Welcome back!

Inheritance

- Every child object is an instance of its parent
  ```java
  class A {}
  class B extends A {}
  A a1 = new A();
  B b1 = new B();
  A a2 = new B();
  B b2 = new A();
  ```
- A parent object is not an instance of the child class

Overloading

- Method overloading occurs when two methods have the same name but different parameters
- Happens at compile time
  ```java
  int a(int x);  
  int a(int x, int y);
  int a(int x);  
  int a(float 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
  ```java
  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
  • \texttt{ball.x} = 5;

– Define methods to access them
  • \texttt{int getX()}\{return x;\}
  • \texttt{int getFoo()}\{return foo;\}
  • \texttt{void setX(int x)}\{this.x = x;\}
  • \texttt{void setFoo(int foo)}\{this.foo = foo;\}
  • \texttt{ball.setX(5)};

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.

\texttt{scale(factor);}  
\texttt{scale(x-factor, y-factor);}
```java
void setup() {
  size(500, 500);
  noLoop();
  line(1, 1, 25, 25);
}

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

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

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

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

grid1.pde
```
**Translate**
- The coordinate system is shifted by the given amount in the x and y directions.

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

**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).

\[
\text{rotate}(\text{radians});
\]
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[]{

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

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));
  }
}

A starfield using matrix transformations

starfield.pde

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

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix}
= \begin{bmatrix}
  M_{11} & M_{12} & M_{13} & 0 \\
  M_{21} & M_{22} & M_{23} & 0 \\
  M_{31} & M_{32} & M_{33} & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]