Review cs1graphics

Things to know:

- All our graphics programs should start with: `from cs1graphics import *`, so that names of objects and functions defined in this library can be recognized by Python, when we use them in our programs.

- As next step, we need to create a Canvas object which creates the window where we'll draw. See slide 1-22 in the Big Ideas lecture.

- `cs1graphics` defines many graphics shapes that we can create by calling the respective special functions that do this, such as `Rectangle`, `Circle`, `Polygon`, etc.

- The created shapes (known as objects) are referred through the names we assign them with the assignment statements. Using these names and the . (dot) operator, we call special functions (known as methods) to manipulate the state of the shape (position, color, size, borders, etc.). For example: `torso.setFillColor("gray")`.

- The coordinate system for drawing shapes has its origin (0, 0) in the top-left corner of the canvas window.

- By default, every new shape you create without a reference point (or center point), will be assigned the reference (0, 0).

- To make objects appear on the canvas window, we need to add them through the method `add`.

Motivating Problem

Sometimes graphics scenes contain repeated elements. But, a programmer should follow the DRY principle: Don't Repeat Yourself. This makes our code easy to read and maintain.

The shown fishtank has six scaled, rotated, and flipped versions of the fish pattern in the picture. How to make use of it?

In preparation to learning to building scenes like this, we will discuss:

1. **Cloning** (making a copy of a shape)
2. **Transformation operation:**
   1. **Scaling** (increase or decrease the size by a factor)
   2. **Rotating** (moving around a center with a certain angle)
   3. **Flipping** (turning over, usually across one of three axes)
Cloning

We can make a copy of a drawable object using the clone method.

Cloning example:

```python
wedge = Polygon(Point(0, 100), Point(0, 0), Point(200, 100))
wedge.setFillColor('green')
wedge.moveTo(100, 200)

wedge2 = wedge.clone()
wedge2.moveTo(525, 200)
```

* On this and subsequent slides, to avoid clutter we omit invocations of the Canvas add method for each of the Polygon objects, e.g. `paper.add(wedge)` etc.

Scaling

We can change the size of a drawable object using the scale method. This method takes one number argument. A number > 1 increases the size, a number < 1 decreases the size.

Scaling example:

```python
wedge = Polygon(Point(0, 100), Point(0, 0), Point(200, 100))
wedge.setFillColor('green')
wedge.moveTo(100, 200)

wedge4 = wedge.clone()
wedge4.moveTo(525, 425)
wedge4.scale(0.5)
```

Rotating

We can rotate a drawable object using the rotate method. This causes the shape to move around its center by a certain angle, which is given as argument to the method.

Rotating example:

```python
wedge = Polygon(Point(0, 100), Point(0, 0), Point(200, 100))
wedge.setFillColor('green')
wedge.moveTo(100, 200)

wedge2 = wedge.clone()
wedge2.moveTo(525, 200)
wedge2.rotate(180)
```

Flipping

We can flip a drawable object around an axis using the flip method.

Flipping example:

```python
wedge = Polygon(Point(0, 100), Point(0, 0), Point(200, 100))
wedge.setFillColor('green')
wedge.moveTo(100, 200)

wedge2 = wedge.clone()
wedge2.moveTo(525, 200)
wedge2.rotate(180)
```

Notice negative value for angle.
from cs1graphics import *
from graphicsState import *

e = Canvas(300,200,'azure3')
d = Circle(25,Point(0,0))
e.add(d)

printState(d) # what is the state of the dot?
d.setFillColor('magenta')
printState(d) # printState is a function defined in the graphicsState module that prints the internal state of the properties of an object.
d.moveTo(200,75)
printState(d) # The cells of the grid are squares 100x100. Notice that 100 was an argument in the function call. You can choose another value. In the picture, observe that the coordinate center (0, 0) in the top left corner of the window.

Tools: cs1graphicsHelper

cs1graphicsHelper is a module that defines two helpful functions: drawReferencePoint, drawGrid.

• The cells of the grid are squares 100x100. Notice that 100 was an argument in the function call. You can choose another value. In the picture, observe that the coordinate center (0, 0) in the top left corner of the window.
• The reference point of an object is depicted as a small orange circle.
• In the slides 5-8, the grid and the reference point are not drawn within Canopy, thus their depiction differs from that of this slide.
• Notice on those slides that the reference point of a polygon is the first point in the list of points passed as arguments.

Returning to our motivating problem

Remember the problem from slide 4: How can we draw this fishtank with six similar fish?

We learned to transform individual shapes by cloning and then performing operations such as rotation, scaling, and flipping to them. However, a fish is composed of three parts: body, tail, and eye. These however, are currently not tied to one-another.

One approach we can try is to separately draw the body, tail, and eye for each of the six fish. Is this easy or hard? What about DRY?

A better way: Abstraction!

Drawing the three parts (yellow body, green fin, and black eye) of every fish over and over again would be tedious, and getting the coordinates right for all the scalings, rotations, and flippings would be extremely challenging. We will make lots of mistakes.

Fortunately, there is a better way! We can abstract over the notion of a fish pattern by creating a Layer object, which you can think of as a mini-canvas for grouping together other shapes.

We can add items to a Layer just like we can add items to a canvas. The "push pin" reference point of a Layer is (0,0) in its own coordinate system.
A Layer containing one fish

# A layer is a "mini-canvas" for combining drawable objects
fish = Layer()

# yellow body of the fish
body = Ellipse(100,50,Point(0,0))
body.setFillColor('yellow')
fish.add(body)

# green tail of the fish
tail = Polygon(Point(-50,0),
Point(-75,25),
Point(-75,-25))
tail.setFillColor('green')
fish.add(tail)

# black eye of the fish
eye = Circle(5,Point(25,-5))
eye.setFillColor('black')
fish.add(eye)

A tank with one fish

# Add fish to a canvas
tank = Canvas(600, 400, 'skyblue', 'Where is Dory?')
tank.add(fish)
fish.moveTo(100, 50)  # reference point of fish layer is (0,0),
# this is why we move it to a new point

Cloning a Layer: a second fish

# Add fish to a canvas
tank = Canvas(600, 400, 'skyblue', 'Where is Dory?')
tank.add(fish)
fish.moveTo(100, 50)

# Add a second, bigger fish
fish2 = fish.clone()
tank.add(fish2)
fish2.moveTo(350, 100)
fish2.scale(2)

Populate the tank

# Add a second, bigger fish
fish2 = fish.clone()
tank.add(fish2)
fish2.moveTo(350, 100)
fish2.scale(2)

# Third fish
fish3 = fish.clone()
tank.add(fish3)

# Fourth fish
fish4 = fish.clone()
tank.add(fish4)

# Fifth fish
fish5 = fish.clone()
tank.add(fish5)

# Sixth fish
fish6 = fish.clone()
tank.add(fish6)
fish6.moveTo(450, 275)

For fish 3 to 6 specify the coordinates and the transformation operations.
The power of abstraction

Suppose we want to add pink hats to all of our fish.

Do we want to change each fish individually?
No! We need only change the prototypical fish (the original fish prior to any cloning).

```python
# Add pink hat *before* any clones are made
hat = Polygon(Point(21,-21),Point(-10,-14),Point(-23,-36),Point(4,-30),Point(30,-46))
hat.setFillColor('pink')
fish.add(hat)
```

Now all our fishies have pink hats.
This illustrates the power of abstraction. When we abstract over a pattern, then a change to the pattern affects all instantiations of the pattern.

---

Drawbacks of Layers

Although Layers are powerful, they do not let us abstract over all the properties of our fish that we might want to change.

What if we want different fish to have different colors?

What if we want different fish to have larger or smaller eyes?

We cannot express these differences with Layers. Why not?

But we can express them with user-defined functions, a more powerful abstraction mechanism that we will study next time.

---

Digging Deeper

In this course we learn problem solving and happen to use Python to write our solutions.

Python has many details that initially seem daunting. Although you might not need to know them right now, you might need them eventually, so we want to make you aware of them. We'll put them in slides marked “Digging Deeper”:

- A class is a description of the shared characteristics (state and behaviors) of a set of objects.
- A class is like a mold for making objects.
- Example of classes include:
  - Canvas
  - Rectangle
  - Point
  - Polygon
  - Circle
  - Text
  - Image
  - Layer
- An object is made from a class by calling the constructor function with the same name as the class. E.g.,
  ```python
  Canvas(900, 600, 'cyan', 'sky')
  Rectangle(75, 300)
  Circle(50, Point(200, 100))
  ```
- Each object made from a class is an instance of the class.

Come back to this material whenever you want to understand Python details you might not have cared about earlier.
**Contracts**

- Every method/function has a **contract** or Application Program Interface (API) that specifies the behavior of the method/function.
- Every class has a contract/API for all of its methods.
- Any user of a method/function/class can expect that it will behave as described in the contract.
- Any implementer of the method/function/class must ensure that the it fulfills the contract.
- Example of contracts include:
  - `cs1graphics` API
  - Python built-in functions API
  - `math` module API

---

**The Canvas Contract**

<table>
<thead>
<tr>
<th>Canvas methods and arguments</th>
<th>Contract methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(w, h, bgColor, title, autoRefresh)</code></td>
<td><code>add(drawable)</code></td>
</tr>
<tr>
<td></td>
<td><code>remove(drawable)</code></td>
</tr>
<tr>
<td></td>
<td><code>clear()</code></td>
</tr>
<tr>
<td></td>
<td><code>open()</code></td>
</tr>
<tr>
<td></td>
<td><code>close()</code></td>
</tr>
<tr>
<td></td>
<td><code>saveToFile(filename)</code></td>
</tr>
<tr>
<td></td>
<td><code>setAutoRefresh(trueOrFalse)</code></td>
</tr>
<tr>
<td></td>
<td><code>refresh()</code></td>
</tr>
<tr>
<td></td>
<td><code>wait()</code></td>
</tr>
</tbody>
</table>

This is just the names of the methods and their arguments in the Canvas class contract. The full Contract specifies what each method does!

---

**Inheritance**

Some classes are generalizations of another class.

We arrange classes in an **inheritance hierarchy** in which more specific **subclasses** below inherit state and behavior from more general **superclasses** above.

---

**Drawable Class Inheritance Hierarchy 1**
Test your knowledge

1. Looking at the diagrams in Slide 3-3, how does the model for storing an object into a variable compare to that of other values we saw in the last lecture?
2. Can you give examples of state and behavior for cs1graphics objects?
3. What is the principle DRY and why it’s important to follow it?
4. Is the reference point always the center point of a shape? Explain.
5. If we were to change the state of a cloned object, would the state of the original object change as well? Explain.
6. What is the difference between rotating and flipping?
7. Were you able to figure out the needed transformations for the fish in slide 3-15?
8. Study the coordinates system in the slide 3-13. How is it different from the one for the Canvas object that we have been working so far?
9. Can you draw the content of the canvas in slide 3-14 before the `moveTo` statement?
10. Which class is more specific: Rectangle or Square?