Building Java Programs

Chapter 5: Program Logic and Indefinite Loops

Chapter outline

- indefinite loops
  - the `while` loop
  - sentinel loops
- generating random numbers with `Random` objects
- Boolean logic
  - `boolean` expressions and variables
  - logical operators
- testing for valid user input
- indefinite loop variations
  - the `do/while` loop
- logical assertions

while loops

reading: 5.1

Definite loops

- **definite loop**: One that executes a known number of times.
  - The `for` loops we have seen so far are definite loops.

  - We often use language like,
    - "Repeat these statements *N* times."
    - "For each of these 10 things, ..."

  - Examples:
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

Indefinite loops

- **indefinite loop**: One where it is not obvious in advance how many times it will execute.
  - The `while` loops in this chapter are indefinite loops.

  - We often use language like,
    - "Keep looping *as long as* or *while* this condition is still true."
    - "Don't stop looping *until* the following happens."

  - Examples:
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Continue looping while the user has not typed "n" to quit.

The while loop statement

- **`while` loop**: Executes as long as a condition is true.
  - well suited to writing indefinite loops

- `while` loop, general syntax:

```
while (<condition>) {
    <statement(s)> ;
}
```

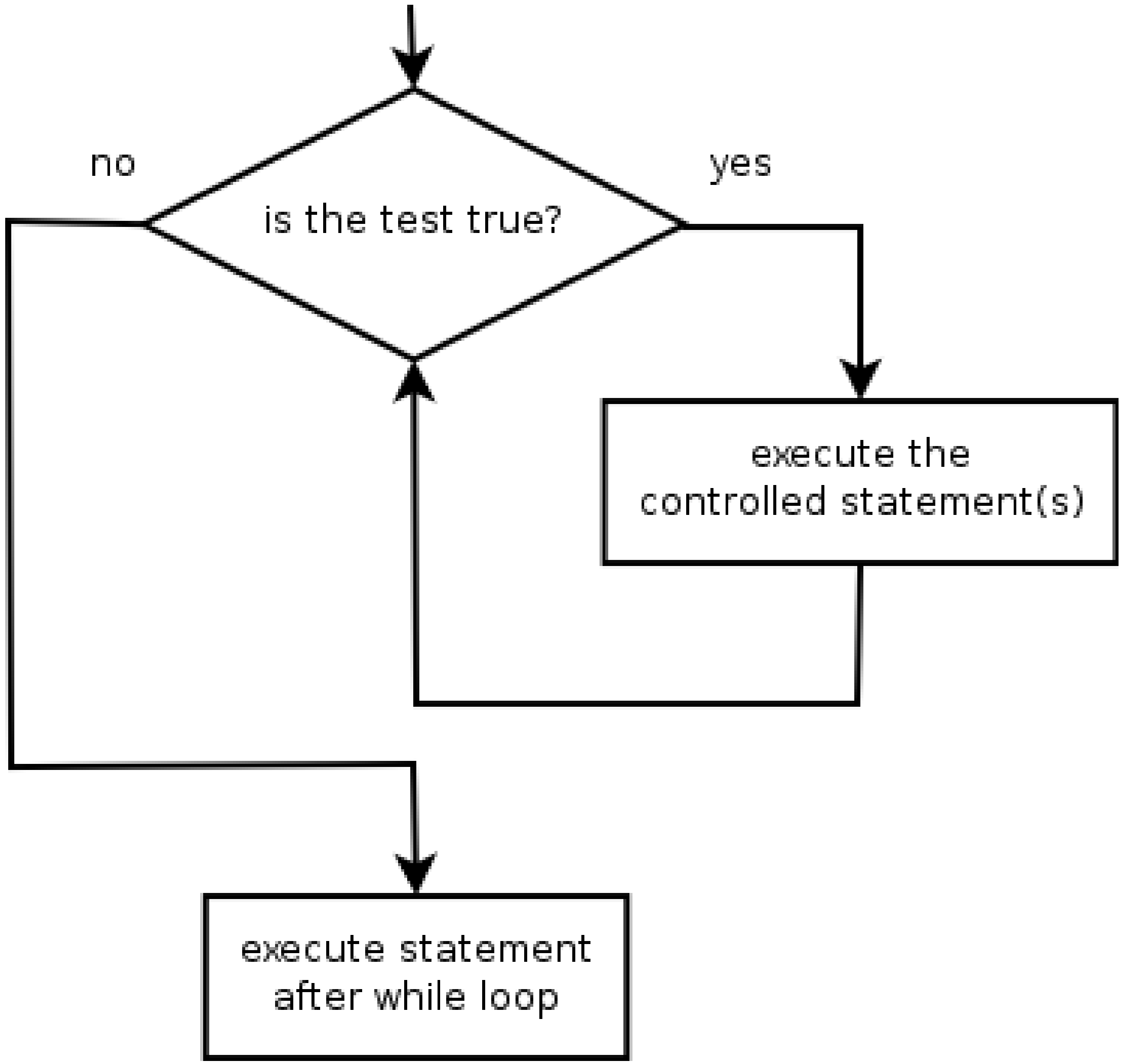
- Example:

```
int number = 1;
while (number <= 200) {
    System.out.print(number + " ");
    number = number * 2;
}
```

  - OUTPUT:

```
1 2 4 8 16 32 64 128
```

While loop flow chart



Example while loop

- Finds and prints a number's first factor other than 1:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int number = console.nextInt();
int factor = 2;
while (number % factor != 0) {
    factor++;
}
System.out.println("First factor: " + factor);
```

- Example log of execution:

```
Type a number: 91
First factor: 7
```

Equivalence of for,while loops

- Any `for` loop of the following form:

```
for (<initialization>; <condition>; <update>) {
    <statement(s)>;
}
```

can be replaced by a `while` loop of the following form:

```
<initialization>;
while (<condition>) {
    <statement(s)>;
    <update>;
}
```

for/while loop example

- What `while` loop is essentially equivalent to the following `for` loop?

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

- ANSWER:

```
int i = 1;
while (i <= 10) {
    System.out.println(i + " squared = " + (i * i));
    i++;
}
```

While loop question

- Write a piece of Java code that uses a `while` loop to repeatedly prompt the user to type a number until the user types a non-negative number, then square it.

  - Example log of execution:

```
Type a non-negative integer: -5
Invalid number, try again: -1
Invalid number, try again: -235
Invalid number, try again: -87
Invalid number, try again: 11
11 squared is 121
```

While loop answer

- Solution:

```
System.out.print("Type a non-negative integer: ");
int number = console.nextInt();

while (number < 0) {
    System.out.print("Invalid number, try again: ");
    number = console.nextInt();
}

int square = number * number;
System.out.println(number + " squared is " + square);
```

- Notice that `number` has to be declared outside the loop in order to remain in scope.

While loop question

- Write a method named `digitSum` that accepts an integer as a parameter and returns the sum of the digits of that number.
  - `digitSum(29107)` returns `2+9+1+0+7` or `19`
  - Assume that the number is non-negative.
  - Hint: Use the `%` operator to extract the last digit of a number.
    - If we do this repeatedly, when should we stop?

While loop answer

- The following code implements the method:

```
public static int digitSum(int n) {
    int sum = 0;
    while (n > 0) {
        sum = sum + (n % 10); // add last digit to sum
        n = n / 10;           // remove last digit
    }
    return sum;
}
```

Sentinel loops

reading: 5.1

Sentinel values

- **sentinel**: A special input value that signals the end of the user's input.

- **sentinel loop**: Repeats until a sentinel value is seen.
  - Example: Write a program that repeatedly prompts the user for numbers to add until the user types 0, then outputs their sum.
    (In this case, 0 is our sentinel value.)

  - Example log of execution:

```
Enter a number (0 to quit): 95
Enter a number (0 to quit): 87
Enter a number (0 to quit): 42
Enter a number (0 to quit): 26
Enter a number (0 to quit): 0
The total was 250
```

Flawed sentinel solution

- What's wrong with this solution?

```
Scanner console = new Scanner(System.in);
int sum = 0;
int inputNumber = 1;   // "dummy value", anything but 0

while (inputNumber != 0) {
    System.out.print("Enter a number (0 to quit): ");
    inputNumber = console.nextInt();
    sum += inputNumber;
}

System.out.println("The total was " + sum);
```

A different sentinel value

- Modify your program to use a sentinel value of **-1**.
  - Example log of execution:

```
Enter a number (-1 to quit): 95
Enter a number (-1 to quit): 87
Enter a number (-1 to quit): 42
Enter a number (-1 to quit): 26
Enter a number (-1 to quit): -1
The total was 250
```

Changing the sentinel value

- To see the problem, change the sentinel's value to -1:

```
Scanner console = new Scanner(System.in);
int sum = 0;
int inputNumber = 1;  // "dummy value", anything but -1

while (inputNumber != -1) {
    System.out.print("Enter a number (-1 to quit): ");
    inputNumber = console.nextInt();
    sum += inputNumber;
}

System.out.println("The total was " + sum);
```

- Now the solution produces the wrong output. Why?

```
The total was 249
```

The problem with our code

- Our code uses a pattern like this:

  sum = 0.

  while (input is not the sentinel) {

      prompt for input; read input.

      add input to the sum.

  }

- On the last pass through the loop, the sentinel value -1 is added to the sum:

      prompt for input; read input (-1).

      add input (-1) to the sum.

- This is a fencepost problem.
  - We want to read *N* numbers (*N* is not known ahead of time), but only sum the first *N* - 1 of them.

A fencepost solution

- We need the code to use a pattern like this:

  sum = 0.
  prompt for input; read input.

  while (input is not the sentinel) {
      add input to the sum.
      prompt for input; read input.
  }

- Sentinel loops often utilize a fencepost-style "loop-and-a-half" solution by pulling some code out of the loop.

Correct code

- This solution produces the correct output:

```
Scanner console = new Scanner(System.in);
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int inputNumber = console.nextInt();

while (inputNumber != -1) {
    sum += inputNumber;        // moved to top of loop
    System.out.print("Enter a number (-1 to quit): ");
    inputNumber = console.nextInt();
}

System.out.println("The total was " + sum);
```

Constant with sentinel

- A better solution creates a constant for the sentinel:

```
public static final int SENTINEL = -1;
```

- This solution uses the constant:

```
Scanner console = new Scanner(System.in);
int sum = 0;
System.out.print("Enter a number (" + SENTINEL + " to quit): ");
int inputNumber = console.nextInt();

while (inputNumber != SENTINEL) {
    sum += inputNumber;

    System.out.print("Enter a number (" + SENTINEL + " to quit): ");
    inputNumber = console.nextInt();
}

System.out.println("The total was " + sum);
```

Generating random numbers

reading: 5.1

The Random class

- Random objects generate pseudo-random numbers.
  - Class `Random` is found in the `java.util` package.

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

Method name	Description
`nextInt()`	returns a random integer
`nextInt(`*max*`)`	returns a random integer in the range [0, *max*) in other words, from 0 through one less than *max*
`nextDouble()`	returns a random real number in the range [0.0, 1.0)

  - Example:

```
Random rand = new Random();
int randomNumber = rand.nextInt(10);
// randomNumber has a random value between 0 and 9
```

Generating random numbers

- Common usage: to get a random number from 1 to *N*
  - Example: A random integer between 1 and 20, inclusive:

```
int n = rand.nextInt(20) + 1;
```

- To get a number in arbitrary range [*min*, max]:

  `nextInt(`***<size of the range>***`) + `***<min>***

  where ***<size of the range>*** equals ***<max>*** - ***<min>*** + 1

  - Example: A random integer between 5 and 10 inclusive:

```
int n = rand.nextInt(6) + 5;
```

Random questions

- Given the following declaration, how would you get:

```
Random rand = new Random();
```

  - A random number between 0 and 100 inclusive?
  - A random number between 1 and 100 inclusive?
  - A random number between 4 and 17 inclusive?

Random answers

- Given the following declaration, how would you get:

```
Random rand = new Random();
```

  - A random number between 0 and 100 inclusive?

```
int random1 = rand.nextInt(101);
```

  - A random number between 1 and 100 inclusive?

```
int random2 = rand.nextInt(100) + 1;
```

  - A random number between 4 and 17 inclusive?

```
int random3 = rand.nextInt(14) + 4;
```

Random question

- Write a program that simulates rolling of two six-sided dice until their combined result comes up as 7.

  - Example log of execution:

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 tries!
```

Random answer

```
// Rolls two dice until a sum of 7 is reached.
import java.util.*;

public class Roll {
    public static void main(String[] args) {
        Random rand = new Random();
        int sum = 0;
        int tries = 0;
        while (sum != 7) {
            int roll1 = rand.nextInt(6) + 1;
            int roll2 = rand.nextInt(6) + 1;
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        }

        System.out.println("You won after " + tries + " tries!");
    }
}
```

Random/while question

- Write a multiplication tutor program. Example log of execution:

```
This program helps you practice multiplication
by asking you random multiplication questions
with numbers ranging from 1 to 20
and counting how many you solve correctly.

14 * 8 = 112
Correct!
5 * 12 = 60
Correct!
8 * 3 = 24
Correct!
5 * 5 = 25
Correct!
20 * 14 = 280
Correct!
19 * 14 = 256
Incorrect; the answer was 266
You solved 5 correctly.
```

Random/while answer

```
// Asks the user to do multiplication problems and scores them.
import java.util.*;

public class MultTutor {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);
        Random rand = new Random();
        int num1 = 0;
        int num2 = 0;
        int guess = 0;
        int correct = 0;

        // loop until user gets one wrong
        while (guess == num1 * num2) {
            // pick two random numbers between 1 and 20 inclusive
            num1 = rand.nextInt(20) + 1;
            num2 = rand.nextInt(20) + 1;

            System.out.print(num1 + " * " + num2 + " = ");
            int guess = console.nextInt();
            if (guess == num1 * num2) {
                System.out.println("Correct!");
            } else {
                System.out.println("Incorrect; the answer was " + (num1 * num2));
            }
        }

        System.out.println("You solved " + correct + " correctly.");
    }
```

Random/while answer 2

```
    ...

    public static void introduction() {
        System.out.println("This program helps you practice multiplication");
        System.out.println("by asking you random multiplication questions");
        System.out.println("with numbers ranging from 1 to 20");
        System.out.println("and counting how many you solve correctly.");
        System.out.println();
    }
}
```

- Consider changing the code to use a class constant for the maximum value of 20.

Random text and others

- Random can be used in text processing.
  - Code to pick a random lowercase letter:
    ```
    char letter = (char) ('a' + rand.nextInt(26));
    ```
  - Code to pick a random character from a string (in this case, a random vowel):
    ```
    String vowels = "aeiou";
    char vow = vowels.charAt(rand.nextInt(vowels.length()));
    ```
  - Another example: code to pick a random letter representing a base in a DNA strand (A, C, G, or T):
    ```
    String bases = "ACGT";
    char base = bases.charAt(rand.nextInt(bases.length()));
    ```

Other random values

- Random can be used with `double`
  - `nextDouble` method returns a `double` between 0.0 and 1.0
  - To get a double in a different range, multiply and/or add
  - Example: Gets a random value between 1.5 and 4.0:

    ```
    double randomGpa = rand.nextDouble() * 2.5 + 1.0;
    ```

- Random can be used to pick between arbitrary choices
  - Code to pick a red, green, or blue color:

    ```
    int r = rand.nextInt(3);
    if (r == 0) {
        g.setColor(Color.RED);
    } else if (r == 1) {
        g.setColor(Color.GREEN);
    } else {
        g.setColor(Color.BLUE);
    }
    ```

Boolean logic

reading: 5.2

Type boolean

- **`boolean`**: A primitive type to represent logical values.
  - A `boolean` expression produces either `true` or `false`.
  - The ***<condition>***s in `if` statements, `for` loops are `boolean`.

  - Examples:

```
boolean minor = (age < 21);
boolean expensive = (iPhonePrice > 500.00);
boolean iLoveCS = true;

if (minor) {
    System.out.println("Can't purchase alcohol!");
}
```

  - You can create `boolean` variables, pass `boolean` parameters, return `boolean` values from methods, ...

Methods that return boolean

- There are methods in Java that return `boolean` values.
  - A call to one of these methods can be used as a ***<condition>*** in a `for` loop, `while` loop, or `if` statement.

  - Examples:

```
Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String line = console.nextLine();

if (line.startsWith("Dr.")) {
    System.out.println("Will you marry me?");
} else if (line.endsWith(", Esq.")) {
    System.out.println("And I am Ted 'Theodore' Logan!");
}
```

String boolean methods

- The following `String` methods return `boolean` values:

Method	Description
`equals(`*`String`*`)`	whether two strings contain exactly the same characters
`equalsIgnoreCase(`*`String`*`)`	whether two strings contain the same characters, ignoring upper vs. lower case differences
`startsWith(`*`String`*`)`	whether one string contains the other's characters at its start
`endsWith(`*`String`*`)`	whether one string contains the other's characters at its end

Writing boolean methods

- Methods can return a `boolean` result.

```
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```

- Calls to such methods can be used as conditions:

```
if (bothOdd(7, 13)) {
    ...
}
```

Writing boolean methods 2

- Another example that returns a `boolean` result:

```
public static boolean isLowerCaseLetter(char ch) {
    if ('a' <= ch && ch <= 'z') {
        return true;
    } else {
        return false;
    }
}
```

- Example call to this method:

```
String name = "e.e. cummings";
char firstLetter = name.charAt(0);
if (isLowerCaseLetter(firstLetter)) {
    System.out.println("You forgot to capitalize your name!");
}
```

"Boolean Zen"

- Methods that return a `boolean` result often have an `if/else` statement:

```
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```

- ... but the `if/else` is sometimes unnecessary.
  - The `if/else`'s condition is itself a `boolean` expression; its value is exactly what you want to return. So do that!

```
public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```

"Boolean Zen" template

- Replace:

```
public static boolean <name>(<parameters>) {
    if (<condition>) {
        return true;
    } else {
        return false;
    }
}
```

- with:

```
public static boolean <name>(<parameters>) {
    return <condition>;
}
```

Boolean practice problems

- Write a method named `isVowel` that returns whether a particular character is a vowel (a, e, i, o, or u). Count only lowercase vowels.
  - `isVowel('q')` returns `false`
  - `isVowel('e')` returns `true`

- Write a method named `allDigitsOdd` that returns whether every digit of a positive integer is odd.
  - `allDigitsOdd(19351)` returns `true`
  - `allDigitsOdd(234)` returns `false`

- Write a method named `countVowels` that returns the number of lowercase vowels in a `String`.
  - `countVowels("zelda")` returns 2
  - `countVowels("E Pluribus Unum")` returns 4

Boolean practice solutions

```
public static boolean isVowel(char c) {
    if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
        return true;
    } else {
        return false;
    }
}
```

or:

```
public static boolean isVowel(char c) {
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}

public static boolean allDigitsOdd(int n) {
    while (n > 0) {
        if (n % 2 == 0) {
            return false;
        }
        n /= 2;
    }
    return true;
}
```

Boolean practice solutions

```
public static int countVowels(String s) {
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
            count++;
        }
    }
    return count;
}
```

or:

```
public static int countVowels(String s) {
    int count = 0;
    for (int i = 0; i < s.length(); i++) {
        if (isVowel(s.charAt(i))) {
            count++;
        }
    }
    return count;
}
```

Boolean question

- Modify your previous multiplication tutor program to use a static method that returns a `boolean` value.

```
This program helps you practice multiplication
by asking you random multiplication questions
with numbers ranging from 1 to 20
and counting how many you solve correctly.

14 * 8 = 112
Correct!
5 * 12 = 60
Correct!
8 * 3 = 24
Correct!
5 * 5 = 25
Correct!
20 * 14 = 280
Correct!
19 * 14 = 256
Incorrect; the answer was 266
You solved 5 correctly.
```

Boolean answer

```
import java.util.*;

// Asks the user to do multiplication problems and scores them.
public class MultTutor {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);
        Random rand = new Random();

        // loop until user gets one wrong
        int correct = 0;
        while (askQuestion(console, rand)) {
            correct++;
        }

        System.out.println("You solved " + correct + " correctly.");
    }

    public static void introduction() {
        System.out.println("This program helps you practice multiplication");
        System.out.println("by asking you random multiplication questions");
        System.out.println("with numbers ranging from 1 to 20");
        System.out.println("and counting how many you solve correctly.");
        System.out.println();
    }

    ...
```

Boolean answer 2

```
    ...

    public static boolean askQuestion(Scanner console, Random rand) {
        // pick two random numbers between 1 and 20 inclusive
        int num1 = rand.nextInt(20) + 1;
        int num2 = rand.nextInt(20) + 1;

        System.out.print(num1 + " * " + num2 + " = ");
        int guess = console.nextInt();
        if (guess == num1 * num2) {
            System.out.println("Correct!");
            return true;
        } else {
            System.out.println("Incorrect; the correct answer was " +
                               (num1 * num2));
            return false;
        }
    }
}
```

Boolean practice problem

- Write a program that compares two words typed by the user to see whether they "*rhyme*" (end with the same last two letters) and/or *alliterate* (begin with the same letter).
  - Use methods with return values to tell whether two words rhyme and/or alliterate.
  - Example logs of execution:

    (run #1)
    ```
    Type two words: car STAR
    They rhyme!
    ```

    (run #2)
    ```
    Type two words: bare bear
    They alliterate!
    ```

    (run #3)
    ```
    Type two words: sell shell
    They alliterate!
    They rhyme!
    ```

Boolean practice solution

```
// Determines whether two words rhyme and/or start with the same letter.
import java.util.*;

public class Rhyme {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Type two words: ");
        String word1 = console.next();
        String word2 = console.next();

        if (rhyme(word1, word2)) {
            System.out.println("They rhyme!");
        }

        if (alliterate(word1, word2)) {
            System.out.println("They alliterate!");
        }
    }

    // Returns true if s1 and s2 end with the same two letters.
    public static boolean rhyme(String s1, String s2) {
        String endOfWord2 = s2.substring(s2.length() - 2);
        return s1.endsWith(endOfWord2);
    }

    // Returns true if s1 and s2 start with the same letter.
    public static boolean alliterate(String s1, String s2) {
        return s1.startsWith(s2.substring(0, 1));
    }
}
```

Boolean practice problem

- Write a program that prompts for a number and tells whether it is prime, and if not, prints the next prime.
  - Example logs of execution: (run #1)

```
Type a number: 29
29 is prime
```

(run #2)

```
Type two numbers: 14
14 is not prime; the next prime after 14 is 17
```

- As part of your solution, write two methods:
  - `isPrime`: Returns `true` if the parameter passed is a prime number
  - `nextPrime`: Returns the next prime number whose value is greater than or equal to the parameter passed. (If the parameter passed is prime, returns that number.)

Boolean practice problem

- Modify your previous program so that it reads two numbers and tells whether each is prime, or if not, gives the next prime after them.
  - Also report whether the two numbers are *relatively prime* (have no common factors).
  - Example logs of execution: (run #1)

```
Type two numbers: 9 16
9 is not prime; the next prime after 9 is 11
16 is not prime; the next prime after 16 is 17
9 and 16 are relatively prime
```

(run #2)

```
Type two numbers: 7 21
7 is prime
21 is not prime; the next prime after 21 is 23
7 and 21 are not relatively prime
```

Boolean practice solution

```
import java.util.*;

public class Primes {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();

        primeTest(num1);
        primeTest(num2);
        if (relativelyPrime(num1, num2)) {
            System.out.println(num1 + " and " + num2 + " are relatively prime");
        } else {
            System.out.println(num1 + " and " + num2 +
                    " are not relatively prime");
        }
    }

    public static void primeTest(int number) {
        if (isPrime(number)) {
            System.out.println(number + " is prime");
        } else {
            System.out.println("the next prime after " + number +
                    " is " + nextPrime(number));
        }
    }

    public static boolean isPrime(int number) {
        return countFactors(number) <= 2;
    }
    ...
```

Boolean practice solution 2

```
...
public static int nextPrime(int number) {
    while (countFactors(number) > 2) {
        number++;
    }
    return number;   // number is now prime
}

public static int countFactors(int number) {
    int count = 1;
    for (int i = 2; i <= number; i++) {
        if (number % i == 0) {
            count++;   // i is a factor
        }
    }
    return count;
}

public static boolean relativelyPrime(int n1, int n2) {
    return commonFactors(n1, n2) == 1;
}

// Returns the number of
public static int commonFactors(int n1, int n2) {
    int count = 1;
    for (int i = 2; i <= Math.min(n1, n2); i++) {
        if (n1 % i == 0 && n2 % i == 0) {
            count++;   // i is a common factor
        }
    }
    return count;
}
}
```

Indefinite loop variations

reading: 5.4

The do/while loop

- **`do/while` loop**: Executes statements repeatedly while a condition is `true`, testing it at the end of each repetition.
  - Similar to a `while` loop, except that its body statement(s) will always execute the first time, regardless of whether the condition is `true` or `false`.

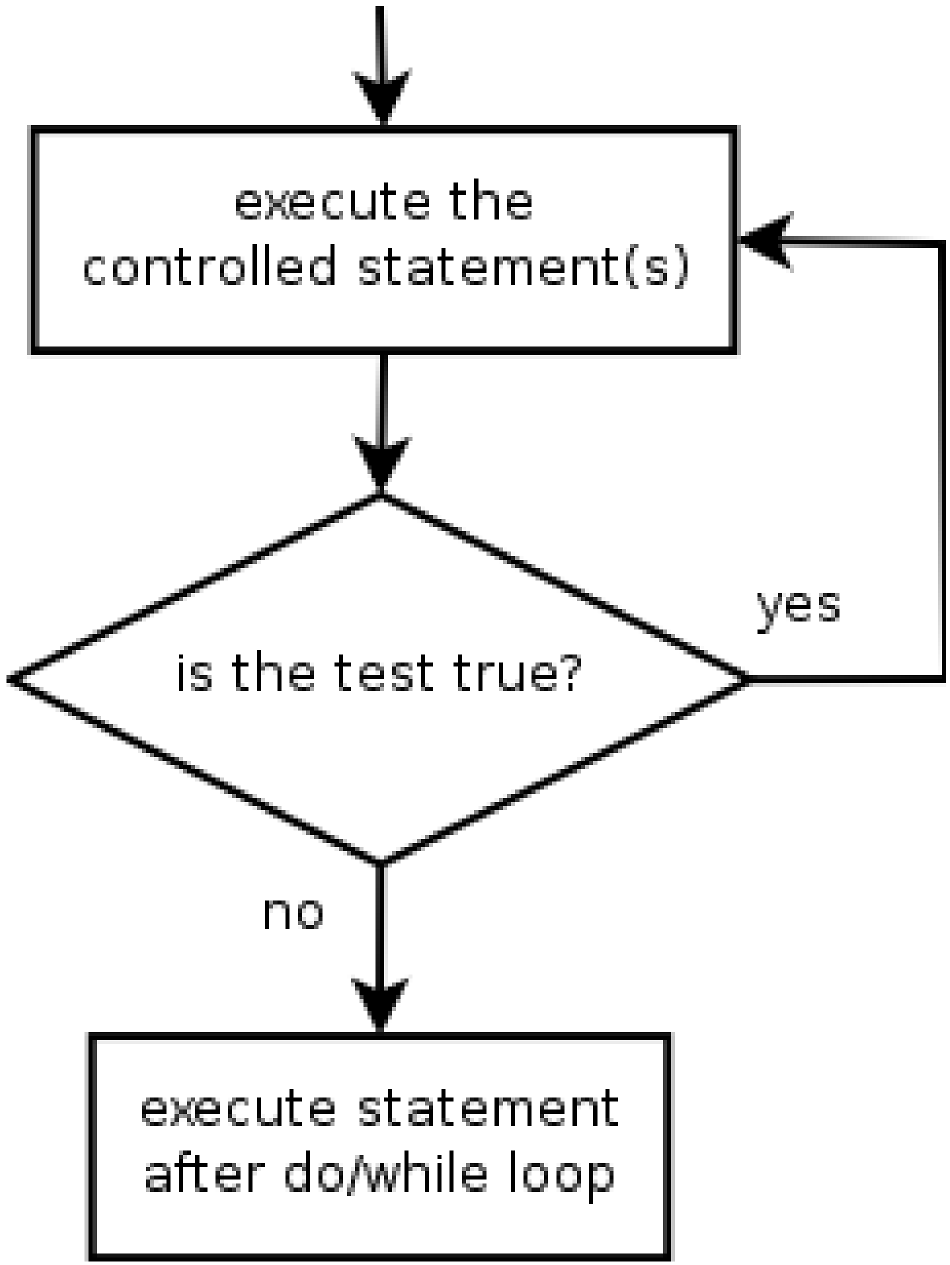
- The `do/while` loop, general syntax:

```
do {
    <statement(s)> ;
} while (<condition>);
```

  - Example:

```
// roll until we get a number other than 3
Random rand = new Random();
int dice;
do {
    dice = rand.nextInt();
} while (dice == 3);
```

do/while loop flow chart



do/while question

- Modify the previous dice program to use a `do/while` loop.

  - Example log of execution:

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 tries!
```

do/while solution

```
// Rolls two dice until a sum of 7 is reached.
import java.util.*;

public class Roll {
    public static void main(String[] args) {
        Random rand = new Random();
        int tries = 0;
        int sum;
        do {
            int roll1 = rand.nextInt(6) + 1;
            int roll2 = rand.nextInt(6) + 1;
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        } while (sum != 7);

        System.out.println("You won after " + tries + " tries!");
    }
}
```

"Forever" loop with break

- **`break` statement**: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - Such loops are often called *"forever" loops* because their header's boolean test is often changed to a trivial `true`.

- "forever" loop, general syntax:

```
while (true) {
    <statement(s)> ;

    if (<condition>) {
        break;
    }

    <statement(s)> ;
}
```

Sentinel loop with break

- A working sentinel loop solution using `break`:

```
Scanner console = new Scanner(System.in);
int sum = 0;
while (true) {
    System.out.print("Enter a number (-1 to quit): ");
    int inputNumber = console.nextInt();
    if (inputNumber == -1) {  // don't add -1 to sum
        break;
    }
    sum += inputNumber;         // inputNumber != -1 here
}

System.out.println("The total was " + sum);
```

Assertions

reading: 5.5

Logical assertions

- **assertion**: A statement that is either true or false.

  Examples:
  - Java was created in 1995.
  - The sky is purple.
  - 23 is a prime number.
  - 10 is greater than 20.
  - x divided by 2 equals 7. *(depends on the value of x)*

Assertions in code

- We can make assertions about our code and ask whether they are true at various points in the code.
  - Valid answers are ALWAYS, NEVER, or SOMETIMES.

```
System.out.print("Type a nonnegative number: ");
double number = console.nextDouble();
// Point A: is number < 0.0 here?       (SOMETIMES)

while (number < 0.0) {
    // Point B: is number < 0.0 here?   (ALWAYS)
    System.out.print("Negative; try again: ");

    number = console.nextDouble();
    // Point C: is number < 0.0 here?   (SOMETIMES)
}

// Point D: is number < 0.0 here?       (NEVER)
```

Assertion example 1

```
public static int mystery(Scanner console) {
    int prev = 0;
    int count = 0;
    int next = console.nextInt();
    // Point A
    while (next != 0) {
        // Point B
        if (next == prev) {
            // Point C
            count++;
        }
        prev = next;
        next = console.nextInt();
        // Point D
    }
    // Point E
    return count;
}
```

Which of the following assertions are true at which point(s) in the code?
Choose ALWAYS, NEVER, or SOMETIMES.

	next == 0	prev == 0	next == prev
Point A	SOMETIMES	ALWAYS	SOMETIMES
Point B	NEVER	SOMETIMES	SOMETIMES
Point C	NEVER	NEVER	ALWAYS
Point D	SOMETIMES	NEVER	SOMETIMES
Point E	ALWAYS	SOMETIMES	SOMETIMES

Assertion example 2

```
public static void mystery(int x, int y) {
    int z = 0;

    // Point A
    while (x >= y) {
        // Point B
        x -= y;

        // Point C
        z++;

        // Point D
    }

    // Point E
    System.out.println(
        z + " " + x);
}
```

Which of the following assertions are true at which point(s) in the code?
Choose ALWAYS, NEVER, or SOMETIMES.

	x < y	x == y	z == 0
Point A	SOMETIMES	SOMETIMES	ALWAYS
Point B	NEVER	SOMETIMES	SOMETIMES
Point C	SOMETIMES	SOMETIMES	SOMETIMES
Point D	SOMETIMES	SOMETIMES	NEVER
Point E	ALWAYS	NEVER	SOMETIMES

Assertion example 3

```
// pre : y >= 0,   post: returns x^y
public static int pow(int x, int y) {
    int prod = 1;

    // Point A
    while (y > 0) {
        // Point B
        if (y % 2 == 0) {
            // Point C
            x *= x;
            y /= 2;

            // Point D
        } else {
            // Point E
            prod *= x;
            y--;
            // Point F
         }
         // Point G
     }
     // Point H
     return prod;
 }
```

Which of the following assertions are true at which point(s) in the code?
Choose ALWAYS, NEVER, or SOMETIMES.

	y == 0	y % 2 == 0
Point A	SOMETIMES	SOMETIMES
Point B	NEVER	SOMETIMES
Point C	NEVER	ALWAYS
Point D	NEVER	SOMETIMES
Point E	NEVER	NEVER
Point F	SOMETIMES	ALWAYS
Point G	SOMETIMES	SOMETIMES
Point H	ALWAYS	ALWAYS