Loop Structures and Strings

Chapter 6
Java
Barrett
The While Statement

• Loop Structure –
  – The while is a loop structure which executes a set of statements over and over again based on a condition. Loop structures are used to:
    • Perform tasks such as summing a set of numbers as they are entered by the end user.
    • Repeatedly prompting the user for a value until valid data is entered.
    • Etc.
The While Statement

• The while statement takes the form:

```java
while (<conditions>) {
    <statements>
}
```

```java
int num = 0
while (num < 5) {
    num += 1;
}
```
The While Statement

• The condition of the **while** loop is a Boolean expression, which is evaluated before the statements are executed.

• When the condition is true the statements are executed. When false, the program flow continues to the next statement after the closing curly brace of the while.

• **Iteration** - each execution of the loop is called an iteration.
The While Statement

• Iterations – Each value represents one iteration

```java
int num = 0

while (num < 5) {
    num += 1;
    System.out.print(num);
}
```

Program Output:

```
12345
```
The While Statement

• Ex.
  – TheWhileStatement
The do-while Statement

- The **do-while** statement is an alternative form of the while statement.
- In the do-while statement, the condition is not evaluated until after the first execution of the loop.
The do-while Statement

- The do-while statement takes the form:

```java
do {
    <statements>
} while (<conditions>);
```

Example code:
```java
do {
    System.out.print("Enter a number less than 4:");
    playerNum = input.nextInt();
} while (playerNum >=4);
```
Do-While Explained

do {
    <statements>
} while (<conditions>);

do {
    this
} while (this is true);
WHEN TOLD - Move to Computers

• Create a **Chapter 6 Programs** project
  – All of the chapter 6 classes you create will be here.
• Grab a textbook or go to the e-book
• Turn to page 133 (p.150, e-book)
• Create a new class in the chapter 6 programs project and call it **Prompter**.
• Write the **Prompter** program.
Infinite Loops

• **int** data range (4 bytes)
  
  \[ -2,147,483,647 \text{ to } 2,147,483,647 \]

• **double** data range (8 bytes)
  
  \[ -1.7E+308 \text{ to } 1.75E+308 \]

• Data range example... *(class name: InfiniteLoops)*
Infinite Loops

• Infinite Loops –
  – when a condition is never false, an *infinite loop* (a loop that continues forever) is the result.

• Infinite Loop example...

• Overflow –
  – occurs when there are not enough bits to store a number.
Counters and Accumulators

Imagine a program that produces the following output. How would this program work? How can we prepare to write this program?

**Pseudocode**

Enter a value (0 to quit): 25
Enter a value (0 to quit): 14
Enter a value (0 to quit): 18
Enter a value (0 to quit): 22
Enter a value (0 to quit): 30
Enter a value (0 to quit): 0
Average is 21.8
Counters and Accumulators

Pseudocode for program on previous slide:

Prompt user for a value
WHILE (value != 0)
    count value
    add value to sum of values
    prompt user for another value
Print average of values (sum/count)
Counters and Accumulators

• A *counter* -
  – A *counter* is utilized to count the number of times a looped statement executes.
    \[ \text{numValues += 1;} \]

• An *accumulator*
  – Each time the statement executes, the value of *newValue* is added to the current value of the variable. The variable is called an *accumulator*.
    \[ \text{sumOFValues += newValue;} \]

• “Flag” aka “Sentinel”
  – A constant value that signifies that a loop should stop iterating
Counters and Accumulators

• As a class we will write a program that produces the following output:

Enter a value (0 to quit): 25
Enter a value (0 to quit): 14
Enter a value (0 to quit): 18
Enter a value (0 to quit): 22
Enter a value (0 to quit): 30
Enter a value (0 to quit): 0
Average is 21.8

Program on page 134
Assignment

- NumberSum
  - P. 134

- Evens
  - P. 134
What is the difference???

We call this an **increment operator**
Our Increment Operators

- $i += 1;$
- $i -= 1;$
- $i *= 1;$
- $i /= 1;$

Where $i$ is our initialized variable.
The for Statement

**Structural**

```java
for (<initialization>; <condition>; <increment>){
    <statements>
    <statements>
    <statements>
}
```

**Contextual**

```java
for (int i =1;  i <= 10;       i++){
    System.out.print(a);
}
```
The for Statement

Without the spacing...

```java
for (int i = 1; i <= 10; i++){
    System.out.print("a");
}
```

• What is this Statement going to accomplish?
• What is our output going to look like?
The for Statement

• The *for* statement is a loop structure that executes a set of statements, a fixed number of times.

• The condition is a Boolean expression which is evaluated before each loop iteration.

```java
for (int i = 1; i <= 10; i++) {
    System.out.print(a);
}
```
The for Statement

- The counter “i” is a loop control variable
- When “i” is greater than 10, looping terminates

```java
for (int i = 1; i <= 10; i++){
    System.out.print(a);
}
```
The for Statement

- The **scope** of the counter is from the initialization to the closing curly brace of the for statement.

```java
for (int i = 1; i <= 10; i++){
    System.out.print(i);
}
```

- The application will not recognize the variable “i” outside of the statement
The for Statement

• Any combination of components can be left out of a `for` statement. This is useful when a counter is declared, and initialized, outside the statement

```java
int num;
System.out.print("Enter the starting number: ");
num = input.nextInt();

for ( ; num <= 10; num++){
    System.out.print(num);
}
```
Assignment

• Factorial
  – P. 136

• OddSum
  – P. 136
The for statement REEVALUATED

• We will use the variable “i” as our loop control variable for all of our for statements.

```java
for (int i =1; i <= 10; i++){
    System.out.print(“i”);
}
```

• The `for` statement is a loop structure that executes a set of statements, a fixed number of times.
Review

Factorial –

```java
int factorial = 1;
for (int i = 1; i <= num; i++)
    factorial = factorial * i;
```

Iteration (1) - Factorial = 1 * 1 = 1
Iteration (2) - Factorial = 1 * 2 = 2
Iteration (3) - Factorial = 2 * 3 = 6
Iteration (4) - Factorial = 6 * 4 = 24
Iteration (5) - Factorial = 24 * 5 = 120
Modified Factorial Output

- Modify your Factorial program’s Output to read:

Enter a number: 5
5*4*3*2*1= 120
The factorial of 5 is 120
Strings and Substrings

Computer Science
Strings

A **string** is a sequence of characters. Ex:

- Barrett
- Barrett is a Man
- Coltrane is Barrett’s Dog
- Centennial’s has sports teams that compete during the Fall, Winter, and Spring.
Positions in Strings

- Positions of characters in strings are numbered
- The first character in a string is at position zero
- So in the string “ABCDE”
  - A is at position zero
  - B is at position 1
  - E is at position 4
Outputting a String

String A = “Mr. Barrett”;  
String B = “Computer Science”;  

System.out.print(A);  
System.out.print(B);  

// output A = Mr. Barrett  
// output B = Computer Science
The Substring Method

`variableName.substring(int start, int end);`

- returns a substring of the a string, which starts at the ‘int start‘ index position, and ends at the index one space prior to the ‘int end’ index position.

`phrase.substring(4, 8);`
Parts of substring

System.out.print(str.substring(0,2));

• Variable name
• followed by a period (.)
• Followed by substring
• An open parenthesis (()
• A number, the position of the character to start the substring
• A comma
• Another number, the position the substring ends before
• A close parenthesis
Finding substrings

• Java gives you a way to find a piece of a string

• Example:
  
  String str = “ABCDE”; 
  System.out.print(str.substring(0,2));
  – Will print AB

• str.substring(0,2) means “the piece of str that starts at position 0 and ends before position 2”
  – Position 0 contains an A
  – Position 2 contains a C
  – So str.substring(0,2) starts at the A and ends before the C
  – Thus AB
Substring example

Example. What will this print?

```java
String s1 = "FGHIJ", s2 = "LMNOP";
System.out.println(s1.substring(1,4));
System.out.println(s2.substring(3,5));
```

- `s1.substring(1,4)` starts at position 1 in S1 and ends before position 4
  - Position 1 of s1 contains a G
  - Position 4 contains a J
  - So s1.substring(1,4) starts at the G and ends before the J
  - Thus GHI

- `s2.substring(3,5)` starts at position 3 in S2 and ends before position 5
  - Position 3 of s2 contains an O
  - Position 5 doesn’t exist
  - But that’s okay, because we’re ending before position 5, so we go to the end of the string
  - Thus OP
Another form of substring

- So far, when we have used substrings, we have put two numbers in between the parenthesis
  1. where the substring starts
  2. where the substring ends

- Substrings can also include only one number between the parenthesis
  – Position where the substring starts
  – Substring continues to the end of the string

- The numbers inside the parentheses are called *parameters*
Example of `substring` with 1 parameter

- Example:
  ```java
  String s = new String("PQRST");
  System.out.print(s.substring(2));
  ```
- Starts at position 2 in `s` contains `R` (recall java starts numbering at zero)
- Ends after `T`, the end of the string
- Thus prints `RST`
What if you go too far?

• What would this do?:
  String str = “WXYZ”;
  System.out.print(str.substring(3,5));

• Starts at position 3 that contains Z
• Ends before position 5. That would mean it would contain whatever is at position 4
• There is no position 4
• What does Java do?
• Answer: **Crash**
  – If a substring requires a non-existent position, Java will stop your program
  – A problem that is serious enough to stop a program is called an **exception**
Other Methods

• Examples
  – String Class Method: Review Sheet
  – Java

```
  stringName.Method();
  stringName.Method(Parameters);
```
Receiving a String from an end user

• After prompting ("Enter your name")

• We will have to use our...

```
    _stringName_ = input.nextLine();
```

method. Remember, we covered different method calls in chapter 4.
Receiving a String from an end user

String name;  //string declaration

System.out.print("enter your name");
__name__ = input.nextLine();
input.close();
Obtaining a value from the user

Review

input.next() – returns a string from the input stream.

input.nextLine() – returns a string up to the end of line character from the input stream
Assignment

• AccountSetup
  – p. 139

• DigitsDisplay
  – P. 151
COMPARING STRINGS
Comparing Strings

- Strings can be compared using the `equals()` method.

```java
    if (stringName.equals(newStringName))
    {
        System.out.print("Good");
    }
    else{
        System.out.print("Bad");
    }
```
Comparing Strings

• ________.equalsIgnoreCase( ________ ) – method returns true when two strings are equal to each other. Casing is not a factor.

Barrett == baRrETT
Comparing Strings

• Our data types:
  - Int
  - double
  - char
  - Boolean

Today’s focus – char, which represents a single character.
Comparing Strings

• We can initialize a character using the statement below

```java
char letter = 'h';
```

```java
System.out.print(letter); //output below
```

```
h
```
Comparing Strings

\[ \text{stringName}.indexOf(\_); \]

- returns the index number corresponding to the first instance of a substring or character.

String exp = “shoes”;  
String b = “floor for our shoes”; 
System.out.println(exp.indexOf(‘e’)); //output below 
System.out.print(b.indexOf(exp)); //output below

3
14
Comparing Strings

```java
String st = "Mississippi";
String foo = "of";
String b = "Maryland is north of Mississippi of";
```

```java
System.out.println(st.lastIndexOf('i'));//output below
System.out.println(b.lastIndexOf(foo)));//output below
```

10

33
Ch 6 Programs

CH 6

• AccountSetup (p. 139)
• DigitsDisplay (p. 151)
• DigitsSum (p. 151)
• ElapsedTimeCalculator (p. 153)
• Necklace (p. 153)