Programming Methodology-Lecture25

Instructor (Mehran Sahami): So welcome back to another fun filled, exciting day of cs106a. A couple quick announcements before we start. First announcement, there is two handouts. One of those handouts, which we'll spend some time talking about today, is your last assignment for the class, which is assignment No. 7. It's worth noting, we sort of talked about it the very first day of class when we talked about late days. The very first handout says no late days can be used on assignment No. 7. So that's important to remember. No late days can be used for assignment No. 7 because it's due the last day of the class.

If you're looking at the syllabus, you might say, I didn't think we had class on Friday, the last day of class. That would still be correct. We don't actually have class, so you only need to do electronic submission for turning in your program, but it's still due at 3:15 on that Friday. We just don't actually have lecture on that Friday. We'll talk about the details of that assignment as we go along.

The other handout that you got is the example of the code that you say last time for the Fly Tunes music store. So you should have all that code, which shows and example of a big data structure. In fact, the data structure for that program is perhaps more complicated than the data structure you're going to need to use for the face pamphlet assignment, No. 7.

So as you know, assignment No. 6 review today. I have a feeling a lot of people may have been taking late days on it because it was, for example, the last assignment to actually be able to use late days on. So we'll just do a quick, painful as to how much time it actually took you. Hopefully it wasn't too painful, but we'll see.

Just wondering, how many people are taking a late day? Wow. So for the six of you who actually are turning in the assignment on time, anyone in the zero to two hour category? I would be frightened if you were. I don't think it's actually possible to do in two hours. Maybe, if you just sat there and wrote all the code. Two to four? Oh, I actually have a couple for two to four. Wow.

Four to six? Good to see. I think all the [inaudible] is just going to be smaller. Six to eight? That's a pretty healthy contingent. Eight to ten? Also good-sized contingent. Ten to 12? Couple of folks. 12 to 14? 14 to 16? 16 to 18? 18 plus? Anyone in 18 plus and still hasn't turned it in, like you're just going for late days. All right. That's good.

So that's good to see. It's a good time. World is still normal, though the normal curve just happens to be a little smaller because there's more people taking late days, but it's a good time. If you haven't done it yet, two things. One is that this should give you an idea about how much time it'll take, perhaps. Two is you don't want to push it. Even if you have late days, don't push it off too late because you really want to give yourself time to do assignment No. 7.

So any questions about anything we've done so far before we kind of delve into today's great topic? All right. So I want to spend the beginning of class actually talking a bit about your next assignment. So if you're still kind of in the mindset of thinking about name surfer, and you're like, oh, I'm just so in the mindset of assignment No. 6, take that little part of your head that's got assignment No. 6. Pop it off the stack. That's my whole head. Yeah, that might be painful. Pop it off the stack for a second, and we're going to think about our friend, the face pamphlet.

So just a quick show of hands, how many people are part of a social network or do some social networking online. That's good to see. So I won't need to spend too much time telling you about what a social network is, but I'll spend a little bit of time in case you're not familiar with social networks.

Basically, a social network is just a way of keeping track of some set of people, or profiles of people, and relationships among people. At the simplest level, that's all it is. So what you can think about is you have some profile for a person. In some sense, you can think of the notion of a profile and a person interchangeably most of the time. But as most of you know, if you've ever read the New Yorker and seen that cartoon that no one on the internet knows you're a dog.

So you don't necessarily have to be a person to have a profile. So when we think about a social network in the abstract sense, we always just talk in terms of profile. But if want to think about it concretely, you can just think of a profile as being synonymous with a person, but it need not be, so we generally refer to a profile.

So what a profile has associated with it, it has some attributes or some things that we keep track of for that profile. For example, it has a name. In the case of your assignment, it's also going to have some status that we care about, keeping track of the person. It might have some other things, like the reason why it might be called face pamphlet is because say you want to keep track of people's faces. You want to actually see what they look like.

So you might actually have some image in there. Then I mentioned you want to keep track of some list of friends. So there's some notion of a list of friends that is also associated with a profile. Real social networks also have other things associated with them, and the assignment talks about extensions you can do, but we're not going to get into extensions right now. But these are things we want to think about.

For a person's profile, there's a name. One of the things based on some of the ideas we talked about in the last lecture, is you can think of name as a unique identifier. What that means is that you will never have two profiles in your social network that have the same name. So once you have a name in your profile, in your social network, that name is yours. That profile will be associated with that name until, at some point we decide, maybe we want to delete the profile to get rid of it. But the whole time the profile's actually live, it will have that unique identifier, which is the name of that profile.

So if you ever try to create another profile with the same name, the application will tell you that you can't. I'll show you some examples of that in a bit. But the name is a unique identifier, and this thing's just some string that we keep track of the name. So it can be like your real name or whatever name you want to have, like Code Dog. That can be your profile name. That's cool, too.

Status is just what you're doing right now. You can think of this as some string which is what you happen to be up to at the time. Like Maron is teaching. I don't know. Ben is contemplating life. It's just some particular string of the thing you happen to be doing. You might want to change this a fair bit.

This image thing is just an image. In our world, that means it's a G image. That means it's some picture. Originally, when you create a profile, you may not have a status, and you may not have an image. You may want to add one. You may want to change one as life goes on. We should be able to allow for that. Then you have a list of friends. When you're born into the world, your list of friends is empty. Over time, you can gradually build up friends.

That's the other part of the social network to consider, besides the fact that we have profiles is that we also have some notion of friendship. The way you can think of friendship is basically some profile or some person just has a list of friends. So an easy way to think of a list of friends is a list of names of friends because names are unique identifiers. So if we keep track of a list of names, we have a list of the unique identifiers that are our friends.

That's an easy way to be able to look them up, for example. The idea behind friendship that's key to us – this is not necessarily true in the real world, sadly enough, but for our intents and purposes, as far as the assignment's concerned, friendship is reciprocal. That means the if your friendship is reciprocal. Not all the time. That's the way it is.

What the really means is if I have two people, like I have Bob over here and I have Alice over here, you can think of those as profiles. If they are friends, we draw a line between their names in our little social network. This is not actually how we keep track of them in the application that you'll see, but this is often how social networks are drawn.

We put in sort of a circle for each profile in the social network. Here's Don over here, and here's Chelsea. Notice there's the ABCD phenomenon. Always a good time. So when we put what we refer to as a link between two people, that means they're friends. Friends are reciprocal. So if Bob is a friend of Alice, that means that Alice is a friend of Bob. That's just the way life is all the time in face pamphlet.

It's just a happier place. Everyone's like, oh, I'll be your friend. Okay. Then you're my friend, too. So that's the way you want to think about it. So if Bob ever adds Alice as a friend, that automatically means Alice becomes Bob's friend. There's never a directionality of friendship. Bob clicks, oh, Alice, be my friend. She's like, no. Talk to the

hand. That happens in real life all the time, but not here because it's just a happier place to be in.

So think of friendships as reciprocal. Again, we don't necessarily have to have anything explicit to model this link. All we need to do is think about just having a list of names. So Bob might have Alice in his list of names of friends that he's keeping track of. That's how I would keep track of the link.

What that also means is that reciprocally, Alice should have Bob in her list of friends because the links are always reciprocal. When we actually draw this out, if we want to draw the stylized picture, we could say Alice is friends with Chelsea and Don, and Don and Chelsea are also friends over here. This is where the name network comes from. When we draw this thing out, eventually you have millions and millions on these things in there, all over the place. Everyone's holding hands and singing Kum Ba Ya. You have a social effect because you have friendship, and what you actually end up drawing is something referred to as a network.

If you're really the mathematical type, this is really a graph. In this case, it's an undirected graph. If that kind of talk makes you hot and bothered, like, oh, a graph, yeah. Take CS103B, which I'll be teaching next quarter. Graphs are just the coolest, most wonderful thing. But that's a thought that's not important right now.

What is important right now is that this is where the name social network actually comes from. We create this network that keeps track of social relations. So if we have that base idea of social networks, how do we actually think about writing an application to keep track of them. So [inaudible] the computer, I'll show you what, at the end of a week and a half's effort, you will have your own social network application.

So we're going to call it Face Pamphlet because it's not really big enough to be a book. It's a tiny book, so it's a pamphlet. So when it starts up, you can see there's interactors galore all over the place. Over on the north side, we got a name. We got some buttons over on the west side. We got some status, some pictures, some adding friends. What does this all mean?

We'll actually go through an example and sort of create a little social network. So what I'm going to do is create a profile for myself. You're like, oh, that's kind of narcissistic. That's okay. I didn't want to – oh, we're going to create a profile for you right now.

So we add the profile on here. Here's what we get in the display. There's a bunch of display elements. We have a name up there with some big font, and it's blue. We don't have an image right now, so we just have a place holder. That's basically just a rectangle with some text. It says no image. We don't have a current status, but what we've done is created an initial profile with the name, Maron Sohami, which is a unique identifier in the system. So right now, our social network has what we refer to as one [inaudible]. There's one profile in there, and I don't have a list of friends.

You might say, okay, that's not so exciting. Let's actually add some stuff. So once I actually have this active profile, I can say, let me change the status. Right now, at this very moment, I happen to be teaching. So when I click change status, it says Maron Sohami's teaching. Sometimes I could just be, no, I just really want to be doing this.

So I can keep changing the status. That's fine. It just updates here whenever I type it into this field and click the button or hit the enter key. I can also add pictures. So I can say, for example, MaronS.jpeg is a picture of Maron. There's a picture that gets loaded. It gets, unfortunately, scaled to always take up the space for the image, which is a 200 by 200 region space.

It's constants that are given to you in the handout, but for these purposes, it's 200 by 200. So this is kind of the bouncer version of Maron in the days of yore, when I was clean shaven. There was actually a time when I shaved. That doesn't happen very often anymore. If you were ever wondering what I look like without the beard, not that I care. But that's what it was like.

So we've created a profile. Not a big deal. You could say, hey, next quarter, I'm planning on taking 106B, and isn't that taught by Julie Zolanski? Yes, in fact, it is. So let's add a profile for her. So we type in Julie Zolanski in the name field up here and click add. We now get a new profile for Julie Zolanski. Julie is just a fiend. She codes like a fiend, so she's coding like a fiend. That's what she's doing.

So you can see down here, not only is all this stuff getting updated as we typed it in, but down here, there's something we refer to as the application message. The application message always tells you what was the last thing you did or tried to do. So your status was updated to coding like a fiend was the last thing you did. Let's give Julie a little picture over here. JulieZ.jpeg, and there's Julie, also scaled to 200 by 200. So she's no longer coding like a fiend. She's kind of feeling stretched.

So now we have her picture. We have her profile. That's all good and well. Let's add a couple more profiles real quickly. So I'll add some of our section leaders. I'll add Sarah K. I'm just going to abbreviate. Notice these fields over here don't necessarily get reset when I create – they just have their old values in them. That's not a big deal. You could add it as an extension if you wanted to, but they just leave their old values.

Sarah K. Sarah, is it okay if I use your picture? All right. Here we go. There she is, and she's actually scaled reasonably well. So we'll also add Ben. There's Ben. We'll add a picture for him. If you're wondering where all these pictures are coming from, in the starter project that you'll get, you'll get a folder of images that contains the images of all the section leaders from the class as well as Julie and myself.

I'm scaling until the cows come home. If you have your own pictures, you can add them into the folder called images, and you, too, can display your image at a 200 by 200 resolution. Ben, where are you? Here's Ben.

Well, Ben's feeling kind of anonymous. So I gave you two anonymous-ish images as well. There was a Stanford logo, which is just a super low-res version of our logo. Then there's also Stanford Tree. So you can change the picture to one of those if you want to. Those just come along for the ride.

If you do the web version, actually, the web version only has these two Stanford logo and Stanford tree images. I didn't put everyone else's image on the web version because for a little bit of privacy. Was there a question?

Student: [Inaudible].

Instructor (Mehran Sahami): No. If I empty the field, one of the things it says in your handout is anytime the field is empty, it just gets ignored when I try to do something. So any field, whether or not it's name or any other field, just ignore it. But I could have something like the nonexistent picture.jpeg, and I try to change the picture to that, and it says unable to open image file.

So in your handout, it actually explains how you can detect if an image exists or not. So you can actually check the case where the image doesn't exist. If it doesn't exist, we just don't update the image to the new image. Really, all we're keeping track of here is the name, which gets created with the initial profile because it never changes, an image that we display, whatever the current status is. If currently there is no current status, and a list of friends.

So far, we've add a bunch of profiles. What else can we do? Well, there's a lot of things we can do. Let's look up Maron again. So if I type in a name I can click look up, and if that profile exists with that name, it brings it up and says you're displaying the profile name down here. Then we can add some friends. So Sarah is a friend. Add friends. So it says Sarah K, added as a friend. Ben is also a friend. So I add Ben.

I can try to add someone who doesn't exist in the system. Like I can try to add – who should I add? Jenny. 867-5309. I add Jenny. Jenny does not exist. As far as I know in my social network, Jenny's not there. I get this message down there.

Remember, friendships are supposed to be reciprocal, so I have Sarah and Ben as friends. I never went to directly update profiles, but if I now look up Ben's profile – Ben N – Ben has me as a friend. So when I added Ben as my friend, at the same time, I got added to Ben's friend list. That's something that you're going to need to manage in the application. Friendship should always be reciprocal. So when you add a friend to a particular profile, you also make sure to go to that friend and add the current profile to that friend.

Now, another thing we can do, besides just creating profiles and done friends, we can look up profiles, like when we had Maron's profile over here. We can look up a profile. If I try to look up a profile that doesn't exist like Maron Sohami Bob doesn't exist. It clears what profile's in there. It says a profile with the name Maron Sohami Bob does not exist.

When there is no profile in the current application – all this stuff, by the way, is explained in the handout. This is just to kind of show you to get a feel for how it works. You don't need to jot this all down in notes. When there is no current profile displayed, I cannot change any of the things in the profile. It says please select a profile to change status because I can't change the status of a non-existent profile or change its picture or add friend to a non-existent profile. But the way I get a new profile displayed in here is I either add a new profile or I look up an existing profile. Then there's a profile there that I can change to whatever I want to.

Now, let's say I've looked up this profile. I say, that's great, and Ben decides he's had enough. Get me out of your social network. So I say, oh, I'm sorry to hear that, Ben. Here's Ben. I'm going to delete Ben from my social network. Even though I'm displaying my own profile. I click Ben up here, and I delete him. Whatever profile was there, it doesn't matter if it was Ben's or whoever else, it cleared, and it says profile of Ben N deleted.

Now what happens when a profile gets deleted? It gets cleared out of the social network so it's no longer there. If I try to look it up, it says a profile with the name Ben N does not exist because, sorry Ben, you've been nuked. You're out of the social network. But Ben was friends with people. He was friends with me, and now it's a sad day. So I go to look up Maron. I should've just used the first name.

Ben has been removed from my list of friends. Sad day, but that's life in the city. You can't be friends with non-existent people. So when someone gets removed from the network, you go through everyone else in the network and say, did they have them as a friend. If they did, they get popped out. I guess you're just not a friend anymore. That's basically the whole social network. That's the whole idea.

So question?

Student: [Inaudible]

Instructor (Mehran Sahami): Okay. Names are not case sensitive, so I can have a Maron Sohami, and then I can have a screaming Maron Sohami, and he's just different. He's angry. I don't know if we have an angry image in here. I don't know if any of the section leaders were actually looking angry, so we'll just make him the Stanford logo. This is different than Maron Sohami, lower case. So you don't need to worry about it like you did in the last assignment. You were worrying about case sensitivity. Here, you don't need to worry about making things insensitive. You can just case – everything is case sensitive, so you don't need to worry about making strings equal to each other, case sensitive and stuff like that.

So any questions about this general idea? You got to think about it, right? Any more questions?

If you kind of think about it, this is the basics of the social network. You have all the people. You can have as many people as you want from the social network. You can maintain friendships. You have attributes of the people, which in this case, just happens to be their name, their image and their status and their list of friends, but you could imagine – you could extend this in a whole variety of ways.

Really, what this is about is data management. There's a display component. In part of the assignment, there's a bunch of milestones that are laid out. The actual only milestone that deals with this display is the very last milestone, which involves a bit of work to get all this stuff displayed at the right locations. There's a bunch of constants, which is all the places, you put things at the right locations.

But it's really about managing data. How do you manage profiles, managing the friendship relationships, being able to remove stuff, add stuff, which is what most applications in the world are really about. So you're going to get a chance to really exercise your data structure skills and your larger-scale application skills in this program. So any questions about this program?

There's also a web demo of it, so if you want to go to the web and play around with it, you can play around with it. Again, in the web version, there's only these two images, the Stanford logo and the Stanford tree.

Student: [Inaudible].

Instructor (Mehran Sahami): No. That's the other thing. It's in big font somewhere, like on page 15 in the assignment. So don't be scared by the fact that the assignment handout is 28 pages like some people are looking at. Oh, my God, this is so big. The code is much shorter than the handout, first of all. Second of all, one of the things – all the details are in there, but one of the important things that comes up in the handout is you don't need to worry about scaling.

So if I shrink or increase, yeah. No worries about size or resizing. I figured you got enough practice with that with name surfer. Either you already got practice with it or you're getting practice with it with name surfer. So we don't need to make you do it again. So you don't need to worry about component listener and resizing and everything. You can just lay this stuff out with t respect to the constants that are given to you, and you're okay. But resizing, you don't need to worry about it.

It can be a cool extension if you want to add it as an extension. That's perfectly fine. There's a bunch of extension ideas that are listed at the end of the assignment. As you can kind of imagine, if most of you are on a social network, you probably already have thousands of ideas for extensions, if you want to add them, but you're not required to add anything in particular.

So any other questions about this? All right. So with that said, I just want to give you one last little – before we moved into the advanced topic for today, one last little side note

about social networks. Has anyone ever heard of the phenomenon, six degrees of separation? A few folks. There was a movie years ago about this phenomenon, and the movie actually had nothing to do with the phenomenon. It just happened to have the name, Six Degrees of Separation.

But it's basically this idea that everyone in the world is linked, potentially, to everyone else by at most, six hops in this graph, which is kind of weird to think about. I won't go into all the details an mathematics and stuff, but there's some very interesting stuff behind it. But the biggest idea is that these social networks, you can sort of think of little hops between people.

You can say Bob is one degree separated from Chelsea or two degrees, depending on how you count, from Chelsea because you can get to Chelsea by going Bob, Alice Chelsea or Bob, Don, Chelsea, whatever the case may be. But just in case you're wondering where the six degree of separation idea came from, Stanley Milgram – anyone ever hear of Milgram's experiments like obedience to authority, if you're a psychology kind of person? Same Milgram.

We won't get into that. That's just a whole bunch of other interesting psychological phenomenon that are discussed on the other side of the quad. But what he did was said – this was back in the '60s, I think. '67, actually. He sent a whole bunch of packages to various people in Nebraska. So let's just say that's Nebraska.

He sent a whole bunch of – yeah, my notion of geography – to random people in Nebraska and said, these packages need to get to someone who's over here in Boston. Yeah, I have no idea. It's Boston. That's Massachusetts, and here's Boston. These packages need to get over here, but he didn't give them an address. He just gave them a name of random people in Boston the packages need to get to.

The only instructions he gave were to say you can't try to figure out who this person is and send them the package directly. What you need to do is send this package to someone that you know who you think will be more likely to know this person than you are. Those were the only instructions. Then when the packages was trackable, how many times it got mailed.

He measured how many times the package had to be mailed to various people before it actually got to its destination. What he found was there was an average of about five to six for the packages to get from a random person in Nebraska to a random person in Boston. That's where the six degrees idea comes from, but that's – now when people talk about social networks, they talk about things like degrees of separation and what we refer to as small world phenomenon. If everyone's linked by only six degrees of separation, then it really is a small world after all. We can all hold hands and be mechanical animals. If you've ever been to Disneyland, that makes sense. If you've never been to Disneyland, you don't need to go on the small world ride now.

So time for something completely different. Are there any questions about social networks before we delve into our next great topic? If you're really interested, you're like, Maron, tell me more, take CS103B. It's a good time.

The advanced topic that you're going to get – and I hesitate a little bit to tell you this, but this is something that you don't need to know for the final exam. As a matter of fact, you don't need to know it for assignment No. 7. It's just something that's so cool with the way programs are actually done today, it's something you should actually know. That's a notion called concurrency.

It's also important for you to know if you want to understand, for example, in next class, for example, I'll show you how to take the programs that you've written in this class and turn them into applets or turn them into executables so you can share them with your friends. You'll need to understand this idea.

So the basic notion of concurrency is that if you think about it, your computer, there's actually multiple things happening on your computer at the same time. It looks like there are multiple things happening at the same time. You're like, there's oftentimes – I got my email program open and I'm IMing, and for those of you out in TV land, I'm watching a the lecture at the same time. Most of the time, I'm not actually watching. I'm just IMing and doing email.

But to you, it looks like all these things are happening simultaneously on your computer, right? In fact, they're not. Your computer only has one – most of you. These days, things are changing a little bit, but most of your machines only have one processor in them, which means at any given time, really, only one thing is happening.

How does it look like multiple things are happening at the same time? That's the notion of concurrency. What's really happening is the computer or your operating system is saying there's all these things that want to be happening at the same time. What I'm going to do is give each a little piece of time to go and execute a little bit, and then I'll do the next one and the next one.

So email gets a few milliseconds and IM gets a few milliseconds. Then the video gets a few milliseconds, but that's enough time to update to the next frame of the video. I just keep cycling through these super fast. So to you, it just looks like they're all happening at the same time.

If you think about this, you actually did this already. Which assignment did you do this on? Breakout, right? Breakout was doing this. You had the ball, and you had the paddle, and you had to check for collisions. So what were you doing? You were saying, hey, ball, move yourself. Wait some time. Hey, paddle, update. Maybe wait a little bit of time. Hey, check for collision, and I just keep doing these over and over.

It was happening so fast that it looked like the paddle and the ball are moving at the same time. But you knew they weren't moving at the same time. You knew that when the ball

was moving, the ball was getting the message to move. So paddle wasn't actually moving. Paddle got moved later when your execution flow got to that place and moved the paddle.

So you've already seen a notion of this. There's a concept around this notion, which actually makes it much more concrete, called the thread. The idea of the thread or a thread of execution, is to say that the way you can think about a program is that rather than a program just doing one thing, a program can say, hey, I'm going to have some thread over here and some thread over here and some thread over here.

I'm just going to kick these off. I'm going to say, hey, there's some piece of a program I want to do here. So just start it and start executing. There's some other piece of a program I want to do at the same time. Just kick it off and start executing. Same thing over here. I don't want to have to worry about doing this cycling myself. I want someone else to say, oh, you get a piece of time and a little piece of time and a little piece of time. It just keeps cycling through. This happens so fast, much faster than my moving hand.

It all looks like it's happening at the same time. That's the notion of a thread of execution. If you can think of a program, what we refer to as being multithreaded, which means it actually has multiple threads of execution that are happening at the same time, which is a little mind bending. Actually, it turns out to be a fairly straight-forward kind of thing.

So how do we make this a little bit more concrete? How do we think about threads in JAVA's world. It turns out, interestingly enough, you've been sort of doing this a bit the whole time.

There's something called the runable interface. What the runable interface says is that anytime I have a class that implements the runable interface, that means that the class can be turned into a thread. It can be kicked off as a thread because it has some notion of running. So I can say, you know how to run. Okay. I'm going to create a thread, and you're going to go run in that thread. It's like having a little baby and sending them off into the world. It's sort of like what your parents did when they sent you to college.

The whole time you were part of your parent's process. They were managing you. They were like, eat, sleep, homework. At a certain point, you were like, okay. I'm done with that cycle. When can I go do my own thing. Then you were like, oh, I'm graduating from high school.

What they did was your parents said, now it's time for you to get a thread. You were like, okay. I'll figure that out when I get to college. Really, they said that. Maybe you just weren't in the room at the time.

Then they sort of kicked you off. They said go do your own thing. You're going to be doing your own thing at the same time everyone else is doing their own thing. I'm not going to manage you anymore. You're just going to go do your own thing. Hopefully, that thing will turn out well.

So what is the runable interface? Let's just look at a simple example. I have public, class, my class, and this is just an interface, so I say implement runable. We can just make up words when we're computer scientists.

This class is going to have some constructor in it, so allow public my class. It needs not actually have a constructor that's explicit, but let's just say it has a constructor. Then there's only one method that a runable interface – the class the implements the runable interface needs to implement. That's public void run. You should see that at this point and go, oh, my God. I've been doing threads the whole time in this class.

That's what you've been doing. Guess what? Program implements the runable interface. So you wrote a method called run, and what we did somewhere in the bowels of the ACM library was we said here's your program. Go run. So it started executing the run method. That's where all the stuff happened in your program.

Now, when you wrote programs that had interactors with them or looked at mouse clicks or whatever the case may be for events, your program – so this is my program over here, for example, that's running. There was another thing running over here that was called the event thread. The event thread was actually the one that was like, mouse click. Yeah, when that mouse click happens, I know about it because I'm managing the mouse clicks.

When that mouse click happened, it said, hey, I got a mouse click. Who's listening for the mouse click out there? You were like, me. I added mouse listeners. I did. It says, okay, I'll let you know there was a mouse click. Then you did something based on that mouse click. You stopped, and it was like this thread kept running. You program only got called when some mouse event happened. It sort of said, okay. I'll let you know that this mouse event actually happened.

So when you were dealing with mouse clicks and mouse movements and all this stuff, there were actually multiple threads running. There was a separate thread that was getting events and sort of telling the people who were listening for those events that they actually happened.

So with that kind of idea, how do we actually create a thread? So I've sort of told you, if you want to create a thread, you need to have a class that implements that runable interface. How do I actually create the thread itself?

So here's how we create the thread. We create something of the class that is runable. So we might say, I have some object, X, that's [inaudible] my class, and my class implements the runable interface. So this is just going to be some new my class, just like creating objects of any class. It's just another class. It just happens to implement the runable interface. So you create one.

Now, once you create one, this is sort of like your birth. You're born/myth up to age 18 or whatever it was. For some of you it's 12. There were actually – I heard that there was someone very young at Stanford this year, but I won't verify that. So let's just say your

life up to age 19 happened. Then at some point, your parents said, hey, I'm going to turn you into your own thread. You can just go an execute independently. You're like, well, what about the tuition checks?

They're like, yeah, we'll handle those. We've got the tuition check listener going on. So it'll actually pick up those events. But we want to spawn you off as a thread. Now, it's kind of the words that we use in the computer science [inaudible]. Spawn a thread.

So what we're going to do is create an object of type thread, which we'll call C. What we'll say is that's going to be a new thread. What you give to thread is the object that you want to execute on a separate thread. So we give it X. That object has to be of some type that implements the runable interface. So that's how it works.

You create a thread. You say, hey, thread, the thing that you're going to execute is this class, X. And the way – what time you're going to start it is, you say, T. So you tell the tread start. When the thread starts, what it does, it says, hey, object. How are you doing? I'm going to call your run method, and you're just going to start executing in your own thread from your run method.

So that's kind of the basic set up. It's fairly simple, actually. You create an object that implements runable. You create a new thread that's passed that object, and then you sort of kick off the thread. It starts executing whenever you say start. So let's make this a little bit more concrete. Let me show you an example.

So what we're going to do is create a super simple example that's basically just a square. It's a rectangle. It's a G rect that's going to slide across the screen. So let me show you the code. We have some class slider that extends G rect. So our class that we're going to create that's going to be runable is just a rectangle. It's going to implement the runable interface. It's going to have some constructor, and what we're going to say is hey, you're going to actually be a square. I'm going to give you some initial size and some initial color.

So the first thing it does, it says, I need to call the constructor for my super class. My super class is G rect. So I'm actually a G rect that's a size-by-size G rect because I'm a square. But to initialize myself as a G rect, I need to call my super class' constructor.

So I say, super size comma size. I set myself to be filled, and I set my color to be whatever color was passed [inaudible]. Then once I'm created, I'm kind of hanging out as an object until I get put into a thread and run, at which point my run method gets called. When my run method gets called, all I'm going to do is go through a sequence of steps where I'm going to pause for 40 milliseconds and the move myself a small amount in the X direction, over and over.

So I'm basically just going to slide across the screen. That's what I'm going to do. My step size is five. That's my whole object that I want to run. So where's the program that I run? Here's the program that runs it. So the program, because it's a graphics program, it is

going to be a thread that someone else, that you have hither to not seen, is going to create and run. So its run method says, hey, I'm going to add a button that's called slide, to the southern region.

I'm going to add an action listener. I'm going to wait for mouse buttons, or I'm going to wait for you to click that button. If you click that button, so if the command I get is slide, what I'm going to do is create one of these sliders. I'm going to create an object called slider that's of type capital slider.

Its initial size is going to be some constant size that I set to 20. Its initial color is going to be some randomly-generated color. So I have some random generator. I'm just going to give it a random color.

Then what I'm going to do is create a thread that's going to have the slider in it, and I'm going to kick off that thread. So if I run this program, here it is. Nothing's gone on. I click slide. The box gets created of a random color, and it's now in a different thread. It slides across the screen. You're like, huh. Okay, Maron, that was a whole lot of work to create a sliding square.

Now, here's the interesting thing. That square is getting added to this canvas. So any squares I create will get added to the same canvas. As a matter of fact, if I look at the code over here, ever time I get a mouse click, I'm going to come and execute this code. Let me make this little thing go away.

I'm going to come execute this code, which means I'm going to create a new slider of a new random color. I'm going to put it in a new thread. So every time I click the button, I'm going to get a new thread that's executing and parallel with all my other threads. You're like, okay. What does that mean? That means I click this once, and I get a box. I click it again, and while that one's still running, I get another box of another random color. They're all sliding across the screen at the same time. These aren't the droids you were looking for.

So that's the beauty of it, right? I didn't need to manage. Oh, you go move yourself. You go move yourself. No, you wait. You go move yourself. I just say, hey, you go run. I want another box. Yeah, you go run, and you go run, and you go run. They're all running at the same time.

Now here's something that's even slightly cooler than that. What's slightly cooler than that is your threads can interact with each other. These threads were all independent. So one thread was executing here, which was a little box that was moving. Another thread was executing here. There was a little box. Never did the two meet. They were just hey, if I run into you or whatever, that's all cool because we're just all treads here, and we're all executing.

Sometimes, you care about knowing what's going on with your threads or threads care about what's going on in some other thread. So there's actually ways that you can think

about keeping track of data, of having shared data between your threads, which means all your threads are sort of running along. There's some piece of data. So these are all just classes. In these classes that are happily running along, in that class, there's actually some variable that's a reference to some object over here.

Guess what? This guy's also got a reference to that object. This guy's got a reference to that object, which means if anyone of these threads updates this object, the other threads can look at that object and see what its updated value is. This is, for example, how ATM machines work. There's different ATMs that can be updating your bank account, but there's only one bank account that they see. They're not looking at different bank accounts. It's the same principal.

So what are we going to do? We're going to create a race. The way the race is going to work is we're going to create something called the racing square. So it's like, yeah, I'm the racing square. I'm ready to go. It's also going to extend the G rect and implement runable. What does it do? Well, it's pathed at index to let it know which square in the race are you. There was a bunch of squares. They're all going to run across the screen at the same time.

Which index are you? So there's some number of squares. We're just going to index them for zero up to the number of squares minus one. We're also going to pass you a shared array. When we pass an array, arrays are always passed by reference. So you don't get a copy of the array. You get where the array lives. So we're going to give you this array bullion finished, and what that array is going to be is an array for all of the different squares that are running. You know what your index is in that array.

When you're done, you should say, hey, my finished is true. Until you're done, you say my finished is false. So we'll get to that in just a second. What it's going to do is create a square of size by size, set itself to be filled, keep track of its index, and keep track of the list of finishers. So it needs to have some internal variable to refer to this reference finished.

So those are just variables that are down here, right? I have some private integer and some private reference to the array, and so when I set finishers equal to finish, what I just have is essentially that diagram over there. I have some reference to this place where the array lives in memory. I have a random generator because I care about having random generators, as you'll see in just a second.

So what's this square going to do? That's all it does to get itself set up. When I gets run, what it's going to say is in the list of finishers, my index, because that's me, it's going to be false because I haven't finished the race yet. When you sort of kick off the thread, I'm just starting to run. I haven't finished.

To run the race, I'm going to move 100 steps, and in each one of those steps, what I'm going to do is pause some random amount of time and then make some step forward. So my step size, I have step size that is just some fixed constant. Basically, what I'm going to

do is take a bunch of steps. I'm basically going to take steps of size five. I'm going to take 100 of them. So I'm going to move 500, but the amount of pause between each one of the steps I take varies.

So I actually move it potentially different rates over time. That's why it's a race. Each one of these guys will be moving at different rates. Then I need to watch out for photo finishes. So this code is actually buggy, and I'll show you why in just a second. When I get to the end of the race, so after I've run the race, I've taken my 100 steps over time, what I do is say this stuff's commented out for right now. I say, I'm finished. So my – the list of finishers at my index is true. I'm like, I crossed the finish line. I finished. Did anyone else finish, or did I win?

So what I do is I want to go through up here and count all of the other people that finished. So before I set my self to having finished, it's sort of like you're about to cross the finish line. You kind of look around. You're like, is it me? Did someone else already finish?

So before I finish, I look around, and I say through the entire list of finishers, if anyone else is finished, it finished so I is true, then I increment the count. So I'm basically just counting how many other people finished before I finished. If no one else finished before I finished, I make myself red because I'm the winner.

No one else should be red because if I cross the line first, my finishers should be true. So when someone else does that count, they should get a count of at least one so no one else should cross the line at the same time. That's the basic idea for this racing square.

The way we run the racing square is we're going to have a bunch of racing squares. So we're going to create an array of racers, just of [inaudible] racer size. That's how many racers we're going to have. We want to have some array of finished, so that's just the same or number of racers, and we're going to have each racer – is going to be in a different thread. So we need to have an array of threads, one to keep track of each one of our racers.

We create a finish line in the race, the numbers aren't important. We'll just see it draw on the screen, and we have a little start button to create all the action. So what's going go on when someone clicks the start button? What I'm going to do when someone clicks the start button is go through all of my racers. If I had an existing racer on the screen, I'm going to remove it. I don't need to worry about that until after the first race.

But if there was already an existing racer, I remove it. Otherwise, I don't worry about it. I create a new racing square at index I because I'm creating all racing squares from index zero up the number of racers. I'm passing them all the same shared array finished. They're all going to be referring to the same shared array, just like the picture over there. Then I'm going to add the racer to the canvas at a particular location. The math basically just has them all sequentially down the screen.

Then what I say is, hey, I've created you, I've put you on the screen. You're revving to go, so I'm going to kick you off. So racer sub I is going to be in a new thread, sub I, and then I'm going to tell thread sub I to sort of go start itself.

So if I run this, I'll show you what it looks like. So if you think about each one of these different threads, you might say, doesn't the first racer have a slight advantage because their thread gets kicked off first? Yeah, but in turns out all the stuff executes so quickly that the random pauses in there actually have a bigger affect.

So here's the thread's example. There's my finish line. I'm ready to start. Yeah, every once in a while, a little bug in JAVA comes up, and it doesn't start. So let me recompile. Not a bug in the program. This was kind of an interesting thing that I just discovered on the side this morning. There's actually a difference in how Windows and the Macintosh happen to deal with this. Sometimes a little error comes up.

Here we go. So here are the two racers. They're just running in different speeds because they're in separate threads. They're running, they're running, they're running. You can see it's just the random pauses between each one of their moves. One crossed the finish line. He's like, oh, I'm the winner. I'm red.

You're like, okay, that's kind of cool, sort of, but how could I potentially extend the program? So instead of two racers, let me make there be ten racers. Instead of ten racers – actually, let me have there be ten racers, and what I'm also going to do is have them take a little victory dance in the end. So when they get to the end of the race, they're going to look around and see who else is around them. Before they cross the line, they're going to be like, I'm done. I'm done. That takes about 50 milliseconds to do that dance. You can just pretend that the square is dancing.

So I go ahead and run this program. Just to show you ten racers because ten racers is kind of cool. You notice I only had to change really one parameter. I just changed the constant from two to ten, and I didn't need to update anything else in the program. Now I got ten racers, and they're all running different. You're like, oh, you can start rooting for you racers. We could put G images in there. You're like, come on, little guy! Oh!

That's way more exciting for me, evidently. So you're like, that's kind of interesting. You just increased the number of racers, but what else can you do? Now, think about this. What if everyone was created equal? So rather than all of them having a random delay between 50 and 150, what if I just said, your pause is always 50. You're going to generate a random number between 50 and 50, so it's always going to be 50.

So that should mean all racers are created equal, right? So if all racers are created equal, all of them run at exactly the same speed. So if you think about everyone running at exactly the same speed, the weirdnesses begin to come up.

So here, we have a race, and we start. Everyone's just cruising. You're like, oh, I can't tell any difference. They're all like, we're all the winner. You look at that, and you get a little

disturbed. You're like, whoa. My code looked when I crossed the finish line who else was crossing. I was supposed to – if no one else had crossed before me, I was red. If someone else crossed before me, I wasn't supposed to turn red. So how come we're all turning red?

Here's where the real trick with multi-threaded programming comes in. What happens is everyone finishes the race at the same time. They all do their count at the same time with the shared rate. Some of them may do their count slightly earlier, but the ones who finish their count slightly earlier, they do a victory dance.

So everyone's at the finish line dancing, and everyone, when they looked around, at the time they looked around, no one else had crossed the finish line. Then they all say, I finished at the same time, and no one else finished before me, so they all turn red. So you might say, okay, Maron. That's kind of funky. What happens if the differences are just small?

So I'll give you one last example, just to show you the full funkiness of threads. So rather than the difference in each time step being between 50 and 150, it's just between 50 and 65. That means a whole bunch of people are going to finish the race at close to the same time. So if they finish the race at close to the same time, here's what happens.

We just tickled that bug again. Let's do it one more time. That's the other problem that you'll see with multi-threaded programming is there's a lot more things that can go wrong, which is why you're not required to know it for this class. Let me just show you the last example. Oh. It's just toying with me now. Green button. Ah. Most of the time, the green button doesn't work, but now it does.

So now, we start the race, and there's very tiny differences between these racers. What happens when there's tiny differences? Let's do it again. Yeah. Some people finish first, and some people don't. This is what makes multi-threaded programming hard. It's how to keep track of stuff like this, but now you know. So I'll see you on Friday.

[End of Audio]

Duration: 50 minutes