Basic Methods (DrawingGizmo class)

- Update Methods
  - public void draw()
    - post: The DrawingGizmo is set to drawing mode (and is colored green).
  - public void dontDraw()
    - post: The DrawingGizmo is set to moving mode (and is colored red).
  - public void turnClockwise()
    - post: The DrawingGizmo is rotated 30 degrees clockwise from its current heading.
  - public void turnCounterclockwise()
    - post: The DrawingGizmo is rotated 30 degrees counterclockwise from its current heading.

Some methods have post-conditions only
- Nothing special is required for them to work
- Only need DrawingGizmo object instance to call them

Class Specifications

- A class diagram only gives a “sketch” of that class
- More detail is given by the full specification of a class
- Every useful Java class should have such a specification
  - When you write code, it’s up to you to create this document, so others can make proper use of your code

Two main parts:
1. Invariants: things that are always true of class objects
2. Methods: all the operations involving the class, with:
   - Pre-conditions: Anything that needs to be true before the method will work properly
   - Post-conditions: Anything that will be true after the method is done
More Complex Methods

- **DrawingGizmo** object has simple methods, each of which does the same thing, the same way, every time.
  - But it also has some more complex methods, which:
    - Take input parameters
    - Change what they do based on those parameters

A Parameterized Method

- The `turnBy(int)` method wants us to insert an integer value ("int" in Java) as an input parameter.
  - We must give it an integer when we call it, or it won't work!
    - `pen.turnBy(90);`
    - `pen.turnBy(9000);`
    - `pen.turnBy(-180);`
    - `pen.turnBy(3.6);`
    - `pen.turnBy();`

Parameterized Methods vs. Non-parameterized Methods

- The parentheses () in a method specification serve two purposes:
  1. They indicate that it is a method, so compiler and coder know what it is.
  2. They show whether or not the method has any input parameters.
- If the method is not parameterized, then nothing is shown in the parentheses, and when we use it, we cannot put anything there, or our code won't compile.
- Similarly, if the method is parameterized, then we have to put actual parameters into the code when we use it.
  - In this description, the diagram shows the type of the input parameter.
  - It is up to us to actually choose what particular input value we will put in there when we call the method.

Two More Parameterized Methods

- These new **DrawingGizmo** methods change the color of the background window and drawing line.
  - E.g., for a yellow line on a blue background:
    - `pen.setBackground(java.awt.Color.blue);`
    - `pen.setForeground(java.awt.Color.yellow);`
Void vs Non-void Methods

- **Void Method**: outputs something or changes one or more objects but **does not return a value**
  - `System.out.println()`
  - `window.setBackground(Color.blue)`
- **Non-void Method**: may make some change to object but primary effect is that it **returns a value**
  - `window.getWidth()`
  - `oval.getBackground()`
- **In general:**
  - Setters are void methods
  - Getters are non-void methods

Calling Void and Non-void Methods

- **Void method**: standalone call
  - `win.setWidth(500);`
  - `win.setHeight(600);`
  - `System.out.println("Window size set.");`
- **Non-void method**: make use of returned value
  - `int w = win.getWidth();`
  - `if(win.getHeight() > w)`
  - `System.out.println("Portrait mode.");`

Methods in the String Class

- **Strings are objects with many methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String(String)</code></td>
<td><strong>String</strong> constructor</td>
</tr>
<tr>
<td><code>charAt(int)</code></td>
<td>Returns character at input position</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>Returns number of chars in String</td>
</tr>
<tr>
<td><code>substring(int)</code></td>
<td>Returns subpart of String, starting from input position, going all the way to end</td>
</tr>
<tr>
<td><code>toLowerCase()</code></td>
<td>Returns lower/upper case version</td>
</tr>
<tr>
<td><code>toUpperCase()</code></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```java
String first, last;
first = "Ronald";
last = "McDonald";
System.out.print( first.charAt( 1 ) );  // o
System.out.print( last.length() );  // 8
System.out.print( last.substring( 2 ) );  // Donald
System.out.print( first.substring( 0, 3 ) );  // Ron

first = first.toLowerCase();
last = last.toUpperCase();
System.out.println( first + last );  // ronald MCDONALD
```

Full list online at:
http://docs.oracle.com/javase/8/docs/api/java/lang/String.html
Non-Void Methods and Operators

- These methods have **non-void return types**, meaning that after they have completed running, they **output** a value.

- The basic rule of such methods is that they can be called and used **anywhere** in our code that a value of the return type can be used.

---

**String**

- *<<constructor>>*
  - `String(String)`
- *<<query>>*
  - `char charAt(int)`
  - `int length()`
  - `String substring(int)`
  - `String substring(int, int)`
- *<<translate>>*
  - `String toLowerCase()`
  - `String toUpperCase()`

---

This Week & Next

- **Meetings this week:**
  - Monday/Wednesday: regular classroom
  - Tuesday: in the CS Lab (16 Wing)
  - Friday: No class

- **Reading 05:** Ch. 6 due Thursday, Oct 31 at Noon

- **Program 05:** due 11:59 PM, Wednesday October 30

- **Office Hours:** Wing 212
  - Monday/Friday: 2:15 PM–3:15 PM
  - Tuesday: 2:30 PM–1:30 PM
  - Wednesday: 12:05 PM–1:00 PM