Chapter 3

How to work with the primitive data types

Objectives

Applied

1. Given the specifications for an application that uses any of the eight primitive data types presented in this chapter, write the application.
2. Use the Integer, Double, Math, NumberFormat, and BigDecimal classes to work with data.
3. Given the Java code for an application that uses any of the language elements presented in this chapter, explain what each statement in the application does.

Objectives (cont.)

Knowledge

1. Describe any one of the eight primitive types.
2. Distinguish between a variable and a constant.
3. Given a list of names, identify the ones that follow the naming recommendations for constants presented in this chapter.
4. Explain the difference between a binary operator and a unary operator and give an example of each.
5. Explain the difference between prefixing and postfixing an increment or decrement operator.
6. Explain what a shortcut assignment operator is and how you use one in an assignment statement.
7. List the order of precedence for arithmetic operations and explain how you can change the order in which operations are performed.

Objectives (cont.)

8. Explain what casting is, when it's performed implicitly, and when you must perform it explicitly.
9. Describe how casting between int and double types can affect the decimal value in a result.
10. Describe the primary uses of these classes: Integer, Double, Math, and NumberFormat.
11. List two reasons for using the BigDecimal class.

The eight primitive data types

Type	Bytes	Use
`byte`	1	Very short integers from -128 to 127.
`short`	2	Short integers from -32,768 to 32,767.
`int`	4	Integers from -2,147,483,648 to 2,147,483,647.
`long`	8	Long integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
`float`	4	Single-precision, floating-point numbers from -3.4E38 to 3.4E38 with up to 7 significant digits.
`double`	8	Double-precision, floating-point numbers from -1.7E308 to 1.7E308 with up to 16 significant digits.
`char`	2	A single Unicode character that's stored in two bytes.
`boolean`	1	A *true* or *false* value.

Technical notes

- To express the value of a floating-point number, you can use *scientific notation* like 2.382E+5, which means 2.382 times 105 (a value of 238,200), or 3.25E-8, which means 3.25 times 10-8 (a value of .0000000325). Java sometimes uses this notation to display the value of a floating-point number.
- Because of the way floating-point numbers are stored internally, they can't represent the exact value of the decimal places in some numbers. This can cause a rounding problem.
- By default, Java uses Intel 80-bit extended precision floating-point when it is available from the CPU. As a result, code that uses floating-point numbers may produce slightly different results on different systems.

The assignment operator

Operator	Name
=	Assignment

How to declare a variable and assign a value to it in two statements

Syntax

```
type variableName;
variableName = value;
```

Example

```
int counter;                    // declaration statement
counter = 1;                    // assignment statement
```

How to declare a variable and assign a value to it in one statement

Syntax

```
type variableName = value;
```

Examples

```
int counter = 1;
                // initialize an int variable
double price = 14.95;
                // initialize a double variable
float interestRate = 8.125F;
                // F indicates a floating-point value
long numberOfBytes = 20000L;
                // L indicates a long integer
int population1 = 1734323;
                // initialize an int variable
int population2 = 1_734_323;
                // improve readability – Java 7 and later
double distance = 3.65e+9;
                // scientific notation
```

How to declare a variable and assign a value to it in one statement (cont.)

Examples

```
char letter1 = 'A';
               // stored as a two-byte Unicode character
char letter2 = 65;
               // integer value for a Unicode character
boolean valid = false;
               // where false is a keyword
int x = 0, y = 0;
               // initialize 2 variables with 1 statement
```

Naming conventions for variables

- Start variable names with a lowercase letter and capitalize the first letter in all words after the first word. This naming convention is known as *camel case*.
- Try to use meaningful names that are easy to remember as you code.

How to declare and initialize a constant

Syntax

```
final type CONSTANT_NAME = value;
```

Examples

```
final int DAYS_IN_NOVEMBER = 30;
final float SALES_TAX = .075F;
final double LIGHT_YEAR_MILES = 5.879e+12;
```

Naming conventions for constants

- Capitalize all of the letters in constants and separate words with underscores.
- Try to use meaningful names that are easy to remember.

The arithmetic binary operators

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Code that initializes two integer values

```
int x = 14;
int y = 8;
```

How to perform addition and subtraction

```
int result1 = x + y;        // result1 = 22
int result2 = x - y;        // result2 = 6
```

How to perform multiplication

```
int result3 = x * y;        // result3 = 112
```

How to perform integer division

```
int result4 = x / y;        // result4 = 1
int result5 = x % y;        // result5 = 6
```

Code that initializes two double value

```
double a = 8.5;
double b = 3.4;
```

How to perform decimal division

```
double result6 = a / b;    // result6 = 2.5
```

The arithmetic unary operators

Operator	Name
++	Increment
- -	Decrement
+	Positive sign
-	Negative sign

A statement that uses the increment operator

```
int i = 1;
i++;                // after execution, i = 2
```

A statement that uses the decrement operator

```
int i = 10;
i--;                // after execution, i = 9
```

How to postfix an increment operator

```
int x = 14;
int result = x++;   // after execution, x = 15,
                    // result = 14
```

How to prefix an increment operator

```
int x = 14;
int result = ++x;   // after execution, x = 15,
                    // result = 15
```

How to reverse the value of a number

```
int x = 14;
int result = -x;   // result = -14
```

How to perform an arithmetic operation on a character

```
char letter1 = 'C';         // letter1 = 'C'  Unicode
integer is 67
char letter2 = ++letter1;  // letter2 = 'D'  Unicode
integer is 68
```

The compound assignment operators

Operator	Name
`+=`	Addition
`-=`	Subtraction
`*=`	Multiplication
`/=`	Division
`%=`	Modulus

Statements that use the same variable on both sides of the equals sign

```
count = count + 1;          // count is increased by 1
count = count – 1;          // count is decreased by 1
total = total + 100.0;      // total is increased by 100.0
total = total – 100.0;      // total is decreased by 100.0
price = price * .8;         // price is multiplied by .8
sum = sum + nextNumber;     // sum is increased by the
                            // value of nextNumber
```

Statements that use the compound assignment operators

```
count += 1;                 // count is increased by 1
count -= 1;                 // count is decreased by 1
total += 100.0;             // total is increased by 100.0
total -= 100.0;             // total is decreased by 100.0
price *= .8;                // price is multipled by .8
sum += nextNumber;          // sum is increased by the
                            // value of nextNumber
```

The order of precedence for arithmetic operations

1. Increment and decrement
2. Positive and negative
3. Multiplication, division, and remainder
4. Addition and subtraction

Code that calculates a discounted price

Using the default order of precedence

```
double discountPercent = .2;              // 20% discount
double price = 100;                       // $100 price
price = price * 1 - discountPercent;      // price = $99.8
```

Using parentheses to specify the order of precedence

```
price = price * (1 – discountPercent);  // price = $80
```

Code that calculates the current value of a monthly investment

```
double currentValue = 5000;
                    // current value of investment account
double monthlyInvestment = 100;
                    // amount added each month
double yearlyInterestRate = .12;
                    // yearly interest rate
```

Using parentheses to specify the order of precedence

```
currentValue = (currentValue + monthlyInvestment) *
               (1 + (yearlyInterestRate / 12));
```

Without using parentheses

```
currentValue += monthlyInvestment;      // add investment
double monthlyInterestRate = yearlyInterestRate / 12;
double monthlyInterest = currentValue *
monthlyInterestRate;
currentValue += monthlyInterest;        // add interest
```

How implicit casting works

Data types

byte➔short➔int➔long➔float➔double

Examples

```
double grade = 93;   // convert int to double

double d = 95.0;
int i = 86, j = 91;
double average = (d+i+j)/3;
                     // convert i and j to double values
                     // average = 90.666666...
```

How to code an explicit cast

Syntax

```
(type) expression
```

Examples

```
int grade = (int) 93.75;
                  // convert double to int (grade = 93)

double d = 95.0;
int i = 86, j = 91;
int average = ((int)d+i+j)/3;
                  // convert d to int value (average = 90)
int remainder = ((int)d+i+j)%3;
                  // convert d to int value (remainer = 2)

double result = (double) i / (double) j;
                  // result has decimal places
```

How to cast between char and int types

```
char letterChar = 65;
                // convert int to char (letterChar = 'A')
char letterChar2 = (char) 65;
                // this works too
int letterInt = 'A';
                // convert char to int (letterInt = 65)
int letterInt2 = (int) 'A';
                // this works too
```

How the compound assignment operator can cause an explicit cast

```
int i = 4;
double d = 4.5;
i += d;              // i = 8 (4.5 is cast to the int type)
```

The Java Shell after testing some code

```
[1]-> int x = 14
|  x ==> 14
[2]-> int y = 8
|  y ==> 8
[3]-> x + y
|  $1 ==> 22
[4]-> x / y
|  $2 ==> 1
[5]-> (double) x / y
|  $6 ==> 1.75
[6]-> x++
|  $8 ==> 14
[7]-> x
|  x ==> 15
[8]-> ++x
|  $9 ==> 16
[9]-> x
|  x ==> 16
[10]-> x += 20
|  $11 ==> 36
[11]-> 3 + 4 * 10
```

The Java Shell after testing some code (cont.)

```
|  $15 ==> 43
[12]-> (3 + 4) * 10
|  $19 ==> 70
[13]-> X + 10
|  Error:
|  cannot find symbol
|    symbol:   variable X
|  X + 10
|  ^
[13]->
```

How to use the Java Shell

- With NetBeans 9.0, you can start the Java Shell by selecting the Tools→Open Java Platform Shell command.
- To test a statement or expression, type it at the prompt and press the Enter key.
- When you enter a statement that declares a variable, the shell shows the name of the variable, followed by an arrow (**==>**), and the value of the variable.
- Any of the variables you create remain active for the rest of the session. As a result, you can use them later in the session.
- When you enter an expression, the shell automatically creates a variable that begins with $, and it stores the result of the expression in that variable.
- You can type the name of a variable at the prompt to view its value.

Constructors for the Integer and Double classes

```
Integer(int)
Double(double)
```

How to create Integer and Double objects

```
Integer quantityIntegerObject = new Integer(quantity);
Double priceDoubleObject = new Double(price);
```

Two static methods of the Integer class

```
parseInt(stringName)
toString(intName)
```

Two static methods of the Double class

```
parseDouble(stringName)
toString(doubleName)
```

How to use static methods to convert primitive types to String objects

```
String counterString = Integer.toString(counter);
String priceString = Double.toString(price);
```

How to use static methods to convert String objects to primitive types

```
int quantity = Integer.parseInt(quantityString);
double price = Double.parseDouble(priceString);
```

The Math class

`java.lang.Math`

Common static methods

```
round(number)
pow(number, power)
sqrt(number)
max(a, b)
min(a, b)
random()
```

The round method

```
long result = Math.round(1.667);          // result is 2
int result = Math.round(1.49F);           // result is 1
```

How to round a double value to a specified number of decimal places

```
double x = 10.315;
x = (double) Math.round(x * 100) / 100;  // x is 10.32
x = (double) Math.round(x * 10) / 10;    // x is 10.3
```

The pow method

```
double result = Math.pow(2, 2);
                                  // result is 4.0 (2*2)
double result = Math.pow(2, 3);
                                  // result is 8.0 (2*2*2)
double result = Math.pow(5, 2);
                                  // result is 25.0 (5 squared)
int result = (int) Math.pow(5, 2);
                                  // result is 25 (5 squared)
```

The sqrt method

```
double result = Math.sqrt(20.25);          // result is 4.5
```

The max and min methods

```
int x = 67;
int y = 23;
int max = Math.max(x, y);                  // max is 67
int min = Math.min(x, y);                  // min is 23
```

The random method

```
double x = Math.random() * 100;
                // result is a value >= 0.0 and < 100.0
long result = (long) x;
                // converts the result from double to long
```

The NumberFormat class

```
java.text.NumberFormat
```

Three static methods of the NumberFormat class

```
getCurrencyInstance()
getPercentInstance()
getNumberInstance()
```

Three methods of a NumberFormat object

```
format(anyNumberType)
setMinimumFractionDigits(int)
setMaximumFractionDigits(int)
```

The currency format

```
double price = 11.575;
NumberFormat currency =
NumberFormat.getCurrencyInstance();
String priceString = currency.format(price);
                                          // returns $11.58
```

The percent format

```
double majority = .505;
NumberFormat percent = NumberFormat.getPercentInstance();
String majorityString = percent.format(majority);
                                          // returns 50%
```

The number format with one decimal place

```
double miles = 15341.253;
NumberFormat number = NumberFormat.getNumberInstance();
number.setMaximumFractionDigits(1);
String milesString = number.format(miles);
                                        // returns 15,341.3
```

Two NumberFormat methods that are coded in one statement

```
String majorityString =
    NumberFormat.getPercentInstance().format(majority);
```

The console for the formatted Invoice application

```
Enter subtotal:   150.50
Discount percent: 10%
Discount amount:  $15.05
Total before tax: $135.45
Sales tax:        $6.77
Invoice total:    $142.22

Continue? (y/n):
```

The code for the formatted Invoice application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class InvoiceApp {

    public static void main(String[] args) {
        final double SALES_TAX_PCT = .05;

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            System.out.print("Enter subtotal:   ");
            double subtotal = sc.nextDouble();

            double discountPercent = 0.0;
            if (subtotal >= 100) {
                discountPercent = .1;
            } else {
                discountPercent = 0.0;
            }
```

The formatted Invoice application (cont.)

```
// calculate the results
double discountAmount =
    subtotal * discountPercent;
double totalBeforeTax =
    subtotal - discountAmount;
double salesTax =
    totalBeforeTax * SALES_TAX_PCT;
double total = totalBeforeTax + salesTax;

// format and display the results
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
NumberFormat percent =
    NumberFormat.getPercentInstance();
```

The formatted Invoice application (cont.)

```
            String message =
                "Discount percent: " +
                percent.format(discountPercent) + "\n" +
                "Discount amount:  " +
                currency.format(discountAmount) + "\n" +
                "Total before tax: " +
                currency.format(totalBeforeTax) + "\n" +
                "Sales tax:        " +
                currency.format(salesTax) + "\n" +
                "Invoice total:    " +
                currency.format(total) + "\n";
            System.out.println(message);

            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }
}
```

The console with a rounding error

```
Enter subtotal:   100.05
Discount percent: 10%
Discount amount:  $10.00
Total before tax: $90.04
Sales tax:        $4.50
Invoice total:    $94.55

Continue? (y/n):
```

Debugging statements added to the code

```
String debugMessage = "\nUNFORMATTED RESULTS\n"
                    + "Discount percent: "
                    + discountPercent + "\n"
                    + "Discount amount:  "
                    + discountAmount + "\n"
                    + "Total before tax: "
                    + totalBeforeTax + "\n"
                    + "Sales tax:        "
                    + salesTax + "\n"
                    + "Invoice total:    " + total + "\n"
                    + "\nFORMATTED RESULTS";
System.out.println(debugMessage);
```

The console with debugging information

```
Enter subtotal:   100.05

UNFORMATTED RESULTS
Discount percent: 0.1
Discount amount:  10.005
Total before tax: 90.045
Sales tax:        4.50225
Invoice total:    94.54725

FORMATTED RESULTS
Discount percent: 10%
Discount amount:  $10.00
Total before tax: $90.04
Sales tax:        $4.50
Invoice total:    $94.55

Continue? (y/n):
```

Code that fixes the error using rounding

```
double discountAmount = subtotal * discountPercent;
discountAmount =
    (double) Math.round(discountAmount * 100) / 100;
double totalBeforeTax = subtotal - discountAmount;
double salesTax = totalBeforeTax * SALES_TAX_PCT;
salesTax = (double) Math.round(salesTax * 100) / 100;
double total = totalBeforeTax + salesTax;
```

The BigDecimal class

java.math.BigDecimal

Constructors

```
BigDecimal(String)
BigDecimal(int)
BigDecimal(double)
```

Methods

```
add(value)
subtract(value)
multiply(value)
divide(value, scale, roundingMode)
setScale(scale, roundingMode)
doubleValue()
toString()
```

The RoundingMode enumeration

`java.math.RoundingMode`

Two of the values in the enumeration

`HALF_UP`

`HALF_EVEN`

How to import the classes for working with BigDecimal objects

```
import java.math.BigDecimal;
import java.math.RoundingMode;
```

How to create variables for BigDecimal numbers

```
BigDecimal subtotal = new BigDecimal("100.05");
BigDecimal discountPercent = new BigDecimal(".1");
```

How to create a constant for a BigDecimal number

```
final BigDecimal SALES_TAX_PCT = new BigDecimal(".05");
```

How to multiply and round BigDecimal numbers

```
BigDecimal discountAmount =
    subtotal.multiply(discountPercent);
discountAmount =
    discountAmount.setScale(2, RoundingMode.HALF_UP);
```

Another way to multiply and round BigDecimal numbers

```
BigDecimal discountAmount =
    subtotal.multiply(discountPercent)
        .setScale(2, RoundingMode.HALF_UP);
```

How to add and subtract decimal numbers

```
BigDecimal totalBeforeTax =
    subtotal.subtract(discountAmount);
BigDecimal total = totalBeforeTax.add(salesTax);
```

How to use the BigDecimal class to round a double value

```
discountAmount = new BigDecimal(discountAmount)
        .setScale(2, RoundingMode.HALF_UP)
        .doubleValue();
```

How to create one BigDecimal object from another

```
BigDecimal subtotal2 =
    new BigDecimal(subtotal.toString());
```

The console for the Invoice application with BigDecimal arithmetic

```
Enter subtotal:   100.05
Discount percent: 10%
Discount amount:  $10.01
Total before tax: $90.04
Sales tax:        $4.50
Invoice total:    $94.54

Continue? (y/n):
```

The code for the Invoice application with BigDecimal arithmetic

```
import java.util.Scanner;
import java.text.NumberFormat;
import java.math.BigDecimal;
import java.math.RoundingMode;

public class InvoiceApp {

    public static void main(String[] args) {
        final BigDecimal SALES_TAX_PCT =
            new BigDecimal(".05");

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            System.out.print("Enter subtotal:   ");
            String subtotalString = sc.next();

            // create the BigDecimal objects
            // for subtotal and discount percent
            BigDecimal subtotal =
                new BigDecimal(subtotalString);
            BigDecimal discountPercent;
```

The code for the Invoice application with BigDecimal arithmetic (cont.)

```
if (subtotal.doubleValue() >= 100) {
    discountPercent = new BigDecimal(".1");
} else {
    discountPercent = new BigDecimal("0.0");
}

// calculate the results
BigDecimal discountAmount =
    subtotal.multiply(discountPercent)
        .setScale(2, RoundingMode.HALF_UP);
BigDecimal totalBeforeTax =
    subtotal.subtract(discountAmount);
BigDecimal salesTax =
    SALES_TAX_PCT.multiply(totalBeforeTax)
        .setScale(2, RoundingMode.HALF_UP);
BigDecimal total = totalBeforeTax.add(salesTax);

// the rest of the code for this class
// is the same as figure 3-13
```