

LCC 6310

The Computer as an Expressive Medium

Lecture 9

Overview

Programming questions related to project 2? (due Friday 6pm!)

Programming concepts

- super and this

- Java SDK classes

 - Lists

 - Reading the Java docs

Free time to work on project 2

Project 2

Create your own drawing tool, emphasizing algorithmic generation/modification/manipulation. Explore the balance of control between the tool and the person using the tool. The tool should do something different when moving vs. dragging (moving with the mouse button down). The code for your tool should use at least one class.

Revisiting our example

So far we have a rocket that flies around in a field of asteroids and fires missiles

Now we want our missiles to blow up asteroids

- This means we need a variable number of asteroids

- How do we do this with an array? (They have a fixed size)

- We can use a special Java object called an ArrayList!

- We'll also need to figure out when we have a collision

The Java SDK

(Java SDK = Java Software Developer's Kit)

Java comes with thousands of classes in the Java Platform API

Documentation is available on Sun's website

<http://java.sun.com/j2se/1.4.2/docs/api/index.html>

Note that Processing supports Java 1.4.2 features (not yet 1.5)

You may want to download the documentation...

Let's look at ArrayList

ArrayList

ArrayList is a resizable list

This means we can add and delete things from the list without worrying about declaring the size up front

The main methods we care about are `add()`, `get()`, `remove()`, and `size()`

<http://java.sun.com/j2se/1.4.2/docs/api/index.html>

Steps for using ArrayList

Declare a variable of type ArrayList

Create a new ArrayList and assign it to the variable

Call `add()`, `get()`, `remove()` and `size()` on the ArrayList as you need them

Parents and children

Remember that we declared a child class `ArmedRocket` whose parent was `Rocket`, and remember that classes are types

So `ArmedRocket` is a type and `Rocket` is a type

So, here are some legal assignments:

```
ArmedRocket r1 = new ArmedRocket(50, 60, 0);
```

```
Rocket r2 = new Rocket(50, 60, 0);
```

```
Rocket r3 = new ArmedRocket(50, 60, 0);
```

But this one is illegal:

```
ArmedRocket r4 = new Rocket(50, 60, 0);
```

Same goes for method arguments as well...

Using ArrayList.add()

The argument type of the `add` method is `Object`

`Object` is the parent class of **all classes**

With an object argument type, you can pass in an object of any class

So, let's declare our asteroids ArrayList...

```
ArrayList asteroids;
```

And then initialize our asteroids...

```
asteroids = new ArrayList();  
for (int i = 0; i < NUM_ASTEROIDS; i++) {  
    asteroids.add(new Asteroid());  
}
```

Let's try this in [Processing](#)...

Getting things out of an ArrayList

Looking at the API doc for ArrayList...

<http://java.sun.com/j2se/1.4.2/docs/api/index.html>

`ArrayList.get(i)`

Returns the i-th object
(starting with 0, i is an int)

But this doesn't work!

```
asteroids.get(i).drawMe();  
Why not?
```

Need to cast back from Object

Since things are put into an ArrayList as Object, they also come back out as Object

It's like they forget their more detailed type

So, when using ArrayList (or any container class), we need to **cast** back to the more detailed type. Casting is the conversion of something from one data type to another - requires care!!

```
Asteroid asteroid = (Asteroid)asteroids.get(i);  
if (!asteroid.collison(r1)) {  
    asteroid.drawMe();  
}
```

Hmmm... we haven't seen this way of doing collision checking yet. Let's work on that now...

Putting collision detection in Asteroid

Currently, detecting collision takes place in the main draw() method
But it would be cleaner (more object-oriented) if Asteroid itself knew how to detect collision

Detecting collision depends on knowing the boundaries of the asteroid, which properly belongs in the asteroid class

So to detect collision with a Rocket...

```
boolean collision(Rocket r) {  
    if ((r.xPos >= xPos - 26 && r.xPos <= xPos + 22) &&  
        (r.yPos >= yPos - 24 && r.yPos <= yPos + 26)) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

We also want to detect collisions with a Missile...

More collision detection

We can use the same method name with a different type of parameter, Java will know how to resolve this

```
boolean collision(Missile m) {  
    if ((m.xPos >= xPos - 26 && m.xPos <= xPos + 22) &&  
        (m.yPos >= yPos - 24 && m.yPos <= yPos + 26)) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Let's put these methods into [Processing](#) (remember to update the old collision detection code in the draw() method)...

```
if(!a[i].collison(r1)) {  
    a[i].drawMe();  
}
```

Destroying asteroids

When a missile hits an Asteroid, we need to destroy it

This was the whole reason for using ArrayList!

Big asteroids turn into two small asteroids

Small asteroids disappear

```
void destroy(ArrayList asteroids) {
    asteroids.remove(this);
    if (large) {
        asteroids.add(new Asteroid(false, xPos, yPos, lastDrawMillis));
        asteroids.add(new Asteroid(false, xPos, yPos, lastDrawMillis));
    }
}
```

Hmm... we've added some new things here:

We need to store the size of the Asteroid (is it large or small?)

We need to be able to create new smaller Asteroids at a specified position

Asteroid size

Since we only want two sizes (big and small) we can use a boolean:

```
boolean large;
```

We also need to put something in the Asteroid's drawMe() method to make sure that it will get drawn at the right size

Let's make small Asteroids at half-scale

```
if (!large) {
    scale(0.5);
}
```

Now let's think about how to create the new smaller Asteroids at a given (x,y) position...

super and this

this is a special variable keyword that always refers to the current instance (object)

Useful in methods to refer to yourself

this.method() – calls a method on yourself (but normally you just directly call the method)

this() – calls a constructor on yourself (useful for one version of a constructor to call another)

this.someVariable – refers to the class variable (useful if a variable of local scope has the same name as some class variable)

super is a special variable that always refers to the superclass portion of an object (the object cast into it's superclass)

super.method() – calls a method on the superclass

super() – calls a constructor on the superclass

A new Asteroid constructor

```
Asteroid(boolean isLarge, float initialX, float initialY, long previousDrawTime) {
    // set the initial x and y pos
    xPos = initialX; yPos = initialY;

    // get a random orientation for this asteroid
    rotation = random(0, TWO_PI);

    // get the asteroid's velocity in x and y directions
    // based on its orientation and size (bigger moves slower)
    if (isLarge) {
        velocityX = sin(rotation)*10;
        velocityY = -cos(rotation)*10;
    } else {
        velocityX = sin(rotation)*25;
        velocityY = -cos(rotation)*25;
    }

    large = isLarge;
    lastDrawMillis = previousDrawTime;
}
```

But what about the old constructor?

We still want to be able to create an Asteroid in a random location

So we want to keep our original constructor without parameters, but we need to update it to take into account the asteroid size (they start big)

We can just use `this` to call our new constructor and let it do all the work

```
Asteroid() {  
    // create an asteroid at a random location on the display  
    this (true, random(0, XSIZE), random(0, YSIZE), 0);  
}
```

Let's assemble all these Asteroid changes together and put them in [Processing...](#)

The size variable...

Storing our size

```
boolean large;
```

Drawing at the correct size

```
// after translating and rotating, we should  
// make sure to scale ourselves too  
if (!large) {  
    scale(0.5);  
}
```

The constructors...

```
Asteroid() {  
    this (true, random(0, XSIZE), random(0, YSIZE), 0);  
}  
  
Asteroid(boolean isLarge, float initialX, float initialY, long previousDrawTime) {  
    xPos = initialX; yPos = initialY;  
    rotation = random(0, TWO_PI);  
    if (isLarge) {  
        velocityX = sin(rotation)*10;  
        velocityY = -cos(rotation)*10;  
    } else {  
        velocityX = sin(rotation)*25;  
        velocityY = -cos(rotation)*25;  
    }  
    large = isLarge;  
    lastDrawMillis = previousDrawTime;  
}
```

The destroy method...

```
void destroy(ArrayList asteroids) {  
    asteroids.remove(this);  
    if (large) {  
        asteroids.add(new Asteroid(false, xPos, yPos, lastDrawMillis));  
        asteroids.add(new Asteroid(false, xPos, yPos, lastDrawMillis));  
    }  
}
```

Transition to ArrayList

Now that we've created a destroy method to split large Asteroids into two smaller ones, we want to move to our ArrayList data structure

Remember to remove the previous fixed-size Asteroid array!

We've already declared and initialized the ArrayList, so next we need to draw it:

```
for(int i = 0; i < asteroids.size(); i++) {
    Asteroid asteroid = (Asteroid)asteroids.get(i);
    if (!asteroid.collission(r1)) {
        asteroid.drawMe();
    }
}
```

Last but not least: calling destroy!

Currently we check the Missile array to draw visible missiles, but now we also want to check if a Missile has collided with an Asteroid

If it has, we should destroy the Asteroid and remove the Missile

```
for(int i = 0; i < MAX_MISSILES; i++) {
    if (m[i] != null) {
        m[i].drawMe();
        for (int j = 0; j < asteroids.size(); j++) {
            Asteroid asteroid = (Asteroid)asteroids.get(j);
            if (asteroid.collission(m[i])) {
                asteroid.destroy(asteroids);
                m[i] = null;
                break;
            }
        }
        if (m[i] != null && !m[i].isVisible()) {
            m[i] = null;
        }
    }
}
```

Summary

Learned how to use **ArrayList**, our first Java Platform class

Learned about super and subclasses as types

Any instance of a subclass is an instance of the superclass, but not vice-versa

Can cast more abstract classes (parents) into more concrete classes (children)

The Java keywords **super** and **this**

Special variables that can be used within a method to refer to yourself (the superclass portion of yourself and all of yourself)

Remember...

For **Thursday** this week: Theory Readings

Two presenters (you know who you are!)

Everyone else: prepare one discussion question for each reading

Happenings in the New York Scene - Kaprow, NMR pp. 83-86

The Cut-Up Method of Brion Gysin - Burroughs, NMR pp. 89-91

Six Selections by the Oulipo - NMR pp. 147-189

Finally today: free time to work on your drawing tools!