Programming Methodology-Lecture10

Instructor (Mehran Sahami): Okay, I would just, even at this point, just email text. Might be easier. I think we need to get started.

Let's go ahead and get started. Couple quick announcements before we start. So one of them is that there are three handouts in the back, including your next assignment. And your next assignment's a little game called Breakout. How many people have ever heard of a game called Breakout?

Yeah, when I was a wee tyke, just [inaudible] to the game, actually. It was my favorite game ever. And it was just fun. And you're actually gonna be implementing it. How cool is that?

So that's one of your handouts. There's a couple other handouts. There's a section handout and a handout of some code examples.

One thing, just so you know, in the handout on Breakout, it actually will refer to a Web demo. So if you wanna sort of see how the game plays – because it's hard to capture an actual game in just screenshots. There's a few screenshots in your handout.

But if you wanna actually get an idea for how the game plays, there's a demo of the actual working game on the Web page. So if you go to the CS106a Web page, both on the Announcements page, as well as on the Assignments page for Assignments 3, there's a link you can click on for demo, and that will actually take you to an executable demo of the game so you can play it.

Assignment No. 2, as you all know, is due today. So quick, painful – hopefully it wasn't too painful for you. But we'll just see how long it took.

Anyone in the zero- to two-hour category? A couple folks, actually. Two to four? Good to see. Four to six? Ooh. Six to eight? Eight to ten? Ten to 12? Twelve to fourteen?

Student: [Inaudible] a lot.

Instructor (Mehran Sahami): Few folks on 12 to 14. Fourteen to 16? And 16-plus? Oh, wait, do I have any 14 to 16? No, there's a couple section leaders in the back row – 14 to 16. They're just – they're pulling for you because they're like, "Marilyn, you're making the assignments too hard." Hopefully I'm not. Anyone in 16-plus?

Student: [Inaudible]

Instructor (Mehran Sahami): All righty. Thanks for letting me know. Once again, the world is normal, and the world is where you would expect it to be, which is hopefully, most folks are below kind of this 10-hour mark because it's what we're shooting for. Good times if you are. If you're not, if there was some kind of – I'd ask you to please be

quiet if you're just coming in right now. In terms of if you're in one of these higher sections, if there was some bug that just caused a problem for you, and you figured it out, and it wasn't a big issue, that's fine.

Sometimes, that will cause you a couple of extra hours. If it was really a conceptual issue, that there was something where you just didn't know what was going on – maybe you just wrote a bunch of code, and eventually, it worked, or you needed to get a lot of help for it to work, please come talk to me. Talk to the TA, Ben, or talk to your section leader to make sure those issues are clarified because we're just gonna keep building on the stuff that you've done. You'll actually see a lot more graphics today.

So with that said, I actually want to kind of ask you a sort of a "where we're at" kind of question. So last time, we talked about perhaps the most critical concept in the class, which is classes, oddly enough for the name. So I wanted to ask, actually, a quick question. So far, with where we're at and what we've done in the class, how many people are just generally feeling okay? If you're feeling okay, raise your hand.

Okay. How many people think we're going too slow? Couple folks.

How many people are just – are feeling like we're going too fast, or you're feeling overwhelmed? Okay.

So first of all, if you are feeling overwhelmed, if you're out of the category of just fast and into the category of overwhelmed, that's kinda why I asked those together. If you're feeling overwhelmed, please come talk to me, or talk to your section leader, or talk to Ben.

We're happy to spend as much time as you need to make sure you don't feel overwhelmed, and you feel like you understand everything that's going on in the class. Have no qualms about coming to talk to us. That's why we're here. It's more fun when we actually get a chance to talk to you if you're having any problems.

And in that sense, I think we're kinda going at a reasonable pace, perhaps a little bit quickly. But that's kinda how we have to be at to actually cover all the material in the class.

So with that said, are there any questions about classes right now, just [inaudible] anything we've done?

I wanna spend a little bit of time touching on classes before we dive into our next great topic today, which is graphics. So if we actually have the computer for a second, I just wanna briefly review the classes that we wrote last time.

So we wrote a class that hopefully is near and dear to many of you because many of you are instances of this class, which is a student. And we went through, and we talked about all the things that you actually wanna have in a class. And hopefully this is just review.

You have some constructors in the class. You can have multiple different constructors in the class. But here, we just have one where we take [inaudible] a name and ID, and that name and ID gets set to some instance variables or ibars that all students have, namely student name and student ID.

So I'll scroll down to the bottom, and here we have our private instance variables, student names, student ID, and units earned. These are the variables that every student's object has their own copy of. That's why they're instance variables as opposed to, say, a class variable.

This guy over here, because it's got a static, is called a class variable because it's got a final that's actually a constant, as hopefully, you've seen many times by now. And so all units – all students share units to graduate as the same constant value, 180.

So it's both a constant and a class variable. Most things that are constants make sense to be class variables because all elements of that class sort of share the same constant value. All students require 180 units to graduate.

So we did a bunch of things in here, and we kinda went through the constructor. We went through a few places where we said, "Hey, if you wanna be able to access portions of this class which are private, you can't directly access them from outside." That's why they're private. So no one can touch your private parts. And so what you needed to have was you needed to have these functions if you wanted to allow people to access them, that we refer to as getters.

And the reason why we refer to them as getters is because they start with the name "get." And what they do is they're public methods. So someone can actually call these methods from outside of your class. And they just return the value for some appropriate thing that the person would want. GetID would return student ID.

Now, you might say that's kinda weird. Why do I have these getters when student ID, I could just make public and let people access it directly? Why would I do this?

Student: [Inaudible]

Instructor (Mehran Sahami): So you can't change it. If student ID was public, that's a public variable. People cannot only read what the value is; they can set what the value is. If I make it private, I control access to it, and here, I could let you read what the idea is by giving you a copy of the ID. I don't let you set it. The only way the ID actually gets set is when a student first gets created, and you get a student ID up here, which you have for life. So when you get created, you have the ID for life. This is actually true at Stanford.

As a faculty member, now, I have the same ID number, no joke, that I had when I was a student. It's kinda funky, and it's just like, "You are in our universal system, and you will be here for the rest of your life. And it doesn't matter if you go away, and then when you

come back, you will still be that same ID number because you can never change it once you're created." Same kinda idea going on over here with this class.

Now, sometimes we do allow someone to change something. And we'll have what we call setters as well. So for units, we not only have GetUnits, which returns the number of units earned. We also have what we refer to as a setter, which sets the number of units.

And you might say, "Okay, that's kinda odd, Marilyn. If you're allowing someone to set the units and get the units, why don't you just make that variable public?" because at this point, you're allowing someone to both set its value and get its value. What else are you gonna do with that variable?

And the reason why we still don't make it public is because of information encapsulation. The person who is getting the number of units you have and setting the number of units you have doesn't need to know how we keep track of that information. It could actually turn out that the way we get your units is we go through your whole transcript and total up all your units.

And we never actually store it as one value, and that's how we get it back for the person. And when they try to set it, if they try to set more units than you already have, we just create some dummy classes. Or if they try to reduce the number of units, we drop some of your classes, and that happens occasionally.

So it doesn't matter how it's implemented underneath the hood. It's information hiding. All the person needs to know is that they can get it and set it. In this case, it's simple because it is just referring to one variable. But – and you'll see that oftentimes is the case, but not always, which is why we still have this encapsulation. And so for some things, we have getters and setters.

And then there was a few other things here where we could increment the number of units someone has, or we could check to see if they have enough units to graduate by just checking if their unit count is greater than graduating – the number of units they need to graduate.

And last thing, which I said all classes need – and again, I will repeat this because all classes you write that are not programs. And any classes you write that does not extend, say, console program or graphics program, should always have something called ToString.

String is just text, and what this does is it just returns, basically, some piece of text that tells the person using this class what this object actually encapsulated. It doesn't need to contain all the information in the object. But here, we're gonna return the student's name and then the student ID number inside parens.

So that's a quick review of the class. And part of the reason I want to do a quick review is we're actually gonna extend this class, which means remember when we talked about classes in the days of yore, and we had classes and subclasses and superclasses?

Here's a student. All of you are students. Now, there's some specializations of students, as well. As a matter of fact, some of you are, for example, frosh, which is a kind of student. And some of you are sophomores. And some of you are juniors. And some of you are seniors.

The size of the boxes don't actually mean anything other than it's smaller. Some of you are grad students. Some of you are the dreaded other student. We won't talk about that. We'll just call you the other student because occasionally, you may not be any of those things, and we still need a way of keeping track of you.

But you're – all of these things are classes, which are specializations of Student, which means you have all the properties that a student does, and perhaps some special properties that you may have by being in one of these particular classes.

So let's actually write one of these – oh, let's say – let's write the frosh subclass. How many people in here are freshmen, frosh? Woo-hoo. So hopefully, I picked the majority class.

So we're just gonna go ahead and pick a – create a new class by extending the student class. So what we're gonna do is we're gonna build a class that extends the functionality of our existing student class. And in this case, what we wanna do is create a Frosh subclass of students. So we wanna create the subclass here called Frosh that is a Student, which means it's gonna extend Student.

So Frosh will have all the same properties that regular Students do, except for the fact that they start with 0 units. And I'm sure like many of you are sitting out there going, "Oh, no, no, no, Marilyn. There's a wonderful thing called the AP exam."

Yeah, let's just pretend the AP exam didn't exist, for the time being because there actually are some people, as sad as that may be, that start with 0 units. And so we're just gonna say that all Frosh start with 0 units.

Think of this as the number of units you've earned at Stanford. And I know there's even a few of you out there who've earned a non-0 number of units at Stanford. But let's just say it's 0. Be thankful it's not negative. You're like, "Oh, you're a freshman. You get negative 5 units. Thanks for playing."

Now, the one thing we're gonna do is we're gonna specially designate Frosh in the strings that display an object. So that two-string method we talked about – it has the name of a person and their ID number. For the case of Frosh in particular, we're going to actually designate them as Frosh just because your RAs love you, and they wanna know who are the Frosh?

So we're gonna have to find a way of saying, "Hey, if you want the string version of this object, we're gonna actually have the string contain the word "Frosh" in it.

So here's how we might do that. First thing we're gonna do is we need to define the Frosh class. So it's a public class, Frosh, that extends Student. So every Frosh is a Student. And then inside here, we need to have a constructor for the Frosh class or, well, for – to create an actual Frosh. It's not gonna bring an entire class into being at once. It's gonna bring each individual freshman into being. And it's gonna take in a name and an ID, just like a regular Student does.

Now, here's the funky thing. What's the super thing all about? If we're gonna initialize a freshman, what we need to say is, "Hey, as a freshman, we're gonna initialize you. You're gonna – we're gonna set your number of units to be 0 because that's one of the properties the freshman have – is their number of units start at 0. But you're also going to have the same initialization done to you that all regular students do."

So how do I refer to the initialization process, or the constructor, that regular Students go through because I'm writing a special constructor for Frosh. And unless I specifically say, "Go and do the same things that you would do for a regular student, they won't happen."

So the way I specify that is I say, "Super," which doesn't necessarily mean, "You're super," although it could. It could because freshmen, you're pretty super. I don't know if when you got your acceptance letter, there was that little handwritten note on the bottom. I remember when I got my acceptance letter, it was a different dean than there is now. And on the bottom, it said, "Super, Marilyn," and I think that was the last handwritten thing I ever got from Stanford.

So "super" means call the constructor of the Super class. So by calling Super a name and ID, what we're doing is saying, "What's the Super class of Frosh?" It's Student. So this will call the Super class of Frosh, which is Student, with name and ID, which means it's gonna go through that initialization process of setting name to be equal or setting student name – that instance variable to be equal name, and student ID to be ID, the stuff that we did back over here.

So let's just hop back over here for a second. The things that we do in the constructor of a Student – so all of this stuff we still wanna happen for freshmen, which is when we say Super name and ID, it's actually calling, essentially, this function, to initialize all of the fields of a regular student. And then it's gonna do some more stuff specifically for freshmen.

So when we come back over here and do our little Super dance, what we get is, essentially, the constructor of a Student being called to set up a Student, which happens to be a freshman, which is the particular case of student. So we're gonna do some additional work by setting the number of units equal to 0.

So that's how we set up the constructor. Now, the other thing I mentioned was we need to set a ToString method.

Now, here's the funky thing. You might say, "But Marilyn, student's already had a ToString method. So if I didn't do anything here, wouldn't all Frosh be able to give me a string version of what they are?" Yeah, they would, but we wanna create a specialization of it.

So what we do is we do something called overriding. And overriding just means even though your Super class already had a method called ToString, in the subclass I'm going to define a new version of that method. If the method has the same name and the same set of parameters, which in this case happens to be none, it does what's called overriding, which means for all objects that are of type Frosh, when you call ToString, you're gonna call this version of the method.

The version that exists in the Student is no longer germane for Frosh. It's still germane, say, for sophomores or juniors who may not implement their own version of ToString, but for Frosh, it's gonna call this version of it.

So overriding just means forget about the old version. I'm sort of overriding it with my new, funky version that's specific to this particular subclass.

Any question about overriding? Has to have the same name for the method and same parameter list.

So what we do here is remember, student ID and student name are private. And now, the funky thing about private is even when I extend the class, when I create a subclass, that subclass does not have access to the private elements.

And you're like, "Whoa, Marilyn, that's kinda weird. I have a Student, and inside a Student, I can play around with Student name and Student ID. But as soon as I create this thing called Frosh, which is a subclass of Student, it doesn't have access to those private elements anymore, which means if I wanna be able to get the name and the ID of the student, I can't just access the variables directly. I need to access the corresponding getters GetName and GetID.

So now if I call ToString, what I'm going to return is a string that says Frosh: and then your name and ID number. So that's how it differentiates it from the standard version of ToString is ToString. The regular one just provided name and ID number. This one actually sort of designates you as a Frosh because we know you're a member of the Frosh class.

Any questions about this notion of public versus private and why in a subclass, you still can't access the private portions?

Uh-huh? Is there a question over there?

Student: Yeah, so if the [inaudible] class [inaudible] Super class?

Instructor (Mehran Sahami): Yeah, so the way you can think of the subclass is the subclass is just another class. So the visibility that it has into its super class is the same visibility that any other class would have. So it can't access the private portions directly, even though it's a subclass. It needs to call public methods to be able to access those elements. It can still access any of the public elements. It just can't access the private elements directly, which is kinda funky.

That sort of freaks some people out, and there's this thing called protected, which eventually we'll get into, but we won't talk about it right now. All you need to worry about it private and public.

But if you sort of extend the class, you create a subclass. You need to understand that the subclass does not have access to the private elements.

Any questions about that? Are you feeling okay with that? If you're feeling okay with that, nod your head. Well, we're mostly heading on in, so that's a good thing.

So what I wanna do now is now it's time for us to draw pictures. It's just a lovely thing. This is really a day that's about graphics. And graphics is all about drawing pictures. And a matter of fact, for your last assignment, you drew a bunch of pictures, right? You – in Assignment No. 1, you drew some stuff that used the graphics library, and life was all good.

So we're gonna revisit that drawing that you just did for your last assignment and just soup it up. We're gonna bump it up so you can do all kinds of funky things like animation and games. And eventually, we'll get into where you can read a mouse clicks and the whole deal. And you will just be good to go. But we gotta build up from where we sorta started before.

So where we started before was the ACM graphics model, which is when you were writing graphics programs, you were using the ACM graphics model. And we talked about this before. It's a collage. You basically start with this empty screen, and you basically put little felt pieces onto the screen. So you're saying, "Give me a square," and, "Give me an oval, and throw it somewhere else." And you just kinda add these things to a little canvas, which is your empty screen to begin with.

Now, a couple things that you sort of may have noticed as you were doing the last assignment, or you'll certainly notice here. Newer objects that we add, so when we add things to our canvas, if it happens to have obscured something else, that's fine. It'll just obscure it. The newer things are sort of laid on top of the old things.

And this is something that we refer to as the stacking order, or sometimes we refer to it as the Z-order because if you're sort of like that Cartesian coordinate system person, the Z-axis comes out toward you. Here's the X-axis; here's the Y-axis. The Z-axis comes

toward you, which imagine if you stack these things on top of each other, if you think in 3D, that's kind of the Z-axis.

But if you don't think about that, think of these as just pancakes you're laying on top of each other. And you're stacking them. And the new stuff occludes the stuff behind it. And that's what we refer to as the stacking order.

So hopefully, this is all review for you. And as a matter of fact, this stuff should all be review. Many moons ago, oh, say two weeks ago, a week ago, we talked about ACM graphics, and we said there's this thing called a GObject, and a bunch of things that inherit from GObject are GLabel, GRect, GOval, and GLine – are all subclasses of GObject. And everyone was like, "Oh, yeah, I remember that. It was a good time. I was creating these objects. I was adding them to my canvas. It was just fun."

And then you look at this diagram, and you say, "Hey, Marilyn, why do you draw it so funky?"

And the reason why I drew it so funky is there's a whole bunch of other stuff that now it's time for you to learn about. So here's kind of we think about the ACM graphics package in the [inaudible], and you're like, "Oh, man, there's points and dimensions and compounds and these 3D rectangles and all this stuff going on."

You don't need to worry about all that stuff. There's just a whole bunch of stuff that comes along with Java and the ACM libraries. And it's kinda like you're out there, and you wanted to buy the basic car. And you got the car, and they put this jet engine on top of it, and you're like, "That's really bad for the environment. I just really wanna be able to drive my car without the jet engine." And they tell you, "Okay, well, the jet engine is there, but you never have to turn it on." And you're like, "All right. That's cool." And that's what we're gonna do.

So all of the stuff you see in yellow is stuff we're gonna talk about. And the rest of the stuff isn't really the jet engine. It's not that cool. Really, the rest of the stuff is '70s racing stripes and a big hood scoop and stuff like that. It's like, yeah, you could have it on your car if you really wanted to, but probably not in the 21st century.

So we're gonna focus on all the stuff that really is kinda important for what we're doing. And there's a few other things. You can read about them in the book, but we're not gonna spend class time on them. You're not gonna worry about them in this class. If you really wanna know, you'll know.

But we're gonna do all kinds of cool stuff and images and polygons and just things that will be interesting, hopefully.

So with that said, let's first talk about this thing called GCanvas. And GCanvas is this thing that kinda sits up there, and you're like, "Yeah, GCanvas is not one of these objects that I sort of put – that I kinda create and put up on my collage. What's that all about?"

So let's talk about that and get that out of the way, and then we can focus on all these other things that we can kinda draw and put up on our board or our little canvas.

So what a GCanvas is, is it represents, in some sense, the background canvas of the collage. It's the big piece of felt that we're gonna put all the other little shapes on top of. And you might say, "But Marilyn, didn't we already have one of these? When I create a graphics program, don't I already get sort of my empty canvas that I put stuff on?"

Yeah, in fact, you do. What a graphics program does for you automatically, just because it loves you that much, is it automatically creates one of these GCanvas objects and makes it large enough to fill the entire program window.

So actually, when you're adding your objects to a graphics program, you're actually adding them to a canvas or a GCanvas object. The graphics program has just created one for you seamlessly, so up until now, you never had to worry about it. As a matter of fact, for about the next five weeks, you're not gonna have to worry about it – oh, two weeks you're not gonna have to worry about it.

And so you might say, "But if that's the case, when I was calling Add, won't my Add call methods? When I was calling those, weren't they going to a graphics program because I never worried about this thing called GCanvas?"

Yeah, in fact, they were going to the graphics program. The graphics program had a method called Add. And when it got it, it said, "Hey, you wanna add an object? I'll pass them over to the GCanvas I created."

So what it was really doing was forwarding, just like you think of call forwarding – get a call from your friend, you're like, "Oh, yeah, you wanna talk to Bill? This isn't Bill. Let me forward you over to Bill, and you can talk to him."

We call the Add method on graphics program. Graphics program said, "Oh, yeah, you wanna add that? Well, the person who really takes care of the adding is the canvas, so I'll just call Canvas passing in the same arguments to Canvas that you've passed to me." That's called forwarding. It's basically just some method that sits there that actually passes all the information that goes [inaudible] someone else to actually do the work. We also refer to that in the working world as management. So you basically forward on to something else that does the real work.

And so forwarding, to put it in the speak of object-oriented programming, is when the receiver of a message, so before, graphics program was the receiver of the message, then call some other object with that same message. So that's how we'd say it to sound real funky and get paid more money. Same kind of idea.

So it turns out that GCanvas and GProgram – or sorry, graphics program, which is really just forwarding a bunch of your calls over to GCanvas, support a whole bunch of methods, some of which you've seen. And I wanna give you the quick tour.

So there's add, and you've certainly seen that. It just takes some object, some GObject like a GRect or a GLabel, and adds it to the screen. And you can add some object at a particular x-y location if that object doesn't already have some existing x-y location. So there's two versions of that method.

Up until now, we've told you had to add things. Now you're sort of old enough to actually say – it's like training wheels. Before, you could add training wheels. Now you can remove the training wheels. So for an object, you can actually say, "Remove," and it will take that object off of the canvas – just rips it right out.

And if you wanna get really funky – if you're having a bad day, and you just come in there, and there's this nice picture of bunnies and flowers and stuff, and you just say, "No, man, that's not my world. RemoveAll." And it's just all gone. All of the objects you put on that canvas is – it takes the canvas outside, shakes it out, and all the little, fluffy bunny pieces go away. And it's RemoveAll. It just clears it.

There's GetElementAt, and this is a very funky thing. You would use this in Breakout. Pay close attention. What GetElementAt does is it – you give it a particular x-y location, a pixel location on the screen. What it will return to you is the frontmost object at that pixel location if one exists on the canvas. If one does not exist on the canvas, it will return to you something called null. And null just basically means no object.

You can assign null to something of type GObject, but that just basically means that GObject is trying to deal with – is just there's nothing there.

Otherwise, it will actually give you back a GObject, which could be a specific thing. It could be a GRect, a GOval, a GLine, whatever it is. But all of those GRect, GOval, GLines are all of type GObject. So in fact, even if it gives you a GRect, it's still giving you a GObject back because the GRect is just a specialization of a GObject. So we'll actually give you back an object that's at that x-y location.

That might be real useful, for example, if you're playing a game that has a bunch of bricks across the screen, and when your ball is supposed to be hitting one of the bricks, you wanna check to see is there actually a brick there? You could call GetElementAt, and if there is some little brick, which is actually, say, a GRect, it will give you back that GRect object, which then, for example, you might wanna remove.

Just a few hints – uh-huh?

Student: On the x-y coordinate, is it – it's not the x-y coordinate that's the top left-hand corner, right? It's just [inaudible] –

Instructor (Mehran Sahami): This is an x-y coordinate of the whole screen of the canvas.

Student: [Inaudible]

Instructor (Mehran Sahami): And if there is some object at that x-y coordinate anywhere – if that point is in – has an object that is – that point is encompassed by that object and will return the object – and just the foremost object. That's why you care about the stacking order because if there's multiple objects there, you get the frontmost one.

GetWith and GetHeight you've probably knew about when you were centering stuff. I guess the width in pixels of the entire canvas and the height in pixels of the entire canvas, which is for the entire screen if you're talking about a graphics program.

And SetBackgroundColor – that's kinda new, kinda funky. You wanna set a different background color than white, you just pass in a color, and it will set the entire color for that canvas or the entire window in the case of a graphics program to that color, which is kinda fun.

Now, there's a couple other things that only graphics programs do. GCanvases do not do these, just graphics programs. And they're very useful for games and animation.

One is called PauseInMilliseconds. What it does is [inaudible] computer these days run real fast. If you ran a game without any pauses in it, people would just not be able to play the game. So you can pause for some number of milliseconds, which is thousandths of a second. So what it does when it gets here is basically just sort of stops execution for some number of thousandths of a second and keeps going. So it allows your programs to kind of slow down a little bit to make them playable.

And sometimes, you wanna wait for a click, and there's, strangely enough, a method called WaitForClick. And it suspends the program. Basically, your program just stops executing until someone hits the mouse button. And then once they click on the mouse, it keeps executing again.

So those are some useful things for games or even for debugging sometimes, if you're doing a graphics program, and at some point you wanna stop and say, "Hey, let me stop the program here and see how far it's gotten," and I don't want it to go any further. So I click. You could just stick a WaitForClick call inside your graphics program.

So here's the class hierarchy for just the GObject. This is the stuff we're gonna focus on here. You've already seen GLabel, GRect, GOval, and GLine. We're gonna spend a little bit more time on GLabel to get into the specific nuances of it. And then we'll talk about some of the other stuff as well.

So here are some methods that are common to all GObjects. So everything that is a GObject, which includes all of this stuff, is gonna inherit all of these methods. So a lot of these you've seen before. SetLocationForAnObject – you've set its x and y location; Move, which you've set its offset by some dx and dy, which is just how much in x direction and how much in the y direction your object should move. That's real useful for animation, and I'll show you an example of that in just a second.

GetXAndY – this just returns the x coordinate of the object and the y coordinate of the object, which, for example, would be the upper left-hand corner for something that's large.

GetWidthAndHeight – this is actually pretty useful. Turns out if you have some rectangle, you know what its width and height are. Later on, you might wanna, if you forget – you forget the [inaudible], you could actually ask it what's your width and height?

This is actually really useful for text because if you wanna draw some piece of text centered in the screen, you actually don't know how big those characters are until you actually figure out what its font is and all that stuff. So you – a lot of times you wanna create your text. I'll show you an example of this momentarily. And then after you've created the GLabel, say, "Hey, what's your height and width so I can center you appropriately on the screen?"

Contains, which is also kind of similar to GetElementAt, but a little bit different – it actually returns true or false. It basically returns true if an object contains the specified point. So this same method you call on a particular object. So you can tell some rectangle, "Hey, do you contain this point, rectangle?" And it'll say "yes" or "no" or "true" or "false" if it actually contains that point.

SetColor and GetColor – hey, it's a pair of setters and getters, just like you saw with students. Uh-huh, question?

Student: [Inaudible] rectangle, or [inaudible]?

Instructor (Mehran Sahami): Pardon?

Student: [Inaudible] contains the area of the shape, or the [inaudible]?

Instructor (Mehran Sahami): It has to do with the – what the particular shape is yourself. So this is one of those things where I would ask you to actually to do it experimentally. So there are a lot of things you can just try out. Just try the experiment. If you want, you can read it in the book, but it's actually more fun to try the experiment because you'll get somewhat funky behavior.

GetColor and SetColor, as you would imagine, sets the color of the object or gets the color of the object.

Here's one that we haven't talked about so far. It's kinda funky – SetVisible and IsVisible. So SetVisible – you set it to be either false or true. If it's false, the thing becomes invisible. If it's true, it becomes visible. You might say, "Hey, Marilyn, how is that different than Remove? I thought Remove takes an object off of the canvas."

This is not taking an object off of the canvas. It's just making that object invisible, which means if you wanna have some object on the canvas and flash it, for example, to be there or not be there, all you need to do is set its visibility to alternate between true and false.

You don't need to keep removing and adding it back on because removing it and adding it back on potentially changes the Z-order because that object now gets tacked onto the front. If this object was in the middle somewhere, you don't wanna change the Z-order by removing it and tightening it back onto the front. You just wanna make it invisible where it is. And you can ask an object for its visibility.

Last, but not least, if you wanna change that Z-order – if you're actually a big fan of drawing programs, a lot of these methods will look familiar to you. SendToFront and SendToBack brings an object either the front or the back of the Z-order. SendForward or MoveForward moves it back one level in the Z-order if you wanna actually just re-order stuff in the Z-order of objects.

Uh-huh?

Student: If you set an object to be invisible, and then you [inaudible]?

Instructor (Mehran Sahami): I knew you were gonna ask that. Try it. Try it, and you'll find out because a lot of these things are actually interesting, and then you realize that you would probably never do this in real life. But if you wanna try it out, it's more fun to be experimental than just giving you the answers for a lot of these things because then you'll never try it. So I want you to at least try a few of them.

Methods defined by interfaces – what does that mean? What is an interface? So there's this funky notion we talk about in computer science – or actually, it has a specific meaning in Java. We talk about it in computer science in general as well. But it's the notion of an interface. And an interface – sometimes people think of, "Oh, is that like a graphical interface? Is that like using my mouse and little things that appear on my screen?"

That's one kind of interface. That's the interface that humans work with. There are interfaces that programs work with. And basically, all an interface is – the way you can think about this is it's a set of methods. That's what, in some sense, defines the interface. And why do you care about defining some particular set of methods? Because what you wanna be able to say is there's a set of object or set of classes that all have these methods.

So you have some set of classes that have that set of methods. That seems like kind of a funky thing. Why would you wanna do that? You might say, "Well, hey, Marilyn, there's kinda a similar concept. Isn't that whole concept of inheritance you talked about sort of like this because if you have over here your GObject, and you have something that's a GLabel, and you have something else that's a GRect, you told me a GObject has some set of methods, so doesn't the fact that GLabel and GRect are both GObject – aren't they some set of classes that have the same set of methods?"

Yes, that would be true. So then you might ask yourself, “So what’s different about this thing called an interface than just inheriting from some class?” And the difference is that sometimes you want this set of methods to be shared by a set of classes, which don’t have this kind of hierarchical relationship.

So some – an example of that might be, for example, a class called an Employee. And students, for example, can be employees, but there are gonna be people who are not students who are employees as well. And there might be some entirely different class of people who are employees. So if I had something called an Employee here, and I might say, “Hey, well, at Stanford, I have a bunch of different specializations. I have my Frosh Employee, and I have my other Employee over here that’s a Senior Employee.”

And then Stanford comes along and says, “Yeah, but there’s also this thing called a Professor, and not all Professors are Employees.” And you’re like, “Really?” Yeah, sometimes it happens. It’s weird, trust me.

But it turns out that sometimes you cannot only have some person who’s a Senior Employee who’s an Employee but some person who’s a Senior Employee who’s also potentially a Professor.

And you’re like, “But Professors have some methods associated with them – not many, but they have some.”

Employees have some methods associated with them. So there’s these different sets of methods that I want, for example, to have one of these guys to be able to share. But there isn’t a direct hierarchical relationship. That’s what you specify in an interface. You basically just say, “If Interface is the set of stuff,” and I’ll tell you which classes actually provide that set of stuff. And they don’t have to have any kind of hierarchical relationship, but just the way of decoupling the hierarchy from having some set of common things that you’d like to do.

So let me show you some examples of that in the graphics world. So there’s something called GFillable. That’s an interface. So GFillable is a set of methods that a certain set of classes have, and the certain classes that have it are a GArch, a GOval, a GPolygon, and a GRect. So notice you might say, “Hey, Marilyn, yeah, those are all GObjects. Why isn’t everything that’s GObject GFillable?” Because there are some things like a GLabel that’s a GObject that’s not fillable. So that direct hierarchical relationship doesn’t hold. I have some subset of the classes that actually have this.

And Fillable are just the things that can be filled. So SetFill I set to be either true or false, just like you’ve done with GRects or GOvals in the past that fills it – that either sets it to be just an outline or filled. I can ask it if it’s filled by saying IsFilled. That returns a Boolean true or false if it’s filled. I can set the fill color. I can get the fill color – getters and setters once again rearing their ugly heads.

So Fillable is just a certain set – it's an interface, and there's a certain set of classes that – what we refer to as implement that interface. That means they provide all the methods in that interface.

And there's some other interfaces. There's Resizable. GImages, GOvals, and GRects are resizable, which is kinda funky. And Resizable just means you can set their size, so you can set the dimensions of the object to be different after you've created the initial version of the object. Or you can set the bounds of the object, which is both its location and its width and height. You can change those after you've created the object. A GLabel I can't do that with.

And there's one more set of interfaces, which is called Scalable or GScalable. And a whole bunch of things actually implement GScalable or provide for you the set of methods in the GScalable interface. GArcs, GCompounds – some of these we haven't even seen before, and you're like, "Marilyn, what's a GArc?" Don't worry, we'll get there. You've seen GLine. You've seen GOval. You've seen GRect. You'll get all the other Gs in there.

It's all about the G today. That's just what's going on. And so you can scale these things, which you give it some scale factor, which is a double – a value of 1 point [inaudible] means leave it unchanged. That's basically 100 percent.

Now, you can potentially scale it. If you give it a value like .5, it means shrink it by 50 percent in the x and y direction if you give it this version. If you give it this version, you can actually set the scaling in the x and y direction to be different, and I'll show you an example of that momentarily. It's kinda funky. It's fun. You can destroy your pictures. It's easy. It's one method.

So let me give you a little animation just to put a few of these things together – a little bouncing ball. So if we come over here – and you should have the code for this in one of your handouts – I'm gonna create a little bouncing ball. And I'll go through some of the constants pretty quickly. The ball has a diameter and some number of pixels. There's a gravity, which is at every time step, how much more quickly is the ball going downward? How much is it affected by gravity? So every cycle, its speed is increased by 3.

[Inaudible] some delay in number of milliseconds for the bouncing ball? Otherwise, it'll just go way too quick, and you won't see it, so it'll have a 50-second millisecond delay between every update of the bouncing ball.

The starting x and y location of the ball is basically just at – y is at 100; x is near the left-hand side of the screen because it's basically the ball's diameter divided by 2. The ball has a constant velocity in the x direction, which is 5. And every time the ball bounces, it loses some speed, so how much of its speed does it keep? It keeps 90 percent of its speed, basically. That's what the ball bounces. It's just – what fraction of its speed does it keep as it goes along?

And so our starting velocities for x and y – or the x velocity's never gonna change. It's just gonna be set to be whatever the starting x velocity is. We're never actually gonna change it. The y velocity, which is how much the ball is moving in this direction, starts at 0, and over time it's gonna increase until it hits the ground under the bottom of our screen, and it'll bounce up. And we'll see how to implement that.

And our ball is just a little GOval, so I'm gonna have some private instance variable that's a GOval that's the ball I'm gonna create. And then I'm gonna move it around.

So what am I gonna do? First of all, in my program, I'm gonna do this thing called Setup. What does Setup actually do? So when I do the Run, I call Setup. What does Setup do? Setup says, "Create a new ball at the starting x and y location with a given diameter in both the height and the width." So it's basically just creating a circle at that location. It sets the ball to be filled, and it says, "Add the ball to the canvas."

So basically, all Setup does is it adds the ball somewhere – an oval, a GOval, that's filled somewhere onto the canvas. That's great. That's all Setup does. It created the ball, added to the canvas. Now, how are we gonna animate it?

The way we're gonna animate it is as long as the ball's x coordinate is not yet the width of the screen, which means the ball's gonna sort of bounce across the screen like this – until it's gone to the end of the screen, I'm just gonna keep running the animation. Once it sort of bounces off the right-hand side of the screen, game over. It's gone past the width of the screen.

So what I'm gonna do on every iteration is I'm gonna move the ball. I'm gonna check for a collision to see if the ball has hit the bottom of the screen. And after I check for collision, which is if it hits the bottom of the screen, I need to send it back up, I'm gonna pause before I move the ball again.

So that's my cycle. Move the ball, check for collision, wait. Move the ball, check for collision, wait, sorta like standard animation.

So what is MoveBall and CheckForCollision do? MoveBall's extremely simple. It just says on every time step, I increase how fast the ball is going down, which is its y velocity, by gravity. So I plus-equal its current y velocity with whatever gravity is.

Uh-huh? Question?

Student: Why do [inaudible] oval at the top and then [inaudible]?

Instructor (Mehran Sahami): Well, when I declare it, all it said is set aside the space for that object. When I say NewGOval, it actually creates the object. So I need to create the object before I can use it. By just declaring the variable, I haven't actually created the object.

So up here, when I declare this GOval up here, this private GOval, that just says, “Give me the box that’s gonna store that object that’s a GOval.” I don’t actually fill it up with a GOval until I do the New. So I have to do that and do that to actually create the object.

So MoveBall, I just showed you. We accelerate by gravity, and we move the ball in the x direction and the y direction. The x direction’s always gonna remain constant, so the ball’s just slowly gonna move across the screen, but its y velocity’s gonna change because it’s gonna bounce.

CheckForCollision looks way more funky than it is. Here’s all that’s going on in CheckForCollision. I check to see if the y coordinate of the ball is greater than the height of the screen minus the ball’s diameter. If it is, that means the ball has not hit the bottom of the screen yet because it’s y – sorry, if it’s greater, it means the ball has hit the bottom of the screen because it would be below the bottom of the screen because y coordinates go down.

So if the ball’s y is greater than the height of the screen minus its diameter – because we have to give the ball some room to be able to hit the bottom of the screen, which is the diameter, then if it hasn’t fallen below the floor, there’s nothing to do.

If it would’ve fallen under the floor, then what I do is say, “Hey, you’re gonna bounce off the floor,” which means when you hit that floor, change your y velocity to be the negative of what it was before, which means if you were going down before – your y was increasing – now you’re gonna be going up, or your y is gonna be decreasing.

So change your velocity to be the negative of what it was before multiplied by the percentage of your bounce that you’re keeping, which is 90 percent. So it’s gonna slow down every time it bounces in terms of how far up it’s gonna go, but just by whatever this percentage is. But the important thing is you need to reverse its direction because it’s bouncing off the ground.

Last but not least, and this is just kind of a minor technical point, but I’ll show you anyway. It’s all in the comments. If the ball would’ve fallen through the floor, like it was actually moving so fast that it actually was here in one time step, and next time step, it would’ve been below the floor, we just say, “Take the difference from the floor, and pretend you already bounced off the bottom.”

So we’ll just kinda add that over to the topside, and you’ll just essentially take that amount that you would’ve bounced through the floor and say that bounce already happened and go back out. Slight technical point, but this is the math that does it – not a big deal.

What does this actually look like? It’s a bouncing ball. Ah, animation. You can play games, and then the simulation ends when it goes off that end of the screen. It’s just so cool, you just wanna run it again, but I won’t because we have more stuff I wanna cover.

But now you, too, can bounce the ball, and you can make it different colors. And every time it hits the bottom, you can set RandomColor, and your ball changes every time it [inaudible]. And you're just like, "Oh, my God, there's so much I can do. It's so cool." Yes, there is. And we won't explore it all right now.

But what we will do is explore a couple more things. So our friend, the GLabel class – you're like, "Oh, but Marilyn, I already saw the GLabel class. I've been doing GLabel since the cows came home. I did GLabel on this last assignment. I know how to create a new label in a particular location, to change its font, to change its color, and to add it. So what new were you gonna tell me?" And on that slide, I'm not gonna tell you anything new.

What I'm going to tell you new is on the next slide, which is what's the actual geometry of this GLabel class if you wanna center stuff on the screen, for example?

So there's a bunch of typesetting concepts that are related to GLine – to GLabel. First of them is the baseline. The baseline is basically the line on which the text appears. But notice there are some things that actually go down below the baseline. That's the way typesetting works. You actually have a baseline – some characters, like a j or a p, go below the baseline.

So when you're setting up a particular GLabel, the value that you set for where that GLabel should actually be placed is the origin of the baseline. It's the far left-hand corner of the baseline. It's not down where you get these descending characters. It's the baseline that most characters are actually sitting on.

So then you say, "Well, how do I get the descent and the ascent and all this other stuff?" Well, before you do that, the height of a font is its distance between two successive baselines.

But more importantly, if you're just trying to center one line in the screen, you care about two things called the ascent and the descent. The ascent is the amount above the baseline by which the tallest character can actually reach. The descent is the distance that any character will drop below the baseline. So for ys and gs and js and stuff, you actually have some non-zero descent, which is the amount below the baseline.

So if you wanna figure out how big something is, usually when we center stuff, we just care about the ascent. We don't actually care about the descent. So if you wanna center something, here's an example of centering something. We again have our little text, "Hello, world." We set it to be big. We set it to be red. We haven't figured out where on the screen to put it yet because we're gonna compute that after we set the size of the font so we know how big it is. Now we can ask this label, "How big are you? How wide are you? How tall are you?"

So to get in – to get it to display right in the center of the screen, its x coordinate is gonna be the width of the whole screen minus the width of the label. So we're gonna take the

width of the whole screen, subtract off the width of the label, and divide the whole thing by 2. That will give us this location right here.

The other thing we need to do is figure out how to center relative to the height of the characters. What we do is we get the ascent. We don't care about the descent of the characters. Standard typesetting kinda concept, but now you know. You get the ascent of your font, which is how high it goes above the baseline. You subtract that off from the width of your window, and you divide by 2. That will tell you what the point is right there in terms of the y direction to be able to figure out how to center this text. And then you add it at that particular y – x-y location.

So now that you know how to actually make your font bigger or whatever, you can figure out the size of your text so you can appropriately space it on the screen.

Couple more things before we wrap up for today. Last thing I wanna cover is this thing called the GArc class. What is a GArc? Actually, there are a couple things I wanna cover. The GArc class is basically just drawing an arc somewhere. It's a little bit more complicated than it looks. Friends of mine that are artists tell me that you can draw everything in the world using just straight lines and arcs. Is that true? Yes? No? Maybe? Let's just pretend it is.

You already know how to draw a line. You have GLine. So now that you know how to draw it – GArc after a couple more slides, you can draw anything in the world. It just might take you a whole bunch of little objects to draw it.

So an arc is formed by taking a section from the perimeter of an oval. What does that mean? Basically, it looks something like this. The steps that are necessary to define that arc is we specify the coordinates and size of some bounding rectangle, just like you did with the GOval. A GOval sits inside some rectangle, and that's what tells you what the dimensions of the oval are. Here, I've drawn it as a circle.

So here's a bounding rectangle, and it's got some upper left-hand location. So we have some x-y upper left-hand location. We have a width and a height. That's one of – what's gonna define for us what essentially the oval is gonna look like.

But we're not gonna draw the whole oval. We're just gonna draw an arc from the oval. How do we specify the arc to draw? We need to specify something called the start angle and the sweep angle, so [inaudible] the start and sweep angle. Both of these angles are things that are measured in degrees starting at this line, so starting at the X-axis.

So if I say for my start is 45 degrees, it goes up 45 degrees and says your start would be here. Your sweep is what portion of the oval you actually draw out in degrees. So if I set a sweep of 180, I will draw out a semicircle because I'll draw 180 degrees starting at the 45-degree mark. And it always draws in the counterclockwise direction if you give it positive values. If you give it a start value or a sweep angle in negative values, it will start on the other side and go in the clockwise direction.

So let me show you some examples of that, because that's kind of a big mouthful. What does that really mean? Here's just a few examples so you know. We're gonna draw some arcs centered in the middle of the screen, so we're gonna figure out dx and dy. Well, let's just assume they're the middle of the window. We already computed them using GetWidth and GetHeight. And d, which is the diameter of the oval we're gonna be using, or the circle in this case, is just preconfigured to be .8 times the height of the screen.

So if we do something like we say, "Create a new GArc that has its bounding box being d and d, that basically means we're gonna have some oval that's height and width are both the same, being d, which means it's just gonna be a circle with diameter d.

We're gonna start at the 0-degree arc. Remember, 3:00 is where we start – 0 degrees. And we're gonna sweep out an arc of 90 degrees, and that goes in the counterclockwise direction because it's positive. So we get an arc that's essentially this little quarter of that circle because it's 90-degree arc starting at 0 degrees.

There are some other things we could do. We could, for example, say, "Hey, you know what I wanna do is start that same circle with the same size. Its height and width are both d. Start at the 45-degree mark, so coming up 45 degrees from sort of this 3:00 mark is right there. Have a sweep of 270 degrees, so we get this big portion of the circle that looks like a giant C. But that's 270 degrees of a circle."

We can do some slightly more funky things using negative numbers, for example, where we say, "Draw a new arc. Your starting point is negative 90 degrees."

What does that mean? It means 90 degrees going clockwise. So I start at 3:00. I go 90 degrees clockwise. That's right here. And then I sweep out an arc of 45 degrees, which is counterclockwise because it's positive, so it sweeps out 45 degrees.

One more for the road – this one says, "Start at 0 degrees, and set your sweep to be negative 180 degrees. So if it's negative 180 degrees, it goes in the clockwise direction," 180 degrees starting from 0, and we get this thing that looks like half of a circle or a smiley face.

So any questions about GArc? There's all kinds of stuff you can do with GArc. But at least if you understand the basic geometry, hopefully it'll make sense.

Uh-huh?

Student: Why [inaudible] fill a GArc?

Instructor (Mehran Sahami): You can fill a GArc.

So last slide before we go is a filled GArc. Good question. Basically if you write a GArc, whatever that GArc is, and you set its fill to be true, what you get instead of just the arc is you get the pie wedge that would've got associated with that.

So just imagine you have some [inaudible] line that sweeps the arc. That's what you get with the filled version of it.

Any questions?

All right, then I will see you on Wednesday.

[End of Audio]

Duration: 51 minutes