A block statement can be used anywhere a single statement is called for in Java syntax. For example, the if portion of an if-else statement could be a block, or the else portion could be a block (as we saw in the Guessing program), or both parts could be block statements. For example:

```java
if (boxes != warehouse.getCount())
{
    System.out.println("Inventory and warehouse do NOT match.");
    System.out.println("Beginning inventory process again!");
    boxes = 0;
}
else
{
    System.out.println("Inventory and warehouse MATCH.");
    warehouse.ship();
}
```

In this if-else statement, the value of boxes is compared to a value obtained by calling the getCount method of the warehouse object (whatever that is). If they do not match exactly, two println statements and an assignment statement are executed. If they do match, a different message is printed and the ship method of warehouse is invoked.

**Nested if Statements**

The statement executed as the result of an if statement could be another if statement. This situation is called a nested if. It allows us to make another decision after determining the results of a previous decision. The program in Listing 5.6, called MinOfThree, uses nested if statements to determine the smallest of three integer values entered by the user.

Carefully trace the logic of the MinOfThree program, using various input sets with the minimum value in all three positions, to see how it determines the lowest value.

An important situation arises with nested if statements. It may seem that an else clause after a nested if could apply to either if statement. For example:

```java
if (code == 'R')
    if (height <= 20)
        System.out.println("Situation Normal");
    else
        System.out.println("Bravo!");
```
import java.util.Scanner;

public class MinOfThree {
    public static void main(String[] args) {
        int num1, num2, num3, min = 0;

        Scanner scan = new Scanner(System.in);

        System.out.println("Enter three integers: ");
        num1 = scan.nextInt();
        num2 = scan.nextInt();
        num3 = scan.nextInt();

        if (num1 < num2) {
            if (num1 < num3)
                min = num1;
            else
                min = num3;
        } else {
            if (num2 < num3)
                min = num2;
            else
                min = num3;
        }

        System.out.println("Minimum value: "+ min);
    }
}
Is the else clause matched to the inner if statement or the outer if statement? The indentation in this example implies that it is part of the inner if statement, and that is correct. An else clause is always matched to the closest unmatched if that preceded it. However, if we’re not careful, we can easily mismatch it in our mind and misalign the indentation. This is another reason why accurate, consistent indentation is crucial.

Braces can be used to specify the if statement to which an else clause belongs. For example, if the previous example should have been structured so that the string "Bravo!" is printed if code is not equal to 'R', we could force that relationship (and properly indent) as follows:

```java
if (code == 'R')
{
    if (height <= 20)
        System.out.println("Situation Normal");
} else
    System.out.println("Bravo!");
```

By using the block statement in the first if statement, we establish that the else clause belongs to it.

**Self-Review Questions** *(see answers in Appendix N)*

**SR 5.8** What output is produced by the following code fragment given the assumptions below?

```java
if (num1 < num2)
    System.out.print(" red ");
if ((num1 + 5) < num2)
    System.out.print(" white ");
else
    System.out.print(" blue ");
System.out.println(" yellow ");
```

a. Assuming the value of num1 is 2 and the value of num2 is 10?
b. Assuming the value of num1 is 10 and the value of num2 is 2?
c. Assuming the value of num1 is 2 and the value of num2 is 2?

**SR 5.9** How do block statements help us in the construction of conditionals?

**SR 5.10** What is a nested if statement?

**SR 5.11** For each assumption, what output is produced by the following code fragment?
```java
if (num1 >= num2)
{
    System.out.print(" red ");
    System.out.print(" orange ");
}
if ((num1 + 5) >= num2)
    System.out.print(" white ");
else
    if ((num1 + 10) >= num2)
    {
        System.out.print(" black ");
        System.out.print(" blue ");
    }
    else
        System.out.print(" yellow ");
System.out.println(" green ");
```

a. Assuming the value of `num1` is 5 and the value of `num2` is 4?
b. Assuming the value of `num1` is 5 and the value of `num2` is 12?
c. Assuming the value of `num1` is 5 and the value of `num2` is 27?

**SR 5.12** Write an expression that will print a message based on the value of the int variable named `temperature`. If `temperature` is equal to or less than 50, it prints "It is cool." on one line and "Dress warmly." on the next. If `temperature` is greater than 80, it prints "It is warm." on one line and "Dress coolly." on the next. If `temperature` is in between 50 and 80, it prints "It is pleasant." on one line and "Dress pleasantly." on the next.

### 5.3 Comparing Data

When comparing data using boolean expressions, it’s important to understand some nuances that arise depending on the type of data being examined. Let’s look at a few key situations.

**Comparing Floats**

An interesting situation occurs when comparing floating point data. Two floating point values are equal, according to the `==` operator, only if all the binary digits of their underlying representations match. If the compared values are the results of
5.3 Comparing Data

computation, it may be unlikely that they are exactly equal even if they are close enough for the specific situation. Therefore, you should rarely use the equality operator (==) when comparing floating point values.

A better way to check for floating point equality is to compute the absolute value of the difference between the two values and compare the result to some tolerance level. For example, we may choose a tolerance level of 0.00001. If the two floating point values are so close that their difference is less than the tolerance, then we are willing to consider them equal. Comparing two floating point values, f1 and f2, could be accomplished as follows:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal.");
```

The value of the constant TOLERANCE should be appropriate for the situation.

Comparing Characters

We know what it means when we say that one number is less than another, but what does it mean to say one character is less than another? As we discussed in Chapter 2, characters in Java are based on the Unicode character set, which defines an ordering of all possible characters that can be used. Because the character 'a' comes before the character 'b' in the character set, we can say that 'a' is less than 'b'.

We can use the equality and relational operators on character data. For example, if two character variables ch1 and ch2 hold two characters, we might determine their relative ordering in the Unicode character set with an if statement as follows:

```java
if (ch1 > ch2)
    System.out.println(ch1 + " is greater than " + ch2);
else
    System.out.println(ch1 + " is NOT greater than " + ch2);
```

The Unicode character set is structured so that all lowercase alphabetic characters ('a' through 'z') are contiguous and in alphabetical order. The same is true of uppercase alphabetic characters ('A' through 'Z') and characters that represent digits ('0' through '9'). The digits precede the uppercase alphabetic characters, which precede the lowercase alphabetic characters. Before, after, and in between these groups are other characters. See the chart in Appendix C for details.

Remember that a character and a character string are two different types of information. A char is a primitive value that represents one character. A character string is represented as an object in Java, defined by the String class. While comparing strings is based on comparing the characters in the strings, the comparison is governed by the rules for comparing objects.

KEY CONCEPT

The relative order of characters in Java is defined by the Unicode character set.