Building Java Programs

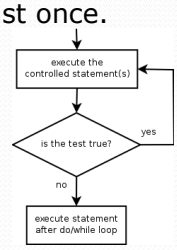
Chapter 5
Lecture 5-4: `do/while` loops, assertions

reading: 5.1, 5.5

The `do/while` loop

- **`do/while` loop**: Performs its test at the *end* of each repetition.
  - Guarantees that the loop's `{}` body will run at least once.

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




```
// Example: prompt until correct password is typed
String phrase;
do {
    System.out.print("Type your password: ");
    phrase = console.next();
} while (!phrase.equals("abracadabra"));
```

do/while question

- Modify the previous `Dice` program to use `do/while`.

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


do/while answer

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

public class Dice {
    public static void main(String[] args) {
        Random rand = new Random();
        int tries = 0;
        int sum;

        do {
            int roll1 = rand.nextInt(6) + 1;   // one roll
            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!");
    }
}
```

break

- **`break` statement**: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - The loop's test is often changed to `true` ("always repeat").

```
while (true) {
    <statement(s)>;
    if (<test>) {
        break;
    }
    <statement(s)>;
}
```

  - `break` is considered to be bad style by some programmers.

5

Sentinel loop with `break`

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

6

Assertions

reading: 5.5

7

Logical assertions

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

  Examples:
  - Java was created in 1995.
  - The sky is purple.
  - The capital of North Dakota is Bismarck.
  - Mr. Marty met a monkey.
  - x divided by 2 equals 7. *(depends on the value of x)*

- An assertion might be false ("The sky is purple" above), but it is still an assertion because it is a true/false statement.

8

Reasoning about assertions

- Suppose you have the following code:

```
if (x > 3) {
    // Point A
    x--;
} else {
    // Point B
    x++;
    // Point C
}
// Point D
```

- What do you know about `x`'s value at the three points?
  - Is `x > 3`? Always? Sometimes? Never?

9

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)
```

10

Reasoning about assertions

- Right after a variable is initialized, its value is known:

```
int x = 3;
// is x > 0?  ALWAYS
```

- In general you know nothing about parameters' values:

```
public static void mystery(int a, int b) {
// is a == 10?  SOMETIMES
```

- But inside an `if`, `while`, etc., you may know something:

```
public static void mystery(int a, int b) {
    if (a < 0) {
        // is a == 10?  NEVER
        ...
    }
}
```

11

Assertions and loops

- At the start of a loop's body, the loop's test must be `true`:

```
while (y < 10) {
    // is y < 10?  ALWAYS
    ...
}
```

- After a loop, the loop's test must be `false`:

```
while (y < 10) {
    ...
}
// is y < 10?  NEVER
```

- Inside a loop's body, the loop's test may become `false`:

```
while (y < 10) {
    y++;
    // is y < 10?  SOMETIMES
}
```

12

"Sometimes"

- Things that cause a variable's value to be unknown (often leads to "sometimes" answers):
  - reading from a `Scanner`
  - reading a number from a `Random` object
  - a parameter's initial value to a method
- If you can reach a part of the program both with the answer being "yes" and the answer being "no", then the correct answer is "sometimes".

Assertion example 1

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

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

        if (x != y) {
            // Point C
            z = z * 2;
        }

        // Point D

    }

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

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	NEVER	NEVER
Point D	SOMETIMES	SOMETIMES	NEVER
Point E	ALWAYS	NEVER	SOMETIMES

Assertion example 2

```
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 3

```
// Assumes y >= 0, and 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 * x;
            y = y / 2;
            // Point D
        } else {
            // Point E
            prod = prod * x;
            y--;
            // Point F
        }
    }
    // Point G
    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	ALWAYS	SOMETIMES
Point C	ALWAYS	ALWAYS
Point D	ALWAYS	SOMETIMES
Point E	ALWAYS	NEVER
Point F	SOMETIMES	ALWAYS
Point G	NEVER	ALWAYS