User Interface

Vocabulary:

- Menu-driven program
- Query-controlled input
- Format specifiers
- Format String
- Format flags
- Exceptions

7.1 A Thermometer Class

- The demonstrations in this lesson involve converting temperatures between Fahrenheit and Celsius.
- To support these conversions we first introduce a Thermometer class
  - Thermometer.java Thermometer.txt
- This class stores the temperature internally in Celsius; however, the temperature can be set and retrieved in either Fahrenheit or Celsius.

7.1 A Thermometer Class

```
public class Thermometer {

   private double degreesCelsius;

   public void setCelsius(double degrees){
      degreesCelsius = degrees;
   }

   public void setFahrenheit(double degrees){
      degreesCelsius = (degrees - 32.0) * 5.0 / 9.0;
   }

   public double getCelsius(){
      return degreesCelsius;
   }

   public double getFahrenheit(){
      return degreesCelsius * 9.0 / 5.0 + 32.0;
   }
}
```

7.2 Repeating Sets of Inputs

- Another technique for handling repeating sets of inputs is called ***query controlled input***.
- Before each set of inputs, after the first, the program asks the user if there are more inputs.
- Figure 7-1 shows an example:

7.2 Repeating Sets of Inputs

```
Enter degrees Fahrenheit: 32
The equivalent in Celsius is 0.0

Do it again (y/n)? y

Enter degrees Fahrenheit: 212
The equivalent in Celsius is 100.0

Do it again (y/n)?
```

7.2 Repeating Sets of Inputs

- The program is implemented by means of two classes -- a class to handle the user ***interface*** and the Thermometer class.
- Following is pseudocode for the interface (client) class:

```
instantiate a thermometer
String doItAgain = "y"
while (doItAgain equals "y" or "Y"){
   read degrees Fahrenheit and set the thermometer
   ask the thermometer for the degrees in Celsius and display
   read doItAgain          //The user responds with "y" or "n"
}
```

7.2 Repeating Sets of Inputs

- The key to this pseudocode is the String variable doItAgain.
- This variable controls how many times the loop repeats.
- Initially, the variable equals “y”.
- As soon as the user enters a character other than "y" or "Y", the program terminates.
- Here is a complete listing of the interface class:

7.2 Repeating Sets of Inputs

```
/* ConvertWithQuery.java     ConvertWithQuery.txt
Repeatedly convert from Fahrenheit to Celsius until the user
signals the end.
*/
import java.util.Scanner;

public class ConvertWithQuery{
   public static void main(String [] args) {
      Scanner reader = new Scanner(System.in);
      Thermometer thermo = new Thermometer();
      String doItAgain = "y";

      while (doItAgain.equals("y") || doItAgain.equals("Y")){
         System.out.print("\nEnter degrees Fahrenheit: ");
         thermo.setFahrenheit(reader.nextDouble());
         // Consume the trailing end of line
         reader.nextLine();
         System.out.println("The equivalent in Celsius is " +
                                thermo.getCelsius());
         System.out.print("\nDo it again (y/n)? ");
         doItAgain = reader.nextLine();
      }
   }
}
```

7.2 Repeating Sets of Inputs

- In the previous code, observe that a String literal is enclosed within double quotes.
- doItAgain = reader.nextLine(); is used to read in the String
- "Y" and "y" are not the same, we need to check for either.
- while (doItAgain.equals("y") || doItAgain.equals("Y")) is used to check if the string is equal to “y” or “Y”.
  - Strings cannot use = = to check for equivalence, you must use .equals

7.3 A Menu-Driven Conversion Program

- Menu-driven programs begin by displaying a list of options from which the user selects one.
- The program then prompts for additional inputs related to that option and performs the needed computations, after which it displays the menu again.
- Figure 7-2 shows how this idea can be used to extend the temperature conversion program.

7.3 A Menu-Driven Conversion Program

```
1) Convert from Fahrenheit to Celsius
2) Convert from Celsius to Fahrenheit
3) Quit
Enter your option: 1

Enter degrees Fahrenheit: 212
The equivalent in Celsius is 100

1) Convert from Fahrenheit to Celsius
2) Convert from Celsius to Fahrenheit
3) Quit
Enter your option: 2

Enter degrees Celsius: 0
The equivalent in Fahrenheit is 32

1) Convert from Fahrenheit to Celsius
2) Convert from Celsius to Fahrenheit
3) Quit
Enter your option: 3
```

7.3 A Menu-Driven Conversion Program

Following is the corresponding pseudocode and the source code:

ConvertWithMenu.java ConvertWithMenu.txt

```
instantiate a thermometer
menuOption = 4
while (menuOption != 3){
   print menu
   read menuOption
   if (menuOption == 1){
      read fahrenheit and set the thermometer
      ask the thermometer to convert and print the results
   }else if (menuOption == 2){
      read celsius and set the thermometer
      ask the thermometer to convert and print the results
   }else if (menuOption != 3)
      print "Invalid option"
}
```

Generic Menu Driven Program

- Here is a generic Menu Driven Program that uses "Stub Programming" as generic place holders:

MenuDrivenStub.java

MenuDrivenStub.txt

7.4 Formatted Output with `printf` and `format`

- Java 5.0 includes method `printf` for formatting output.
  - Requires **format string** and data values
- General form of `printf`:

```
printf(<format string>, <expression-1>, …, <expression-n>)
```

7.4 Formatted Output with `printf` and `format` (cont.)

- Format string is a combination of literal string information and formatting information.
  - Formatting information consists of one or more **format specifiers**.
    - Begin with a '%' character and end with a letter that indicates the format type
    - Format.java Format.txt

```
double dollars = 25;
double tax = dollars * 0.125;
System.out.printf("Income: $%.2f%n", dollars);
System.out.printf("Tax owed: $%.2f%n", tax);
```

7.4 Formatted Output with `printf` and `format` (cont.)

Table 7-1: Commonly used format types

CODE	FORMAT TYPE	EXAMPLE VALUE
d	Decimal integer	34
x	Hexadecimal integer	A6
o	Octal integer	47
f	Fixed floating-point	3.14
e	Exponential floating-point	1.67e+2
g	General floating-point (large numbers in exponential and small numbers in fixed-point)	3.14
s	String	Income:
n	Platform-independent end of line	

7.4 Formatted Output with `printf` and `format` (cont.)

- Symbol `%n` embeds an end-of-line character in a format string.
- Symbol `%%` produces literal '%' character.
- When compiler sees a format specifier, it attempts to match that specifier to an expression following the string.
  - Must match in type and position

7.4 Formatted Output with `printf` and `format` (cont.)

- `printf` can justify text and produce tabular output.
  - **Format flags** support justification and other styles.

Table 7-2: Some commonly used format flags

FLAG	WHAT IT DOES	EXAMPLE VALUE
-	Left justification	34
,	Show decimal separators	20,345,000
0	Show leading zeroes	002.67
^	Convert letters to uppercase	1.56E+3

The Formatter Class

- The full specification for the Formatter Class can be found at the following link:

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html#syntax

- Formatting conversions apply to the following types:
  - General,
  - Character,
  - Numeric,
  - Date/Time,
  - Percent, and
  - Line Separator

7.4 Formatted Output with `printf` and `format` (cont.)

Figure 7-3: Table of sales figures shown with and without formatting

```
NAME SALES COMMISSION
Catherine 23415 2341.5
Ken 321.5 32.15
Martin 4384.75 438.48
Tess 3595.74 359.57
```

Version 1: Unreadable without formatting

NAME	SALES	COMMISSION
Catherine	23415.00	2341.50
Ken	321.50	32.15
Martin	4384.75	438.48
Tess	3595.74	359.57

Version 2: Readable with formatting

7.4 Formatted Output with `printf` and `format` (cont.)

- To output data in formatted columns:
  - Establish the width of each field.
  - Choose appropriate format flags and format specifiers to use with `printf`.
- Width of a field that contains a `double` appears before the decimal point in the format specifier `f`

7.4 Formatted Output with `printf` and `format` (cont.)

Table 7-3: Some example format strings and their outputs

VALUES	FORMAT STRING	OUTPUT
34, 56.7	"%d%7.2f"	34 56.70
34, 56.7	"%4d%7.2f"	34 56.70
34, 56.7	"%-4d$%7.2f"	34 $ 56.70
34, 56.7	"%-4d$%.2f"	34 $56.70

7.4 Formatted Output with `printf` and `format` (cont.)

Example 7.3: Display a table of names and salaries

DisplayTable.java DisplayTable.txt

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

public class DisplayTable{

   public static void main(String[] args) throws IOException{
      Scanner names = new Scanner(new File("names.txt"));
      Scanner salaries = new Scanner(new File("salaries.txt"));
      System.out.printf("%-16s%12s%n", "NAME", "SALARY");
      while (names.hasNext()){
         String name = names.nextLine();
         double salary = salaries.nextDouble();
         System.out.printf("%-16s%,12.2f%n", name, salary);
      }
   }
}
```

7.4 Formatted Output with `printf` and `format` (cont.)

- Formatting with the method (P.256)
- `String.format`:
  - Can be used to build a formatted string
  - Use the same syntax as `printf`
  - Returns a formatted string
  - The difference is that resulting string is not displayed on the console window, but stored in a string variable
  - str = String.format("The price is: $%.2f", price);

7.5 Handling Number Format Exceptions During Input

- If data are found to be invalid after input, the program can display an error message
  - and prompt for the data again
- The program should detect and handle when a number is requested from the user, but the user enters a non-numerical value
  - The Scanner methods nextInt() and nextDouble() will do this, but will crash the program with an error message

7.5 Handling Number Format Exceptions During Input (cont.)

- The `try-catch` construct allows **exceptions** to be caught and handled appropriately.
- Statements within `try` clause executed until an exception is thrown
  - Exceptions sent **immediately** to `catch` clause
    - Skipping remainder of code in `try` clause

```
try{
   <statements that might throw exceptions>
}catch(Exception e){
   <code to recover from an exception if it's thrown>
}
```

7.5 Handling Number Format Exceptions During Input (cont.)

- If no statement throws an exception within the `try` clause, the `catch` clause is skipped.
- Many types of exceptions can be thrown.
  - Catching an `Exception` object will catch them all.

ConvertWithQueryException.java

ConvertWithQueryException.txt

7.5 Handling Number Format Exceptions During Input (cont.)

- Here is the "try catch" statement rewritten without the use of a **break** statement in the while loop.

ConvertWithQueryExceptionNoBreak.java

ConvertWithQueryExceptionNoBreak.txt

Summary

- Terminal-based program: Program controls most of the interaction with the user
- The terminal input/output (I/O) interface can be extended to handle repeated sets of inputs.
  - Query-based pattern
  - Menu-driven pattern