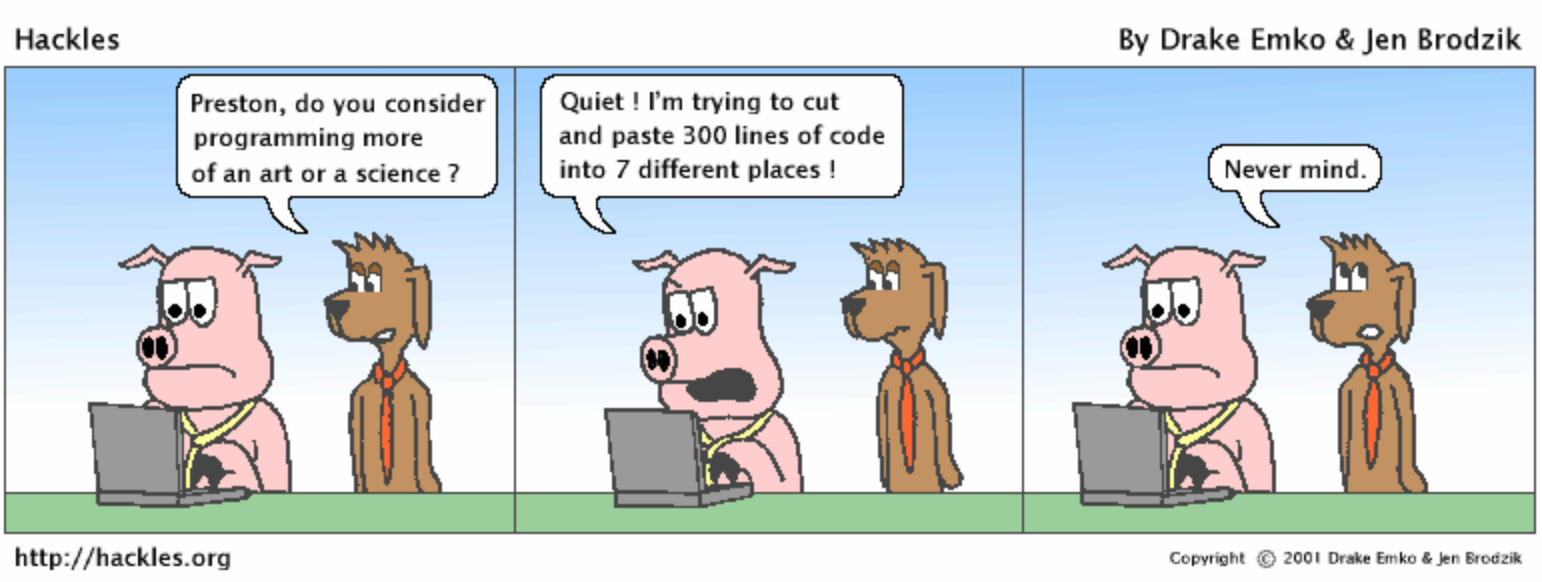
Building Java Programs

Chapter 3
Lecture 7: Return values, `Math`, and `casting`

reading: 3.2, 2.1 - 2.2

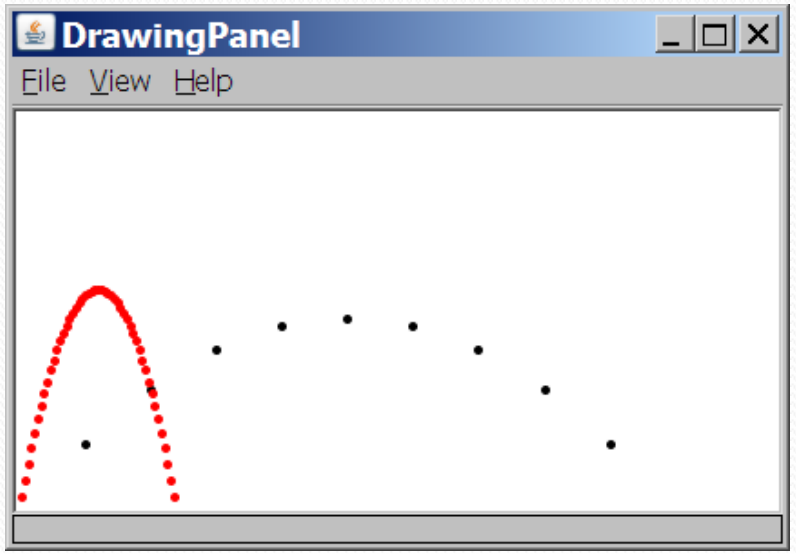
(Slides adapted from Stuart Reges, Hélène Martin, and Marty Stepp)


Hackles
By Drake Emko & Jen Brodzik
Preston, do you consider programming more of an art or a science ?
Quiet ! I'm trying to cut and paste 300 lines of code into 7 different places !
Never mind.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik

Projectile problem

- Write a program that displays (as text and graphics) the paths of projectiles thrown at various velocities and angles.
  - Projectile #1: velocity = 60, angle = 50° , steps = 10
  - Projectile #2: velocity = 50, angle = 80° , steps = 50

```
step         x         y     time
   0      0.00      0.00     0.00
   1     36.14     38.76     0.94
   2     72.28     68.91     1.87
   3    108.42     90.45     2.81
   4    144.56    103.37     3.75
   5    180.70    107.67     4.69
   6    216.84    103.37     5.62
   7    252.98     90.45     6.56
   8    289.12     68.91     7.50
   9    325.26     38.76     8.43
  10    361.40      0.00     9.37

step         x         y     time
   0      0.00      0.00     0.00
   1      1.74      9.69     0.20
   2      3.49     18.98     0.40
  ...
```



Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`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.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

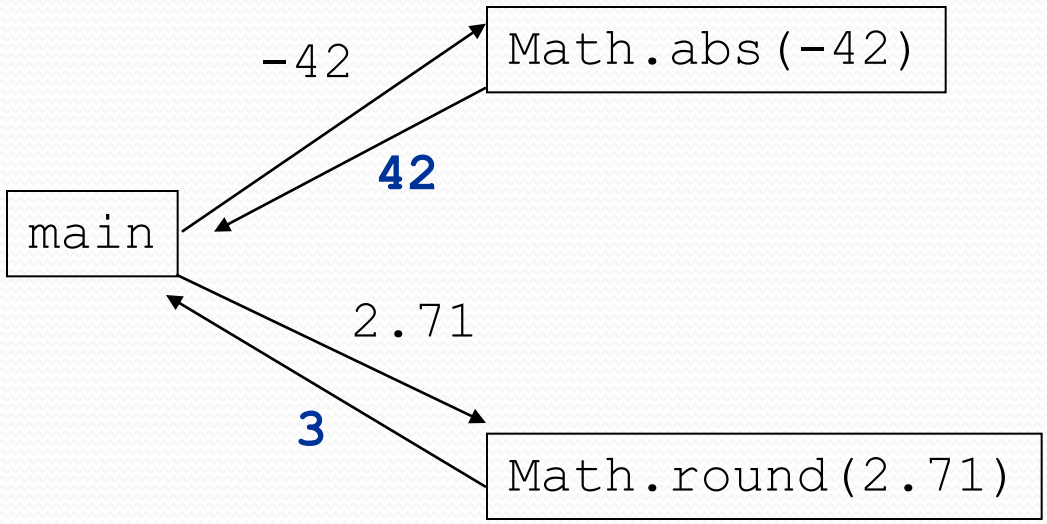
Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

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.
  - Return values send information *out* from a method to its caller.
    - A call to the method can be used as part of an expression.
  - (Compare to parameters which send values *into* a method)



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);     // 0.3333333333333
```

- The computer represents `double`s 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;
}
```

- When Java reaches a return statement:
  - it evaluates the expression
  - it substitutes the return value in place of the call
  - it goes back to the caller and continues after the method call

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

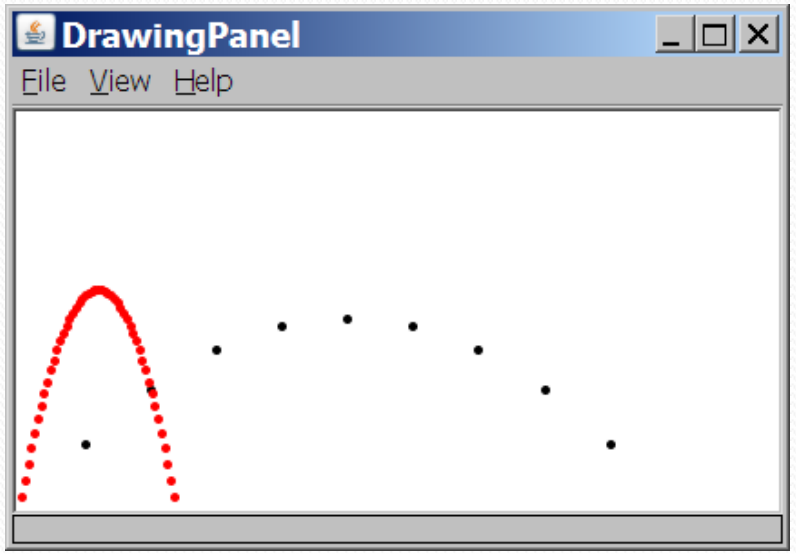
- Returning sends the variable's *value* back. Store the returned value into a variable or use it in an expression.

```
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;
}
```

Projectile problem

- Write a program that displays (as text and graphics) the paths of projectiles thrown at various velocities and angles.
  - Projectile #1: velocity = 60, angle = 50° , steps = 10
  - Projectile #2: velocity = 50, angle = 80° , steps = 50

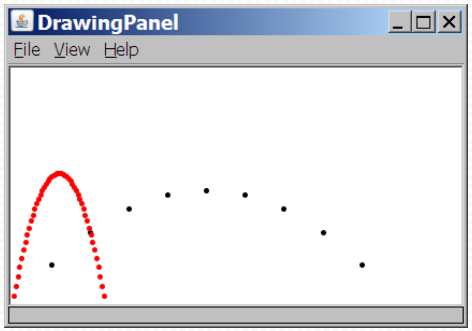
```
step        x        y     time
   0     0.00     0.00     0.00
   1    36.14    38.76     0.94
   2    72.28    68.91     1.87
   3   108.42    90.45     2.81
   4   144.56   103.37     3.75
   5   180.70   107.67     4.69
   6   216.84   103.37     5.62
   7   252.98    90.45     6.56
   8   289.12    68.91     7.50
   9   325.26    38.76     8.43
  10   361.40     0.00     9.37

step        x        y     time
   0     0.00     0.00     0.00
   1     1.74     9.69     0.20
   2     3.49    18.98     0.40
  ...
```



Time observations

- We are given the number of "steps" of time to display.
  - We must figure out how long it takes the projectile to hit the ground, then divide this time into the # of steps requested.

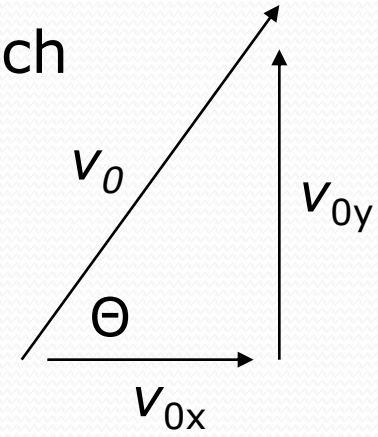
```
step        x         y     time
   0     0.00      0.00     0.00
   1    36.14     38.76     0.94
   2    72.28     68.91     1.87
  ...
  10   361.40      0.00     9.37
```



- Total time is based on the force of gravity on the projectile.
  - Force of gravity (g) $\cong$ 9.81 m/s^2, downward
  - The projectile has an initial upward velocity, which is fought by gravity until the projectile reaches its peak, then it falls.

Velocity and acceleration

- The projectile has a given initial velocity v_0, which can be divided into x and y components.
  - $v_{0x} = v_0 \cos \Theta$
  - $v_{0y} = v_0 \sin \Theta$
    - Example: If v_0=13 and Θ=60° , v_{0x}=12 and v_{0y}=5.



- The velocity v_t of a moving body at time t, given initial velocity v_0 and acceleration a, can be expressed as:
  - $v_t = v_0 + a\,t$
- In our case, because of symmetry, at the end time t the projectile is falling exactly as fast as it was first going up.
  - $v_t = -v_0$
    $-v_0 = v_0 + a\,t$
    $\boldsymbol{t = -2\,v_0 / a}$

X/Y position, displacement

- Based on the previous, we can now display *x* and *time*.
  - $x_t = v_x\, t$ since there is no force in the x direction.

```
step        x        y     time
   0     0.00     ????     0.00
   1    36.14     ????     0.94
   2    72.28     ????     1.87
  ...
  10   361.40     ????     9.37
```

- To display the y, we need to compute the projectile's *displacement* in y direction at each time increment.
  - $y_t = v_{0y}\, t + \frac{1}{2}\, a\, t^2$
  - Since this formula is complicated, let's make it into a method.

System.out.printf

an advanced command for printing formatted text

```
System.out.printf("format string", parameters);
```

- A format string contains *placeholders* to insert parameters into it:
  - `%d` an integer
  - `%f` a real number
  - `%s` a string

  - Example:

```
int x = 3;
int y = 2;
System.out.printf("(%d, %d)\n", x, y);   // (3, 2)
```

System.out.printf cont'd

- A placeholder can specify the parameter's *width* or *precision*:
  - `%8d` an integer, 8 characters wide, right-aligned
  - `%-8d` an integer, 8 characters wide, left-aligned
  - `%.4f` a real number, 4 characters after decimal
  - `%6.2f` a real number, 6 characters wide, 2 after decimal

  - Examples:

```
int age = 45;
double gpa = 1.2345678;

System.out.printf("%-8d %4f\n", age, gpa);
System.out.printf("%8.3f %.1f %.5f", gpa, gpa, gpa);
```

  - Output:

```
45       1.23
   1.234 1.2 1.23457
```

Projectile solution

```
// This program computes and draws the trajectory of a projectile.

import java.awt.*;

public class Projectile {
    // constant for Earth's gravity acceleration in meters/second^2
    public static final double ACCELERATION = -9.81;

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(420, 250);
        Graphics g = panel.getGraphics();

        //       v0  angle  steps
        table(g,  60,  50,  10);

        g.setColor(Color.RED);
        table(g,  50,  80,  50);
    }

    // returns the displacement for a body under acceleration
    public static double displacement(double v0, double t, double a) {
        return v0 * t + 0.5 * a * t * t;
    }

    ...
```

Projectile solution

```
    ...
    // prints a table showing the trajectory of an object given
    // its initial velocity v and angle and number of steps
    public static void table(Graphics g, double v0,
                             double angle, int steps) {
        double v0x = v0 * Math.cos(Math.toRadians(angle));
        double v0y = v0 * Math.sin(Math.toRadians(angle));
        double totalTime = -2.0 * v0y / ACCELERATION;
        double dt = totalTime / steps;

        System.out.println("    step       x       y    time");
        for (int i = 0; i <= steps; i++) {
            double time = i * dt;
            double x = i * v0x * dt;
            double y = displacement(v0y, time, ACCELERATION);
            System.out.printf("%8d%8.2f%8.2f%8.2f\n", i, x, y, time);
            g.fillOval((int) x, (int) (250 - y), 5, 5);
        }
    }
}
```