Building Java Programs

Chapter 3:
Parameters, Return, and Interactive Programs

Lecture 3-2: Return Values, Cumulative Sum
(reading: 3.2, 4.1)

Return values

reading: 3.2

self-check: #7-11

exercises: #4-6

Java's `Math` class

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

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

Math method syntax

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

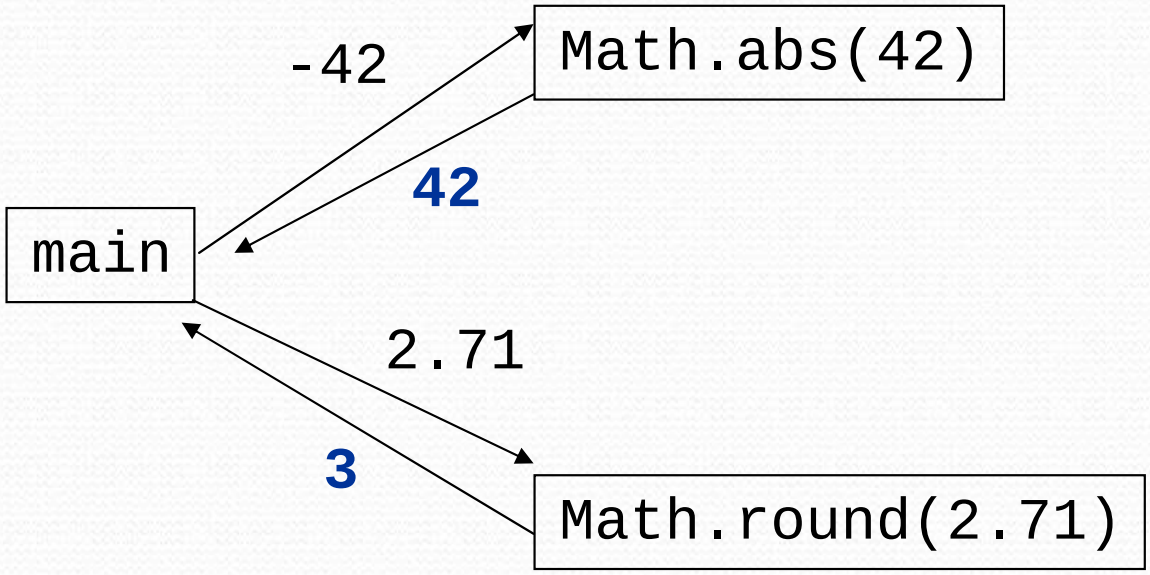
- Examples:

```
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
```

- The Math methods do not print to the console.
  - Each method produces ("**returns**") a numeric result.
  - The results are used as expressions (printed, stored, etc.).

Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Returned values send information *out* from a method to its caller.



Math 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?

Returning 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;
}
```

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);
}
```

Common error: not storing

- Many students incorrectly think that a `return` statement expands a variable's scope to include the calling method.

```
public static void main(String[] args) {
    slope(0, 0, 6, 3);
    System.out.println("The slope is " + result);  // ERROR:
}                                                  // result not defined

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

Fixing the common error

- The returned result must be stored into a variable or used in an expression to be useful to the caller.

```
public static void main(String[] args) {
    double s = slope(0, 0, 6, 3);
    System.out.println("The slope is " + s);
}

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```

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.

Cumulative sum

reading: 4.1

self-check: Ch. 4 #1-3

exercises: Ch. 4 #1-6

Adding many numbers

- How would you 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 from 1-1,000,000?
  Or the sum up to any maximum?

- We could write a method that accepts the max 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);
```

  - `sum`'s scope is in the `for` loop, so the code does not compile.

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly 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);
```

Key idea:

- Cumulative sum variables must be declared *outside* the loops that update them, so that they will exist after the loop.

Cumulative product

- This cumulative idea can be used with other operators:

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

  - How would we make the base and exponent adjustable?

Cumulative sum question

- Write a program that computes the sum of several ranges of numbers and shows the difference between those sums.
  - Example log of execution:

```
first sum, from 1 to 7, is 28
second sum, from -3 to 8, is 30
difference is 2
```

Cumulative sum answer

```
// Computes/displays the sum of several ranges of numbers.
public class Sum {
    public static void main(String[] args) {
        int sum1 = sum(1, 7);
        int sum2 = sum(-3, 8);
        System.out.println("first sum, from 1 to 7, is " + sum1);
        System.out.println("second sum, from -3 to 8, is " + sum2);
        System.out.println("difference is " + Math.abs(sum2 - sum1));
    }

    // Returns the sum between the given minimum and maximum.
    public static int sum(int min, int max) {
        int total = 0;
        for (int i = min; i <= max; i++) {
            total = total + i;
        }
        return total;
    }
}
```

Cumulative 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.