Submission Title: Shape Drawing System

Contact Information: Nick Parlante <nick.parlante@cs.stanford.edu>

Java has many little features that I thought might be a problem in CS1, but in almost every case they worked out fine. Graphics was the one exception. You can do it subclassing off JComponent or whatever, but it's nice to have a system where you can start using graphics on the first day, and I think JComponent is too complex for that. Or put another way, graphics is the one area I'd love to see some standard approach -- all the other stuff (i/o, main(), Integer.parseInt(), ...) seemed bearable.

At Stanford we've used a simple, custom "shape" graphics layer (as many schools do) for the first 5 weeks or so, and it seems to work pretty well in our intro course with about 150 students. I'm just now teaching it for the second time, so it's too early to say anything definitive. It is certainly simple enough to use in the very early lectures, and we make some fairly nifty graphical assignments with it. I'll mention its main features and my impressions about Java graphics and CS1 from using it. (The API docs are available at http://www.stanford.edu/class/cs106a/handouts/HO09Graphics.pdf, or email me.)

-Our model is of a window to which persistent shapes are added. All the shapes -- rect, oval, image, string -- are defined by an x,y,width,height. All the shapes respond to setX(), setWidth() type messages to manipulate them -- they are in the typical tidy little graphics inheritance hierarchy. They react on-screen when sent messages. The shapes make a nice Blue-J like example for use in the very early lectures -- creating objects, sending them messages, seeing them react. There is no exposed notion of repaint() or paintComponent() -- it's far simpler than.

-To give a quick sense of the API, here's what typical client code looks like...

```
public static void main(String[] args) {
    DWindow window = new DWindow("Poodle", 500, 200);

    // add an oval to the window
    DOval body = new DOval(10, 40, 200, 100);
    window.add(body);

    // put a head picture on the oval
    DImage head = new DImage(150, 5, 80, 80, "hennessy.jpg");
    window.add(head);

    window.waitClick();

    // add a wand image, and animate it over towards the head
    DImage wand = new DImage(300, 10, "wand.png");
    window.add(wand);

    window.waitClick();

    DWindow.waitSeconds(0.1);
    wand.moveBy(-20, 0);
    DWindow.waitSeconds(0.1);
    wand.moveBy(-20, 0);
    ....
}
```

-Persistent on-screen shapes that respond to messages like getX(), setX() etc. make animation really easy. You do not have to build a whole data-model for what's on screen. Just keep pointers to the things you want to move and message them at will.

-We echo the "real" Java graphics style where possible -- origin upper left, use pixels, use the Java notion of Color, Font name, etc. Though the native Java style for many issues is not perfect pedagogically, I think it's still the best path to teach them the "real" style they will end up using later, unless it is really irretrievably awful.

-Single-threaded paradigm -- control starts in main() and runs forward in the normal way. Stepping forward, each setColor(), etc. appears to happen on screen synchronously. There is no notion or repaint(). When we switched to JComponent, this loss of sequentiality was a slight problem for the students. IMHO, the lack of a sequential model is one thing that makes JComponent tough to use in the first weeks, and hence the need for something simple in the early weeks.

-For input, we used a waitClick() message that blocks until a click, isMouseDown() that checks if the mouse is down but does not wait, and the standard JOptionPane to prompt for strings.

-You need some sort of static waitSeconds(0.5) -- crucial for any sort of animation

-With a custom window class, it's easy to provide a "make screenshot" command that saves a file. The window can have debug facilities to help diagnose common drawing problems.

-Having "image" be one of the supported primitive shapes that can go on screen is fantastic -- it opens up what you and the students can do with very little effort. It's easy for the students to play around making JPEGs or whatever and they can just pull them into their programs.

-Have some sort of textual on-screen shape is quite handy. It responds to getText(), setText(), so you can sprinkle them on screen for your program and it can setText() them as needed. Trick: rather than use font size, just use the notion of height. If it's 80 pixels high or whatever, it just figures out the right font size internally to fill out that height. Or maybe this is bad, since it's not the "real" way of the Graphics object.

-We did not have a line type, although it would be handy. Lines do not quite fit neatly into the x,y,width,height paradigm. I was content to leave that until we got to the real JComponent way, and then the whole space of drawing would be open to them.

-In the 2nd half of the quarter, we switched them to JComponent/subclassing style. The switch cost did not seem bad. Probably because the time required for our simple graphics was so low, and many of the concepts (co-ordinates, pixels, colors) transfered pretty well.

-There's a natural angle to allowing the students to create their own shape subclasses to work with the system. We used JComponent for that sort of subclassing, but clearly it could be done with the earlier shape stuff too.