Review
• Inheritance
• Overloading and overriding

Up until now ...
• All movement and sizing of graphical objects have been accomplished by modifying object coordinate values.

Going forward, we have a new option...
• We can leave coordinate values unchanged, and modify the coordinate system in which we draw.

Three ways to transform the coordinate system:
1. Scale
   — Magnify, zoom in, zoom out ...
2. Translate
   — Move axes left, right, up, down ...
3. Rotate
   — Tilt clockwise, tilt counter-clockwise ...

Scale
— All coordinates are multiplied by an x-scale-factor and a y-scale-factor.
— Stroke thickness is also scaled.

scale( factor );
scale( x-factor, y-factor );
void setup() {
  size(500, 500);
  smooth();
  noLoop();
  line(1, 1, 25, 25);
}

void setup() {
  size(500, 500);
  smooth();
  noLoop();
  scale(2,2);
  line(1, 1, 25, 25);
}

void setup() {
  size(500, 500);
  smooth();
  noLoop();
  scale(20,20);
  line(1, 1, 25, 25);
}

void setup() {
  size(500, 500);
  smooth();
  noLoop();
  scale(2,5);
  line(1, 1, 25, 25);
}

void setup() {
  size(500, 500);
  background(255);
  smooth();
  noLoop();
}

void draw() {
  grid();
  scale(2,2);
  grid();
}

void draw() {
  grid();
  fill(255);
  ellipse(50,50,40,30);
  scale(2,2);
  grid();
  fill(255);
  ellipse(50,50,40,30);
  }
Translate
– The coordinate system is shifted by the given amount in the x and y directions.

\[ \text{translate}(\text{x-shift, y-shift}); \]

Transformations can be combined
– Combine Scale and Translate to create a coordinate system with the y-axis that increases in the upward direction
– Axes can be flipped using negative scale factors

Rotate
– The coordinate system is rotated around the origin by the given angle (in radians).

\[ \text{rotate}(\text{radians}); \]
```java
void draw() {
    rotate( 25.0 * (PI/180.0) );
    grid();
}
```

```java
void draw() {
    translate(250.0, 250.0);
    //rotate( 25.0 * (PI/180.0) );
    //scale( 2 );
    grid();
}
```

```java
void draw() {
    translate(250.0, 250.0);
    rotate( 25.0 * (PI/180.0) );
    //scale( 2 );
    grid();
}
```

```java
void draw() {
    grid();
    fill(255);
    ellipse(50, 50, 40, 30);
    translate(250.0, 250.0);
    rotate( 25.0 * (PI/180.0) );
    scale(2);
    grid();
    fill(255);
    ellipse(50, 50, 40, 30);
}
```

Some things to note:
- Transformations do NOT work within `beginShape()`/`endShape()`;
- Transformations are cumulative.
- All transformations are cancelled prior to calling `draw()`.
- You can save and restore the current state of the coordinate system by calling
  - `pushMatrix()`;
  - `popMatrix();`
```java
String[] word = new String[] {
};

void setup() {
    size(500, 500);
    smooth();
    noLoop();
}

void draw() {
    background(255);
    translate(250, 250);
    for (int i = 0; i < word.length; i++) {
        text(word[i], 0.0, -150.0);
        rotate(radians(10));
    }
}
```

Each time through the loop an additional 10 degrees is added to the rotation angle. Total rotation accumulates.

```
float start = 0.0;
void setup() {
    size(500, 500);
    smooth();
}

void draw() {
    background(255);
    translate(250, 250);
    fill(0);
    rotate(start);
    for (int i = 0; i < word.length; i++) {
        text(word[i], 0.0, -150.0);
        rotate(radians(10));
    }
    start += radians(1);  
}
```

Each time through the loop an initial rotation angle is set, incremented, and saved in a global. Transformations reset each time draw() is called.

- Transformations work in 3D
  - Z is depth (into or out of the screen)
  - Negative z goes into the screen
  - translate(0, 0, -100);
  - Translate(0, 0, 100);
- If using 3D transformations
  - Change to 3D coordinates
  - Add a third argument to size to change the default renderer to P3D or OPENGL
  - import processing.opengl

A starfield using matrix transformations

```java
starfield.pde
```

We want to find the point where each star is projected on our viewport.

```
x' = x/z
y' = y/z
x'' = x/z
```

We use matrix transformations to transform the stars into our view.

```java
A starfield using matrix transformations
```