Chapter 2

Values, Variables, and Types

In this chapter we explore the primitive building blocks that are used to develop Java programs. We experiment with the following concepts within DrJava’s interactive environment:

- numeric and nonnumeric values
- variables
- expressions

Next chapter we use these primitive concepts to build more sophisticated, computation-rich objects.

2.1 Java Values in DrJava’s Interaction Pane

The DrJava environment provides an Interactions pane that is convenient for experimenting with Java’s primitive types. When you select the Interactions tab in the lower window panel, you see

```
Welcome to DrJava. Working directory is /Users/rick/java
>
```

The greater than (>) prompt indicates the Interactions interpreter is ready to receive input from the user. If you type the number four

```
Welcome to DrJava. Working directory is /Users/rick/java
> 4
4
>
```

the value four is displayed, and the prompt reappears waiting for additional input. In these simulated Interactions panes the user’s input appears to the right of the prompt (>), and the response of the interpreter appears on a line without a prompt; this is exactly how it works in DrJava’s Interactions pane. Also, when the message
Welcome to DrJava. Working directory is /Users/rick/java

appears in the Interactions pane it means that this is a fresh session. In a fresh session any prior user interactions are forgotten. A fresh session is instituted in one of several ways:

- A new interactive session is started when DrJava begins running.
- When the user issues the "Reset interactions" command from the DrJava menu the session is reinitialized.
- Compiling a Java source file within the Editor pane resets the Interactions pane.

The number four is an integer value. Java supports a variety of numeric value types and several kinds of nonnumeric values as well. Like mathematical integers, Java integers represent whole numbers, both positive and negative:

```
> -93
-93
```

Unlike mathematical integers, however, Java’s integers have a limited range. Consider the following interactive session:

```
> 2147483646
2147483646
> 2147483647
2147483647
> 2147483648
NumberFormatException: For input string: "2147483648"
at java.lang.NumberFormatException.forInputString(NumberFormat...
at java.lang.Integer.parseInt(Integer.java:463)
at java.lang.Integer.valueOf(Integer.java:553)
```

2,147,483,647 is the largest integer that can be represented by Java’s “normal” integer type. Java’s standard integer type is called int. This limited range is common among programming languages since each number is stored in a fixed amount of memory. Larger numbers require more storage in memory. In Java (as in many other languages), ints require four bytes (32 bits) of memory. 2,147,483,647 is the largest integer that can be stored in four bytes. In order to model the infinite set of mathematical integers an infinite amount of memory would be needed! As we will see later, Java supports an integer type with a greater range and also provides support for arbitrary-precision integers.

Now try to enter numbers with decimal places:

```
> 3.14159
3.14159
> -0.0045
-0.0045
```
These are not integers but are more like mathematical real numbers. The key here is “like.” The name of this decimal type is double, which stands for double-precision floating point number. The term floating point means that the decimal point can “float” so the number of digits behind the decimal point can vary. As the double-precision label implies, Java also supports single-precision floating point numbers.

Just like with ints, doubles use a fixed amount of memory, in this case eight bytes per value. This means that both the range and precision of doubles is limited. The largest double is

\[1.7976931348623157 \times 10^{308}\]

the smallest positive double is

\[4.9 \times 10^{-324}\]

doubles can be both positive and negative, and maintain a minimum of 15 digits of precision. Java’s doubles are therefore only an approximation of mathematical real numbers. An irrational number like π cannot be represented exactly since π has an infinite number of digits. While integers can be represented exactly within the range of values, because of finite precision not all floating point values can be represented. Consider:

```
> 1.227835364547718468456
1.2278353645477185
```

The double type cannot store 22 decimal digits of precision, so the desired value is rounded to just 17 digits. Programmers must take care when performing complex calculations involving doubles to ensure that cumulative rounding errors do not lead to meaningless results.

Scientific notation can be used:

```
> 6.023e23
6.023E23
```

The number to the left of the e (capital E can be used as well) is the mantissa, and the number to the right of the e is the exponent of 10. 6.023e23 thus stands for 6.023 \times 10^{23}.

One type of nonnumeric value is the boolean type. It has only two values: true and false. These values must be typed in exactly, as Java is case sensitive (capitalization matters):

```
> true
true
> True
Error: Undefined class 'True'
> TRUE
Error: Undefined class 'TRUE'
> false
false
> False
Error: Undefined class 'False'
> FALSE
Error: Undefined class 'FALSE'
```
The word Boolean comes from George Boole, a mathematician that founded the algebra of mathematical logic. At first glance, the boolean type may appear rather limited and useless, but as we will see it is essential for building powerful programs.

Java supports other primitive data types. A comprehensive list can be found in Table 2.1, but we will have little need for them until later. For the time being we will restrict our attention to ints, doubles, and booleans.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Range</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>byte</td>
<td>(-128 \ldots +127)</td>
<td>8 bits</td>
</tr>
<tr>
<td>short</td>
<td>short integer</td>
<td>(-32,768 \ldots +32,767)</td>
<td>16 bits</td>
</tr>
<tr>
<td>char</td>
<td>Unicode character</td>
<td>(0 \ldots +65,536)</td>
<td>16 bits</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>(-2,147,483,648 \ldots +2,147,483,647)</td>
<td>32 bits</td>
</tr>
<tr>
<td>long</td>
<td>long integer</td>
<td>(-9,223,372,036,854,775,808 \ldots +9,223,372,036,854,775,807)</td>
<td>64 bits</td>
</tr>
<tr>
<td>float</td>
<td>single-precision floating point</td>
<td>(\pm 3.4 \times 10^{-38} \ldots \pm 1.4 \times 10^{-45}) with at least 7 decimal digits of precision</td>
<td>32 bits</td>
</tr>
<tr>
<td>double</td>
<td>double-precision floating point</td>
<td>(\pm 1.7 \times 10^{-308} \ldots \pm 4.9 \times 10^{-324}) with at least 15 decimal digits of precision</td>
<td>64 bits</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
<td>false or true</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

Table 2.1: Primitive data types

One non-primitive type that is worth noting at this point is String. A string is a sequence of characters. In Java, string literals are enclosed within quotation marks:

Welcome to DrJava. Working directory is /Users/rick/java
> "This is a string"
"This is a string"

A string is an example of a Java object, and so it is not a primitive type. This means that strings have capabilities that exceed those of primitive types. Some of these capabilities will be explored later.

### 2.2 Variables and Assignment

In algebra, variables are used to represent numbers. The same is true in Java, except Java variables also can represent values other than numbers.

Welcome to DrJava. Working directory is /Users/rick/java
> x = 5
5
> x
5

In an actual Java program the first two lines of the above interaction would be terminated with semicolons and be called *statements*. The semicolons can be used in the Interactions pane as well.
Notice how the terminating semicolon suppresses the evaluation of the entered expression; consider:

The statement

\[ x = 5; \]

is called an assignment statement because it assigns a value to a variable. The = operator is called the assignment operator, and its meaning is different from equality in mathematics. In mathematics, = asserts that the expression on its left is equal to the expression on its right. In Java, = makes the variable on its left take on the value of the expression on its right. It is best to read \( x = 5 \) as “\( x \) is assigned the value 5,” or “\( x \) gets the value 5.” This distinction is important since in mathematics equality is symmetric: if \( x = 5 \), we know 5 = \( x \). In Java, this symmetry does not exist:

The command 5 = \( x \) attempts to reassign the value of the literal integer value 5, but this cannot be done because 5 is always 5 and cannot be changed. To further illustrate the point consider:

In mathematics no number satisfies the equation \( x = x + 1 \). In Java the statement \( x = x + 1 \) works as follows:

- The expression to the right of the = operator, \( x + 1 \), is evaluated. The current value of \( x \) (5 in this case) is added to 1, and so the right-hand side of the = evaluates to 6.
• The value 6 is then assigned to $x$.

Variables can be reassigned different values as needed:

```java
Welcome to DrJava. Working directory is /Users/rick/java
> int x = 10;
> x
10
> x = 20;
> x
20
```

Notice a variable may not be declared twice in the same interactive session.

Variables may be of type `double`, `boolean`, and `String`:

```java
Welcome to DrJava. Working directory is /Users/rick/java
> amount = 12.08;
> fact = true;
> name = "Lance";
> amount
12.08
> fact
true
> name
"Lance"
```

While unnecessary in DrJava’s Interactions pane, in a Java program variables must be declared before they are used. A declaration specifies a variable’s type to the compiler. The compiler will issue an error if a programmer attempts to use an undeclared variable in a Java program. Variables can optionally be declared in an Interactions pane:

```java
Welcome to DrJava. Working directory is /Users/rick/java
> int x;
> x = 5;
> x
5
```

The statement

```java
int x;
```

is a declaration statement. We saw how variables may be assigned different values as often as we like, but a given variable may be declared only once in a given context:
A variable may not be declared twice in the same interactive session.

Variables of type `double`, `boolean`, and `String` can be declared interactively and must be declared within a Java program:

```java
> double amount = 12.08;
> boolean fact = true;
> String name = "Lance";
> amount
12.08
> fact
true
> name
"Lance"
```

Note that even though we use the term `string` in a generic sense, the type name for string variables is `String` with a capital `S`.

Sometimes programmers need to have numeric objects with immutable values; that is, their values cannot be changed. Such `constants` are specified by the keyword `final`:

```java
> final int C = 100
> C
100
> C = 45
Error: This object cannot be modified
```

Here `C` is a constant that may not be modified. We say that `C` is read only; once its value has been specified you may look at and use its value, but you may not change its value. This notion of using names for unvarying values is common in mathematics and the sciences; for example, $\pi \approx 3.14159$ (ratio of a circle’s circumference to its diameter) and $c = 2.998 \times 10^8 \frac{\text{m}}{\text{sec}}$ (speed of light). While not required by the language, the convention in Java is to use all capital letters for constant names. These symbolic constants can make Java source code more readable, as in:

```java
> final double PI = 3.14159
> PI
3.14159
```
Here $\pi$ stands for the mathematical constant $\pi$.

Table 2.2 shows various ways in which declaration statements can be written. Whether in a Java program or in the Interactions pane, a variable may not change its type during its lifetime. When an undeclared variable is used within the Interactions environment DrJava automatically determines the variable’s type based on the type of the expression on the right-hand side of the assignment operator. Just as in Java programs, though, during a given interactive session a variable’s type is fixed:

<table>
<thead>
<tr>
<th>Description</th>
<th>General Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare one variable and do not give it an initial value</td>
<td>type variable;</td>
<td>int x;</td>
</tr>
<tr>
<td>Declare one variable and give it an initial value</td>
<td>type variable = value;</td>
<td>int x = 5;</td>
</tr>
<tr>
<td>Declare multiple variables of the same type and give none of them initial values</td>
<td>type variable&lt;sub&gt;1&lt;/sub&gt;, variable&lt;sub&gt;2&lt;/sub&gt;, ..., variable&lt;sub&gt;n&lt;/sub&gt;;</td>
<td>int x, sum, height;</td>
</tr>
<tr>
<td>Declare multiple variables of the same type and give some or all of them initial values</td>
<td>type variable&lt;sub&gt;1&lt;/sub&gt; = value&lt;sub&gt;1&lt;/sub&gt;, variable&lt;sub&gt;2&lt;/sub&gt; = value&lt;sub&gt;2&lt;/sub&gt;, ..., variable&lt;sub&gt;n&lt;/sub&gt; = value&lt;sub&gt;n&lt;/sub&gt;;</td>
<td>int x = 5, sum, height = 0;</td>
</tr>
</tbody>
</table>

Table 2.2: Structure of declaration statements. Commas separate variables, and the statement ends with a semicolon.

Here the interpreter correctly determined that $y$ has type `int` when it assigned 19 to $x$. An attempt to assign a `double` value to $y$ then resulted in an error since it is illegal to assign a `double` value to an `int` variable. Notice that the opposite assignment is acceptable:

```
Welcome to DrJava. Working directory is /Users/rick/java
> y = 19;
> y
19
> y = 12.4;
Error: Bad types in assignment
```

```
Welcome to DrJava. Working directory is /Users/rick/java
> z = 12.4;
> z
12.4
> z = 19;
```
Why is this so? Remember that the range of doubles is much larger than the range of ints. In fact, any int value can be represented as a double, so it makes sense to automatically convert an int value to a double for the purposes of assignment. The conversion in the other direction has potential problems because not all double values can be represented as ints because most legal double values fall outside the range of ints. The compiler (and interpreter) thus prohibit the automatic assignment of a double value to an int variable.

What if the double value is in the range of ints? The programmer can force a double value to be assigned to an int variable through an explicit cast. Consider

```java
Welcome to DrJava. Working directory is /Users/rick/java
> int x;
> x = (int) 12.7;
> x
12
```

The name of the type, in this case int, is placed in parentheses before the expression to convert. Notice that the value is truncated; this means the decimal digits are simply dropped and the value is not rounded properly. The programmer must be careful, though:

```java
Welcome to DrJava. Working directory is /Users/rick/java
> int x;
> x = (int) 4.29e15;
> x
2147483647
```

The value $4.29 \times 10^{15}$ is a legitimate mathematical integer, but it is much too big to be a Java int. Notice that the compiler does the best it can under the circumstances—it gives x the biggest value it can.

A boolean value may not be assigned to a numeric type and vice-versa. Similarly, strings are assignment incompatible with numbers and boolean values.

### 2.3 Identifiers

Java has strict rules for variable names. A variable name is one example of an identifier. An identifier is a word that can be used to name a variable. (As we will see in later chapters, identifiers are also used to name methods and classes.) Identifiers can be created as follows:

- Identifiers must contain at least one character
- The first character must be an alphabetic letter (upper or lower case), the underscore, or the dollar sign
  
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_$

- The remaining characters (if any) may be alphabetic characters (upper or lower case), the underscore, the dollar sign, or a digit
  
  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_$/0123456789
2.3. IDENTIFIERS

- No other characters (including spaces) are permitted in identifiers
- A reserved word cannot be used as an identifier

Java reserves a number of words for special use that could otherwise be used as identifiers. Called reserved words, these words are used to define the structure of Java programs and statements. Table 2.3 lists all the Java reserved words.

<table>
<thead>
<tr>
<th>reserved word</th>
<th>reserved word</th>
<th>reserved word</th>
<th>reserved word</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>else</td>
<td>interface</td>
<td>super</td>
</tr>
<tr>
<td>boolean</td>
<td>extends</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>break</td>
<td>false</td>
<td>native</td>
<td>synchronized</td>
</tr>
<tr>
<td>byte</td>
<td>final</td>
<td>new</td>
<td>this</td>
</tr>
<tr>
<td>case</td>
<td>finally</td>
<td>null</td>
<td>throw</td>
</tr>
<tr>
<td>catch</td>
<td>float</td>
<td>package</td>
<td>throws</td>
</tr>
<tr>
<td>char</td>
<td>for</td>
<td>private</td>
<td>transient</td>
</tr>
<tr>
<td>class</td>
<td>goto</td>
<td>protected</td>
<td>true</td>
</tr>
<tr>
<td>const</td>
<td>if</td>
<td>public</td>
<td>try</td>
</tr>
<tr>
<td>continue</td>
<td>implements</td>
<td>return</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>import</td>
<td>short</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>instanceof</td>
<td>static</td>
<td>while</td>
</tr>
<tr>
<td>double</td>
<td>int</td>
<td>strictfp</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Java reserved words

In most programming languages the terms reserved word and keyword are used interchangeably; in Java, a keyword is a reserved word that has special meaning when describing the structure of a program [2]. All of the reserved words in Table 2.3 are keywords except null, true, and false. These three reserved words are technically classified as special literal values, not keywords. None of the reserved keys in Table 2.3 can be used as identifiers. Fortunately, if you accidentally use one of the reserved words in Table 2.3 as a variable name, the compiler (and the interactions pane) will issue an error:

Welcome to DrJava. Working directory is /Users/rick/java
> void = 11
Syntax Error: "void ="

(§ 8.5). The purposes of many of these reserved words are revealed throughout this book.

While mathematicians are content with giving their variables one-letter names (like \(x\)), programmers should use longer, more descriptive variable names. Names such as \(\text{sum}, \text{height}, \text{and subTotal}\) are much better than the equally permissible \(s, h, \text{and st}\). A variable’s name should be related to its purpose within the program. Good variable names make programs more readable by humans. Since programs often contain many variables, well-chosen variable names can render an otherwise obscure collection of symbols quite understandable.

Java is a case-sensitive language. This means that capitalization matters. \texttt{if} is a reserved word, but none of \texttt{If}, \texttt{IF}, or \texttt{iF} are reserved words. Identifiers are case sensitive also:

Welcome to DrJava. Working directory is /Users/rick/java
> String name = "Clyde";
The variable `name` was declared in all lowercase letters, so the variable `Name` does not exist in this session.

### 2.4 Summary

- Java supports a variety of numeric types.
- Numbers represented on a computer have limitations based on the finite nature of computer systems.
- Nonnumeric types include Boolean values, strings, and other object types.
- Variables are used to store values.
- In a Java program, all variables must be declared; in DrJava’s Interactions pane, variables may be declared.
- Variables are assigned values in assignment statements using the `=` operator.
- Variable names follow the rules for naming identifiers as indicated in § 2.3.
- Constants are defined with the `final` specifier.
- Reserved words cannot be used as variable names.
- Floating point values can be assigned to integer variables via a casting operation. Programmers must be careful using casts because it is easy to introduce errors into a program via a legal but bad cast.

### 2.5 Exercises

1. How does DrJava’s Interactions Pane respond if you enter a number? (For example, type 35 and press `Enter`.)
2. How do integer values differ from floating point values?
3. What is Java’s “standard” integer type?
4. What is Java’s “standard” floating-point type?
5. How many primitive types does Java support? List them.
6. What is the largest `int` value supported in Java?
7. What is the largest floating point value supported by Java’s built-in floating point types?
8. How, in Java source code, can you represent the largest floating point value supported by Java’s built-in floating point types?
9. List all possible `boolean` values.
10. Explain how Java’s `=` operator differs from the same symbol used in mathematics. Provide one legitimate example from mathematics that would be an illegal Java expression.
11. How can you declare the variable named `total` to be type integer?
12. What is a string? Is a string a primitive type?

13. What does the final specifier do when declaring a variable?

14. In our example, why did we declare PI to be final?

15. In our example, why did we use the name PI instead pi, Pi, or pI? Could we have used those names instead?

16. How is DrJava’s Interaction’s Pane different from a Java program in terms of variable declarations?

17. What is the difference between a variable name and an identifier?

18. Classify each of the following as either a legal or illegal Java identifier:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Legal/Illegal</th>
</tr>
</thead>
<tbody>
<tr>
<td>fred</td>
<td>Legal</td>
</tr>
<tr>
<td>sum_total</td>
<td>Illegal</td>
</tr>
<tr>
<td>sumTotal</td>
<td>Illegal</td>
</tr>
<tr>
<td>sum-total</td>
<td>Illegal</td>
</tr>
<tr>
<td>sum total</td>
<td>Illegal</td>
</tr>
<tr>
<td>if</td>
<td>Legal</td>
</tr>
<tr>
<td>While</td>
<td>Legal</td>
</tr>
<tr>
<td>$16</td>
<td>Illegal</td>
</tr>
<tr>
<td>_4</td>
<td>Illegal</td>
</tr>
<tr>
<td>___</td>
<td>Illegal</td>
</tr>
</tbody>
</table>

19. What can you do if a variable name you would like to use is the same as a reserved word?