

Programming Methodology-Lecture11

Instructor (Mehran Sahami):Hi, all. Welcome to CS106A. Let's go ahead and get started. If you're on your way in, just feel free to come on in. You can always get the handouts at the end.

A couple quick announcements before we dive into things. There's two handouts today. There's a whole bunch of coding examples. There's coding examples up the wazoo. They're sort of like – I gave you a bunch of code that might just be useful for you to look out for Breakout, just because it's good times.

So also related to Breakout – no, I think we just change the way we say this. It's just Breakout, and so any time you see people on campus – every quarter I come up with some random thing that I wanted to try to get people on campus to say. And I've never gotten one to take off seriously. Well, let's see if we can actually get Breakout to work. So just practice that.

On the count of three: One, two, three. Students:

Breakout.

Instructor (Mehran Sahami):Yeah, good times. All right. See, sometimes you gotta find these little [inaudible] of humor yourself.

Anyway, the note on Breakout is that first of all, there's actually a version of it that's available on the Web site, so you can actually play around with. The other one is we've talked a lot about classes so far. And with the initial project that we give you for Breakout, there's one class in there, you don't actually need to write a whole bunch of separate classes for Breakout. It's perfectly reasonable to write all your code in that one class.

If you want to add additional classes and have additional files in the project, and do it like we did in class, that's perfectly fine, and if it makes sense for you. But you shouldn't necessarily feel obligated that, "Oh, we talked about classes, so we have to do that for Breakout." For Breakout, you're actually fine. Classes can potentially provide some [inaudible], but they're not a requirement for the assignment.

So just couple other quick questions. In class recently – so now that we've had a few classes, you can sort of get the feel for – we do some stuff on slides. Occasionally, we do some stuff on the board. And so I wanna get a preference. If you have a – if you get a notion – if you have a preference for slide versus board.

How many people prefer slides?

How many people prefer the board?

Interesting. All right. How many people like some combination of the two? You can vote again if you want.

All right. So we'll try to do that. All righty. So today's gonna be a little slide-heavy day because we didn't have the vote so far. But we'll try to do some more board stuff as well.

So here are the class hierarchy you saw before. And we covered a lot of the class hierarchy last time. But there were still a few bits that we have left to cover, which is GImage, GPolygon, and GCompound. So you'll see those. And then you'll be ready to do all kinds of cool stuff today.

As a matter of fact, today we're gonna break out things you can do with a mouse and graphic – and more graphics kind of stuff with the mouse. And it's just a whole bunch fun [inaudible] doing Breakout.

So GImage: All the GImage is, and this is a funky way that you can put pictures inside your program, like pictures that you take with a camera or whatever or your phone. You can just stick them in your program. And the way you do this is there's a class called GImage. And so you can create an object of type GImage by saying, "new GImage." You give it the name of the file. That's the full file name that with a dot and whatever ending it has, and then the xy location for where you wanna display that image. The xy is the upper left-hand corner of the image.

So the file – image file is the name of the file containing the image, and x and y, as I just mentioned, are those coordinates. Now, where you have to stick this file – that's something to remember. So when it looks for the this file, what it will do is it will look in your current project folder, and if it does not find that file name in your current project folder, then it will look in a subdirectory named "Images."

So you can have the subfolder if you have a whole bunch of pictures called "Images" inside the folder for your project. Put all your images there. It will also look there as well.

If it's not in either one of these places, bad things happen. And if you wanna know what bad things happen, you can actually try it. But I wouldn't encourage you to try to do bad things. So just stick in the folder with your project, and you'll be fine. And you'll get the image.

So let me show you an example of what that actually looks like. Just so you know, both .gif or "jif" formats. Some people say "jif," even though the "g" stands for "graphics" over here, which is kind of a hard "g," and then .jpg, which is another graphics standard, and you'll see the files either in .jpg or .jpeg, which just another format. Both of them are supported, and so most cameras or phones or whatever actually end up taking pictures in one of these formats.

So here's a simple example. So if we had some program run, we can say I wanna have some image that's a GImage. So I create one, and let's say I have to have the Stanford

seal, which hopefully we have rights to, I would hope. It's probably some guy in Kansas actually has rights to the Stanford seal and charges every time we use it. It would not be the first time something's like that has happened. But that's not important right now.

What is important is that Stanfordseal.gif [inaudible] – it looks for that file inside the folder for our project. And then it's gonna – we're gonna add that particular image and that location 00. So here, I did it the way that I just specify the file name without giving the x and y coordinates. You can give it the x and y coordinates if you wanna begin with.

But if you don't, it's kind of funky to say, "Hey, load that image." And now I can get the size of that image so I can center it in the screen or whatever, if you wanna do that.

You can also – what's also sorta fun is you can remember GImage [inaudible] implements this particular interface called resizable. So we can actually say, "Hey, image, I want you to scale one and a half times in the x-direction and .5 times in the y-direction." It's kind of like Stanford comes to school and checks into the dorm and gets on the meal plan and puts on the freshman 15 and is just like, "Oh, oh, oh, I'm Fat Stanford now."

But you can take pictures of your friends and just distort them in funky ways. It's just a good time. And so you can do all kinds of – a bunch of different options on things like images. That's just one of them.

There's also this class called Polygon. And Polygon's kinda a fun, kinda a interesting, cool class. And what it lets you do is it lets you represent graphical objects bound by line segments. So we need to get a little bit more formal. But here's a little diamond. Here's a hexagon. They're just polygons. And all a polygon basically is is just something that has multiple sides on it. That's where "poly" comes from. It just means many.

So the basic idea that's interesting about polygons is a polygon has a reference point. So when I create a polygon, I'm actually gonna tell the computer sort of all the points on the polygon. All of those points that I specify, like let's say the four corners of a diamond, are all in some relation to some particular reference point that I pick.

And so in this case, I might, for example, pick the center of the diamond. And in fact, in most cases, all of the vertices that I lay out, since they're gonna be relative to some reference point, that reference point often is convenient to pick the center. It just turns out, and it doesn't have to be. You could actually have your reference point be upper-left corner of the bounding box or whatever you want.

You just gotta make sure that when you sort of tell it what the vertices are that you lay them out in the right order. But oftentimes, the center of the polygon, if it's a regular polygon, which means it's sort of symmetric all around, then the center is often easiest.

So I'll show you some examples of this, how you actually construct one of these GPolygon objects.

So the first thing you do is you say, "Create an empty polygon." So you create empty polygon object. And then what you're gonna do is you're gonna specify the vertices of that polygon one at a time using a method called `addVertex`. And `addVertex` can take an x and y coordinate. These x and y coordinates are relative to your reference point. So if you think of your reference point as being the center of the object, and you say, "Oh, my xy is 1,1," that's one pixel over and one pixel down from wherever you think of your reference point as being.

You never actually specify the reference point to the `GPolygon`. The reference point's just in your mind. All of the vertices are just relative to whatever reference point you happen to pick. And I'll show you an example of this to make it concrete in just a second.

After you set an initial vertex – so you need to set the first one with something called `addVertex`. After you set the initial vertex, you can set all the remaining vertices in one of two ways because we just give you options. We like you.

One way is just to keep adding vertices. And what – when you call a `addVertex` again, what it does is it adds a new vertex, again, relative to the reference point. And it essentially creates an imaginary line between the last vertex you just added and the first vertex you added. That's how you're getting edges of your polygon. You're sort of specifying the corner points.

Another way, instead of specifying corner points, is after you specify the first one, you can explicitly `addEdge`. And `addEdge` adds a new vertex relative to the preceding one. And here, it's going to do it with offsets of `dx` and `dy`.

So relative to where we were before, we actually specify sort of an offset in the x-direction, offset in the y-direction. It's kinda like you were almost creating a line, and it's gonna add an edge. This one here is using absolute coordinates relative to the reference point. That's kind of a – the key difference between these two things. And I'll show you an example of both of them.

One final thing you should know is the polygon is, in some sense, closed for you. So if you were drawing a diamond, and – let's back up so I can show you the diamond. If you're drawing a diamond, you might specify that as the first vertex, then this is the second, then this is the third, and then this is the fourth.

How did that know what the fourth and first are what should be attached to each other? It just closes it for you. So after you've added the last vertex to the polygon, basically what it does is it just automatically sort of links up the first and last segments, or first and last vertices, to sort of close the polygon for you. That's how you actually specify. So you never actually need to go back to the first vertex again. It just does it.

So that's kinda a whole bunch of stuff in theory. What does it look like in practice.

So we're gonna create a diamond. We're actually gonna create it using some method that we'll see called createDiamond. And so when we call createDiamond, we're expected to give us back a GPolygon.

Here's where all the interesting stuff's going on. So here's the create-diamond method. CreateDiamond's gonna return a GPolygon. So it needs to create one because it's gonna return it back to us. And what it's getting is the width and height of what that diamond should look like. So the first thing it does is it says, "Create a new polygon." It creates a new, empty polygon, just like you sort of saw before.

Then what does it need to do? It needs to add the first vertex. So it adds the first vertex. And it's gonna do this relative to some imaginary reference point that I pick. The reference point that I'm gonna pick is going to be the center of the diamond, which means my first vertex in the x-direction is going to be minus width over two. So let me just show you where that vertex would be.

So if I think of the imaginary center, and I've just executed that line to add a vertex, I'll say relative to my imaginary center, my first vertex is at half the width of the diamond over in the negative x-direction, and on that same line, so 0 in the y-direction.

Then where do I go from there? Well, I'm gonna add the next vertex. The next vertex I'm gonna add, relative to my center, is going to be at the same x coordinate as the center, but it's going to be minus height divided by two upward. So what I get, essentially, is this is an x vertex, and it just kinda creates the segment in between them.

Where's my next vertex? It's gonna be over here. It's width divided by two for my imaginary center point and on the same line as the imaginary center point. So the y offset is 0.

And then last, but not least, the last vertex I'm gonna add is on the same x coordinate as the imaginary center. But the height is half the height of the diamond, downward. So I get that.

And now if I say, "Return diamond," it matches up the first and last vertex. So what I get back is a closed polygon. I actually get that diamond. And that's what's returned back to me in my "Run" function over here. So diamond is now this little polygon.

Question?

Student: Do you add [inaudible] you add the bottom to the top [inaudible]?

Instructor (Mehran Sahami): It depends on if you do something like addEdges versus addVertex. So again, I would try it. And especially if you try things like where your lines would've crossed, you'll see sort of funky things between addEdge and addVertex.

Uh-huh?

Student: Do you ever actually [inaudible]?

Instructor (Mehran Sahami): You never specify the reference point. All of the computation we did here are all relative to some imaginary reference point. But we never specified a reference point exactly. It's always a good idea, if you're thinking about creating a GPolygon, to identify for yourself what that reference point is before you draw the line segments.

Uh-huh?

Student: [Inaudible] is there a way to try to [inaudible]?

Instructor (Mehran Sahami): Yeah, you just figure out what the size of the image is by asking its height and width. And then you need to specify scaling factor to make it large enough to do the whole screen.

Uh-huh?

Student: [Inaudible] addEdge [inaudible]?

Instructor (Mehran Sahami): Yeah, so let me actually show that to you right now. Here's the – so addVertex. We just returned the polygon. We set its fill to be true. We set its color to be magenta. We write it out in the middle of the screen – not a whole lot of excitement going on. The one thing that's important to note is when we add the diamond to our canvas, the location that we give for adding it to the canvas is the location of the reference point. So every time that we do something with the polygon once we've created it, the xy location that we specify for the canvas is the imaginary reference point.

So here are – middle point was get width divided by two – or sorry, get width divided by two is this way. Get height divided by two is the center of the screen. And because we use the center as the imaginary reference point for the polygon, it actually shows up in the center of the screen.

So actually, using GPolygons a lot of times is just convenient even if you wanna circle because you wanna specify the reference point of the circle to just be in the center of the screen as opposed to its corner.

Uh-huh?

Student: [Inaudible]

Instructor (Mehran Sahami): If you really wanna do – we should talk offline. We should talk offline.

Student: [Inaudible]

Instructor (Mehran Sahami): Yeah. There's a bunch of things we could do. Essentially, there's a bunch of stuff with polar coordinates that I should mention. We're not gonna deal – so if you read in the book about polar coordinates and fade and all that, we're actually not gonna do it in this class.

So if you're like, "Oh, polar coordinates." You're like, "Yeah, polar scares me," it scares me, too. Polar coordinates, you only need to worry about if you're a bear, or you're in the Antarctic.

We just not gonna worry about polar coordinates. We're just gonna do Cartesian coordinates. But there are – it's easier to do rotations if you think polar.

So in terms of adding an edge, we're gonna do the same thing for createDiamond using edges instead of vertices. So if we wanna create a diamond adding edges, we have to still specify the first vertex, just like we did before.

And now the edges – what I'm specifying for the values for the edge is an offset in the x and y direction. And my offset in the x-direction is width over two, so that would move here. And my height is negative height over two, so that would essentially move over here because it's relative to my last point. It's not relative to the imaginary reference point.

And so basically, if I sort of follow this all the way around, what I do is I addEdges. Again, after I add that last edge, the polygon is closed for me automatically. I get back the same diamond. I'm just now doing it with edges instead of with vertices, same sort of deal. I make it magenta because magenta's a fun color, and I write it out on the screen.

So that's GPolygon. Now, things get a little more funky. And the thing that gets the funkiest of all – it's kind of like the George Clinton of graphics class, is GCompound.

And so what GCompound's gonna do – anyone know George Clinton? All right, a couple, but at least it's not totally lost. GCompound basically is as you'd imagine. It's a compound shape. It allows you to take multiple other shapes that you've seen and put them all together and treat that one thing as one object, which is very convenient sometimes if you wanna draw something complicated and then move it all around the screen, for example, or rescale it.

So adding objects – the way a GCompound works is you add objects to a GCompound just like it was a canvas. So just like you've done before where you say, "I have some canvas, and I put these little objects on it, and it draws pictures," if I have some GCompound, and I add a bunch of objects to it, I now have this compound thing that encompasses all of those objects.

So you can now treat the whole compound as one object, though, which is the whole reason for having this, and I'll show you the example of this in just a second.

So similar to a GPolygon, a GCompound also has a reference point that when you add objects to the GCompound, you add all the objects relative to some imaginary reference point that the GCompound has.

And finally, how do you display this thing? So when you add things to a GCompound, they're not displaying on the screen. You need other take the GCompound and add it to the canvas just like all the other [inaudible] or ovals or whatever you did before. And when you place it, you place it relative to its reference points, and it will draw all the objects that it encompasses.

So let's actually just see what that looks like. So let's draw a little face. Oh, little face, and so you can see we're gonna use some different things here. We're gonna have an oval. We're gonna have a couple of other ovals as the eyes. Hey, triangle – what should we use for the triangle?

Student:[Inaudible]

Instructor (Mehran Sahami):Polygon, right. Some people are like, "Three lines." No, GPolygon. It's your friend, really. It's a fun time. And here's a little rectangle for the mouth.

So what we're gonna do with this is we're just gonna go ahead and take a look at a class GFace. So the GFace class that I'm creating is just a whole separate class. I'm going to create this class GFace as extending GCompound. So a GFace is going to be a GCompound. And what I'm gonna do is I'm gonna have a whole bunch of constants here that just specify the width of the eyes and height of the eyes and all the sort of features of the face. And I'm gonna have some objects that comprise my little GCompound.

So I'm gonna say, "Well, the head's an oval, and the left eye and the right eye are both oval, and the nose is gonna be a polygon. And the mouth is gonna be rectangle." These are all my private instance variables. They're just the variables I'm gonna use to keep track of the things I create and add to my GCompound.

So the compound has a constructor, which is just the same name of the class. It's GFace, and what it's gonna take is the width and the height because I wanna allow the face to automatically be resizable, which means all the stuff that I create that's part of the face needs to be in terms of the width and height that are past, and so it will resize itself depending on whatever width and height the user gives me.

So the first thing I do is I say, "Hey, the head's gonna be an oval." And the oval is basically just a circle that's off-size width, height. It's gonna be the size of the whole head, so whatever size you give me is the size of the head.

The left eye is gonna have the eye width times width, so it's gonna scale by whatever width you give me for the face. I'll scale it by eye width, and same thing for the height. I'll scale it by eye height. And I'm gonna create two eyes, left eye and right eye. Notice

that at this point, I haven't placed where my left eye and right eye are. I've just created them.

The nose is going to have a function, or a method, associated with it called createNose that's going to return for me a GPolygon. And so we'll take a brief digression down here to take a look at createNose. So what's createNose gonna do? It's gonna have some width and some height for the nose. It's gonna create a new polygon and add vertices to construct a nose.

So the first vertex it's gonna add, and we're gonna think of the center of the nose as being the reference point, the imaginary reference point. So the first vertex is sort of the height or the bridge of the nose. It has the same x coordinate as the center of the nose, if we think of some imaginary triangle here.

Actually, let me just draw a little triangle so we can use the board. Here's the nose. Here's our imaginary reference point. Where's our first vertex gonna be? It's gonna be up here. That's the same x coordinate as our imaginary reference point, and half the height. You can just imagine these two are now equal – waving of hands. It's gonna be half the height upwards. So that's gonna set the first vertex here.

And then where am I gonna set the next vertex? The next vertex, relative to the center, is gonna be width divided by two over, so width divided by two over.

And then in terms of height, if we can see it over here, it's gonna be height divided by two downward. So this is the next vertex, and then the last vertex is over here. And that's gonna give you your little triangle for the nose.

So when we create the nose – losing my chalk. We're gonna return that polygon that's created here because this method is returning a GPolygon. And the place that gets assigned is back up here where we actually have a nose.

Student:[Inaudible]

Instructor (Mehran Sahami):So the nose is whatever I get back from createNose. It's going to be the polygon for my nose. And the nose is, of course, scaled by the size of the face. So we have nose with the nose height to scale the size of the face.

And the mouth, last but not least, is just a rectangle – again, is scaled by the height and width of the face given the mouth width and the mouth height.

So now I've created all my little pieces. How do I put them together to create the face?

Well, when I call add methods inside of here, what I'm creating, I'm creating a new GCompound. What I'm creating here is GFace, which extends GCompound. Since it extends GCompound, that means all the methods that exist for GCompound extend here. This is not a graphics program. This is a GCompound.

So when I call add here, I'm not adding it to the graphics window. I'm adding it to the GCompound. So addHead and 0,0 – my imaginary reference point for this face is going to be this upper-left-hand corner of the bounding box.

So if I say add, head is 0,0. If I think of this as 0,0, and this is an oval, it's going to add the oval here because what I'm specifying is the upper-left-hand corner of the oval.

Now where am I gonna add other stuff? I'm gonna add the left eye to this kinda funky equation over here. But basically, all it's saying is I take a quarter of the width of the face and subtract from it the eye width scaled by the size of the face divided by two. So basically, what it's gonna do for the x coordinate is do something on sort of this part of the face. It's gonna bring it over a quarter from where it would've been relative to the center of the screen.

And then I'm also gonna do something relative to the height wherefore the left eye and the right eye – they're both gonna be at the same height, where I basically just look at a quarter of the height down the face is where those eyes are gonna show up. And I have to do some accounting for the eye height. So if my eyes – if my face were really big, my eyes still show up at the right place.

So it's just a little bit of math. You can sort of trace through it by hand if you're interested. And the exact coordinates for where these things are going is really not that important as long as we get a light – right layout that looks like a face. The important thing is all these things are relative to our little reference point at 0,0, which is the reference point for the face. We haven't put anything on the canvas yet.

The nose is just gonna be at the center, so nose is gonna be halfway over on the width and halfway over on the height. So if this is our reference point over here, we go halfway over on the width, halfway down on the height, that's the very center of the circle. And because when we created our GPolygon, the imaginary reference point of our triangle was the center, it places it relative to the imaginary center point of the triangle. So we get the triangle right in the middle of the face.

And last, but not least, we're gonna put in – we're gonna add the mouth. And the mouth is basically gonna go near the bottom of the face. It's actually gonna go in the center of the face. This is just basically finding the center point for the x shifted over by the sides of the rectangle, and then it's gonna be three quarters of the way down the circle, which is kinda how we get this.

So we add all these things relative to this reference point. And now, if we actually want to display this, we need to write some program that displays it. And so we'll have a program called DrawFace. And all DrawFace does is pretty simple. It says, "I'm gonna pop up a little pop-up in the middle of your screen."

I'm gonna have a graphics program that's gonna have some face width and face height. And what I'm gonna do is I'm gonna create a new face, giving it the width and height.

And where do I wanna place this face on the screen? I wanna center it on the screen. The reference point for the face, though, is not the center of the face. It's this upper-left-hand corner. So I need to figure out, relative to that upper-left-hand corner – I'm losing another piece of chalk – how to center the face.

So what do I do? I get the width of the screen. I subtract from it the face width, and I divide by two. That's kinda the classic thing you did when you were trying to center, say, a rectangle. You would take the width of the screen, subtract off the width of the rectangle, divide by two. What it will essentially do is figure out the coordinate to place this guy so this whole thing is centered in the screen. And we do essentially the same thing in the y-direction as well.

So we look at the height of the screen. We subtract off the height of the face and divide by two, and that gives us how much we should go down.

So again, when we finally place this on the canvas, we're placing it – when we specify the point on the canvas, that point is the reference point of the whole GCompound.

Instructor (Mehran Sahami):

Are there any questions about that?

Uh-huh?

Student:[Inaudible] the face [inaudible]?

Instructor (Mehran Sahami):I could've actually done it as just creating a GPolygon object and adding everything inside of another program. But I wanted you to actually see it as extending a class just because that's a common way of doing the decomposition. You say, "This thing that I'm gonna create really is a GCompound, so I'm gonna create it as a derived class of GCompound." It's the more common way of doing it.

Uh-huh?

Student:[Inaudible] –

Instructor (Mehran Sahami):Oh, sorry.

Uh-huh?

Student:[Inaudible] compound [inaudible] in the compound [inaudible]?

Instructor (Mehran Sahami):Yeah, so you can think of there's a z-ordering of the compound that's just like a canvas. So you actually have it – you can have layering of objects. And objects will include other objects in the compound. And it's the order in which they're added.

So the funky thing also about doing this is remember our friend, the bouncing ball? Let me just refresh your memory with the bouncing ball. I like the bouncing ball. Doo, doo, doo, doo, doo, so I'm running. I'm running. I'm running. Here's our friend, the bouncing ball. Come on, bouncing ball. We're bouncing. We're bouncing. And you're like, "Ooh, the bouncing ball got bigger since last time." Yeah, I changed one constant. It got a little bigger. That's okay. You wouldn't have remembered if I didn't tell you. How many people were like, "Oh, I measured the number of pixels."

Well, here's bouncing ball. And what's the whole beauty of having GCompound and creating a new class out of GCompound because I don't want bouncing ball anymore. Ball is boring. I want a bouncing face. How do I create a bouncing face? Hey, I got this class over here, GFace. That's a good time. Instead of a ball, I'm gonna have this thing be a GFace.

You're like, "Don't do it, Marilyn." All right, well, now that I do that, I'm still gonna call that ball just so I don't have to change the ball everywhere in my program. There's one other thing that happens, though. When I create the ball, I'm no longer creating a G-oval. I'm creating a GFace. Notice GFace still takes a width and a height, so my parameters are unchanged. The only other problem is a compound, a GCompound, does not satisfy the criteria or the interface for being fillable.

So this little thing that make the ball fillable and set the fill of the ball to be true, you'll see, "Oh, little air condition over here," because GFace or GCompound in general doesn't have a notion of filling the whole compound, so we get rid of that.

And now we save this puppy off, and we're gonna run. And let's see what happens.

First thing, with just modifying 20 seconds worth of code, we didn't get an error, and we'll run bouncing ball. Dun, dun, dun, and I have bouncing face. Yeah, scaled to be just the right size. Life is good. Three lines of code to modify. That's the beauty of object-oriented composition is when I have something that's a ball, and I have – and a ball, a GObject or g-oval's a GObject, and I have something else that's a GObject, I can just slam one in for the other.

Bouncing Stanford logo? Yeah. Bouncing face of your roommate? Yeah. You can do it, all right? We're just gonna not bounce anything else for the time being.

But one thing that's even more interesting than just being able to bounce stuff around is what we refer to as event-driven programs. And all an event is is when you do something in the world, that's an event. When you went to Stanford, that was a big event. You got your big packet in the mail, and you opened it up, and you're like, "Oh, I got into Stanford. I'm going to Stanford, or maybe you gnashed your teeth and tried to figure out where you were gonna go or whatever." Some of you were just like, "Oh, yeah, I'm going to Stanford for sure."

And that was an event, and there was cake and celebrating and the whole maybe – I don't know – maybe you were like, "Oh, I gotta go to Stanford. Bummer."

But it was an event. And in some sense, your life is a program, and it's driven by events. These little things happen, and it changes the course of your life. Hopefully coming to Stanford changed the course of your life in a good way.

In terms of actual programs you write, there will also be events that will change the course of your program. But these events are a little bit more minor than, say, getting into Stanford. There are things like when the user interacts with a computer, like clicking a mouse or pressing a key. Those are events that happen on a computer. And your program should be able to detect certain events that are going on and respond to them by doing various kinds of things, say when you're writing your Breakout program.

The way we be able to determine if an event has taken place is kinda a clandestine operation. It's kinda like we're in the CIA. And we gotta say, "Oh, did an event take place? Well, I don't know [inaudible] if an event took place. I need to send out my minions."

And your minions are called your listeners. Your listeners are something you put out there that says, "I'm here. I'm listening for events. Is anything going on?"

And unless you put out your listeners, you're deaf to what's going on in the world. It's kind of like you weren't listening. You weren't watching the mail for that Stanford packet, and the Stanford packet came, and you just never checked the mail, and you never knew. And no one was gonna tell you because you didn't put out your listener for the Stanford packet.

Well, on the computer, there are two different kinds of listeners we think about. There's a mouse listener and a key listener, which gets events from the mouse and events from the keyboard. And so the way we create our listeners is in our program very early on, we either say `addMouseListeners` or `addKeyListener`. We can actually say "both" – listen for both things. And these are methods that are supported by graphics programs, for example.

Now, in order to use these listeners, you need to have a library added in your program that deals with events, and that's the Java `awt.event.*`. So on your programs that use events like listeners, and we'll look at events in just a second, you'll need to have this little import line in there. And this is all in the book. You don't have to hurriedly copy it down.

A bunch of people have asked for slides. And I was hesitant to post slides on the Web because I think if slides are up there, people will not come to class.

So first of all, how many people would like me to post the slides on the Web?

How many people would continue to come to class if I post the slides on the Web?

Yeah, it should be all those same hands. Good times. All right. I'll probably post them on the Web, then.

All right. So there are certain methods of a listener that get called when an event happens, and this is the funky thing. Up until this time, all of your programs have been – have been what we refer to as synchronous. There are some series of events that happen, and you know the next event is gonna happen, or the next method's gonna get called after the last method got called. You're now entering the world called the asynchronous world, which means you don't know when things are gonna happen.

Events happen. You might've been waiting for weeks for that Stanford little acceptance letter to come. You didn't know when it was gonna come. Eventually, it came, but you didn't know. And there were some sad people who we won't talk about right now, but if you're out there watching on video, it's fine. It's perfectly okay. You were waiting for the Stanford acceptance letter, and it never came. That happens. It's not a bad thing. You're still listening. You were active. You took a participatory role in life. That's what you should do. It's a good thing. But you didn't know when or if it was gonna happen.

And that's what an asynchronous event is. It happens, and you don't know when it's gonna happen. You just gotta be prepared when it happens. You're like, "I'm gung ho. When it happens, it's gonna happen, and I'm gonna be there because I'm listening for it." That's what you wanna think about.

So let's look at a simple example of a event-driven program called ClickForFace. You're like, "What does that mean?" But it probably involves the face. Yes, in fact, it does.

So click can mean anything. ClickForFace. Here's an example of an event-driven program. What this is gonna do is it's gonna bring up a blank screen, and every place we click on the screen, it's gonna put a little face there. That's why it's ClickForFace, all right?

So how does this work? Well, first of all, ClickForFace is a graphics program. It's gonna drawFace wherever the user clicks the mouse. Now, here's the funky thing. There's a couple funky things that happen in this program. This program has no run method because there is nothing we need to do in terms of a bunch of sequential steps in this program. All we're doing is we're waiting for events. We're waiting for someone to click the mouse. Until they click the mouse, we got nothing to do except for we gotta be listening for that mouse to get clicked.

So there's a method called a init(), and init() automatically gets called when a program starts. And you might say, "Well, that sounds a lot like run. What's the difference between run and init()?" The way you wanna think about the difference between these is run is when you're actually doing some real work.

Init() is generally when you're just saying, "There's a few things I need to initialize. I need to put my little ear out there to be listening for things. But there's not any real work

I'm gonna be doing in the program." And [inaudible] distinction is not a big deal if you mix the two up. But there is a difference, just so you know.

And we won't ding you for it heavily or anything like that, just so you know. Most of your programs will probably have the run method. In Breakout, you'll have a run method. You won't have init(). It's not a big deal.

So all init() does is it says, "Hey, I gotta be listening for those mouse events." So it adds the mouseListeners. It just says that. And that means now I'm listening for the mouse.

What happens when the mouse gets clicked? Here's the funky part. There is a method called mouseClicked(). And mouseClicked() takes any parameter of type mouse events. Now, if you're paying very careful attention to this program, you will realize that nowhere in your program do you actually call the mouse click method. You're like, "Huh. If I never call the mouse click method, why am I writing it?"

This is what asynchronous events are all about. This particular method has a special name that is understood by the listener. When it hears a mouse click, it will call your mouse click method for you. That's why it's asynchronous. You don't know when it's gonna get called. All you know is if there's a click, it will get called, and you will get this parameter called mouse events.

And what a mouse event has is information about that mouse click. Namely, you can ask the mouse event, which we're just calling e for events, GetX() and GetY(), and that gets the xy location of where the mouse was clicked on the screen.

If the mouse was not clicked in your graphics window, you don't hear that because your listener can only listen in your graphics program. It's not listening to your whole computer. So don't worry if you're like, "Oh, and I'm reading email. Is it listening?" No. It's okay. It's just looking for little clicks in your graphics window.

And so what we're to do when we get to mouse event is, "Hey, our friend the GFace." We're gonna create a new GFace. That sounds like it should be a rap group, don't you – GFace. Maybe that should be – all right, all right, we won't get into that.

And it's gonna be a circle with a face diameter of just some constant I specify – not a big deal. Thirty – I'm gonna create some new round GFace, and I'm gonna place it at the xy location where the mouse was clicked. Note that the xy location that I set the face is this relative location of the face. The face will not show up exactly in the middle where I clicked the mouse. It will show up slightly to the right and down of where I clicked the mouse.

But just to show you that this works, let's click for face. It's like bowling for dollars. We're clicking for face. Doo, we're running. We're feeling good. Any questions about asynchronous events, by the way? Let me just run ClickForFace.

Question?

Student:Do you have to [inaudible] click to [inaudible]?

Instructor (Mehran Sahami):If you have two programs running simultaneously?

Student:[Inaudible] if you have [inaudible].

Instructor (Mehran Sahami):Uh-huh?

Student:[Inaudible]?

Instructor (Mehran Sahami):Yeah, so while your program's running, if you get these asynchronous mouse clicks, your – the mouse-click method will get called for you, yeah.

Student:[Inaudible] will it stop [inaudible] method [inaudible]?

Instructor (Mehran Sahami):It will stop momentarily for you to deal with that method called `mouseClicked()`, and then it will continue execution.

Student:[Inaudible]

Instructor (Mehran Sahami):Yeah. So here we have – there's nothing in the program yet because we haven't clicked yet. Ah, `ClickForFace`. So you can hit it, and you can spend weeks. You're like, "Yeah, there's a click here. I'm gonna put some face on top of each other. It's like a party over here." Yeah, `ClickForFace`. That's it.

If I click out here, I don't get any faces. I [inaudible] into some other application because it only listens for events inside here. So that's `ClickForFace`. There we go.

Uh-huh?

Student:[Inaudible] a [inaudible]?

Instructor (Mehran Sahami):No, because you don't know where it would return it to. You're not the one that called it. Someone else called you, so you're like, "Hey, you called me. Here's a good time. Have a candy bar." And it's like, "I don't wanna candy bar." And it gets really upset, so – but I hope you wanted the candy bar.

So yeah, you don't return anything. But that's a good thing to think about.

So that's `ClickForFace`. Now, there's a whole bunch of things you can actually listen for. It's not just clicks. The general idea is first, you add your `mouseListeners`. That's critical. Common thing people do is they forget their listeners. And they're like, "Hey, I'm clicking for face. And I'm not getting any face."

And it's not because the face doesn't love you. The face loves you. I know. I've was writing the face at 3:00 a.m. last night. It loves you. And then what you need to do after you have your mouse listener in there, and then you add definitions for any listeners you wanna have.

So here are the common set of listeners for mouses. There's mouse-click, which you just saw. That's called whenever the user clicks the mouse. There's mouse-pressed. What's the difference between a click and a press? The press is that the mouse button has been pressed down. It has not yet been released. So if you wanna do something like a drag where you hit a button, and you move the mouse around, and then you release it, you can check for press and released. So a press and a release together is a click.

So you can actually kinda do multiple things. You can do something when the mouse is pressed. You can do something when the mouse is released, and that'll also count as a click. So mouseClicked(), mousePressed(), mouse released. MouseMoved() – every single pixel that the mouse moves, it's like, "Oh, mouse move, mouse move, mouse move." Yeah, I can't even do it fast enough. That's how fast it goes. It's how quickly the mouse moved.

And mouseDragged() is when someone actually clicks on something and then moves the mouse while holding down the button. In fact, we'll say that the mouse has been dragged, which is like mouseMoved() except that this is only happening when the button's down, and someone's moving the mouse.

But all these take the same kind of mouse events, and so you can get the xy location for where that click happened, or where the mouse recently moved to. And that's gonna provide you the data about the event.

So let's look at another one. Let's actually track the mouse. And you're like, "Oh, that wily mouse. It has gotten away from me. How do I track the mouse?" MouseTracker might be real useful for Breakout, just in case it wasn't clear.

What the MouseTracker's gonna do is basically, I'm gonna have a run method here because there's actually some work I wanna do other than just having listeners. What am I gonna do? I'm gonna create some empty label. I'm gonna make a font for that label real big so that you can see it. I'm gonna add that label at a particular location on the screen because it's an empty label. When I start out, nothing will show up. And then I listen. I set out my mouseListeners, and I'm like, "Woo, mouse? Mouse?" And what I'm checking for is mouseMoved(). That's the only listener that I'm setting up here.

If mouse moves, I get some events. I'm going to change the text for that label to be a mouse with the xy location it moved to. And I need to keep track of my label, both here and here, so I need to keep track of the label in between method calls so I actually have my label be a private-instance variable.

So if I run this puppy, it's ten lines of code, but check out just how cool it is – well, after it runs. You're like, "How cool is it, Marilyn?" We're gonna track the mouse. Doo, doo, doo. We're running. Oh, yeah, see, it moves off the screen, stops tracking, moves back into the tracking to start tracking. One thing you can do if you're just totally bored is can you get `y(0,0)`.

[Inaudible] last night. It was 3:00. I'm like, "I can be preparing a lecture, but I wanted to see, can I get to 0,0?" I'm like – an hour and a half later, I was like, "There it is." And then I wept.

So besides the mouse, there's also things you can do with the keyboard. So the keyboard is also your friend. So, returning to our friend, the slide, besides tracking the mouse, we can also do things with the keyboard. And you're like, "Hey, Marilyn, for Breakout, do I need to do anything with the keyboard?"

Not in the basic version of Breakout, but if you wanna do the cooler version of Breakout, where your paddle [inaudible] can shoot at the bricks and take them down? Some people have done that in the past. It makes the game much easier. But it's fun. You can listen to keyboard events.

So keyboard events work just like mouse events. You add a listener. But the listener you're gonna add is `KeyListener`s. It's perfectly fine for program to add both `MouseListener`s and `KeyListener`s. Why do you have to add these to begin with? And the reason why you have to add these is because your program really needs to know, "Do I need to pay attention to these things?"

Because it actually requires some work on the part of the computer to pay attention to these things. Because one thing you might just think [inaudible], "This whole listener thing is just dumb, Marilyn." And I'm like, "Any program I run, yeah, if I'm pressing the keyboard, it should just know about it, right?"

No, not necessarily. If you're writing a piece of software that needs to run really fast and doesn't care where the mouse is, why should you have it worry about where the mouse is or worry about what keys are being pressed?

So that's why this thing that's important to remember that you need to put in there, but doesn't come in by default because it's actually programs where we care about efficiency, and we don't care about what the user's doing, which, sadly enough, is many programs. We don't have listeners.

So for the key listener, there's three things I check for. These are the most common ones. There's actually a few more that are in the book, but these are the ones you really need to worry about for most things. There's `KeyPressed()`, which is called when the user presses a key. There's `KeyReleased()`, which is when they let go of the key. So `KeyPressed()` is when they push it down. They have not necessarily let go of the key. `KeyReleased()` is they've let up on the key.

So you can actually do funky things what – when you press something, like the screen goes black, and then your roommate leaves and doesn't know what you're doing, and then you lift up the key, and it comes back. You can actually do that.

KeyTyped() is basically a click. It's press and release together. So when you press and release a key, that generates a keyTyped() event. What you get in this E that you're getting is an event. It's not a mouse event. It's a key event. And key event is an object which has information about, for example, which key was pressed. And the book goes into details of which key was pressed.

But I wanna show you some simple example of this where we don't even care what key was pressed. All we care is that a key was pressed. So yet another example. Woo-hoo. And I can actually just close PowerPoint.

So what we're gonna do is we're going to DragObjects around. What does DragObjects do? So DragObjects is a graphics program, and what I'm gonna do is create a rectangle on the screen and add it, create an oval on the screen and add it, and add mouseListeners and keyListeners.

And you might see this and go, "Marilyn, why is it a init() method as opposed to a run method? Shouldn't that really be a run method because you're doing some work?" Yeah, probably. You can – it's [inaudible] the other. The book actually has a similar program for this. It's not the same program but a similar program that they call it init() over there. So I just made it init() just kinda to match the book so you're not like, "Oh, Marilyn, slides in the book are just completely wrong," and again, this death struggle.

No, it just – it doesn't really matter that much. But we create the rectangle. We create the oval. We add them both to the screen. We add the listeners, both the mouse listener and the key listener. And now here's what we're gonna do that's funky. We're going to allow objects to be dragged on the screen.

What does that mean? That means when you click on a rectangle on the screen, and you move your mouse holding down the button – that's called a drag – we're gonna have the rectangle move with you. Woo-hoo. Rock on. So mouse drag. Mouse drag is gonna get some mouse events.

Now, there's a little bit of funkiness in here with this little GObject thing. What does that mean? So I'll get back to that in just a second by first showing you what this GObject actually is. So what I'm gonna keep track of in my program is a generic GObject. And you might say, "But Marilyn, we don't create generic GObjects." Yeah, that's because its variable GObject is either gonna be the rectangle or the oval. It can be either one. They're both GObjects. So the common denominator is that they're GObjects.

So this object – this variable is gonna keep track of what object is currently being dragged on the screen – what object I clicked on so I know what I'm actually dragging, whether it's the oval or rectangle or that I haven't clicked on any object at all.

GPoint is basically just something that holds the x and y location. It's a very simple object that we haven't talked about till right now. But basically, all the GPoint is is it's a little tiny object, and it has an x and a y location in it. And so I can set the x and y location in the point, and I can say, "Give me both getters and setters." Get the x and y location. It's just a little convenient encapsulation. I could've actually had this be a separate variable x and a separate variable y. I just created the GPoint so you could see a GPoint.

And I'm gonna have little randomness in my program, too. So the random generator will once again rear its ugly head. And so I have – I get an instance of the random generator. You'll see we're gonna generate some random colors in just a second.

So with that said, how did this mouse press and mouse drag thing work? MousePressed() is when someone clicked the button. They have not yet released the button. It is not a full click. It's just the mouse was pressed. The mouse was pushed down. I get this event. And that event, E, basically has an x and y location associated with it.

As you saw in the previous example, I can get the x and y location separately. There is a way I can just say e.get point(), and it gets the x and y location together in a little point object and gives that back to you, which is why I created this thing called the GPoint, so you could see it. It's just a nice [inaudible] that gives you x and y at the same time.

And when I say, "Hey, get that point, and what I wanna do is get the elements at that point." So wherever you clicked, call getElement() at. That will return to me the topmost object at the point you clicked. If it's the rectangle, it will give me back the rectangle. If it's the oval, it will give me back the oval.

If there is no object at the point I clicked, I get back something called null, N-U-L-L, and that gets assigned to GObject, which means your GObject – there is no thing. There is no object there. Null is just a way of referring essentially to no object. But you can assign any object the value null to say, "There's not really an object here."

And so getElement() will either give you the oval, the rectangle, or null. Get element – that's something you're gonna use for Breakout. Pay close attention.

So what happens after the user clicks? If I get an object at the place they clicked, I now see are they moving the mouse? If they're dragging the mouse – after they click, if they move the mouse at all, that generates events that are mouse-dragged events. So gets called the position of the mouse that the mouse has been dragged to.

First thing I checked, are you dragging a natural object? When you click the mouse, did you click it on top of an object? If you did, then the object is not null. If you didn't click it on an object, then it is null, so hey, nothing to see here. You didn't click on an object, and now you're trying to drag around. You've got nothing to drag, buddy. Sorry. Thanks for playing. I'm not gonna do anything.

But if you did click on an object, then what I'm gonna do is move that object. And remember, move is a relative coordinates. It says, "Move this object relative to where it was before." So I say, "Get the x location and the y location that the mouse has moved to from this event, and subtract off essentially the last place that the mouse was, which is where the mouse actually clicked on this object." So I get some relative amount of movement.

And now after you've moved the mouse, I need to say the mouse is actually at a new point now. So I update the last points that the mouse was at to be equal to wherever the mouse was actually moved to.

So last is always the last location of the mouse was, either dragged to or got clicked on.

Last but not least is I wanna change the color of the object that you last clicked on or that you're dragging to a random color if you type any key. So I don't care what key you type. I'm not actually gonna look at what the key event was. But if you typed any key, `keyTyped()` will get called, and if your object is not null, I'm gonna set it to object color to be – or the color of that object to be some random color that I'm gonna get my random color generator or my random number generator gonna give me random colors.

So let me run this so you get a sense of what's actually going on – how these pieces fit together. Doo, doo, doo, doo, doo. It's like you can create a whole drawing program now, all right?

So we're gonna drag some objects around. So when this puppy starts off, we get a oval and a square, which both start off black because I didn't set their color. And they happen to be slightly on top of each other. If I click on the rectangle and move it, oh, yeah. Notice how it tracks the mouse when I move it. When I now click off the rectangle and try to move around, nothing happens.

Or if I click here and try to drag, I'm holding the mouse button down, nothing happens. If I click on the oval and move it around, oh, it gets moved around. Now I wanna be funky. I press a key. The last object I dragged gets assigned a random color. Oh, that's kinda fun for about three seconds, all right?

If I click on the other object and move it around, you can actually see the oval was put in front of the rectangle, and the z-ordering is not changing by me moving the object around. I never say, "Send to front," or "Send to back," or anything like that. I could do that if I wanted to, to bring an object up to the front, which is kind of the behavior you're used to from other applications. When you click on something, it comes to the front. Not here because I'm not changing the z-order, so it's just back over here.

But the last thing I clicked on, I can also get random colors and change its colors. Yeah, that's – it's just – yeah, oh, so much fun, all right?

Any questions about that? We're dragging. We're clicking. We're changing colors.

Student:[Inaudible] object [inaudible].

Instructor (Mehran Sahami): Yeah, I could've done set location. I just need to do a little bit more math for set location because I need to figure out to set the xy of the object relative to where the mouse was clicked inside of the object and its coordinates sort of off in this corner.

So one final thing I wanna show you before you go, because now you've seen all the codes to actually be able to create a simple game like "Breakout." But here's another simple game, just to show you another example of stuff going on.

I call it the UFO game. I gave you all the code for the UFO game. And you may recognize this game. Oh, yeah, it's sort of like Space Invaders Lite. You remember – anyone remember Space Invaders? Two people. Yeah, this thing is coming down the screen toward you, and you're in the middle. You get to shoot these little shots out at every time you click on the mouse. And if it ever reaches the bottom of the screen, you die.

But if you hit it – oh, I'm coming so close – it's gone. Thank you. Hours, hours of work. Let me just show you a little bit of the code so you can understand. This will be important for Breakout. When the code runs, how are you gonna create effective animation and also other stuff going on at the same time?

I'm going to give each object in the world a chance to do something. I'm gonna call `moveUFO()` to allow it to move, `move bullet` to allow it to move if there's a bullet in the air. Check to see if there's a collision between these two things, and wait before I do this cycle again. So I just continue to do this cycle over and over until my game is over.

And the way I check my game is over is basically that little square got onto the bottom, or I actually shot it and got a collision. I'll leave it to you to look at the code because if I look – go look at the code in excruciating detail, I'll probably get Breakout questions, which you have all the code so you can understand how hopefully it works. And you can leverage it for Breakout. So I'll see you on Friday.

[End of Audio]

Duration: 52 minutes