What we will cover:

*switch* Statements
*Ternary Operator*
*GUI confirmation dialogs*
*String and Output Formatting*

**switch** Statements

The *switch* statement is useful for handling conditions that have multiple results and would result in a complex nested *if* statement. If you have three or more cases to handle, a switch statement might be a better choice than nested *if*. The general format of a switch statement is

```java
switch (expression) {
    case value1: statement(s)1;
    break;
    case value2: statement(s)2;
    break;
    ...
    case valuen: statement(s)n;
    break;
    default:  statement(s);
}
```

The expression must evaluate to a value of type char, byte, short, or int. The resulting value of the expression is matched against each case's value until there is a match. Whenever a case is matched, the associated statements are executed. The break statement forces the program to exit the switch (goes to instruction after the ending brace). If you forget to put the break after the case, the program keeps on executing down the list of cases. If no matches are found with the cases the default statement is executed. A common use of the switch statement is to selectively execute instructions based on a program menu choice, consider for example, writing a program to display a menu and then perform an operation based on the user choice from the menu.

```java
import javax.swing.JOptionPane;

public class Menu {
    public static void main(String[] args) {
        String temp, menuText, message;
        int menuChoice;

        // display menu and get user choice
        menuText = "Enter your choice: \n" + "1 - Choice 1 \n" + "2 - Choice 2 \n" + "3 - Choice 3 \n";
        temp = JOptionPane.showInputDialog(null, menuText);
        menuChoice = Integer.parseInt(temp);

        message = "You chose ";

        // perform operation based on menu choice
        switch (menuChoice) {
            case 1:
                message += "option 1";
                break;
            case 2:
                message += "option 2";
                break;
            case 3:
```


message += "option 3";
break;
default:
    message = "an invalid option";
}

// display result
JOptionPane.showMessageDialog(null, message);
System.exit(0);

Remove some of the breaks and observe the behavior of the program. Remember that the breaks tell the switch statement to jump to the end. Without these, control just “falls through” to the other cases.

Ternary Operator
There is only one ternary operator in Java. A ternary operator works on three operands. This ternary operator works as a shortcut to a simple if-else statement. The general format is:
<boolean-expression> ? expression1 : expression2;

The result of this expression is expression1 if the boolean-expression is true or expression2 if the Boolean-expression is false. A couple of examples follow:
y = (x > 0) ? 1 : -1;
max = (num1 > num2) ? num1: num2;

GUI Confirmation Dialogs
With conditional statements, we can now direct the flow of code by asking for confirmation from the user. This type of dialog box offers the user a Yes, No, and a Cancel option. Confirmation dialog boxes return an integer that correlates with one of three constants: JOptionPane.YES_OPTION, JOptionPane.NO_OPTION, and JOptionPane.CANCEL_OPTION.

import javax.swing.JOptionPane;

public class ConfirmDialog {  
    public static void main(String[] args) {  
        String message;
        int option = 0;

        option = JOptionPane.showConfirmDialog(null, "Choose an option: ");
        message = "You chose ";

        switch (option) {
            case JOptionPane.YES_OPTION:
                message += "Yes";
                break;
            case JOptionPane.NO_OPTION:
                message += "No";
                break;
            case JOptionPane.CANCEL_OPTION:
                message += "Cancel";
                break;
            default:
                message += "none of the options";
        }

        message += ".";
        JOptionPane.showMessageDialog(null, message);
        System.exit(0);
    }
}
In this particular example, the default case is activated when the 'X' button is used instead of picking an option.

**String and Output Formatting**
The command line output from `System.out.print` (and `println`) and strings can be formatted as a single string with format specifiers as opposed to simply appending strings together. Also see §3.17 in your book for the use of `System.out.printf(...)` which is very similar to `String.format(...)`.  

The String class has a format method that takes, as arguments, a control string with one or more format specifiers and a comma-separated item list that corresponds to each specifier. The item data is inserted, in order, into the string at the location of the format specifiers. The items are formatted depending on which specifier they are replacing. Format specifiers are indicated by a % sign and contain formatting information.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output</th>
<th>Item example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>Boolean value</td>
<td>true</td>
</tr>
<tr>
<td>%c</td>
<td>character</td>
<td>‘a’</td>
</tr>
<tr>
<td>%d</td>
<td>decimal integer</td>
<td>13</td>
</tr>
<tr>
<td>%f</td>
<td>real number (floating-point)</td>
<td>37.36</td>
</tr>
<tr>
<td>%e</td>
<td>number in scientific notation</td>
<td>3.736e+01</td>
</tr>
<tr>
<td>%s</td>
<td>string</td>
<td>“Bob”</td>
</tr>
</tbody>
</table>

For example:
```java
String temp = String.format("The answer to question %d is %f", 2, -5.8 );
System.out.println(temp);
```
creates and then prints a string with an integer and real number inserted into it:

The answer to question 2 is -5.800000

The width and precision of data can be indicated in the format specifier by adding that precede the specifier. These numbers will indicate the requested width and precision.

For example:
```java
String temp = String.format("The answer to question %4d is %6.2f",2, -5.8 );
System.out.println(temp);
```
creates and then prints a string with an integer and real number inserted into it. The integer has a minimum width of 4 (if there are less than 4 digits, spaces will be inserted) and the real number has a minimum width of 6 with two decimal places. The width is automatically increased if the number requires more space than specified; trailing zeros are appended after the decimal point to give the appropriate number of places.

The answer to question 2 is -5.80

Similar width formatting can be used with characters and strings. The main advantage to using a method like this is that you do not have to concatenate strings together to effectively print variable values.