NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#6: Looping Statements

Objectives

- Introducing the idea of looping statements and conditions.
- Introducing *for loop statement*
- Introducing *while statement*
- Introducing *do… while statement*
- Learning the difference between looping statements
- Learning from examples of loops in Java

Processing Statements

- Tree methods of processing statements in a program
  - *In sequence*
  - *Branching*
  - *Looping*
- *Branching:* Altering the flow of program execution by making a selection or choice (if else … else if …)
- *Looping:* Altering the flow of program execution by repetition of a particular block of statement(s)

Source: The Art and Science of Java, Eric Roberts

Statement Types in Java

- Programs in Java consist of a set of **classes**. Those classes contain **methods**, and each of those methods consists of a sequence of **statements**. (we will see this again)
- Statements in Java fall into three basic types:
  - Simple statements
  - Compound statements
  - Control statements
- **Simple statements** are formed by adding a semicolon to the end of a Java expression.
- **Compound statements** (also called **blocks**) consist of a sequence of statements enclosed in curly braces.
- **Control statements** fall into two categories:
  - **Conditional statements** that specify some kind of test
  - **Iterative statements that specify repetition**

Source: The Art and Science of Java, Eric Roberts

The Repeat-N-Times Paradigm

One strategy for generalizing the addition program is to use the Repeat-N-Times idiom, which executes a set of statements a specified number of times. The general form of the idiom is

```
for (int i = 0; i < repetitions; i++) {
   statements to be repeated
}
```

The information about the number of repetitions is specified by the first line in the pattern, which is called the **header line**.

The statements to be repeated are called the **body** of the **for** statement and are indented with respect to the header line.

A control statement that repeats a section of code is called a **loop**.

Each execution of the body of a loop is called a **cycle**.

Source: The Art and Science of Java, Eric Roberts

The for Statement Template

The **`for`** statement in Java is a particularly powerful tool for specifying the control structure of a loop independently from the operations the loop body performs. The syntax looks like this:

```
for ( init ; test ; step ) {
    statements to be repeated
}
```

Java evaluates a **`for`** statement by executing the following steps:

1. Evaluate *init*, which typically declares a **control variable**.
2. Evaluate *test* and exit from the loop if the value is **`false`**.
3. Execute the statements in the body of the loop.
4. Evaluate *step*, which usually updates the control variable.
5. Return to step 2 to begin the next loop cycle.

Source: The Art and Science of Java, Eric Roberts

The while Statement

The **while** statement is the simplest of Java's iterative control statements and has the following form:

```
while ( condition ) {
    statements to be repeated
}
```

When Java encounters a **while** statement, it begins by evaluating the condition in parentheses, which must have a **boolean** value.

If the value of *condition* is **true**, Java executes the statements in the body of the loop.

At the end of each cycle, Java reevaluates *condition* to see whether its value has changed. If *condition* evaluates to **false**, Java exits from the loop and continues with the statement following the closing brace at the end of the **while** body.

Source: The Art and Science of Java, Eric Roberts

These statements are equivalent

```
for (i = 0; i < N; ++i)
    s += i;
```

```
i = 0;
while (i < N) {
        s += i;
        ++i;
}
```

The Repeat-Until-Sentinel Idiom

A better approach for the addition program that works for any number of values is to use the Repeat-Until-Sentinel idiom, which **executes a set of statements until the user enters a specific value called a sentinel to signal the end of the list:**

```
while (true) {
   prompt user and read in a value
   if (value == sentinel) break;
   rest of loop body
}
```

You should choose a sentinel value that is not likely to occur in the input data. It also makes sense to define the sentinel as a named constant to make the sentinel value easy to change.

Source: The Art and Science of Java, Eric Roberts

The Loop-and-a-Half Pattern

The **`while`** statement in Java always tests the condition at the beginning of each cycle of the loop. Sometimes, however, you need to perform some computation *before* you can make the test. In those situations, the **loop-and-a-half** pattern is very useful:

```
while (true) {
    computation necessary to make the test
    if (test for completion) break;
    computation for the rest of the loop cycle
}
```

Because the condition in the **`while`** statement itself is always **`true`**, this loop would continue forever without some other strategy to indicate completion. The loop-and-a-half pattern uses the **`if`** and **`break`** statements to exit the loop. When the test for completion becomes **`true`**, Java executes the **`break`** statement, which causes the loop to exit, skipping the rest of the cycle.

Source: The Art and Science of Java, Eric Roberts

The Repeat-Until-Sentinel Idiom

A better approach for the addition program that works for any number of values is to use the Repeat-Until-Sentinel idiom, which **executes a set of statements until the user enters a specific value called a sentinel to signal the end of the list:**

```
while (true) {
    prompt user and read in a value
    if (value == sentinel) break;
    rest of loop body
}
```

You should choose a sentinel value that is not likely to occur in the input data. It also makes sense to define the sentinel as a named constant to make the sentinel value easy to change.

Source: The Art and Science of Java, Eric Robert

do .. while statement

Use: After executing the repeating statements, then check the conditional expression

Syntax of the Do-While Statements

```
do {
    statement 1
    statement 2
    …
} while (<conditional expression>);
```

Important: *Although the conditional expression is false, The statement (s) will be executed at least one.*

Source: The Art and Science of Java, Eric Roberts

Review
Conditions and Bitwise operations

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

&& *and*

|| *or*

! *not*

```
n >= 1 && n <= 10
1 <= n && n <= 10
```

Sample Programs

(Please make sure you understand the following programs we did in class)

```java
import java.util.Scanner;
public class UnlockAccount {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String actualpassword = "12345";
        String actualusername = "edu";
        String enteredpassword;
        String enteredusername;

        int num_times = 0;

        Scanner myscanner = new Scanner(System.in);

        while(true){
        System.out.println("Please enter your password.");
        enteredpassword = myscanner.next();

        System.out.println("Please enter your username.");
        enteredusername = myscanner.next();

        if(!(actualpassword.equals(enteredpassword))||(!actualusername.equals(enteredusername)))
        {
            num_times++;


        }
        if(num_times == 3){
            System.out.println("Your account has been locked"); break;
        }


        }
```

AddSeveralNumbers.java

```java
/*
 *          Control Statements
 * This program uses the While Loop statement to add several numbers
 */
import java.util.Scanner;

public class AddSeveralNumbers {

        public static void main (String args[]) {
            int N=0;
            int x;
            int sum = 0;
            Scanner Scan = new Scanner(System.in);
            System.out.println("How many numbers are you adding?");
            N = Scan.nextInt();
            int count = 0;

            while(count <=N) {

                System.out.println("?");
                x = Scan.nextInt();
                sum = sum + x;
                count++;
            }
            System.out.println("sum " + sum);

}
}
```

*AddSeveralNumbers.java

```java
/*
 *          Control Statements
 * This program uses the While - sentinel statement to add several numbers
 */
import java.util.Scanner;

public class AddSeveralNumbers {

        public static void main (String args[]) {
            int N=0;
            int x;
            int sum = 0;
            Scanner Scan = new Scanner(System.in);
            System.out.println("How many numbers are you adding?");
            N = Scan.nextInt();
            int count = 0;

            while(true) {

                System.out.println("?");
                x = Scan.nextInt();
                sum = sum + x;
                count++;
                if(count == N) break; //break: exit the while loop
            }
            System.out.println("sum " + sum);

}
}

```

*AddSeveralNumbers.java

```java
/*
 *          Control Statements
 * This program uses For statement(for loop) to add several numbers
 */
import java.util.Scanner;

public class AddSeveralNumbers {

        public static void main (String args[]) {
            int N=0;
            int x;
            int sum = 0;
            Scanner Scan = new Scanner(System.in);
            System.out.println("How many numbers are you adding?");
            N = Scan.nextInt();


            for(int count=0;count<N;count++) {

                System.out.println("?");
                x = Scan.nextInt();
                sum = sum + x;

            }
            System.out.println("sum " + sum);

}
}
```

AddSeveralNumbers.java

```java
/*
 *          Control Statements
 * This program uses the do while statement to add several numbers the user inputs
 * the program stops until answers "yes" to Are you done entering the number you want to add?
 */
import java.util.Scanner;

public class AddSeveralNumbers {

        public static void main (String args[]) {
            int x;
            int sum = 0;
            String userEnteredMessage = " ";


            do {

                Scanner ourScanner = new Scanner(System.in);
                System.out.println(" Enter you number");
                x = ourScanner.nextInt();
                sum = sum + x;
                System.out.println("Are you done entering the numbers you want to add");
                userEnteredMessage = ourScanner.next();

            } while(!userEnteredMessage.equals("yes"));

            System.out.println("sum " + sum);

}
}
```

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#6: Looping Statements