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();
    };
}

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

Assignment 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

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

grid1.pde

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

grid1.pde

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

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

Rotate

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

rotate(radans);
Some things to note:
• Transformations do NOT work in the middle of 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();

Example
• squareGrid
• squares
```java
String[] words = new String[] {

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

void draw() {
    background(255);
    translate(250, 250);
    fill(0);
    for (int i = 0; i < words.length; i++) {
        text(words[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.

Transformations reset each time draw() is called.

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

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} = \begin{bmatrix}
  z & x & z \\
  x & z & x \\
  1 & 1 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]

A starfield using matrix transformations

```
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 < words.length; i++) {
        text(words[i], 0.0, -150.0);
        rotate(radians(10));
    }
    start += radians(1);
}
```