Graphics with libGDX

Dr. Andrew Vardy

Adapted from the following sources:

libGDX slides by Jussi Pohjolainen (Tampere Unversity of Applied Sciences)

transformation slides by Dr. Paul Gillard (Memorial University)

ENGI 5895

Software Design

Memorial University

Introduction

- The goal here is to illustrate some concepts in computer graphics
- The tool we will use is libGDX, a cross-platform game development environment
  - libGDX library provides six interfaces to abstract away platform details
    - Application, Files, Input, Net, Audio, **Graphics**
  - The graphics library wraps OpenGL ES or WebGL
    - OpenGL has emerged as a standard library for graphics; ES = Embedded Systems
  - OpenGL ES available on Android and iOS
  - WebGL is a Javascript API that conforms to OpenGL ES

Cross-platform

- libGDX targets Desktop, Android, HTML5, and iOS
  - Desktop via LWJGL (Lightweight Java Game Library)
  - Android via Android SDK
  - HTML5 via GWT (Google Web Toolkit)
    - Java -> Javascript
  - iOS via RoboVM
    - Java -> Objective-C

- For an alternate intro to libGDX try “2D Game Development with libGDX” from Udacity

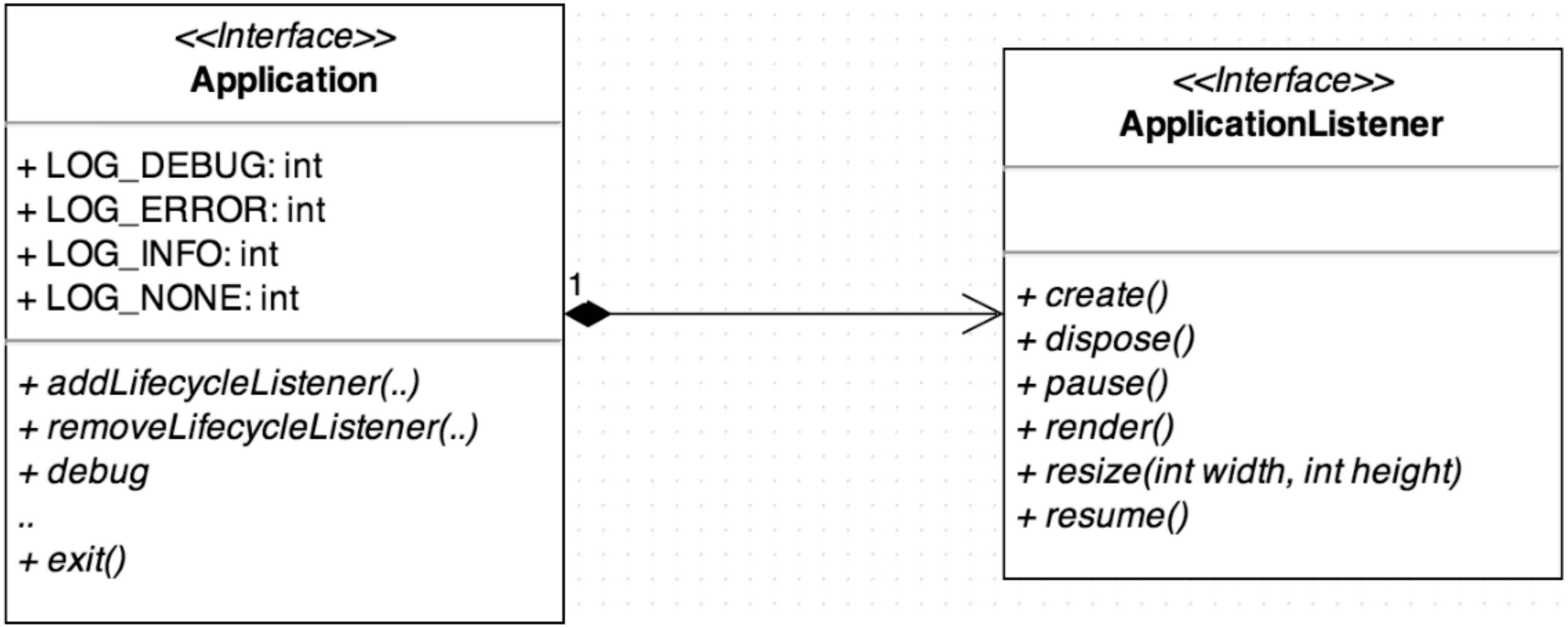
interface Application

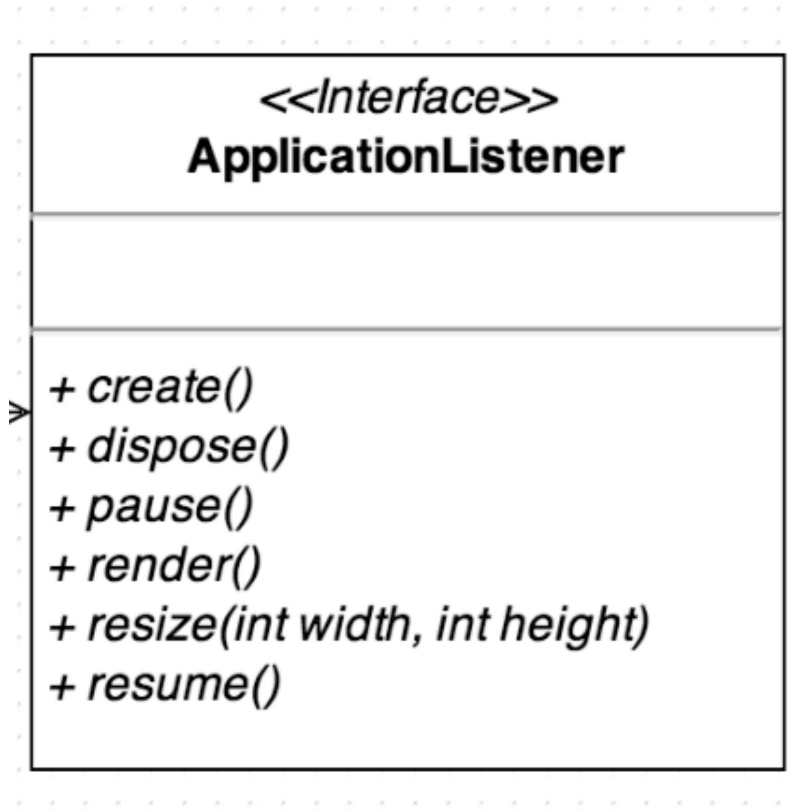
- According to the libGDX API:

  “An Application is the main entry point of your project. It sets up a window and rendering surface and manages the different aspects of your application, namely Graphics, Audio, Input and Files. Think of an Application being equivalent to Swing's JFrame or Android's Activity.”

- Application is an interface which is implemented by one of the following:
  - JglfwApplication (Desktop)
  - AndroidApplication (Android)
  - GwtApplication (HTML5)
  - IOSApplication (iOS)

- The Application interface and the corresponding XXXApplication (e.g. AndroidApplication) classes exist and don't need to be modified
- Create your own app by implementing ApplicationListener





`create()`
Called when the `Application` is first created.

`dispose()`
Called when the `Application` is destroyed.

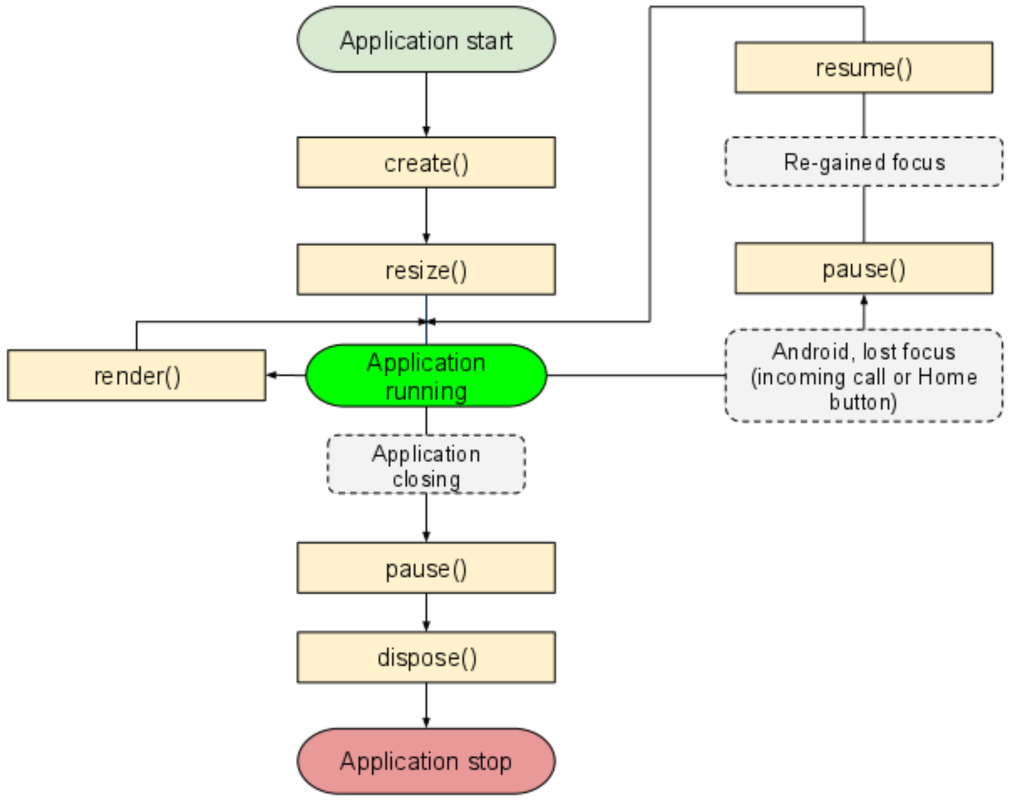
`pause()`
Called when the `Application` is paused, usually when it's not active or visible on screen.

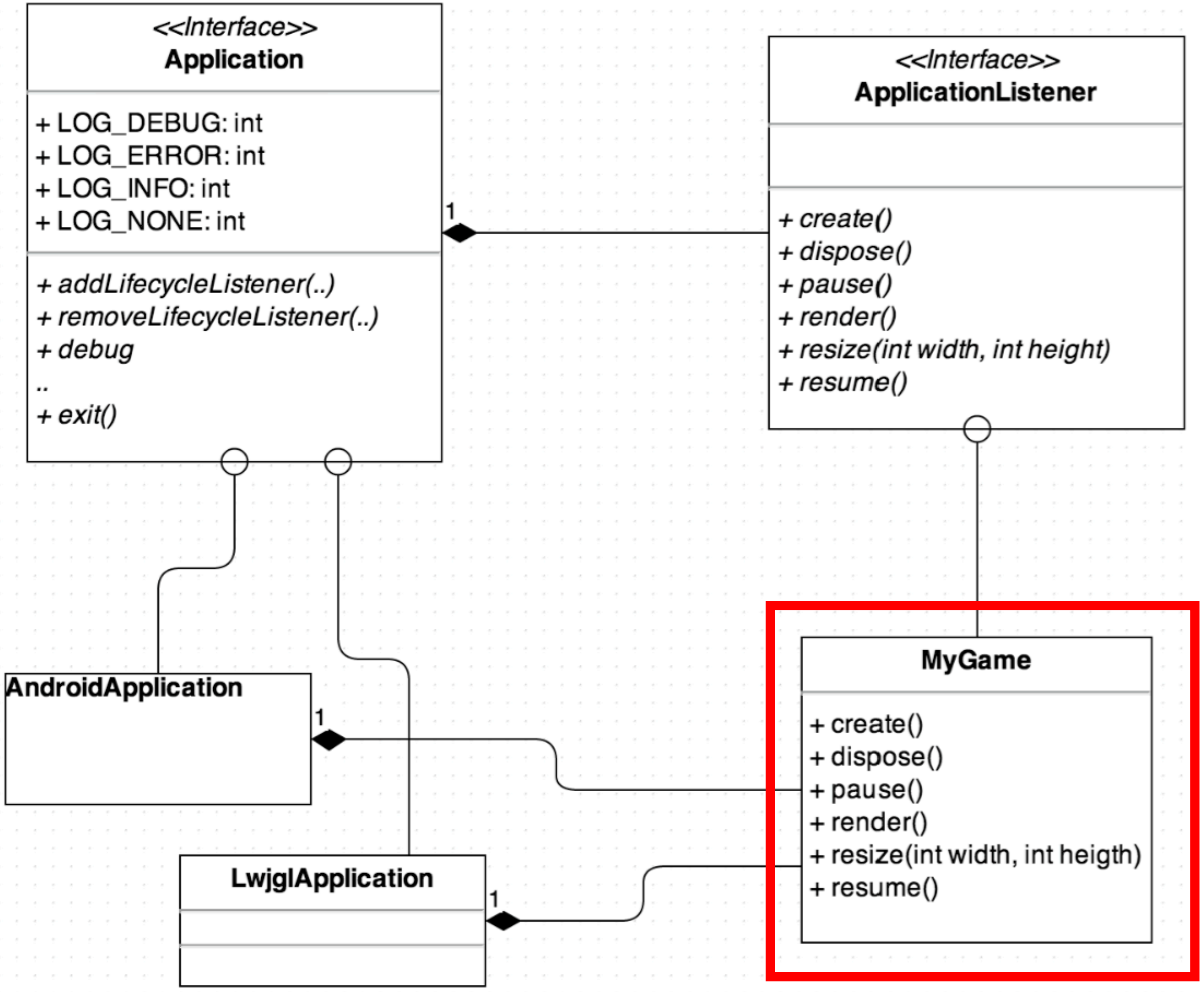
`render()`
Called when the `Application` should render itself.

`resize(int width, int height)`
Called when the `Application` is resized.

`resume()`
Called when the `Application` is resumed from a paused state, usually when it regains focus.

App Lifecycle





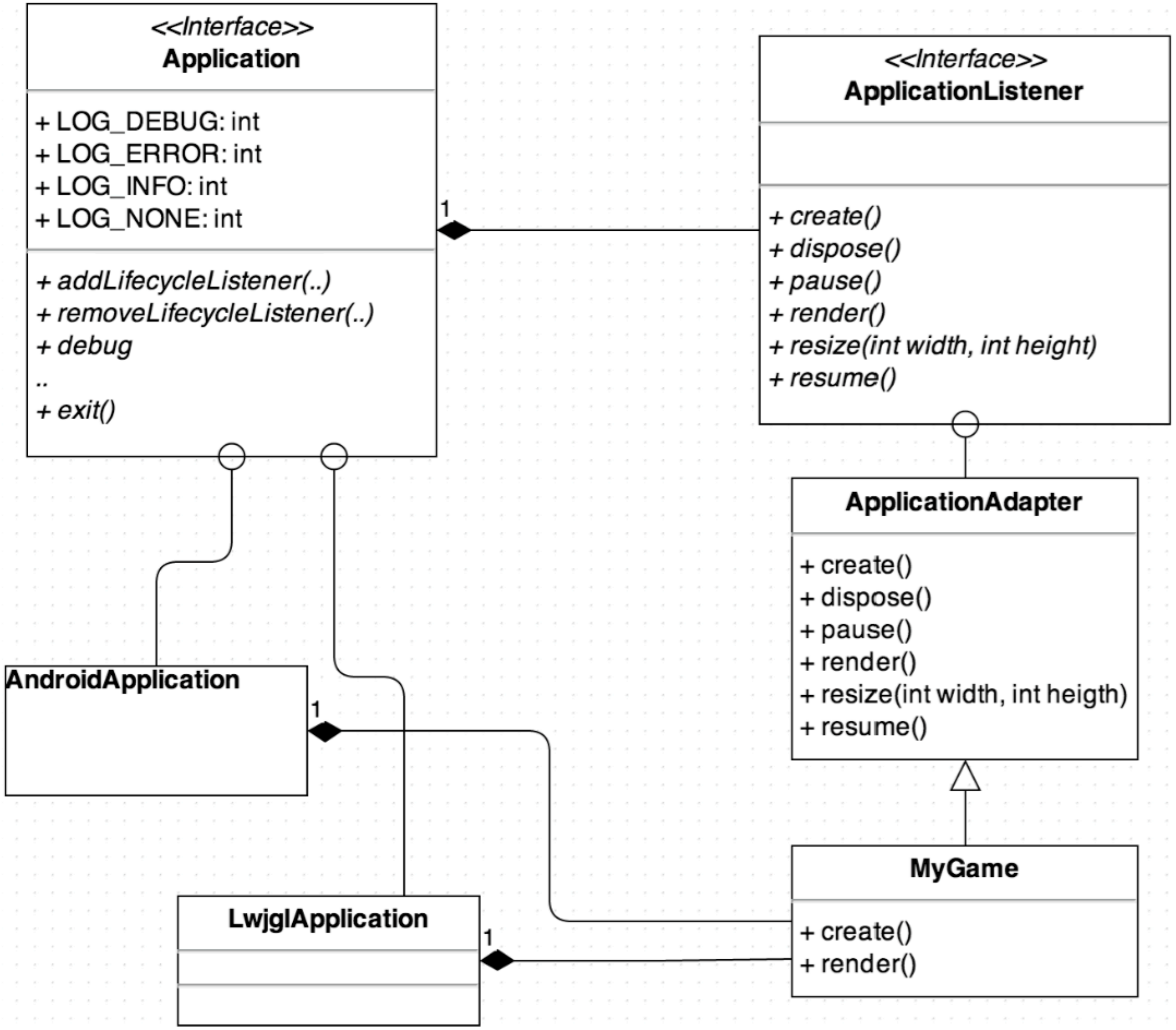
This is what you have to implement to make your own game/app

Listeners and Adapters (Java Concept)

- Usually a “Listener” in Java responds to events
  - e.g. in Swing interface MouseListener defines the following methods:

    mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased

- This is really just another flavour of the Observer pattern
- But what if you only care about “mouseClicked” events? Your concrete Listener has to define all 5 of the methods above
- To avoid this the abstract class MouseAdapter is defined which provides empty methods for all of these
- Now your concrete Listener can extend MouseAdapter instead of implementing MouseListener and you define only the methods you want


<<Interface>>
Application
+ LOG_DEBUG: int
+ LOG_ERROR: int
+ LOG_INFO: int
+ LOG_NONE: int
+ addLifecycleListener(..)
+ removeLifecycleListener(..)
+ debug
..
+ exit()
1
<<Interface>>
ApplicationListener
+ create()
+ dispose()
+ pause()
+ render()
+ resize(int width, int height)
+ resume()
ApplicationAdapter
+ create()
+ dispose()
+ pause()
+ render()
+ resize(int width, int heigth)
+ resume()
AndroidApplication
1
LwjglApplication
1
MyGame
+ create()
+ render()

About Starter Classes

- For **each platform** (iOS, Android, Desktop ..) a starter class must be written
- Starter classes are platform dependent
- We will focus on
  - **Desktop** (LWJGL)
  - **Android**

Starter Classes: Desktop

```java
// This is platform specific: Java SE
public class DesktopStarter {
   public static void main(String[] argv) {
      LwjglApplicationConfiguration config
             = new LwjglApplicationConfiguration();
      config.title = “…”;
      config.width = 480;
      config.heigth = 320;
      new LwjglApplication(new MyGame(), config);
   }
}
```

Starter Classes: Android

```java
import android.os.Bundle;
import com.badlogic.gdx.backends.android.AndroidApplication;
import com.badlogic.gdx.backends.android.AndroidApplicationConfiguration;

// This is platform specific: Android
// No main
public class AndroidLauncher extends AndroidApplication {
    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();
        MyGame game = new MyGame();

        initialize(game, config);

        if(this.getApplicationListener() == game) {
            this.log("test", "success");
        }
    }
}
```

Android Manifest

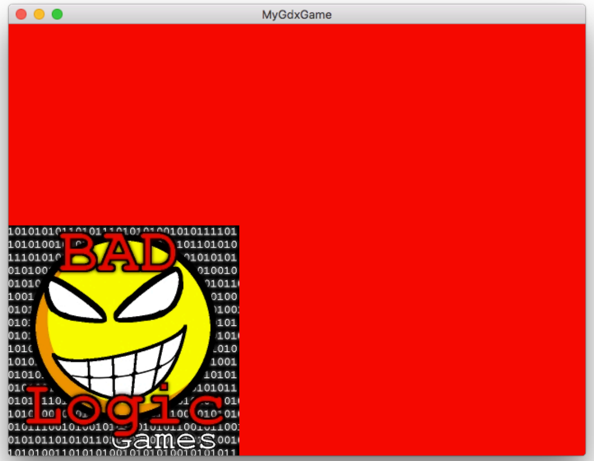
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mygdx.game.android"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="20" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/GdxTheme" >
        <activity
            android:name="com.mygdx.game.android.AndroidLauncher"
            android:label="@string/app_name"
            android:screenOrientation="landscape"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Android Permissions

- Add permissions if your android app requires certain functionality
  - `<uses-permission android:name=”android.permission.RECORD_AUDIO”/>`
  - `<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE”/>`
  - `<uses-permission android:name=”android.permission.VIBRATE”/>`
- Add these to manifest file
- See
  - `http://developer.android.com/guide/topics/manifest/manifest-intro.html#perms`

Project Setup

- Rather than craft your own Starter Classes, I recommend using the project generator (gdx-setup.jar) referred to here:
  - https://github.com/libgdx/libgdx/wiki/Project-Setup-Gradle
- To run on the desktop:
  - Click Run -> “Edit Configurations...”
  - Click ”+” in upper-left corner
  - Select “Gradle” (the build system)
    - Change “Name” to Desktop
    - Set “Tasks” to desktop:run
  - With “Desktop” selected, hit the play button
  - If everything works, this should appear:



- The following class will be defined in core/src/com.mygdx.game

```
public class MyGdxGame extends ApplicationAdapter {
  SpriteBatch batch;
  Texture img;

  @Override
  public void create () {
    batch = new SpriteBatch();
    img = new Texture("badlogic.jpg");
  }

  @Override
  public void render () {
    Gdx.gl.glClearColor(1, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    batch.begin();
    batch.draw(img, 0, 0);
    batch.end();
  }

  @Override
  public void dispose () {
    batch.dispose();
    img.dispose();
  }
}
```

Located in android/assets (other platforms link to this dir)

Two lines of actual OpenGL;
Colour specified as red, green, blue, alpha (opacity)

Entities (here an image) drawn in a batch to optimize them for processing by the GPU

Isn't Java supposed to have garbage collection (GC)? Unfortunately, GC is unpredictable and costly. If large resources (e.g. images) were subject to GC it could cause game lag. Also, objects allocated outside the JVM (e.g. by calling C++ code) are **not** GC'd.

Example: “A Simple Game”

- This example was adapted from

  https://github.com/libgdx/libgdx/wiki/A-simple-game

- A simple zen-like game with no end:
  - Catch raindrops with a bucket on the bottom of the screen.
  - Raindrops spawn randomly at the top of the screen every second and accelerate downwards.
  - Player drags the bucket horizontally via the mouse/touch or by the keyboard using left and right cursor keys.

```
package com.mygdx.game;

import /* NOT SHOWN */

public class Drop extends ApplicationAdapter {
   private Texture dropImage;
   private Texture bucketImage;
   private SpriteBatch batch;
   private OrthographicCamera camera;
   private Sprite bucket;
   private Array<Sprite> raindrops;
   private long lastDropTime;

   // The width and height of the screen --- assumed not
   // to change (otherwise define resize).
   private int width, height;

   ...
```

Images to be loaded from files

Having a camera enables manipulating the view independent of the world. Two choices:
- PerspectiveCamera: Distant objects will appear smaller. Good for 3D.
- OrthographicCamera: The scene is projected onto a plane. Good for 2D.

```java
@Override
public void create() {
  // load the images for the droplet and the bucket, 64x64 pixels each
  dropImage = new Texture(Gdx.files.internal("droplet.png"));
  bucketImage = new Texture(Gdx.files.internal("bucket.png"));

  width  = Gdx.graphics.getWidth();
  height = Gdx.graphics.getHeight();

  // create the camera and the SpriteBatch
  camera = new OrthographicCamera();
  camera.setToOrtho(false, width, height);
  batch = new SpriteBatch();

  // create a Sprite to logically represent the bucket
  bucket = new Sprite(bucketImage);
  bucket.setX(width / 2 - bucket.getWidth() / 2); // center the bucket horizontally
  bucket.setY(20); // bottom left corner of the bucket is 20 pixels above the bottom screen edge

  // create the raindrops array and spawn the first raindrop
  raindrops = new Array<Sprite>();
  spawnRaindrop();
}
```

This Sprite is created with new. GC will still happen as normal and as long as the objects created are small, there shouldn't be a big impact.

```
private void spawnRaindrop() {
   Sprite raindrop = new Sprite(dropImage);
   raindrop.setX(MathUtils.random(0, width-raindrop.getRegionWidth()));
   raindrop.setY(height);
   raindrops.add(raindrop);
   lastDropTime = TimeUtils.nanoTime();
}
```

render method: part 1 / 3

```
@Override
public void render() {
  // clear the screen with a dark blue color. The arguments to glClearColor are the red, green blue and alpha component
  // in the range [0,1] of the color to be used to clear the screen.
  Gdx.gl.glClearColor(0, 0, 0.2f, 1);
  Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

  // tell the camera to update its matrices.
  camera.update();

  // tell the SpriteBatch to render in the coordinate system specified by the camera.
  batch.setProjectionMatrix(camera.combined);

  // begin a new batch and draw the bucket and all drops
  batch.begin();
  bucket.draw(batch);
  for(Sprite raindrop: raindrops) {
        raindrop.draw(batch);
  }
  batch.end();

  ...
```

Not really necessary here since the camera is not changing

render method: part 2 / 3

```
...

// process user input
if(Gdx.input.isTouched())  {
   Vector3 touchPos = new Vector3();
   touchPos.set(Gdx.input.getX(),  Gdx.input.getY(), 0);
   camera.unproject(touchPos);
   bucket.setX(touchPos.x - bucket.getWidth() / 2);
}
if(Gdx.input.isKeyPressed(Keys.LEFT))  bucket.translateX( -400 * Gdx.graphics.getDeltaTime());
if(Gdx.input.isKeyPressed(Keys.RIGHT))  bucket.translateX( 400 * Gdx.graphics.getDeltaTime());

// make sure the bucket stays within the screen bounds
if(bucket.getX() < 0) bucket.setX(0);
if(bucket.getX() > width - bucket.getWidth()) bucket.setX(width - bucket.getWidth());

// check if we need to create a new raindrop
if(TimeUtils.nanoTime() - lastDropTime > 1000000000) spawnRaindrop();

...
```

render method: part 3 / 3

```
  ...

  // move the raindrops, remove any that are beneath the bottom edge of
  // the screen or that hit the bucket.
  Iterator<Sprite> iter = raindrops.iterator();
  while(iter.hasNext()) {
    Sprite raindrop = iter.next();
    raindrop.translateY(-200 * Gdx.graphics.getDeltaTime());
    if(raindrop.getY() + raindrop.getHeight() < 0) iter.remove();
    if(raindrop.getBoundingRectangle().overlaps(bucket.getBoundingRectangle())) {
      iter.remove();
    }
  }
}
```

```
 @Override
  public void dispose() {
    // dispose of all the native resources
    dropImage.dispose();
    bucketImage.dispose();
    batch.dispose();
  }
}
```

What needs to be disposed of? Any classes that implement **Disposable.**

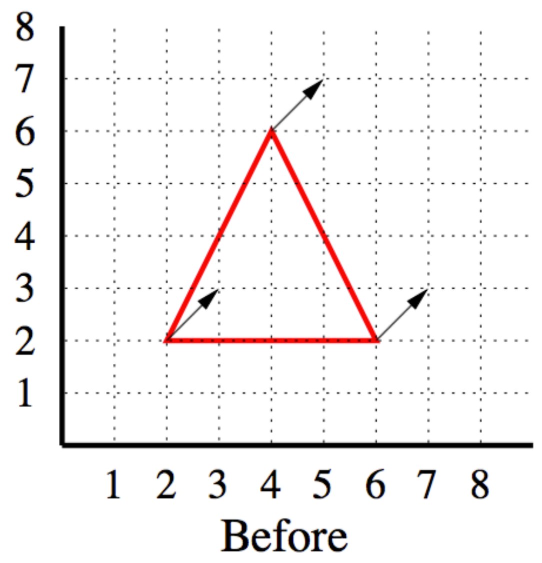
Vectors and transformations

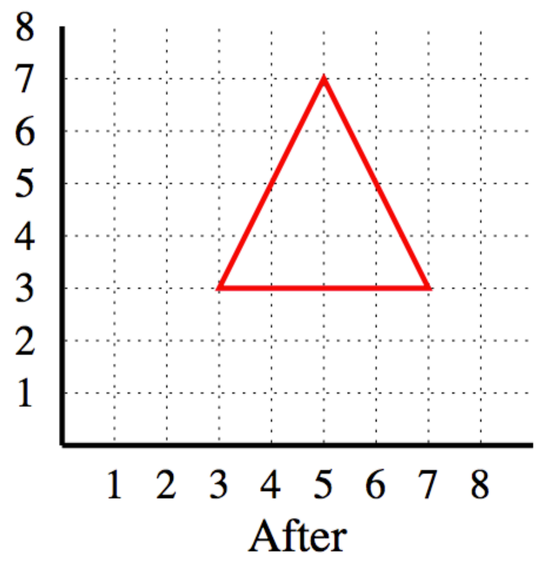
For the case of OpenGL, everything that we want to visualize must be composed of primitives. To display anything interesting we will have to take our basic primitives and **transform** them to form the object of interest. Therefore, transformations are fundamental to computer graphics. We begin with the most common transformations (translation, rotation, and scaling) in 2-D...

Translation

If we represent the vertices of primitives as vectors, translation is easily accomplished by vector addition.

Example: Given a triangle with a set of vertex vectors $V = \{(2,2), (4,6), (6,2)\}$ and a displacement vector $T = (1,1)$ the resultant vertex set for the triangle is $V' = \{(3,3), (5,7), (7,3)\}$



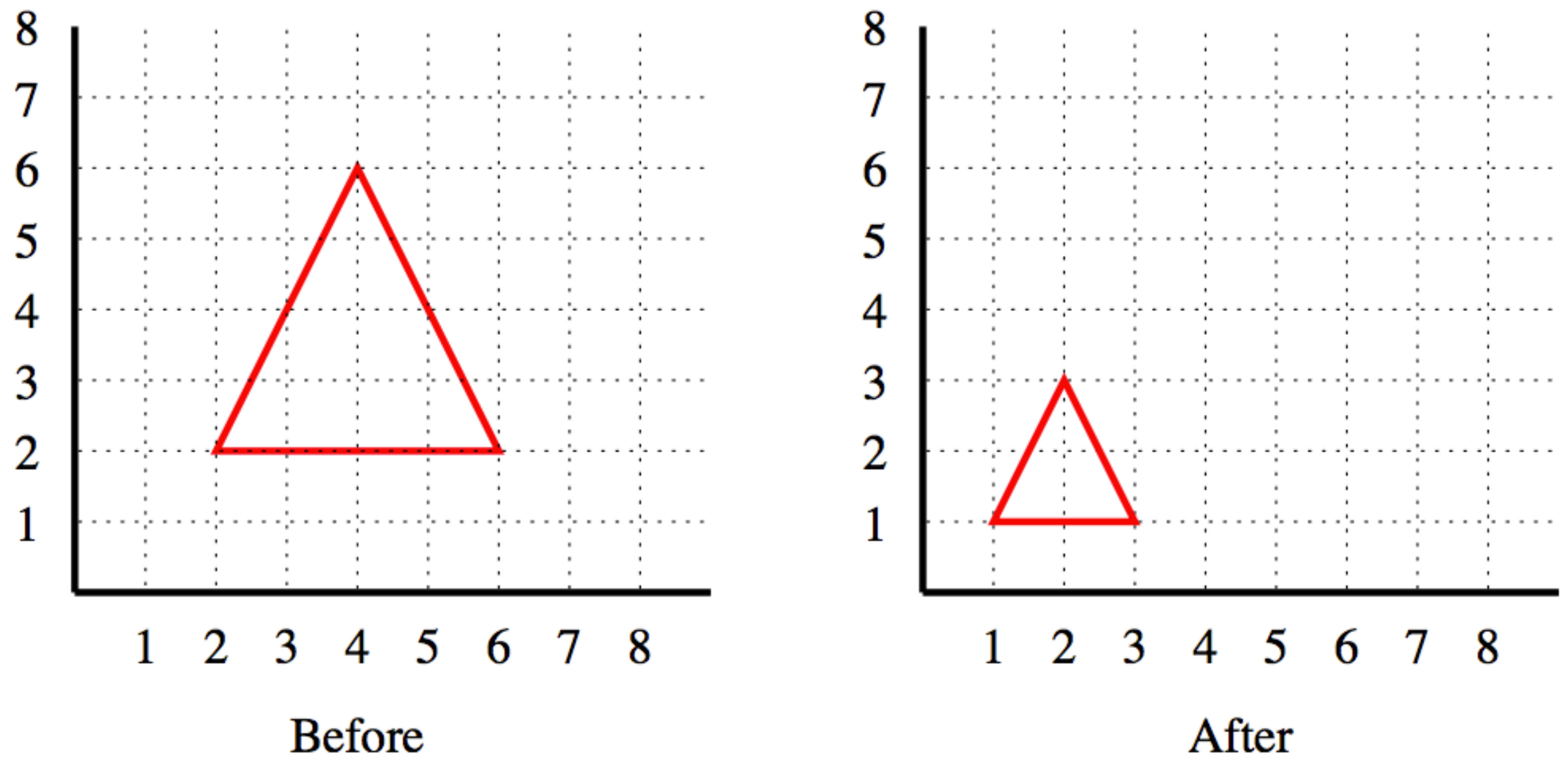


$$V' = V + T$$

Scaling

The vectors can be uniformly scaled by simply multiplying each vector by a scalar constant. Note that the full vector (from the origin to the point) will be scaled, so the image will change in both size and position.

Scaling by 0.5



A differential scaling is also possible, where x and y are multiplied by two different factors s_x and s_y.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

It will be convenient to write this as a matrix multiplication.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

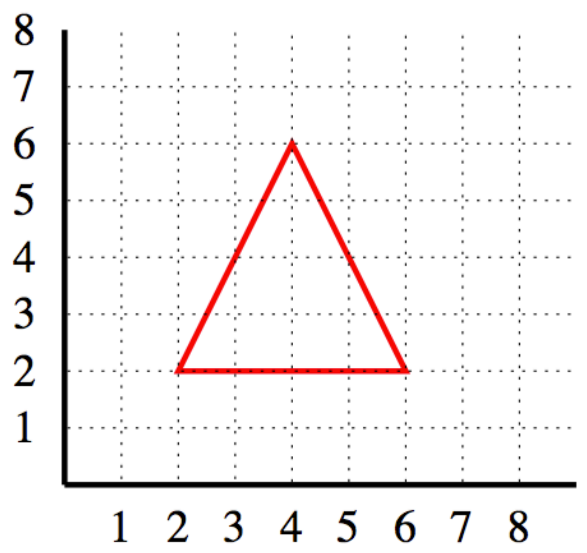
This can be written as

$$V' = S \cdot V$$

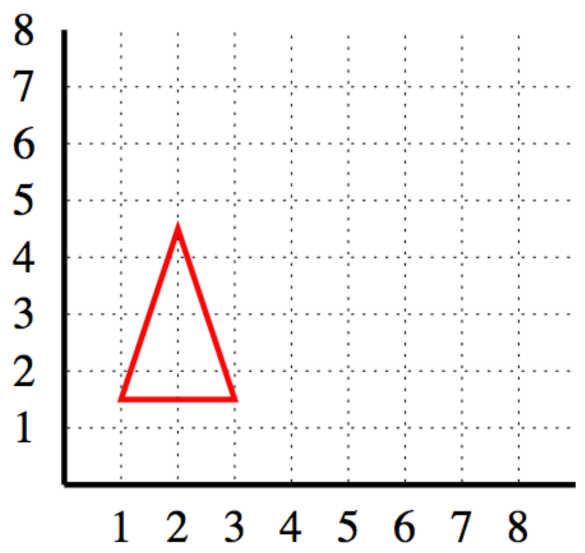
where x and y are the components of V.

For differential scaling the size, position, and shape may all change. The following is the figure seen earlier scaled by the matrix

$$S = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.75 \end{bmatrix}$$



Before

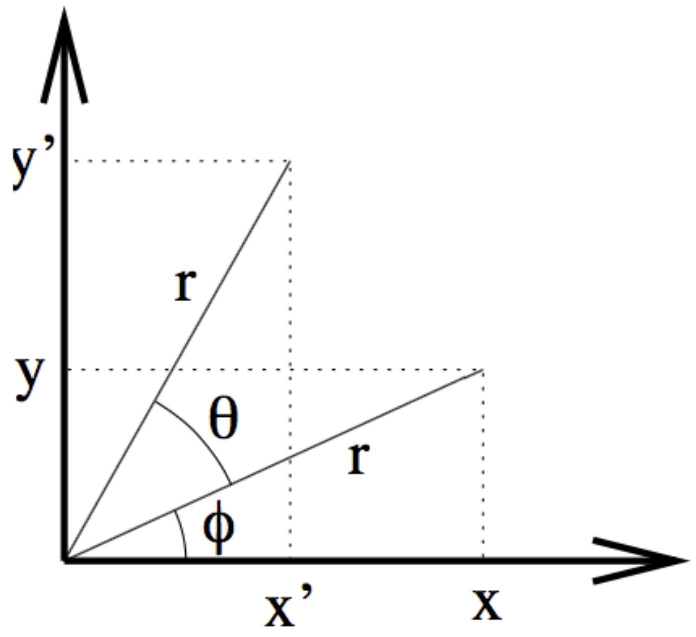


After

Rotation

Assume we have a vertex at (x, y) which is to be rotated counterclockwise about the origin by an angle θ. In polar coordinates, this vertex is at (r, ϕ). We can express the Cartesian coordinates in these terms:

$$x = r\cos\phi$$
$$y = r\sin\phi$$



Now the rotation by θ can be understood as an addition of angles:

$$x' = r\cos(\theta + \phi)$$
$$y' = r\sin(\theta + \phi)$$

We can now make use of the following trigonometric identities:

$\cos(a+b) = \cos a \cos b - \sin a \sin b$

$\sin(a+b) = \sin a \cos b + \cos a \sin b$

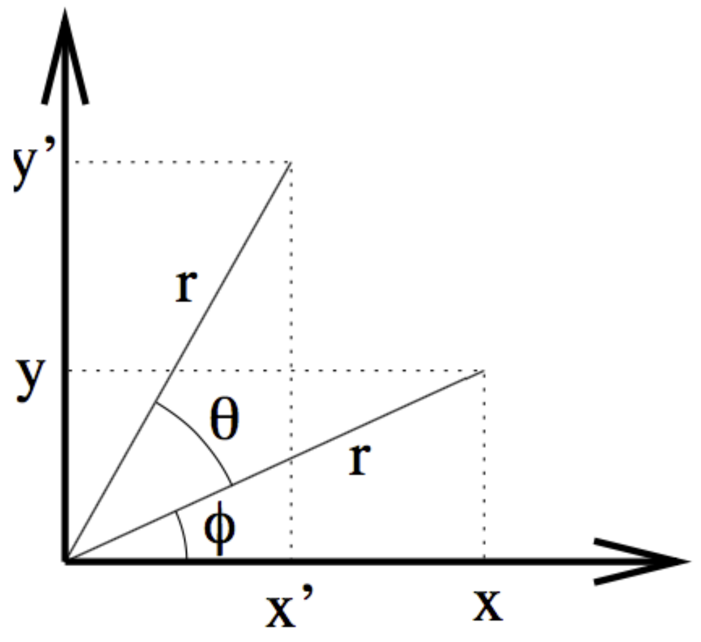
To obtain, $x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

In vector form, this is written as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

or $V' = R(\theta) \cdot V$



The three types of transformations have the form:

Translation	$V' = V + T$
Scaling	$V' = S \cdot V$
Rotation	$V' = R \cdot V$

The results of successive rotations and scalings can be obtained by matrix multiplication, but translation cannot.

We may have a long sequence of transforms to apply to a vertex V. For example,

$$V' = S_0 R_0 S_1 (R_1 S_2 V + T_0) + T_1$$

$$V' = S_0R_0S_1(R_1S_2V + T_0) + T_1$$

The same transforms will be applied to many vertices so they should be done quickly. Observe that the matrices can be composed:

$$V' = M_0(M_1V + T_0) + T_1$$

where $M_0 = S_0R_0S_1$ and $M_1 = R_1S_2$. This improves efficiency somewhat, but the translations prevent us from optimizing any more than this.

If translation could be described as a matrix multiplication then we could combine all transformations into a single matrix M,

$$V' = MV$$

Homogeneous coordinates

By using homogeneous coordinates we can use matrix multiplication to implement all three basic transformations.

With homogeneous coordinates, a third coordinate is added to a point; point (x, y) is represented as (x, y, W).

If we set W to be 1, (the point would be $(x/W, y/W, 1)$) then we have homogenized the point.

Translation can now be performed with matrix multiplication.

Translation by (d_x, d_y) would be represented as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The transformation for scaling remains much the same:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

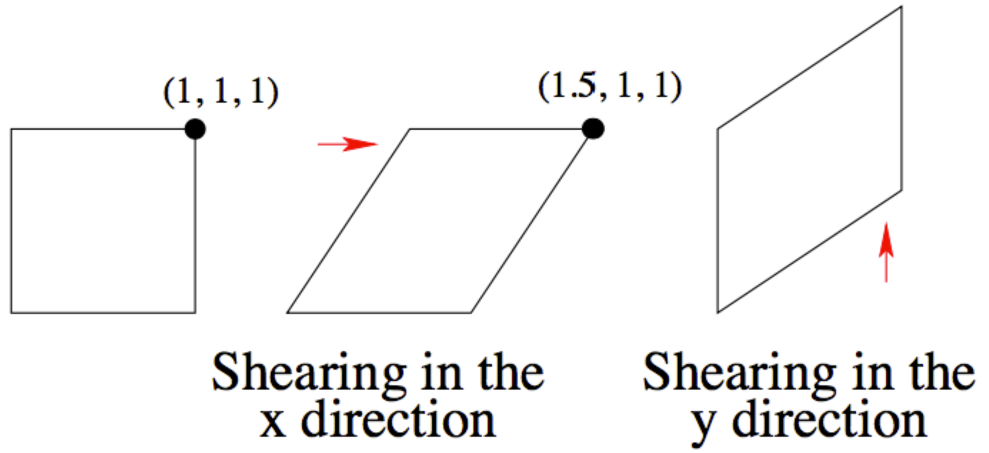
...and for rotation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Successive translations, scalings, and rotations can now be implemented with matrix multiplication.

Remember, though, that if the order of transformations is changed the result may also change. (Matrix multiplication is not commutative — $AB \neq BA$, in general.)

Shearing is another common transformation. Shearing distorts a primitive by "pushing" it in one direction, as shown:



The matrix for shearing is as follows, where a_x and a_y are the factors for shearing in the x and y directions, respectively.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a_x & 0 \\ a_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The general form for transformations derived from translation, scaling, rotation, and shearing is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where the r_{ij} correspond to some combination of rotation, scaling, and shearing, and the t's correspond to translation.

The general form for transformations derived from translation, scaling, rotation, and shearing is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where the r_{ij} correspond to some combination of rotation, scaling, and shearing, and the t's correspond to translation.

Recall again that the order of operations is important in applying transformations.

For example, if we have a point (or vector) V and wish to apply translation, then scaling, then rotation, then translation again, we would perform the operations as $T(R(S(T(V))))$

Three dimensional transformations

All of the transformations we have seen have similar representations in 3 dimensions.

In fact, all operations can be combined to give a general transformation of the form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Example: “Zen Garden”

- ”A Simple Game”
  - Rain falls randomly from the sky
  - User controls a bucket to catch raindrops
- Here we invert this setup
  - User controls a cloud in the sky, from which raindrops fall
  - If rain falls on a tree, the tree grows

- Download the code for this example (see notes page)
- The main files:
  - ZenGardenGame (extends ApplicationAdapter)
  - Tree (interface)
  - SimpleTree (implements Tree)
  - RecursiveTree (implements Tree)

- First, consider SimpleTree’s draw method…

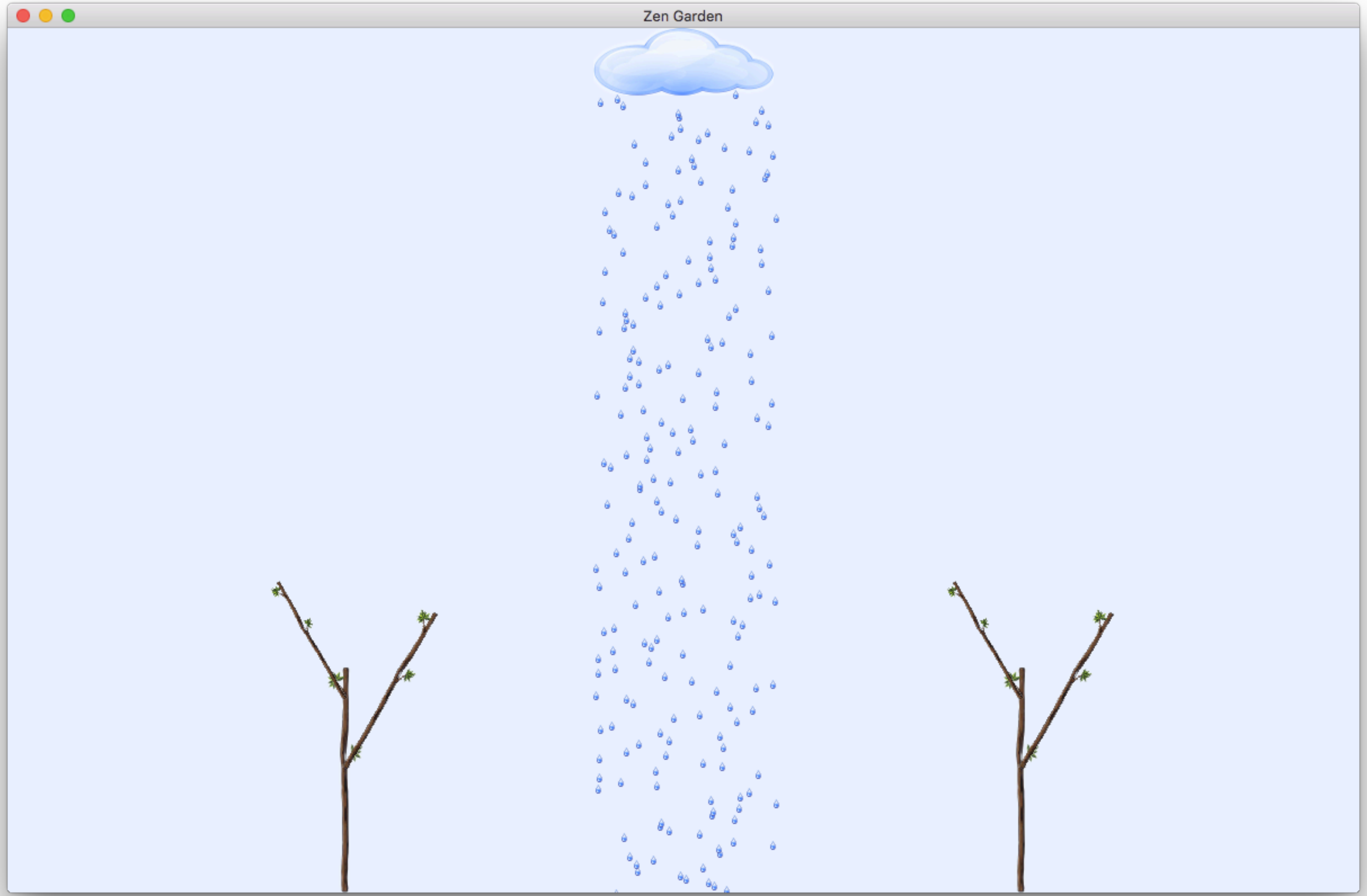
draw method: part 1 / 2

```java
public void draw(SpriteBatch batch) {
    // An affine transform is used to represent translation, rotation, and scaling operations.
    Affine2 transform = new Affine2();

    // Initial translation and rotation, bringing us to the base of the tree, pointed upwards.
    transform.translate(baseX, baseY);
    transform.rotate(90.0f);

    // Store the current transform state for use below.
    Affine2 savedTransform = new Affine2(transform);

    drawBranch(batch, transform);

    ....
```

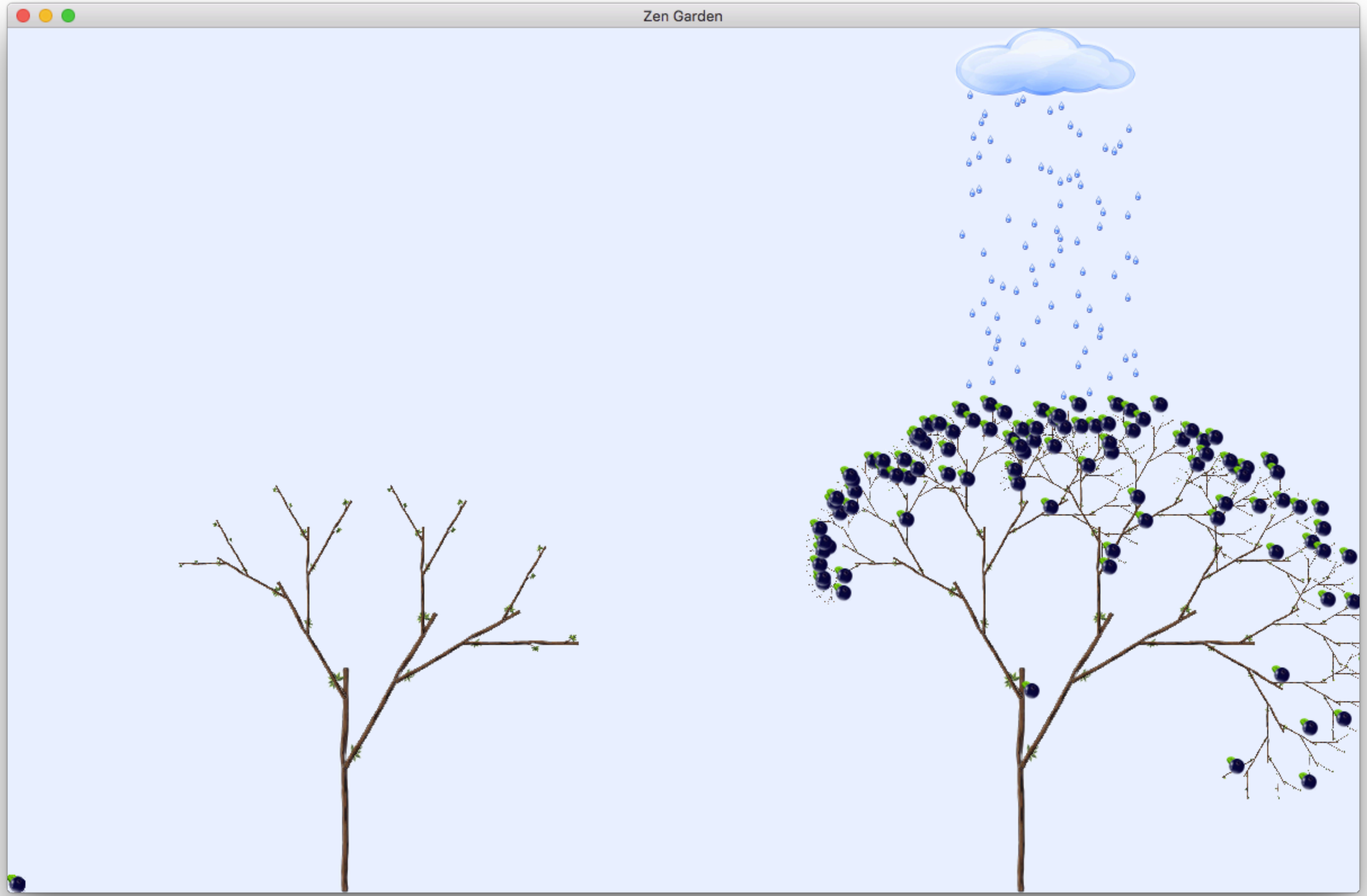
```java
private void drawBranch(SpriteBatch batch, Affine2 transform) {
    // Draw the current branch.  We draw it as two halves because transformations such as
    // rotation are made with respect to the lower left corner of the image.
    batch.draw(stickLeft, stickLeftWidth, stickLeftHeight, transform);
    transform.scale(1f, -1f);
    batch.draw(stickRight, stickRightWidth, stickRightHeight, transform);
    transform.scale(1f, -1f);
}
```

draw method: part 2 / 2

```
    // Translate to the first branching point
    transform.translate(stickLeftWidth * 0.86f, 0);

    // Draw the first branch
    transform.rotate(30.0f);
    transform.scale(FIRST_BRANCH_SCALE, FIRST_BRANCH_SCALE);
    drawBranch(batch, transform);

    // Reposition to second branching point by restoring the saved transform.
    transform = savedTransform;
    transform.translate(stickLeftWidth * 0.55f, 0);

    // Draw the second branch
    transform.rotate(-30.0f);
    transform.scale(SECOND_BRANCH_SCALE, SECOND_BRANCH_SCALE);
    drawBranch(batch, transform);
}
```


Zen Garden

- On the previous slide we are using the SimpleTree class which draws the basic trunk of the tree and two branches.
- RecursiveTree adds the following:
    - Tree grows in response to water drops falling on it
    - Tree grows fractally by recursive branching
    - When at the maximum growth level, berries emerge!
- Please see the attached code for details…


Zen Garden