Welcome back!

Inheritance

- Every child object is an instance of its parent
- A parent object is not an instance of the child class

```
class A {}
class B extends A {}
A a1 = new A();
B b1 = new B();
A a2 = new B();
B b2 = new A();
```

Overloading

- Method overloading occurs when two methods have the same name but different parameters
- Happens at compile time

```
int a(int x) ;
int a(int x, int y);

int a(int x) ;
int a(float y);

int a(int x) ;
float a(int x);

int a(int x) ;
int a(int y);
```

Overriding

- Method overriding occurs when a child class redefines a parent method, but keeps the method signature unchanged – only change is in the method body
- Overriding happens at run time

```
class A {
  void a(int i){};
}

class B extends A {
  void a(int i){
    println();
  };
}
```

Assignment 3 Feedback

- Parameterization of functions
  - use parameters instead of global variables or hard-coded values
  - pass those variable values in as function arguments
- Designing parameters and functions is fundamental
- Class examples
  - it is NOT acceptable to take class examples and turn them in assignments
  - you must cite any code taken, including mine!
- Comment your functions
  - header
  - parameters
  - return value

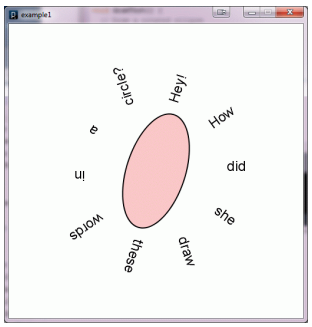
Object Oriented Design and Assignment 4

- Class variables keep track of the states of an object
- Methods assume all fields are always up-to-date
- Each method is responsible for one task and updating the related fields only
- Your Assignment 4 object
  - `x`, `y`, `size`, `angle`, `t`
  - `display()` draws the object (at current `x`, `y`, `size` and `angle`). It is NOT responsible for updating those variables!
  - `step()` updates the timer `t` every frame (`draw()` loop)
  - `move()` updates `x` and `y` based on current `t`

Getters and setters

- Instead of accessing data fields directly
  - `ball.x = 5;`
- Define methods to access them
  - `int getX(){return x;}`
  - `int getFoo(){return foo;}`
  - `void setX(int x){this.x = x;}`
  - `void setFoo(int foo){this.foo = foo;}`
  - **`ball.setX(5);`**

example1.pde

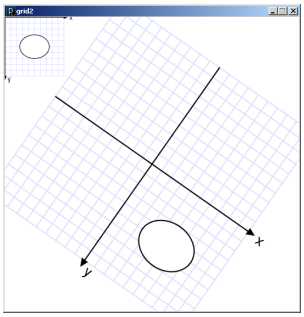


Up until now …

- *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.*



The commands that draw these two ellipses are identical.

What has changed is the coordinate system in which they are drawn.

Three ways to transform the coordinate system:

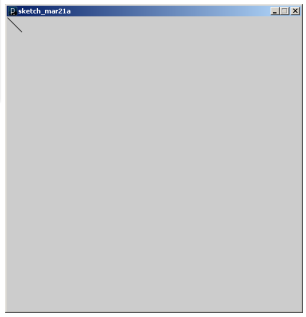
1. **Scale**
   - Magnify, zoom in, zoom out …
2. **Translate**
   - Move axes left, right, up, down …
3. **Rotate**
   - Tilt clockwise, tilt counter-clockwise …

Scale

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

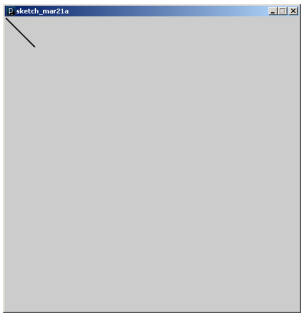
```
scale(factor);
scale(x-factor, y-factor);
```

```
void setup() {
  size(500, 500);
  noLoop();

  line(1, 1, 25, 25);
}
```



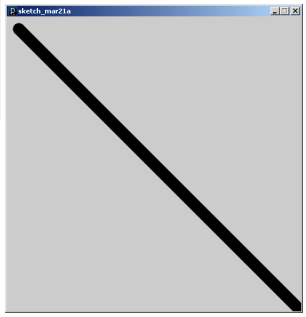
example2.pde

```
void setup() {
  size(500, 500);
  noLoop();

  scale(2,2);
  line(1, 1, 25, 25);
}
```



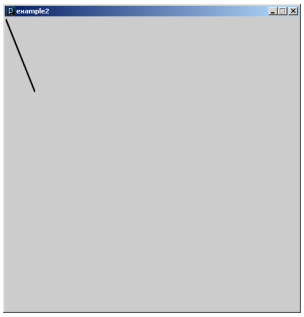
example2.pde

```
void setup() {
  size(500, 500);
  noLoop();

  scale(20,20);
  line(1, 1, 25, 25);
}
```



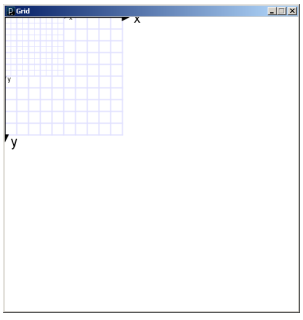
example2.pde

```
void setup() {
  size(500, 500);
  noLoop();

  scale(2,5);
  line(1, 1, 25, 25);
}
```



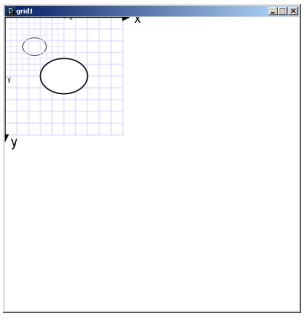
example2.pde

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

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



grid1.pde

```
void draw() {
  grid();
  fill(255);
  ellipse(50,50,40,30);

  scale(2,2);
  grid();
  fill(255);
  ellipse(50,50,40,30);
}
```



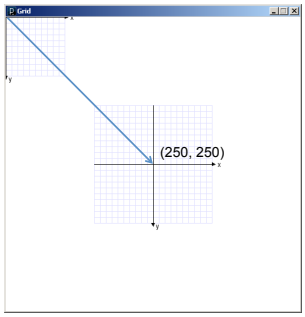
grid1.pde

Translate

- The coordinate system is shifted by the given amount in the x and y directions.

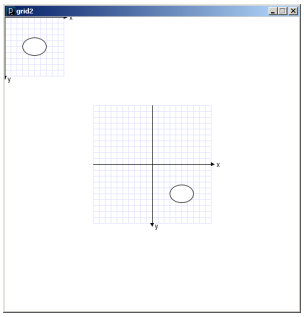
```
translate(x-shift, y-shift);
```

```
void draw() {
  grid();
  translate(250,250);
  grid();
}
```



grid2.pde

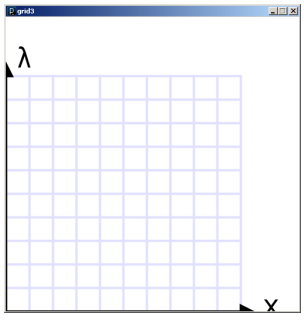
```
void draw() {
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);

  translate(250, 250);
  grid();
  fill(255);
  ellipse(50, 50, 40, 30);
}
```



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

```
void draw() {
  translate(0,height);
  scale(4,-4);
  grid();
}
```



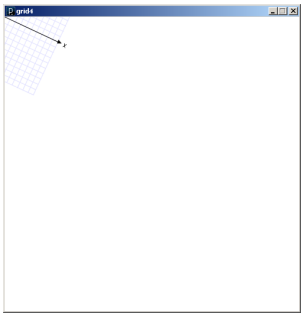
grid3.pde

Rotate

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

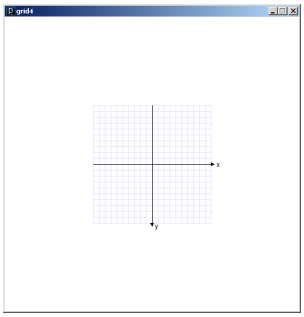
```
rotate(radians);
```

```
void draw() {
  rotate(25.0 * (PI/180.0));
  grid();
}
```



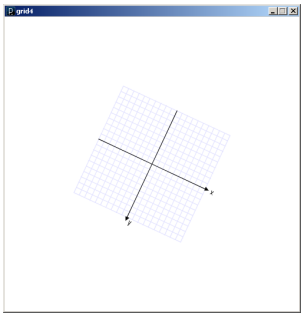
grid4.pde

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



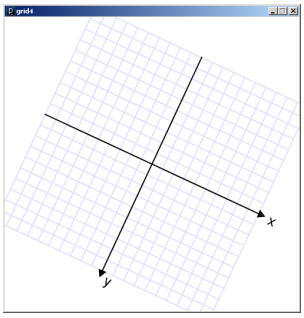
grid4.pde

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



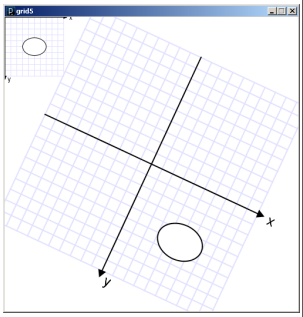
grid4.pde

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



grid4.pde

```
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);
}
```

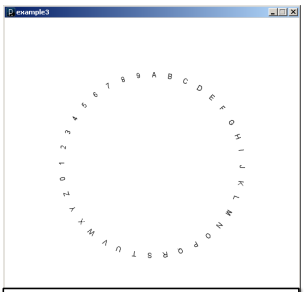


grid5.pde

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();

```
String[] word = new String[]
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S
","T","U","V","W","X","Y","Z","0","1","2","3","4","5","6","7","8","9"};

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

void draw() {
  background(255);
  translate(250,250);

  fill(0);
  for (int i=0; i<word.length; i++) {
    text(word[i], 0.0, -150.0);
    rotate(radians(10));
  }
}
```

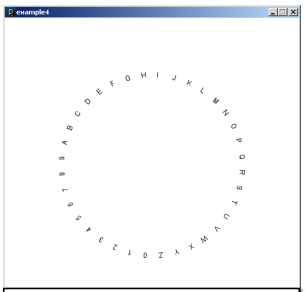


Each time through the loop an additional 10 degrees is added to the rotation angle.

Total rotation accumulates.

example3.pde

```
String[] word = new String[]
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T",
"U","V","W","X","Y","Z","0","1","2","3","4","5","6","7","8","9"};

float start = 0.0;

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

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);
}
```

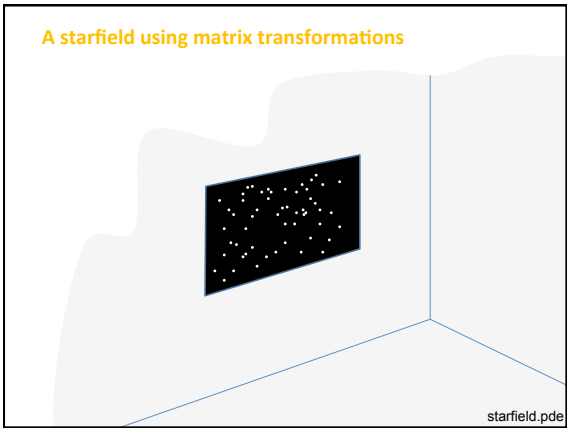


Each time through the loop an initial rotation angle is set, incremented, and saved in a global.

Transformations reset each time draw() is called.

example4.pde

A starfield using matrix transformations



starfield.pde