Midterm announcements

- Next week on Friday May 8
- Must bring an ID
- Open book, open notes, closed electronics
- Must attend correct section unless you fill out web-site form in advance
- Sample exam(s) posted Friday
  - Another sample in section next week
  - And problems on Practice-It!
- Exam will have 2-3 programming problems
  - harder, after other problems
  - See sample exam(s)
- Review session next week
  - Time/place to-be-determined
  - probably Thursday late afternoon

Building Java Programs

Chapter 5
Lecture 5-3: Assertions, `do/while` loops

reading: 5.4 - 5.5
self-check: 22-24, 26-28

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)*

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

Reasoning about assertions

- Suppose you have the following code:

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

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

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)
```

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
          ...
      }
  }
  ```

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
    ...
}
```

- Immediately 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
}
```

More on loops

- Remember that a loop might execute 0 or more times

```
public static void m(int y) {
 int x = 0;
 while (y < 10) {
     ++x;
     ... // no other changes to x
 }
 // is x > 0?   SOMETIMES
}
```

8

"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".

- If you're unsure, "Sometimes" is a good guess.

Perspective

- Assertions are a great way to think about your program
  - And what all our language constructs are actually good at

- Purpose of assignments: change whether assertions hold
- Purpose of tests: learn more about what assertions hold
- Purpose of ifs/loops: have different code points for different possibilities

"If I get here, then x must be less than y, so it's okay to…"

- (Plus, it's on the midterm)

Assertion example 0

```java
public static void mystery(int x) {
    int y = 10;

    // Point A
    while (x < y) {
        // Point B
        x++;
        // Point C
    }

    // Point D
    System.out.println(y);
}
```

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

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

Assertion example 1

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

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

        // Point C
        z++;

        // 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	SOMETIMES	SOMETIMES
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

13

Asserlion 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

Another non-useless example

```
/* This method program prompts the user for numbers until -1 is
   typed, then returns the largest number typed (or -1 if
   that was the only number typed). */

   public static int biggest(Scanner console) {
     System.out.print("Type a number (or -1 to quit): ");
     int number = console.nextInt();
     int max = number;
     // max >= number: ALWAYS
     while (number != -1) {
       // max >= number: SOMETIMES
       if (number > max) {
         max = number;
       }
       // max >= number: ALWAYS
       System.out.print("Type a number (or -1 to quit): ");
       number = console.nextInt();
     }
     // max >= number: SOMETIMES (!)
     return max;
   }
}
```

Another non-useless example

```java
/* Prompts the user for numbers until -1 is typed. Returns the
   largest positive number typed. Requires at least one positive.*/
  public static int biggest(Scanner console) {
    int number = -1;
    int max = -1;
    while(number <= 0) {
      System.out.print("Type a positive number: ");
      number = console.nextInt();
    }
    max = number;
    // max >= number: ALWAYS, max > 0: ALWAYS
    while (number != -1) {
      // max >= number: SOMETIMES, max > 0: ALWAYS
      if (number > max) {
        max = number;
      }
      // max >= number: ALWAYS, max > 0: ALWAYS
      System.out.print("Type a number (or -1 to quit): ");
      number = console.nextInt();
    }
    // max >= number: ALWAYS, max > 0: ALWAYS
    return max;
  }
}
```

16

`while` loop variations

reading: 5.4
self-checks: #22-24
exercises: #6

The `do/while` loop

- **`do/while` loop**: Executes statements repeatedly while a condition is `true`, testing it at the *end* of each repetition.
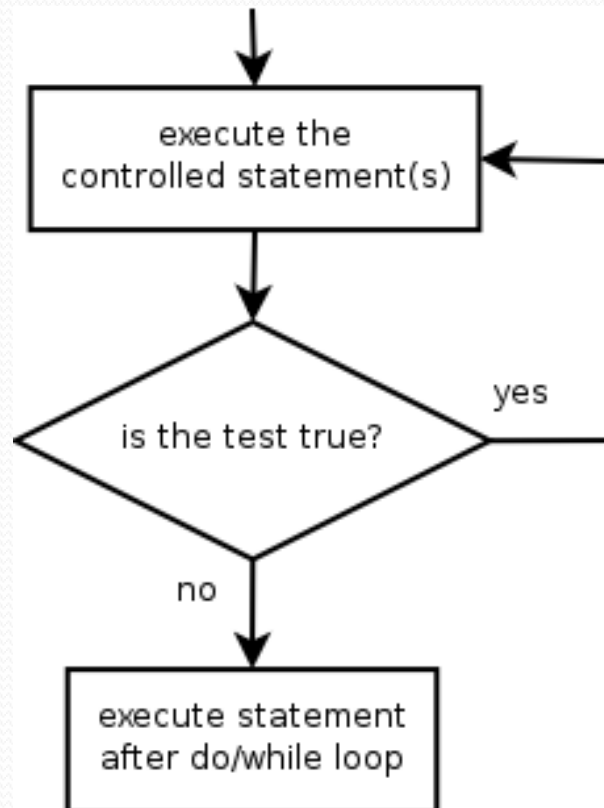
  ```
  do {
      statement(s);
  } while (test);
  ```

  - Example:

    ```
    // prompt until the user gets the right password
    String phrase;
    do {
        System.out.print("Password: ");
        phrase = console.next();
    } while (!phrase.equals("abracadabra"));
    ```

do/while flow chart

- How does this differ from the while loop?
  - The controlled **statement(s)** will always execute the first time, regardless of whether the **test** is `true` or `false`.

Thoughts on do/while

- Not used very often; optional in 142

- Affects assertions:
  - body always executes at least once
  - body executes once before test

```
public static void m(int y) {
  int x = 0;
  do {
    // is y < 10? SOMETIMES
    ++x;
    ... // no other changes to x
  } while(y < 10);
  // is x > 0?  ALWAYS
}
```

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`.

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

        if (test) {
            break;
        }

        statement(s);
    }
    ```

  - `break` is often bad style! *Do not use it on CSE 142 homework!*

Sentinel loop with `break`

- A working sentinel loop solution using `break`:

```java
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);
```

Thoughts on break

- Literal meaning is go to after the loop *right now*

- Affects assertions: No longer know the loop test is false right after the loop

```
public static void m(int y, int x) {
    while(y < 10) {
      if(y==x) {
        break;
      }
      ++y;
    }
    // is y >= 10?   SOMETIMES
}
```

- Can also use `return` anywhere in a method
  - Returns *right now*