Mat 2170
Week 6

Methods

Spring 2014
Student Responsibilities

- Reading: Textbook, Chapter 5.1 – 5.3
- Lab
- Attendance
Chapter Five Overview: 5.1 – 5.3

Methods

- 5.1 Quick overview of methods
- 5.2 Writing our own methods
- 5.3 Mechanics of the method–calling process
Quick Overview of Methods

- You’ve been working with methods ever since Lab 1 and the HelloWorld program.

- The `run()` method in every program is one example.

- Other examples are `println()`, `setColor()`, and `getHeighit()`. 
Methods

Basically a method is a sequence of statements collected together and given a name.

- The name makes it possible to execute the statements more easily.
- Instead of copying the entire list of statements, we can simply provide the method name.
Useful Terms

- **Invoking** a method (using its name) is known as **calling** that method.

- The part of a program or code that invokes a method is named the **caller** or **calling program**.

- The caller can **pass** information to a method by using **arguments** (the java expressions in the parentheses), e.g.:

  
  ```java
  R.setFilled(true)
  R.setFilled((row + col) % 2 == 0)
  Color myColor = Color.green;
  R.setColor(myColor)
  ```
When a method completes its operation, it returns to the code which invoked it (i.e., the caller).

A method can pass information back to the caller by returning a single result.
Method Calls as Messages

- The act of **calling a method** is often described in terms of **sending a message to an object**.

- The method call: `R.setColor(Color.RED);`

  is regarded metaphorically as **sending a message to** the `GRect` object `R` telling it to change its color to red.
The object to which a message is sent is called the **receiver**.

The general pattern for sending a message to an object is:

```java
receiver.methodName(arguement_list)
```
Information Hiding

- **One important advantage of methods:**
  They allow us to **ignore the inner workings** of complex operations.

- When a method is used, it is **more important** to know what the method does than to understand exactly **how** it does it.

- The underlying details of a method are of interest only to the programmer who implements and maintains it.
Programmers who utilize a method are concerned about:

1. The method interface
   1.1 return type
   1.2 method name
   1.3 order, number and type of arguments

2. Whether the method is correct.

They can usually ignore the implementation altogether.

The idea that callers should be insulated from the details of a method is the principle of information hiding, one of the cornerstones of software engineering.
Methods as Tools for Programmers

- A method provides a service to a programmer, who is typically creating some kind of application.

- A programmer utilizes methods to reduce the amount of work he or she must do, and to organize the software they are writing.

- Methods like `readInt()` and `println()` are used to communicate with and obtain information from the user.
Method Calls as Expressions

- Syntactically, method calls in Java are part of the expression framework.

- Methods that **return a value** can be used as **terms** in an expression, just like variables and constants.

- The **Math** class in the **java.lang** package defines several methods that are useful in writing mathematical expressions.

- Methods that **belong to the entire class** are called **static** methods.
Math Library Method Calls

- You must **include the name of the class**, along with the method name, for example: `Math.sqrt()`.

- Suppose we need to compute the distance from the origin to the point \((x, y)\), which is given by the formula:

\[ d = \sqrt{x^2 + y^2} \]

- We can apply the square root function by calling the `sqrt()` method from the **Math** class:

```java
double distance = Math.sqrt(x * x + y * y);
```
## Useful Methods in the `Math` Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Math.abs(x)</code></td>
<td>Returns the absolute value of <code>x</code></td>
</tr>
<tr>
<td><code>Math.min(x, y)</code></td>
<td>Returns the smaller of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>Math.max(x, y)</code></td>
<td>Returns the larger of <code>x</code> and <code>y</code></td>
</tr>
<tr>
<td><code>Math.sqrt(x)</code></td>
<td>Returns $\sqrt{x}$, the square root of <code>x</code></td>
</tr>
<tr>
<td><code>Math.log(x)</code></td>
<td>Returns $\log_e x$, the natural logarithm of <code>x</code></td>
</tr>
<tr>
<td><code>Math.exp(x)</code></td>
<td>Returns $e^x$, the inverse logarithm of <code>x</code></td>
</tr>
<tr>
<td><code>Math.pow(x, y)</code></td>
<td>Returns $x^y$, the value of <code>x</code> raised to the <code>y</code> power</td>
</tr>
</tbody>
</table>
### The Math Class — Trig Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Math.sin(theta)</strong></td>
<td>Returns the sine of <em>theta</em>, measured in radians</td>
</tr>
<tr>
<td><strong>Math.cos(theta)</strong></td>
<td>Returns the cosine of <em>theta</em></td>
</tr>
<tr>
<td><strong>Math.tan(theta)</strong></td>
<td>Returns the tangent of <em>theta</em></td>
</tr>
<tr>
<td><strong>Math.asin(x)</strong></td>
<td>Returns the angle whose sine is <em>x</em></td>
</tr>
<tr>
<td><strong>Math.acos(x)</strong></td>
<td>Returns the angle whose cosine is <em>x</em></td>
</tr>
<tr>
<td><strong>Math.atan(x)</strong></td>
<td>Returns the angle whose tangent is <em>x</em></td>
</tr>
</tbody>
</table>
### The Math Class — Conversion Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.toRadians(degrees)</td>
<td>Converts an angle from degrees to radians</td>
</tr>
<tr>
<td>Math.toDegrees(radians)</td>
<td>Converts an angle from radians to degrees</td>
</tr>
</tbody>
</table>
Solving Quadratic Equations

- The standard **quadratic** equation is:

\[ ax^2 + bx + c = 0 \]

- The quadratic **formula**, to solve for the roots, is:

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

- Of interest to us is the **radicand**, since it determines the **number** of solutions:

\[ b^2 - 4ac \]

- How many **real** solutions are there when the radicand is...
  1. positive?
  2. zero?
  3. negative?
## Also of Use in Lab 6

More messages available for **GRect** and **GOval** objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getX()</code></td>
<td>returns the x component of the object’s position</td>
</tr>
<tr>
<td><code>getY()</code></td>
<td>returns the y component of the object’s position</td>
</tr>
<tr>
<td><code>getWidth()</code></td>
<td>returns the width of the object</td>
</tr>
<tr>
<td><code>getHeight()</code></td>
<td>returns the height of the object</td>
</tr>
<tr>
<td><code>move(dx, dy)</code></td>
<td>moves the object using the displacements dx and dy</td>
</tr>
</tbody>
</table>
### Examples

Assume **R** is a **GRect** and **C** is a **GOval**:

<table>
<thead>
<tr>
<th>Method</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.getX()</td>
<td>C.getX()</td>
</tr>
<tr>
<td>R.getY()</td>
<td>C.getY()</td>
</tr>
<tr>
<td>R.getWidth()</td>
<td>C.getWidth()</td>
</tr>
<tr>
<td>R.getHeight()</td>
<td>C.getHeight()</td>
</tr>
<tr>
<td>R.move(1.0, −1.5)</td>
<td>C.move(1.0, −1.5)</td>
</tr>
<tr>
<td>R.move(dx, dy)</td>
<td>C.move(dx, dy)</td>
</tr>
</tbody>
</table>
Problem Solving — in Class

Design algorithms, then programs, to:

- Create a table of values for $x$, $\sqrt{x}$, and $x^2$ running from $x = 0$ to $x = 100$ by 10s

- Create a table of values for $x$, $\sin(x)$, and $\cos(x)$ as $x$ runs from 0 to $2\pi$ by $\frac{\pi}{4}$ increments.

- Solve for the $n^{th}$ Fibonacci number, defined by the sequence 0, 1, 1, 2, 3, 5, 8, 13 ... The first two terms are 0 and 1, and every subsequent term is the sum of the preceding two.

- Modify the previous program to display all the terms in the Fibonacci sequence that are smaller than 1,000.