

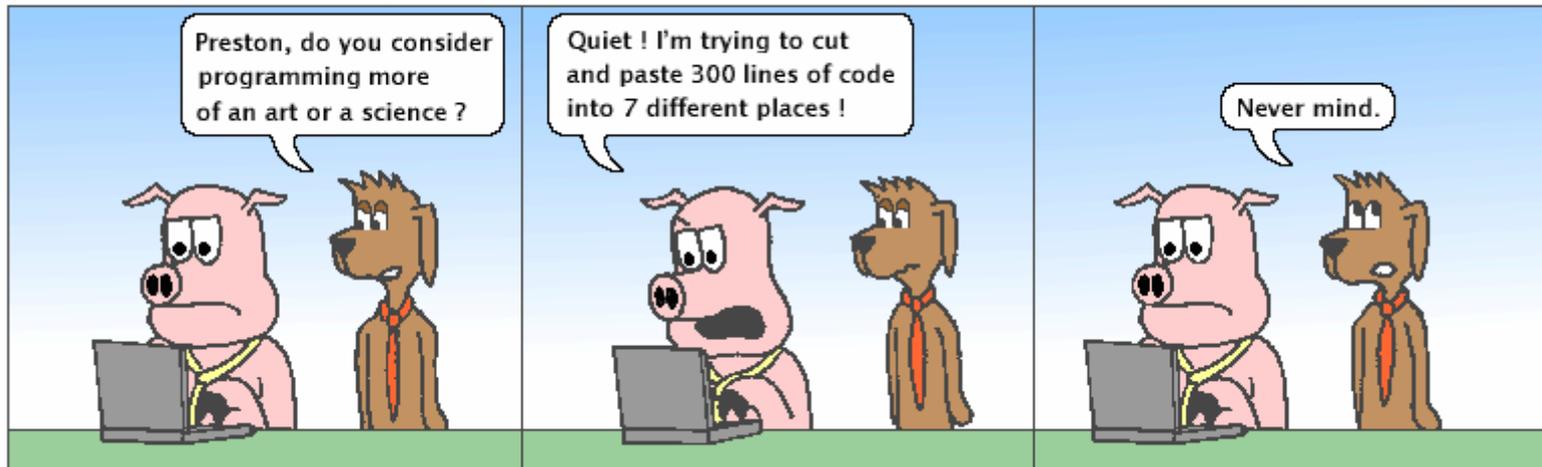
# Building Java Programs

Chapter 3

Lecture 3-2: Return values, `Math`, and `double`

**reading: 3.2, 2.1 - 2.2**

Hackles



By Drake Emko & Jen Brodzik

<http://hackles.org>

Copyright © 2001 Drake Emko & Jen Brodzik

# Java's Math class

Method name	Description
<code>Math.abs(<i>value</i>)</code>	absolute value
<code>Math.ceil(<i>value</i>)</code>	rounds up
<code>Math.floor(<i>value</i>)</code>	rounds down
<code>Math.log10(<i>value</i>)</code>	logarithm, base 10
<code>Math.max(<i>value1</i>, <i>value2</i>)</code>	larger of two values
<code>Math.min(<i>value1</i>, <i>value2</i>)</code>	smaller of two values
<code>Math.pow(<i>base</i>, <i>exp</i>)</code>	<i>base</i> to the <i>exp</i> power
<code>Math.random()</code>	random double between 0 and 1
<code>Math.round(<i>value</i>)</code>	nearest whole number
<code>Math.sqrt(<i>value</i>)</code>	square root
<code>Math.sin(<i>value</i>)</code> <code>Math.cos(<i>value</i>)</code> <code>Math.tan(<i>value</i>)</code>	sine/cosine/tangent of an angle in radians
<code>Math.toDegrees(<i>value</i>)</code> <code>Math.toRadians(<i>value</i>)</code>	convert degrees to radians and back

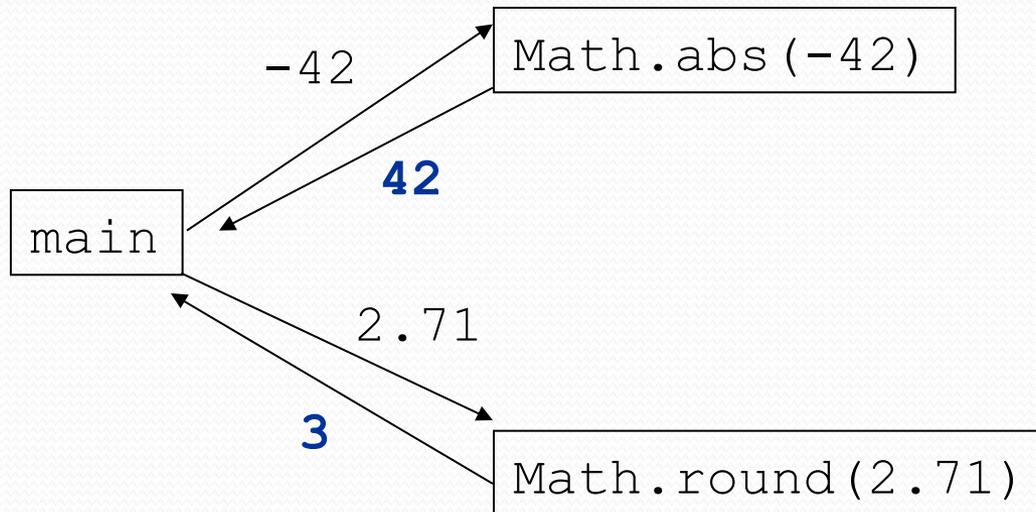
Constant	Description
<code>Math.E</code>	2.7182818...
<code>Math.PI</code>	3.1415926...

# No output?

- Simply calling these methods produces no visible result.
  - `Math.pow(3, 4); // no output`
- Math method calls use a Java feature called *return values* that cause them to be treated as expressions.
- The program runs the method, computes the answer, and then "replaces" the call with its computed result value.
  - ~~`Math.pow(3, 4); // no output`~~  
`81.0; // no output`
- To see the result, we must print it or store it in a variable.
  - `double result = Math.pow(3, 4);`
  - `System.out.println(result); // 81.0`

# 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.
    - Return values send information *out* from a method to its caller.
      - A call to the method can be used as part of an expression.



# Why return and not print?

- It might seem more useful for the `Math` methods to print their results rather than returning them. Why don't they?
- Answer: Returning is more flexible than printing.
  - We can compute several things before printing:

```
double pow1 = Math.pow(3, 4);  
double pow2 = Math.pow(10, 6);  
System.out.println("Powers are " + pow1 + " and " + pow2);
```

- We can combine the results of many computations:

```
double k = 13 * Math.pow(3, 4) + 5 - Math.sqrt(17.8);
```

# 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?

# Quirks of real numbers

- Some Math methods return double or other non-int types.

```
int x = Math.pow(10, 3);    // ERROR: incompat. types
```

- Some double values print poorly (too many digits).

```
double result = 1.0 / 3.0;  
System.out.println(result);    // 0.3333333333333333
```

- The computer represents doubles in an imprecise way.

```
System.out.println(0.1 + 0.2);
```

- Instead of 0.3, the output is 0.30000000000000004

# Type casting

- **type cast:** A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer
- Syntax:

**(type) expression**

Examples:

```
double result = (double) 19 / 5;           // 3.8
int result2 = (int) result;                // 3
int x = (int) Math.pow(10, 3);            // 1000
```

# More about type casting

- Type casting has high precedence and only casts the item immediately next to it.

- `double x = (double) 1 + 1 / 2; // 1.0`

- `double y = 1 + (double) 1 / 2; // 1.5`

- You can use parentheses to force evaluation order.

- `double average = (double) (a + b + c) / 3;`

- A conversion to `double` can be achieved in other ways.

- `double average = 1.0 * (a + b + c) / 3;`

# Returning a value

```
public static type name(parameters) {  
    statements;  
    ...  
    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;  
}
```

- `slope(5, 11, 1, 3)` returns 2.0

# Return examples

**// Converts degrees Fahrenheit to Celsius.**

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

**// Computes triangle hypotenuse length given its side lengths.**

```
public static double hypotenuse(int a, int b) {  
    double c = Math.sqrt(a * a + b * b);  
    return c;  
}
```

- You can shorten the examples by returning an expression:

```
public static double fToC(double degreesF) {  
    return 5.0 / 9.0 * (degreesF - 32);  
}
```

# Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method.

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

```
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

- Returning sends the variable's *value* back. Store the returned value into a variable or use it in an expression.

```
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;  
}
```

# Exercise

- In physics, the *displacement* of a moving body represents its change in position over time while accelerating.
  - Given initial velocity  $v_0$  in m/s, acceleration  $a$  in  $\text{m/s}^2$ , and elapsed time  $t$  in s, the displacement of the body is:
    - Displacement =  $v_0 t + \frac{1}{2} a t^2$
- Write a method `displacement` that accepts  $v_0$ ,  $a$ , and  $t$  and computes and returns the change in position.
  - example: `displacement(3.0, 4.0, 5.0)` returns 65.0

# Exercise solution

```
public static double displacement(double v0, double a, double t) {  
    double d = v0 * t + 0.5 * a * Math.pow(t, 2);  
    return d;  
}
```

# Exercise

- If you drop two balls, which will hit the ground first?
  - Ball 1: height of 600m, initial velocity = 25 m/sec downward
  - Ball 2: height of 500m, initial velocity = 15 m/sec downward
- Write a program that determines how long each ball takes to hit the ground (and draws each ball falling).
- Total time is based on the force of gravity on each ball.
  - Acceleration due to gravity  $\cong 9.81 \text{ m/s}^2$ , downward
  - Displacement =  $v_0 t + \frac{1}{2} a t^2$

# Ball solution

```
// Simulates the dropping of two balls from various heights.
import java.awt.*;

public class Balls {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(600, 600);
        Graphics g = panel.getGraphics();

        int ball1x = 100, ball1y = 0, v01 = 25;
        int ball2x = 200, ball2y = 100, v02 = 15;

        // draw the balls at each time increment
        for (double t = 0; t <= 10.0; t = t + 0.1) {
            double disp1 = displacement(v01, t, 9.81);
            g.fillOval(ball1x, ball1y + (int) disp1, 10, 10);
            double disp2 = displacement(v02, t, 9.81);
            g.fillOval(ball2x, ball2y + (int) disp2, 10, 10);

            panel.sleep(50);    // pause for 50 ms
            panel.clear();
        }
    }
    ...
}
```