3.2 The Type **`boolean`**

- Boolean Expressions and Variables
- Truth Tables and Precedence Rules
- Input and Output of Boolean Values

The Type **`boolean`**

- A [redacted] type
- Can have expressions, values, constants, and variables just as with any other primitive type
- Only two values: **`true`** and **`false`**

The Type **boolean**

- Can use a `boolean` variable as the condition in an if statement
- Using a boolean variable as the condition can make an if statement easier to read by avoiding a complicated expression.
- Boolean variables can make programs more readable.

```
if (                    )
```

instead of

```
if((temperature <= 100) && (thrust >= 12000) &&
   (cabinPressure > 30) && …)
```

```
if (systemsAreOK)
  System.out.println("Initiate launch sequence.");
else
  System.out.println("Abort launching sequence");
```

`boolean` Variables in [redacted]

- A `boolean` expression evaluates to one of the two values true or false.
- The value of a `boolean` expression can be assigned to a `boolean` variable:

```
int number = -5;
boolean isPositive;
isPositive = (number > 0);
if (isPositive)
  System.out.println("positive");
else
  System.out.println("negative or zero");
```

Parentheses are not [redacted] here.

Parentheses are necessary here.

- There are simpler and easier ways to write this small program, but `boolean` variables are useful in keeping track of conditions that depend on a number of factors.

Naming Boolean Variables

- Choose names such as `isPositive` or `systemsAreOk`.
- Avoid names such as `numberSign` or `systemStatus`.

Truth Tables for `boolean` Operators

&& (and)

Value of A	Value of B	A && B
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

|| (or)

Value of A	Value of B	A || B
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

! (not)

Value of A	!A
true	**false**
false	**true**

Precedence

An example of using precedence rules to see which operators in following expression should be done first:

```
score < min/2 - 10 || score > 90
```

- [redacted] operator has highest precedence of all operators used here so treat it as if it were parenthesized:

```
score < (min/2) - 10 || score > 90
```

- [redacted] operator has next highest precedence :

```
score < ((min/2) - 10) || score > 90
```

- The < and > operators have equal precedence and are done in left-to-right order :

```
(score < ((min/2) - 10)) || (score > 90)
```

- The last expression is a fully parenthesized expression that is equivalent to the original. It shows the order in which the operators in the original will be evaluated.

Precedence Rules

Highest Precedence

- First: the [redacted] operators: `+`, `-`, `++`, `--`, and `!`
- Second: the binary [redacted] operators: `*`, `/`, `%`
- Third: the binary arithmetic operators: `+`, `-`
- Fourth: the boolean operators: `<`, `>`, `=<`, `>=`
- Fifth: the boolean operators: `==`, `!=`
- Sixth: the boolean operator `&`
- Seventh: the boolean operator `|`
- Eighth: the boolean operator `&&`
- Ninth: the boolean operator `||`

Lowest Precedence

Precedence Rules, cont

- In what order are the operations performed?

```
score < min/2 - 10 || score > 90
score < (min/2) - 10 || score > 90
score < ((min/2) - 10) || score > 90
(score < ((min/2) - 10)) || score > 90
(score < ((min/2) - 10)) || (score >
  90)
```

Evaluation

- *evaluation*—only evaluating as much of a boolean expression as necessary.
- Example:

```
if ((assign > 0) && ((total/assign) > 60))
   System.out.println("Good work");
else
   System.out.println("Work harder.");
```

- If `assign > 0` is false, then the complete expression cannot be true because AND is only true if both operands are true.
- Java will not evaluate the second part of the expression.
- Short-circuit evaluation prevents a divide-by-zero exception when `assign` is 0.

[redacted] evaluation

- Both subexpressions are always evaluated.
- & rather than &&
- | rather than ||

Input and output of Boolean Values

```
boolean booleanVar = false;
System.out.println(booleanVar);
System.out.println("Enter a boolean value: ");
booleanVar = Scanner.
System.out.println("You entered " + booleanVar);
```

```
Enter a boolean value:
true
You entered true
```

```
import java.util.* ;
public class NextBoolean
{
   public static void main(String[] args)
   {
         Scanner scannerobject = new Scanner(System.in);
         boolean booleanVar = false;
         System.out.println(booleanVar);
         System.out.println("Enter a boolean value: ");
         booleanVar = scannerobject.n
         System.out.println("You entered" + booleanVar);
   }
}
```

```
false
Enter a boolean value:
true
You entered true
```

C:₩WINDOWS₩system32₩cmd.exe

```
false
Enter a boolean value:
f
```

nextBoolean

```
public boolean nextBoolean()
```

Scans the next token of the input into a boolean value and returns that value. This method will throw `InputMismatchException` if the next token cannot be translated into a valid boolean value. If the match is successful, the scanner advances past the input that matched.

Returns:

the boolean scanned from the input

Throws:

- `InputMismatchException` - if the next token is not a valid boolean
- `NoSuchElementException` - if input is exhausted
- `IllegalStateException` - if this scanner is closed

Using a Boolean Variable to End a Loop

- example

```
boolean numbersLeftToRead = true
while (numbersLeftToRead)
{
   next = keyboard.nextInt()
   if (next < 0)
         numbersLeftToRead = false;
   else
         Process_Next_Number
}
```

```
//  Use of a Boolean Variable to End a Loop
/**
 Illustrates the use of a boolean variable to control loop ending.
*/
import java.util.Scanner;

public class BooleanDemo
{
   public static void main(String[] args)
   {
      System.out.println(
                 "Enter nonnegative numbers, one per line.");
      System.out.println("Place a negative number at the end");
      System.out.println("to serve as an end marker.");

      int next, sum = 0;
      boolean numbersLeft = true;
      Scanner keyboard = new Scanner(System.in);
      while (numbersLeft)
      {
         next = keyboard.nextInt( );
         if (next < 0)
            numbersLeft = false;
         else
            sum = sum + next;
     }

      System.out.println("The sum of the numbers is " + sum);
   }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter nonnegative numbers, one per line.
Place a negative number at the end
to serve as an end marker.
1
2
3
-1
The sum of the numbers is 6
계속하려면 아무 키나 누르십시오 . . .
```

끝