Chapter 7
Improving the User Interface

Fundamentals of Java

Objectives

- 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.

Vocabulary

- Menu-driven program
- Query-controlled input

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;
   }
}
```

Repeating Sets of Inputs

- Chapter 4 presented count-controlled and sentinel-controlled input.
- **Query-controlled input:** Before each set of inputs after the first, the program asks the user if there are more inputs.

Repeating Sets of Inputs (cont.)

```
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)?
```

Figure 7-1: Interface for a query-controlled temperature conversion program

Repeating Sets of Inputs (cont.)

- Program implemented by two classes:
  - Interface class
  - `Thermometer` class
- Pseudocode for interface 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
}
```

Repeating Sets of Inputs (cont.)

```
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 newline
         reader.nextLine();
         System.out.println("The equivalent in Celsius is " +
                            thermo.getCelsius());
         System.out.print("\nDo it again (y/n)? ");
         doItAgain = reader.nextLine();
      }
   }
}
```

Example 7.1: ConvertWithQuery.java

Menu-Driven Conversion Program

- **Menu-driven programs:** Display list of options; user selects one
  - Program prompts for additional inputs related to that option and performs computations.
  - Menu is displayed again.

Menu-Driven Conversion Program (cont.)

```
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
```

Figure 7-2: Interface for a menu-driven version of the temperature conversion program

Menu-Driven Conversion Program (cont.)

- Pseudocode:

```
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 "Goodbye!"
 else
    print "Invalid option"
}
```

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>)
```

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

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

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

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	

Table 7-1: Commonly used format types

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

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

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

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

Table 7-2: Some commonly used format flags

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

```
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

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

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`

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

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

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

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

```
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);
      }
   }
}
```

Example 7.3: Display a table of names and salaries

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

- Formatting with `String.format`:
  - Can be used to build a formatted string
  - Same syntax as `printf`
  - Difference is that resulting string is not displayed on the console

Handling Number Format Exceptions During Input

- If data found to be invalid after input, the program can display an error message
  - Prompts for the data again
- Must detect and handle when a number is requested from the user, but the user enters a non-numerical value
  - e.g., during a `Scanner.parseInt` statement

Handling Number Format Exceptions During Input (cont.)

- The `try-catch` construct allows **exceptions** to be caught and handled appropriately.

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

- Statements within `try` clause executed until an exception is thrown
  - Exceptions sent **immediately** to `catch` clause
    - Skipping remainder of code in `try` clause

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.

Graphics and GUIs

- GUIs based on pop-up dialogs can be limiting.
  - Static and rigid
- Better GUI presents the user with entry fields for many of the data values simultaneously
  - Offer many command options
    - Via buttons, drop-down lists, and editable fields

The Model/View/Controller Pattern

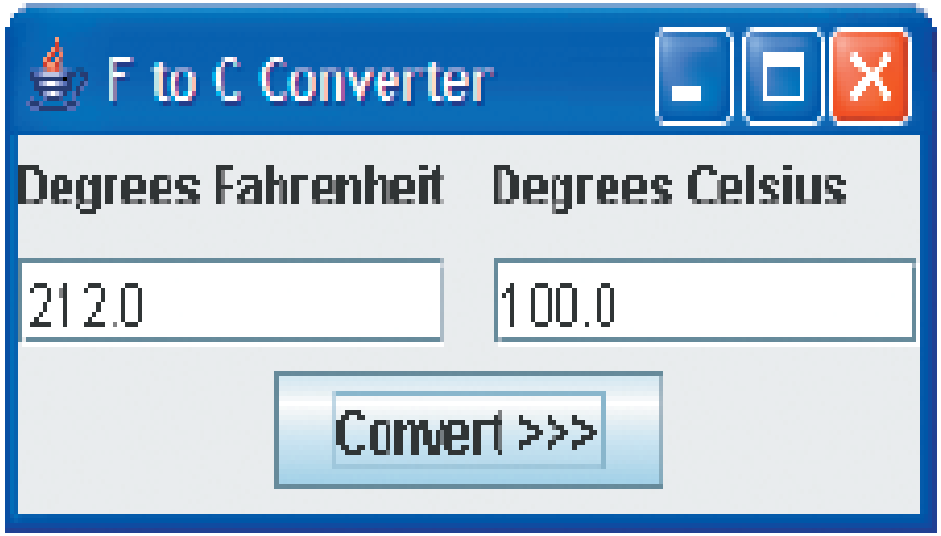


Figure 7-4: Interface for the GUI-based temperature conversion program

The Model/View/Controller Pattern (cont.)

- Create classes to represent:
  - **Model:** The data that the program uses and the operations that can be performed on that data
  - **View:** What the user of the program interacts with
  - **Controller:** Coordinates model and view classes by passing messages and data between them
    - **Listener** classes
    - Can be attached to **widgets** in the view class

The Model/View/Controller Pattern (cont.)

- When a controller class receives an event from a view class, it sends a message to a model class.
- Use a separate class to set up model, view, and controller classes from within a `main` method.
  - The **application**

The Model/View/Controller Pattern (cont.)

- Temperature conversion application class:

```
/* ConvertWithGUI.java
Application class for a GUI-based temperature conversion
program that coverts from Fahrenheit to Celsius.
*/

import javax.swing.*;

public class ConvertWithGUI{

   // Execution begins in the method main as usual.
   public static void main(String[] args){
      GUIWindow theGUI = new GUIWindow();
      theGUI.setTitle("F to C Converter");
      theGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      theGUI.pack();
      theGUI.setVisible(true); //Make the window visible
   }
}
```

The Model/View/Controller Pattern (cont.)

- `GUIWindow` is the main view class
  - Instantiates and maintains reference to the data model
    - A `Thermometer`
  - Instantiates and maintains references to data fields and the command button
  - Adds widgets to window's container
  - Instantiates and attaches a `FarhenheitListener` to the command button

The Model/View/Controller Pattern (cont.)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GUIWindow extends JFrame{

   // >>>>>>> The model <<<<<<<

   // Declare and instantiate the thermometer
   private Thermometer thermo = new Thermometer();

   // >>>>>>> The view <<<<<<<

   // Declare and instantiate the widgets.
   private JLabel fahrLabel        = new JLabel("Degrees Fahrenheit");
   private JLabel celsiusLabel     = new JLabel("Degrees Celsius");
   private JTextField fahrField    = new JTextField("32.0");
   private JTextField celsiusField = new JTextField("0.0");
   private JButton fahrButton      = new JButton("Convert >>>");
```

Example 7.5: GUIWindow.java

The Model/View/Controller Pattern (cont.)

```
// Constructor
public GUIWindow(){
   // Set up panels to organize widgets and
   // add them to the window
   JPanel dataPanel = new JPanel(new GridLayout(2, 2, 12, 6));
   dataPanel.add(fahrLabel);
   dataPanel.add(celsiusLabel);
   dataPanel.add(fahrField);
   dataPanel.add(celsiusField);
   JPanel buttonPanel = new JPanel();
   buttonPanel.add(fahrButton);
   Container container = getContentPane();
   container.add(dataPanel, BorderLayout.CENTER);
   container.add(buttonPanel, BorderLayout.SOUTH);
   // Attach a listener to the convert button
   fahrButton.addActionListener(new FahrenheitListener());
}
```

Example 7.5: GUIWindow.java (cont.)

The Model/View/Controller Pattern (cont.)

```
    // >>>>>>> The controller <<<<<<<<

    private class FahrenheitListener implements ActionListener{
       public void actionPerformed(ActionEvent e){
          String input = fahrField.getText();        // Obtain input
          double fahr = Double.parseDouble(input);   // Convert to a double
          thermo.setFahrenheit(fahr);                // Reset thermometer
          double celsius = thermo.getCelsius();      // Obtain Celsius
          celsiusField.setText("" + celsius);        // Output result
       }
    }
}
```

Example 7.5: GUIWindow.java (cont.)

The Model/View/Controller Pattern (cont.)

JTEXTFIELD METHOD	WHAT IT DOES
`String getText()`	Returns the string currently occupying the field
`void setEditable(boolean b)`	The field is read-only by the user if `b` is `false`. The user can edit the field if `b` is `true`. The default is `true`.
`void setText(String s)`	Displays the string `s` in the field

Table 7-4: Some `JTextField` methods

The Model/View/Controller Pattern (cont.)

- Deciding the layout of the view class(es) is very important.
  - Often necessary to break view into smaller pieces (panels)
    - Each formatted separately
    - Add formatted panels to other panels to build the layout.

The Model/View/Controller Pattern (cont.)

- Real GUI programs are **event-driven**.
  - When program opens, it waits for events
- Events are generated when the user interacts with the GUI.
  - Events invoke controller classes, which in turn invoke model classes.

Making Temperature Conversion Go Both Ways

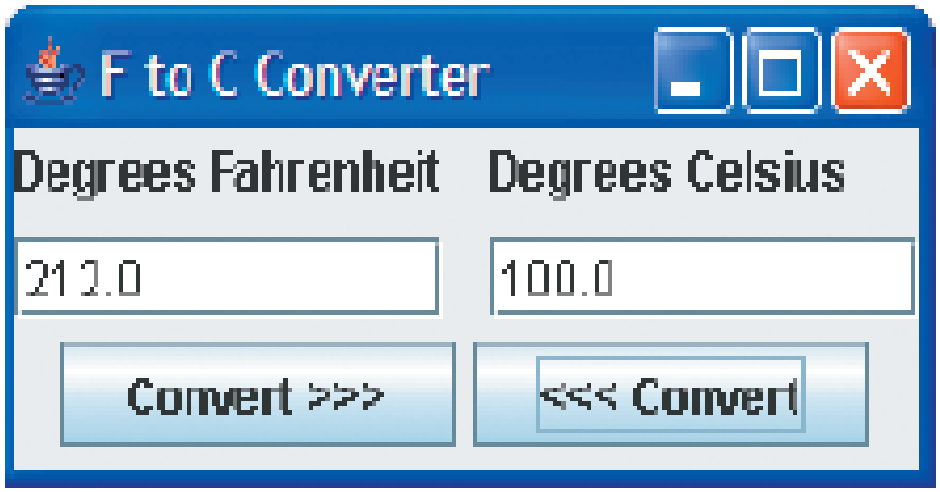


Figure 7-5: Temperature converter that goes both ways

Making Temperature Conversion Go Both Ways (cont.)

- Alterations:
  - Declares and instantiates second button
  - Adds button to button panel
  - Creates listener object and attaches it to button
  - Defines a separate listener class that converts from Celsius to Fahrenheit

Making Temperature Conversion Go Both Ways (cont.)

```
private class CelsiusListener implements ActionListener{
   public void actionPerformed(ActionEvent e){
      String input = celsiusField.getText();          // Obtain input
      double celsius = Double.parseDouble(input);     // Convert to a double
      thermo.setCelsius(celsius);                     // Reset thermometer
      double fahr = thermo.getFahrenheit();           // Obtain Fahrenheit
      fahrField.setText("" + fahr);                   // Output result
   }
}
```

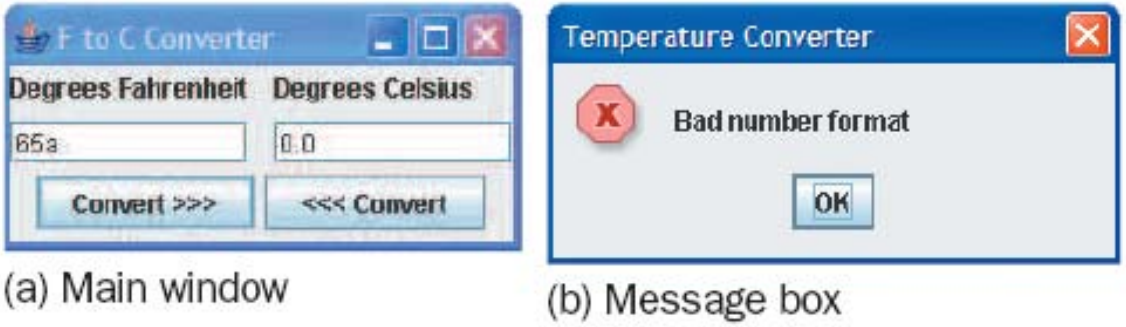
Example 7.6: Listener to convert Celsius to Fahrenheit

Making the Temperature Conversion Robust

```
private class FahrenheitListener implements ActionListener{
   public void actionPerformed(ActionEvent e){
      try{
         String input = fahrField.getText();         // Obtain input
         double fahr = Double.parseDouble(input);    // Convert to a double
         thermo.setFahrenheit(fahr);                 // Reset thermometer
         double celsius = thermo.getCelsius();       // Obtain Celsius
         celsiusField.setText("" + celsius);         // Output result
      }catch(Exception ex){
         JOptionPane.showMessageDialog(GUIWindow.this,
                                       "Bad number format",
                                       "Temperature Converter",
                                       JOptionPane.ERROR_MESSAGE);
      }
   }
}
```

Example 7.7: Robust listener for number format errors

Making the Temperature Conversion Robust (cont.)



(a) Main window (b) Message box

Figure 7-6: Responding to a number format error

Summary

- The terminal input/output (I/O) interface can be extended to handle repeated sets of inputs.
  - Query-based pattern
  - Menu-driven pattern
- The graphical user interface (GUI) allows user to interact with a program by displaying window objects and handling mouse events.

Summary (cont.)

- Terminal-based program: Program controls most of the interaction with the user
- GUI-based program: Driven by user events
- Two primary tasks of a GUI-based program:
  - Arrange window objects
  - Handle user interactions