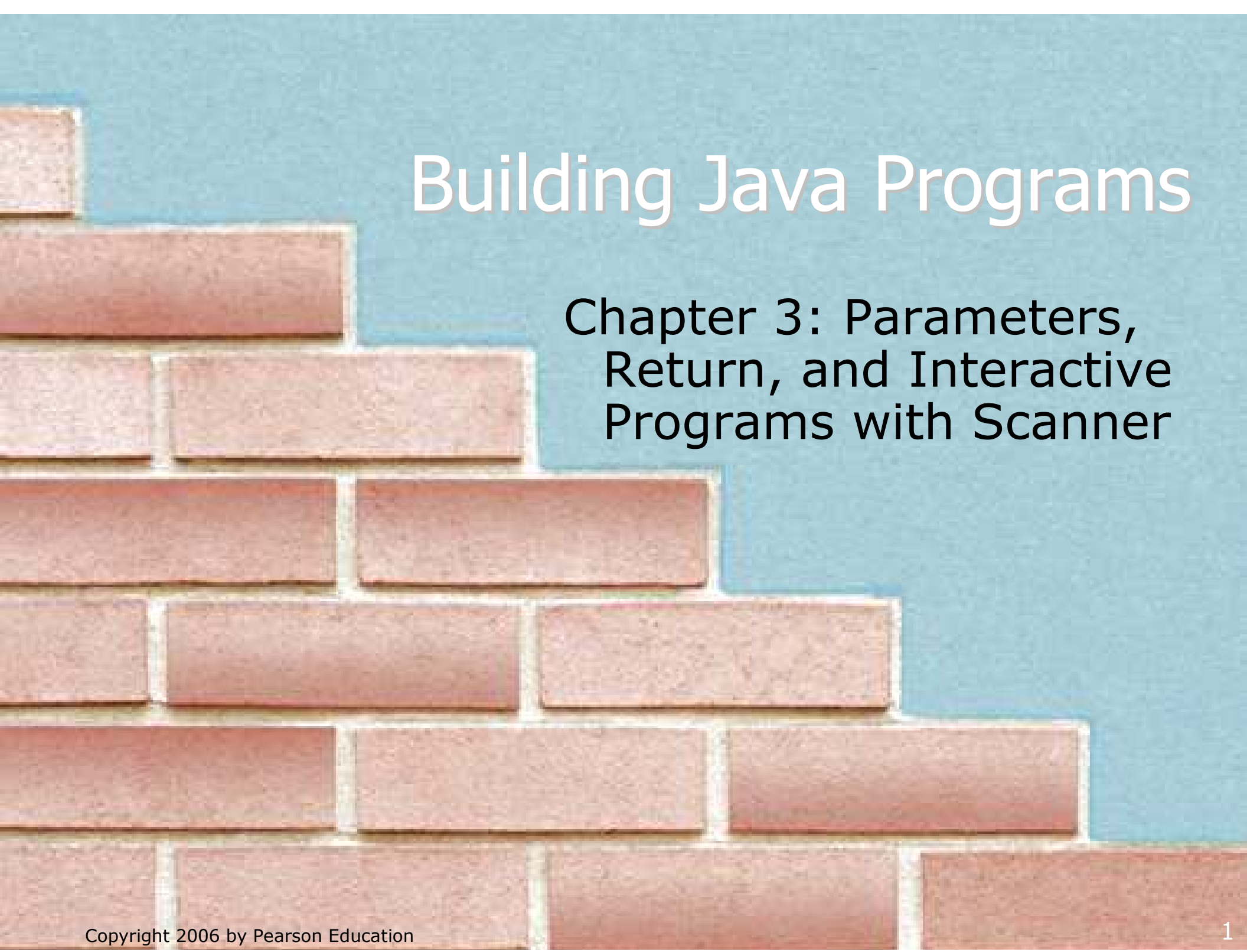
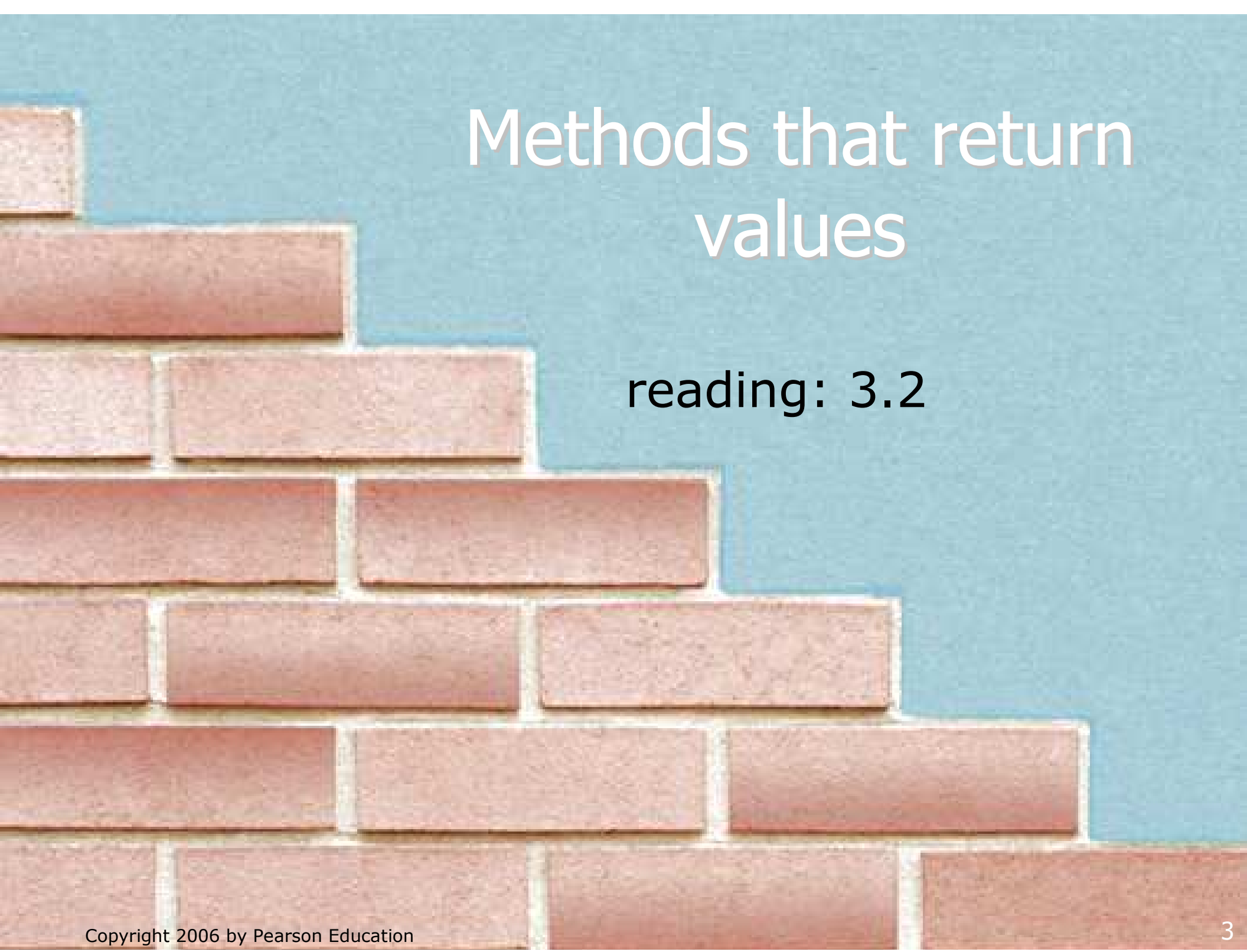


Lecture outline

- methods that return values
  - calling (e.g. the `Math` class)
  - writing

- cumulative sum



Java's `Math` class

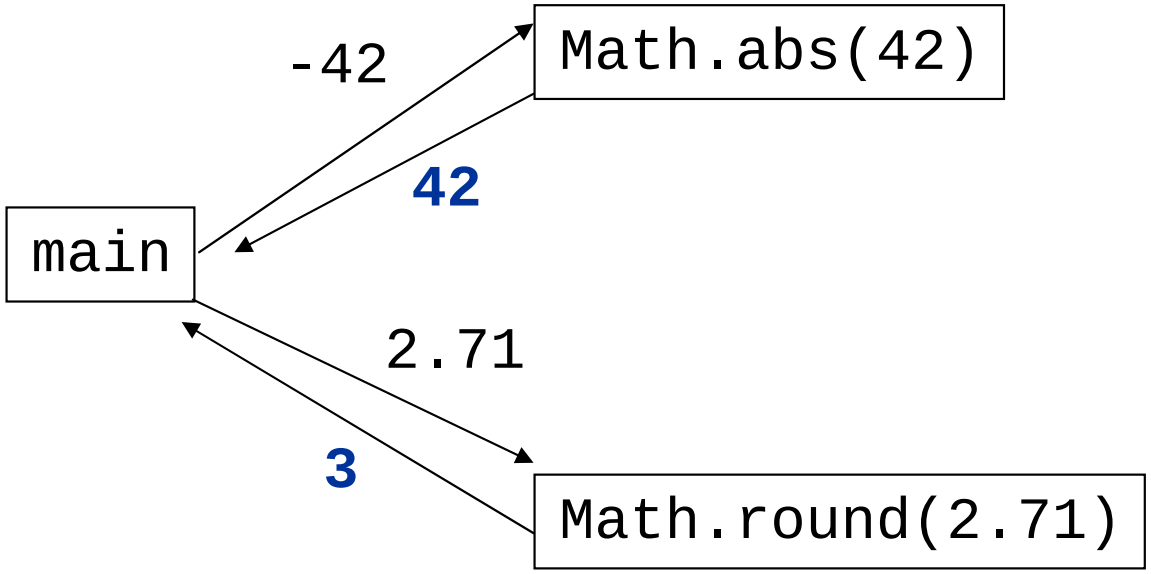
- Java has a class named `Math` with useful static methods and constants for performing calculations.

Method name	Description
`abs(`*value*`)`	absolute value
`ceil(`*value*`)`	rounds up
`cos(`*value*`)`	cosine, in radians
`floor(`*value*`)`	rounds down
`log(`*value*`)`	logarithm, base *e*
`log10(`*value*`)`	logarithm, base 10
`max(`*value1*`, `*value2*`)`	larger of two values
`min(`*value1*`, `*value2*`)`	smaller of two values
`pow(`*base*`, `*exponent*`)`	*base* to the *exponent* power
`random()`	random `double` between 0 and 1
`round(`*value*`)`	nearest whole number
`sin(`*value*`)`	sine, in radians
`sqrt(`*value*`)`	square root

Constant	Description
`E`	2.7182818...
`PI`	3.1415926...

Methods that return values

- **return**: To send a value out as the result of a method, which can be used in an expression.
  - A return is like the opposite of a parameter:
    - Parameters pass information *in* from the caller to the method.
    - Return values pass information *out* from a method to its caller.



  - The `Math` methods do not print results to the console.
    - Instead, each method evaluates to produce (or *return*) a numeric result, which can be used in an expression.

Math method examples

- `Math` method call syntax:

  `Math.` ***`<method name>`*** `(` ***`<parameter(s)>`*** `)`

- Examples:

```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);            // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);         // 50

System.out.println(Math.min(3, 7) + 2);    // 5
```

  - Notice that the preceding calls are used in expressions; they can be printed, stored into a variable, etc.

Math method questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

- `Math.max` and `Math.min` can be used to bound numbers.

  Consider an `int` variable named `age`.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Methods that return values

- Syntax for declaring a method that returns a value:

```
public static <type> <name> ( < parameter(s)> ) {
    < statement(s)> ;
    ...
    return <expression> ;
}
```

- Example:

```
// Returns the slope of the line between the given points.
public static double slope(int x1, int y1, int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx;
}
```

Return examples

```
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes length of triangle hypotenuse given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}

// Rounds the given number to two decimal places.
// Example: round(2.71828183) returns 2.72.
public static double round2(double value) {
    double result = value * 100.0;  // upscale the number
    result = Math.round(result);    // round to nearest integer
    result = result / 100.0;        // downscale the number
    return result;
}
```

Return examples shortened

```
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    return 5.0 / 9.0 * (degreesF - 32);
}


// Computes length of triangle hypotenuse given its side lengths.
public static double hypotenuse(int a, int b) {
    return Math.sqrt(a * a + b * b);
}


// Rounds the given number to two decimal places.
// Example: round(2.71828183) returns 2.72.
public static double round2(double value) {
    return Math.round(value * 100.0) / 100.0;
}
```

Return questions

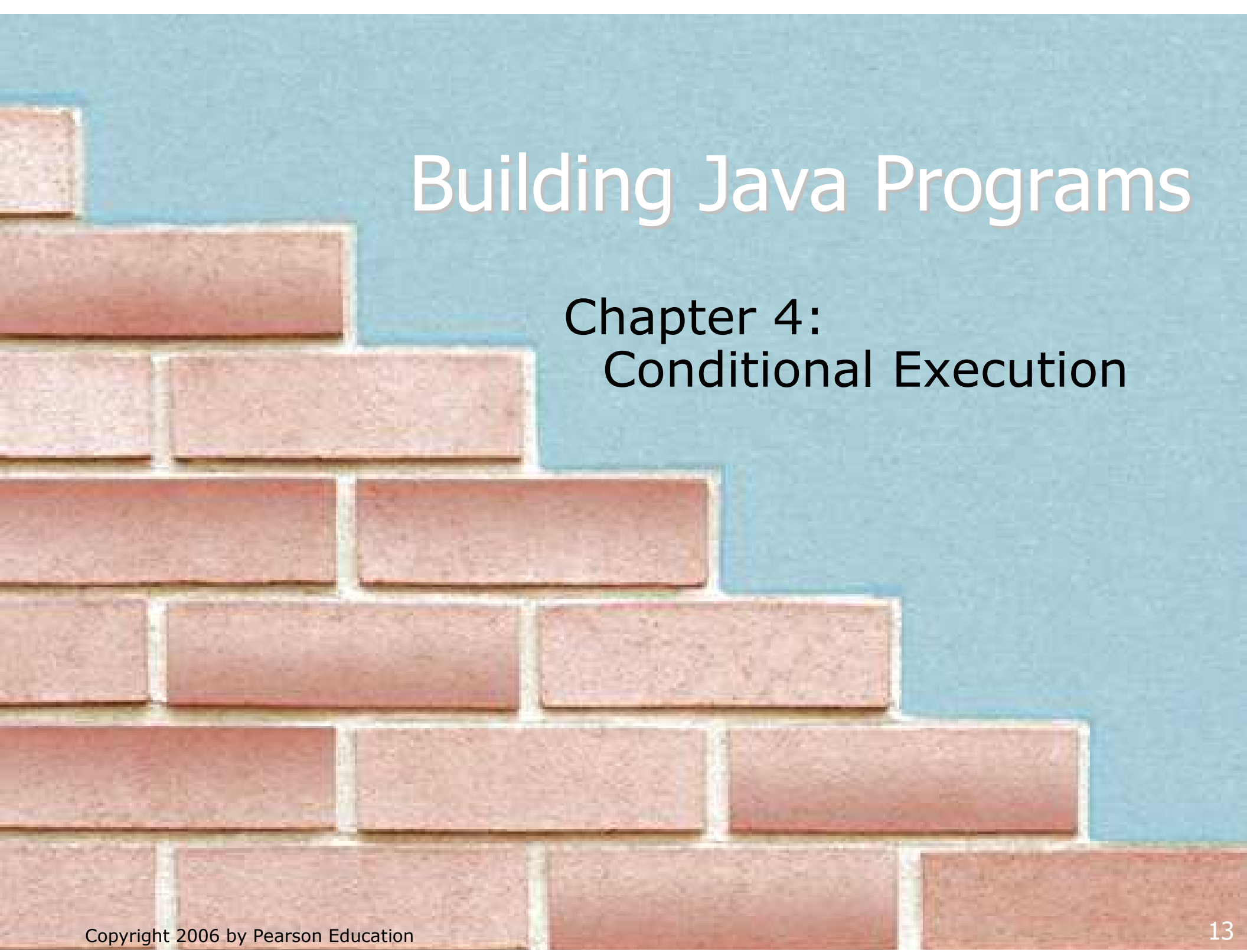
- Write a method named `area` that accepts a circle's radius as a parameter and returns its area.
  - You may wish to use the constant `Math.PI` in your solution.

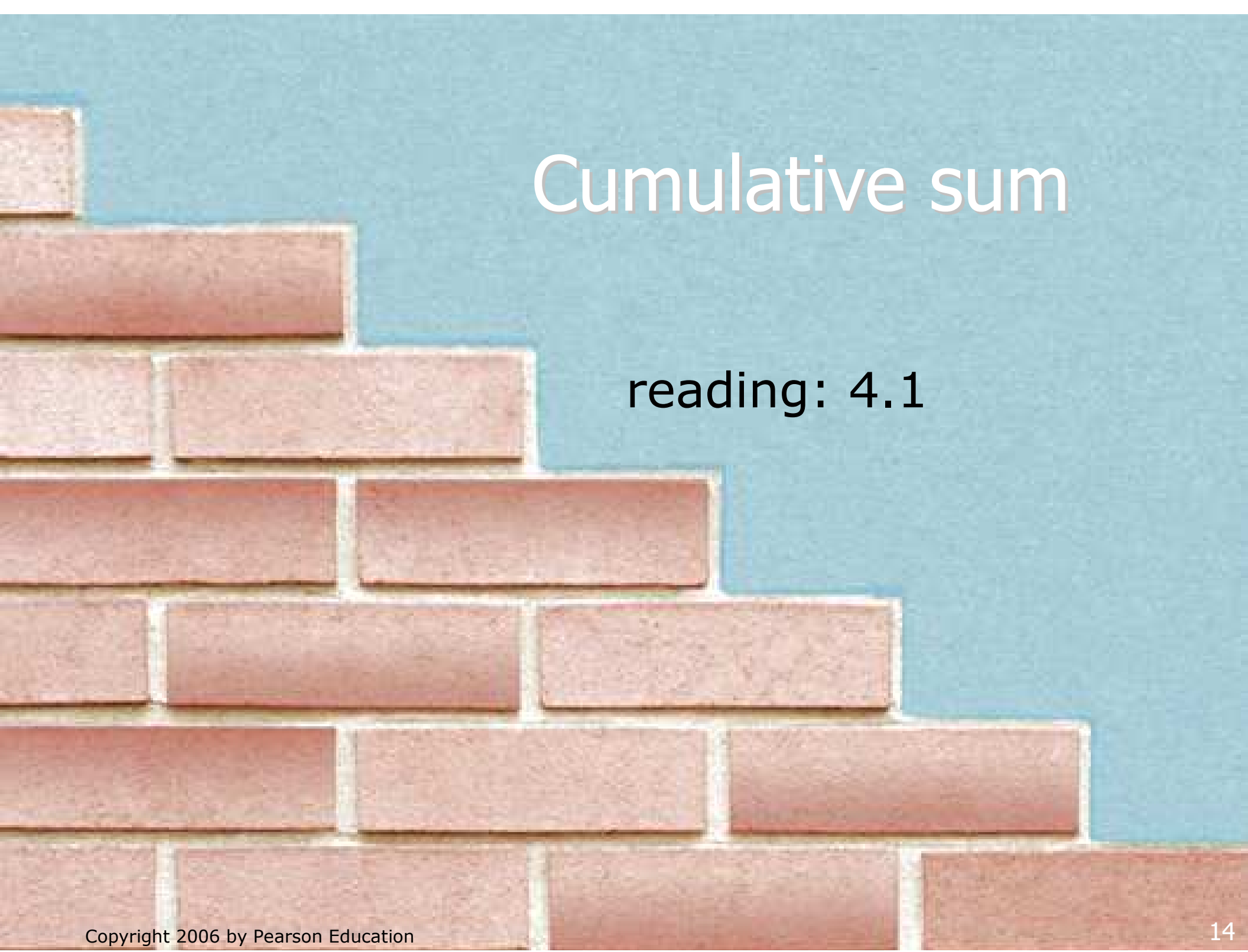
- Write a method named `attendance` that accepts a number of lectures attended by a student, and returns how many points a student receives for attendance.
  - The student receives 2 points for each of the first 5 lectures and 1 point for each subsequent lecture.

Return questions 2

- Write a method named `distanceFromOrigin` that accepts x and y coordinates as parameters and returns the distance between that (x, y) point and the origin.

- Write a method named `medianOf3` that accepts 3 integers as parameters and returns the middle value. For example, `medianOf3(4, 2, 7)` should return `4`.
  - Hint: Use methods from the `Math` class in your solution.





Adding many numbers

- How would you write code to find the sum of all integers from 1-1000?

```
int sum = 1 + 2 + 3 + 4 + ... ;
System.out.println("The sum is " + sum);
```

- What if we want the sum of integers from 1-1,000,000? Or to compute the sum up to any maximum?
  - We could write a method that accepts the maximum value as a parameter and prints the sum.
  - How can we generalize code like the above?

A failed attempt

- An incorrect solution for summing 1-100:

```
for (int i = 1; i <= 100; i++) {
    int sum = 0;
    sum = sum + i;
}

// sum is undefined here
System.out.println("The sum is " + sum);
```

  - The scope of `sum` is inside the `for` loop, so the last line of code fails to compile.

- **cumulative sum**: A variable that keeps a sum-in-progress and is updated until summing is finished.
  - The `sum` in the above code is an attempt at a cumulative sum.

Fixed cumulative sum loop

- A corrected version of the sum loop code:

```
int sum = 0;
for (int i = 1; i <= 100; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

The key idea:

- Cumulative sum variables must always be declared *outside* the loops that update them, so that they will continue to live after the loop is finished.

Cumulative sum question

- Write a method named `sumTo` that accepts an integer parameter *n* and returns the sum from 1 through *n*.
  - For example, `sumTo(5)` returns 1 + 2 + 3 + 4 + 5 = 15.
  - Call your method several times from `main` and print the results.
- Example log of execution:

```
sum to 5 is 15
sum to 10 is 55
```

Cumulative sum answer

```
public class Sum {
    public static void main(String[] args) {
        System.out.println("sum to 5 is " + sumTo(5));
        System.out.println("sum to 10 is " + sumTo(10));
    }

    // Returns the sum from 1 to the given maximum.
    public static int sumTo(int max) {
        int sum = 0;
        for (int i = 1; i <= max; i++) {
            sum = sum + i;
        }
        return sum;
    }
}
```

Variation: cumulative product

- The same idea can be used with other operators, such as multiplication which produces a cumulative product:

```
int exponent = 10;
int product = 1;
for (int i = 1; i <= exponent; i++) {
    product = product * 2;
}
System.out.println("2 to the " + exponent + " = " + product);
```

  - How would we change the above code so that it also allows changing the base, instead of always using 2?

Cumul. sum exercises

- Write a method named `sumSeries` that accepts an integer parameter *k* and computes the sum of the first *k* terms of the following series:
  - 1 + 1/2 + 1/4 + 1/8 + ...

- Write a method named `pow2` that accepts an integer parameter *n* and computes 2^n.

- Write a method named `pow` that accepts integers for a base *a* and an exponent *b* and computes a^b.