Team: Ilan Biala, Nick Roberts, Varun Sharma
API: Processing

Requirements and Use Cases

1. Our wrapper for Processing must be **unusually simple**; it should not exercise any deep knowledge of Java, and should be approachable even for the absolute beginner.
    - Processing has risen to great popularity because its use requires next to no knowledge of Java. It accomplishes this by completely ignoring idioms and design patterns that are prevalent across the Java platform, instead providing a serviceable set of "system calls" the update the state of the canvas.
    - Therefore, using our API should *not* require the user to hook into the class hierarchy of Java beyond implementing the draw loop for their app. **We should do our best to expressly prohibit users from creating subtypes of the types exported by our API.** Any provision for subtyping would lead to a untenable surface area of Java to learn for beginners.
2. The API should, first and foremost, simplify the process of getting shapes and images on the screen, styled as the user wishes.
    - This is the point of a GUI library, after all.
    - This is Processing's biggest failing: styles are set mutably, and it's difficult and error-prone to draw on one part of the canvas.
    - We expect that this design process will involve the **reification of shapes and settings as Java classes**, where before this data was conveyed only via method calls (e.g. to get a rectangle on the screen, you had no choice but to call the *rect* method, which would immediately create a rectangle of the specified dimensions on the screen.)
3. Use cases, revised from our proposal.
    - **Nesting canvases should be as easy as nesting divs.** Drawing on a subcanvas of the main canvas should require the same sort of code as drawing on the main canvas. This is definitely not the case in the existing library, requiring code to set up and tear down mutable state.

      Example program: One of those screensavers where a box bounces around a screen, and the box itself contains some content.

- **Managing multiple shapes with similar settings should not require a whole lot of boilerplate code.** That is, reifying shapes and settings as Java objects should empower developers to reuse much of the setup code for settings.

  Example program: A solar system simulator, with a lot of similar (but not identical) objects orbiting a central sun.

- **It should be impossible to run setup code in the "wrong place."** The state machine corresponding to a Processing program escapes explanation. Trying to create images outside of a certain "magical" method (*setup*) leads to runtime exceptions. We modify the API for creating a processing app to close off pathways to error.

  Example program: A program that loads some images from some files and draws them to the screen.