

Session 2

Student Name _____

Other Identification _____

Data Types and the while Statement

In this laboratory you will:

1. Learn about three of the primitive data types in Java, `int`, `double`, `char`.
2. Learn about the `String` data type.
3. Investigate some of the elementary operations that can be performed on these types.
4. Investigate how to read input from the keyboard.
5. Investigate the `while` control statement.

In preparation for this laboratory session, you should read Chapters 1 and 2 of *Computer Science: An Overview*. In this and in future sessions, you will be provided with programs that you will be modifying. These are organized by session and experiment number, i.e. `J02E01.java` corresponds to Java session 2, Experiment 1. You should download the programs that correspond to the session from the companion web site at www.aw.com/brookshear before you begin.

Primitive Data Types

Intuitively, there are differences between the many kinds of data that we want to store in a computer. When entering an order at McDonald's, a customer chooses to buy a quantity of different kinds of sandwiches at a stated price. The quantity of sandwiches is a whole number such as 1, 2, or 3. The sandwich ordered is one or more words such as "hamburger", "cheeseburger" or "Big Mac". The price is a fractional number such as 1.89, or 0.99. In programming terminology we say that these groups of data are of different *data types*.

Java provides some basic data types, or *primitive types*. One of these is known as `int`, and is the type associated with the integer values consisting of the counting numbers (1, 2, 3 . . .), their negatives, and 0:

. . . -4, -3, -2, -1, 0, 1, 2, 3, 4 . . .

The representation of a specific value is called a *literal*. Examples of integer literals in Java are `-7`, `0`, `345`, and `67954305`. Notice that large numbers cannot contain commas.

Java also provides a primitive type `double` that represents the type associated with numerical values with a fractional component. Examples of real literals are `8.0`, `-0.0005`, and `345.678`.

As explained in *Computer Science: An Overview*, integer values are normally stored in memory using two's complement notation and, real values are normally stored using floating-point notation. The integer `5`, which is stored using two's complement notation, has a different internal representation than `5.0`, which is stored using floating-point notation.

Single characters can be stored using the primitive data type `char`. Java uses the 2-byte Unicode encoding scheme for character storage. Examples of character literals are `'a'`, `'?'`, and `'5'`. The combination `\n` that was introduced in Session 1 represents a single character, `'\n'`. The symbol `\` is called the escape character and means "escape the normal meaning of the next character".

In a Java program, a string of characters such as "Big Mac" is a `String`. Technically, `String` is not one of the primitive data types in Java, it is actually a class from which we create an object. This is why `String` is capitalized while the primitive data types like `int` and `double` are not. Thus, in a Java program, "Big Mac" is stored as an object of type `String`. If this doesn't make sense yet, it's OK. We will be learning about objects later. For now, you can treat a `String` as if it were a primitive type.

Variables

A *variable* is a name used in a program to refer to an item of data. A variable name, also called an *identifier*, is chosen by the programmer according to the following rules. An identifier

- may consist of uppercase and lowercase letters, digits, \$, and the underscore `_`
- may not begin with a digit
- may not be a Java reserved word

Examples of legal identifiers are `choice1`, `r2d2`, `taxRate`, `weight`.

Examples of illegal identifiers are `#calories`, `2Bad`, `tax Rate`, `int`.

A *Java reserved word* is a word that has a special meaning in Java and, therefore, may not be used in any other context. Examples of reserved words are: `int`, `char`, `class`.

A good programming practice is to choose variable names that reflect their meaning in the program. Example: `weight` rather than just `w`. A Java guideline is to choose variable names that

begin with lower case letters with additional words capitalized. Example: `totalWeight` rather than `total_weight`.

Before a variable can be used in a program, it must be declared; that is, the name and the data type must be described. This is done with a variable declaration statement of the form

```
dataType variableList;
```

where *dataType* indicates the type of the data associated with the variable and *variableList* can be a single variable name or multiple variables names that are separated by commas. A program that contains the statements

```
int quantity;
double weight, totalWeight;
char drinkSize;
String sandwichName;
```

declares five variables. From the machine's point of view, this program segment states that the program requires areas of memory for the storage of one value of type `int`, two values of type `double`, one value of type `char` and one object of type `String`. Moreover, it informs the compiler that these memory locations will be referenced in the rest of the program by the names `quantity`, `weight`, `totalWeight`, `drinkSize` and `sandwichName`.

Java allows a value to be assigned to a variable at the time the variable is declared. This initialization occurs when the variable name is followed by the assignment operator (`=`) and the value to be assigned. The statements

```
int x = 1, y = 2;
double z = 3.75;
char initial = 'D';
String name = "Bugs Bunny";
```

not only establish `x`, `y`, `z`, `initial` and `name` as variables of type `int`, `int`, `double`, `char` and `String`, respectively, but assign the variables the starting values of 1, 2, 3.75, 'D' and "Bugs Bunny". Note that assignment statements are read right to left. That is, `x = 1` is read as "assign 1 to x", not as "x equals 1".

Writing Data to the Monitor Screen

A program is expected to communicate with the world outside of the program. For example, it should be able to receive information from a peripheral device such as the keyboard, mouse or scanner. It should be able to send information to the monitor, printer, or disc drive. Information flows into and out of the computer as a sequence of characters called an I/O Stream.

As you learned in Session 1, the object `System.out` can be used to display characters on the monitor. `System.out` is the output stream that is associated with the monitor. The information that is to be displayed, in this case a string of characters, is given to the `println` method in the parenthetical expression. We say the string is "passed to the `println` method."

```
System.out.println("Special: Big Mac");
```

displays

```
Special: Big Mac.
```

The `println` and `print` methods can also be passed a variable. If the variable `number` is 3 then

```
System.out.println(number);
```

displays

```
3
```


Step 2. Replace the single print statement in Step 1 with the two lines

```
System.out.println(num + " scoops of " +  
    flavor + " ice cream");
```

Compile and execute the modified program and record the results.

Step 3. Modify the code in Step 2 so that the following sentence is printed using the third variable `num2`. There are several ways to do this. Compile and execute the program. Record your solution(s) and the results.

```
2 scoops of chocolate ice cream for the price of 1.
```

Step 4. Modify the program by replacing the statement

```
int num = 2;
```

with

```
int num;
```

Compile the program and record the results of this modification.

Reading Data from the Keyboard

Our programs are more interesting when the program's user can enter information from the keyboard. The keyboard produces an input stream of bytes called `System.in`, which is a predefined object available to all Java programs. This stream is typically processed by an object of type `Scanner`—that is, an object built from the `Scanner` class. Such an object is capable of interpreting a byte stream in various ways. Each capability is accessed by means of a call to a method within the object. Three of these methods are `nextLine`, `nextInt`, and `nextDouble`, which we are about to explore.

The `Scanner` class is defined in the Java Application Programming Interface, or Java API (which is a collection of predefined program units) and must be added (or imported) to any program that uses it. This is done by including the statement

```
import java.util.*;
```

at the beginning of the program. This statement tells the compiler to include (`import`) all the definitions (the asterisk means "all definitions") found in the package named `util` (the Java API is divided into packages of related classes) within the Java API (`java`). The package `java.util` contains definitions that allow various utility functions to be performed.

An object of type `Scanner` can be constructed in a Java program by a single statement of the form

```
Scanner keyboard = new Scanner(System.in);
```

that, in this case, declares the variable `keyboard` to be an object of type `Scanner`, which gets its input from the object `System.in`.

Once the object `keyboard` has been declared by the statement above, we can ask it to retrieve a string of characters from the machine's keyboard by sending it the message `nextLine()` as demonstrated by the statement

```
String line = keyboard.nextLine();
```

that declares `line` to be a variable of type `String` and assigns this variable the string that is created by executing the `nextLine` method in the object `keyboard`.

In summary, the following program reads a line of text from the machine's keyboard and stores the string of characters in an object named `line`.

```
import java.util.*;
class className
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        String line = keyboard.nextLine();
    }
}
```

When a numeric value such as 375 is entered at the keyboard, it is received as a string of symbols—not as a numeric value. More precisely, the value 375 would appear as three encoded characters (the digit 3, followed by the digit 7, followed by the digit 5) rather than as a bit pattern representing the value in two's complement notation. To convert the string of symbols into the correct two's complement representation, the string must be processed by the `nextInt` method instead of the `nextLine` method. Thus, if method `nextInt` is given the string "485" as its input, the method will produce the two's complement representation of 485 as its output. Thus, if the characters 4, 8, and 5 are typed at the keyboard, the statement

```
int num = keyboard.nextInt();
```

will cause the two's complement representation of 485 to be stored as the value of the variable num.

Experiment 2.2

Step 1. Compile and execute the program J02E02.java. In response to the first request enter chocolate, and in response to the second request enter 2. Record the results.

```
import java.util.*;
class J02E02
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int num;
        String flavor;
        System.out.print("What flavor? ");
        flavor = keyboard.nextLine();
        System.out.print("How many scoops? ");
        num = keyboard.nextInt();
        System.out.println(num + " scoops of " + flavor + " ice cream");
    }
}
```

Step 2. Execute the program again, this time responding with chocolate and 2.5. Record what happens, indicating the type of Exception thrown.

Step 3. Execute the program again, this time responding with 15 and 2. Record what happens.

An exception is an error that may cause a program to terminate. For example, if an integer is expected and the user types a number with a decimal point, an object of type `Scanner` will generate (the technical term is “throw”) an Exception known as an `InputMismatchException` when the object tries to read the line of characters. When an object of type `Scanner` throws an `InputMismatchException`, the object will not pass the token that caused the exception, so that it may be retrieved or skipped via some other method.

Since fractional values are stored using a different representation than integers, the `nextDouble` method rather than the `nextInt` method must be used to process a string representing a numeric value with a fractional component. Thus, if the string "48.5" is typed at the keyboard, the statement

```
double decimal = keyboard.nextDouble();
```

will cause the value 48.5 represented in floating-point notation to be stored as the value of the variable `decimal`.

Experiment 2.3

Step 1. Compile and execute the program `J02E03.java`, which demonstrates how to read data of type `double` from the keyboard. In response to the request enter `5.678`. Record what happens.

```
import java.util.*;
class J02E03
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double number;
        System.out.print("Enter a real number: ");
        number = keyboard.nextDouble();
        System.out.println("You entered " + number);
    }
}
```

Step 2. Execute the program again. Respond with `87` and record the results.

Step 3. Execute the program again. Respond with your name and record the results.

Elementary Operations and the Assignment Statement

The assignment operator (denoted by an equal sign =) can also be used to assign a new value to a variable. If `weight` is initialized to 1.89,

```
double weight = 1.89;
```

it can later be assigned a new value of 0.99.

```
weight = 0.99;
```

When this is done, the original value is overwritten and can no longer be accessed.

The right side of an assignment statement can be any expression whose value is compatible with the variable on the left side. After these statements are executed, both `fat` and `calories` store 270.

```
int calories, fat;
calories = 270;
fat = calories;
```

In the case of numeric data, the traditional arithmetic operations can be used in an expression on the right side of the assignment statement. First, the calculation is performed and then it is assigned to the variable on the left. Here the symbols `+`, `-`, `*`, `/` are used to represent addition, subtraction, multiplication and division, respectively. Therefore, the statement

```
pounds = calories * fat;
```

assigns the product of `calories` and `fat` to `pounds`.

Parentheses can be used to modify the order in which the expression on the right side of an assignment statement is evaluated. Therefore,

```
servingCalories = (fatCalories + sugarCalories) / servings;
```

will correctly calculate the number of calories per serving (`servingCalories`), whereas

```
servingCalories = fatCalories + sugarCalories / servings;
```

will divide `sugarCalories` by `servings` before adding `fatCalories`, thus producing an incorrect result.

A common operation is that of adding 1 to an existing value. Incrementing a value by 1 is accomplished by the statement `x = x + 1;` where `x` is a numerical variable. Java provides a shorthand notation for this assignment statement. It consists of the name of the variable to be incremented followed by two plus signs

```
x++; //++ is called auto increment
```

A similar expression is a shorthand notation for the statement `x = x - 1;` that decrements the value of `x` by one.

```
x--;          //-- is called auto decrement
```

Expressions such as `x++` and `x--` can be used in more complex expressions such as in the assignment statement

```
y = 5 + x++;
```

Here, `y` is assigned the result of adding 5 to `x` and then `x` is incremented by 1. In particular, the combination

```
x = 2;
y = 5 + x++;
```

results in `y` being assigned the value 7 and `x` being assigned 3. Note that the value of `x` is not incremented until its original value has been used in the computation.

If, instead, we want the incremented value to be used in the computation, we would use the expression `++x` rather than `x++`.

```
x = 2;
y = 5 + ++x;
```

In this case, after the execution, `y` is assigned 8 and `x` is assigned 3. In short, `++x` means that the value of `x` is incremented first and this new `x` value is used in the remaining computation. Whereas, `x++` means that the computation is completed with the original value of `x` and `x` is incremented after.

Experiment 2.4

Step 1. Predict the output of the program

```
class J02E04
{
    public static void main(String[] args)
    {
        System.out.println(7 + 3);           _____
        System.out.println(7 - 3);           _____
        System.out.println(7 * 3);           _____
        System.out.println(7 / 3);           _____
    }
}
```

Step 2. Compile and execute the program `J02E04.java`. Record the results and comment on any results that differ from your prediction.

Step 3. Modify the program in Step 1 by replacing each occurrence of 7 with 7.0 and each occurrence of 3 with 3.0. Compile and execute the program. Record the results, commenting on differences from the results in Step 2.

Step 4. Modify the program in Step 3 by replacing each occurrence of 7.0 with 7. Compile and execute the program. Record the results, commenting on differences from the results in Step 3.

Step 5. Modify the 4 print statements as shown. Predict the results. Then, compile and execute the program, recording the results below. Comment on any unexpected results.

```
System.out.println(5 / 2);           _____  
System.out.println(5.0 / 2);        _____  
System.out.println(5 / 2.5);        _____  
System.out.println(0.5 / 2.5);      _____
```

Experiment 2.5

Step 1. Consider the program J02E05.java. Predict the results.

```
class J02E05
{
    public static void main(String[] args)
    {
        int b = 4, c = 7;
        double average;
        average = (b + c) / 2;
        System.out.println(
            "Average of " + b + " and " + c + " is " + average);
    }
}
```

Step 2. Compile and execute the program. Record and explain the results.

Step 3. Using what you learned from Experiment 2.4, correct the program so that the calculated average is 5.5. List your changes. Compile and execute the program. Record the results.

Step 4. Modify the program by adding a third integer, *d*, that is assigned a value of 2. Find the average of the three numbers. List your changes. Compile and execute the program. Record the results.

Experiment 2.6

Step 1. The operator `%` is called the *modulus* operator. Compile and execute the program `J02E06.java`. Respond with 7 and 3. Record the results in the first row of the following table.

```
import java.util.*;
class J02E06
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int x = keyboard.nextInt();
        System.out.print("Enter an integer: ");
        int y = keyboard.nextInt();

        System.out.println(x + " / " + y + " = " + (x / y));
        System.out.println(x + " % " + y + " = " + (x % y));
    }
}
```

x	y	x / y	x % y
7	3		
12	4		
5	7		
19	7		
99	25		

Step 2. Execute the program several times, each time changing the values you enter for x and y. Compile and execute the program. Record the results in the table above.

Step 3. What is the meaning of `x % y`?

Experiment 2.7

Step 1. Predict the result of executing the program J02E07.

```
class J02E07
{
    public static void main(String[] args)
    {
        int x, y, z;
        x = 4;
        y = x++ + 5;
        System.out.println("x = " + x + " y = " + y);
        x = 4;
        y = ++x + 5;
        System.out.println("x = " + x + " y = " + y);
        x = 4;
        y = x-- * 5;
        z = --x * 5;
        System.out.println("x = " + x + " y = " + y + " z = " + z);
    }
}
```

Step 2. Compile and execute the program. Record the results. Explain any discrepancies with your prediction.

statement following the closing brace of the while statement. Due to its cyclic nature, we refer to the while statement as a while loop. Thus, the following statements

```
int count = 5;
while(count > 0)
{
    System.out.println("Recycle!");
    count--;
}
```

produce

```
Recycle!
Recycle!
Recycle!
Recycle!
Recycle!
```

Initially, count is 5 and the condition is true. Therefore, Recycle! is printed and count is decremented. The loop is executed five times, when count is 5, 4, 3, 2, and 1. When count is 0, the condition is false and the loop is exited.

We will discuss the details of the *condition* part of a while statement in Session 4. For now, we merely note that the symbols <, >, ==, and != are used to represent "is less than", "is greater than", "is equal to", and "is not equal to", respectively, and combined as in <= and >= to represent "is less than or equal to" and "is greater than or equal to".

Experiment 2.8

Step 1. The following program J02E08.java is designed to print the integers 1 through 5. Compile and execute the program. Record the results.

```
class J02E08
{
    public static void main(String[] args)
    {
        int count;
        count = 1;
        while(count < 5)
        {
            count++;
            System.out.println(count);
        }
    }
}
```

Step 2. Explain the changes required to make the program perform as initially intended. Confirm your answer by making these changes and compiling and executing the corrected program.

Step 3. Modify this program as follows:

1. At the beginning of the main method, add: `char c = 'a';`
2. Replace the body of the loop with the three statements below.

```
count++;  
c++;  
System.out.println(c);
```

Compile and execute the program. Record the results.

Step 4. Change the data type of the variable `c`, to `double`. Compile and execute the program. Comment: The `++` operator may be used on variables of what primitive types?

Post-Laboratory Problems

2.1. Write a program with the declarations

```
int x = 3, y = -2, w = 7, z = 20;
```

that computes and prints the results of each of the following arithmetic problems. According to your results, summarize the precedence rules for the mathematical operators $/$, $\%$, $*$, $+$, $-$.

- | | |
|----------------------|------------------------|
| a. $z / x / y$ | b. $z + x * y$ |
| c. $(z + x) * y$ | d. $z / - y * w$ |
| e. $5 + z \% x$ | f. $5 + z / x$ |
| g. $z \% w - z \% w$ | h. $9 - x * (2 + y)$ |

2.2. Write a program that calculates a Celsius temperature from the Fahrenheit temperature supplied by the user. Use the formula $C = 5/9(F - 32)$.

2.3. Write a program that uses a `while` loop to find the sum of the numbers between 1 and N , where N is an integer value supplied by the user.

2.4. In this session you learned a shorthand notation for incrementing/decrementing variables. A similar shorthand consists of an operation symbol followed by an assignment symbol such as `*=`, `+=`, `-=` or `/=`. Write a program that contains statements similar to those below.

```
int x = 5;
x += 3;
System.out.println(x);
x += 10;
System.out.println(x);
```

Based on the results, state what `+=` appears to represent. Then, experiment with the other arithmetic operations to confirm your hypothesis. Present your findings in an organized manner. Extend your investigation by trying expressions such as `x *= 2 + z`.