Programming Methodology-Lecture26

Instructor (Mehran Sahami): All right. Welcome back to – what kind of day is it going to be in 106a? Anyone want to – fun-filled and exciting. It always is. Thanks for playing along. So a couple quick announcements before we start. One announcement is that there is one handout today, so several people have asked in the past, these programs that I make in these classes are kind of cool. I would like to be able to share them with my friends and relatives and whoever else. We're going to talk a little bit about how you do that today and what that means underneath the hood, but the handout actually explains it all. It's this notion of a jar file. We'll talk more about a jar file as we go along.

The graphics context, for those of you who are doing it, is due today. Just wondering, quick show of hands, how many people entered the graphics contest. Wow. Not as many as I would've thought. There could be a couple people who are at home, even if you don't win of getting 100 on the final in a random drawing. So that's a good sign.

One thing I do want to check, I just heard a little bit before class that some folks were having some trouble submitting their graphics contest because there actually might have been an issue with the server that takes submissions. So if you submitted to the graphics contest, whether you're in here or you happen to be watching the video, email me, and let me know what the name of your contest entry was. That way, I know for sure that we actually got all the contest entries that we think we had, and if we didn't get one, I can email you back. The thing I would ask you is, if you can't email me any time this weekend because this weekend is actually, when we're going to make the first pass looking over all the contest entries, and then we're going to have a small pool that we'll take to the section leaders. They will vote and give the winner. I'll announce the winner in class next week.

I might show a demo of the winning two as well. We might do the random drawing in class as well to see who actually gets the third coveted random drawing spot, even if you don't win. So please email me if you entered the graphics contest, just to make sure.

One other thing with email for SCPD students, I know it's still a little too early to think about final exams, but if you're an SCPD student, it's not too early. If you're not an SCPD student, it's not too early, either. But for SCPD students, if you're taking the final exam, if you plan of taking it at your sites and you're not going to come on campus to take it, email me by 5:00 p.m. December 5, letting me know that you're taking it at your site and the name and email of your site coordinator, just like the midterm. That way, I can get the information to your site coordinator for the final well before the final.

If you're planning on coming on campus to take the final, you can feel free to send me an email to say you're coming on campus. If I don't hear from you, I will assume you're coming on campus. So you only need to email me if you're taking it at your site, so please do that if you're an SCPD student and you plan on taking it at your site.

Any questions about anything we've done so far before we dive into our next great topic? All right.

One of the things that we've done the whole time in this class is we use these things called the ACM libraries. The ACM libraries are a set of libraries that are actually created by a task force of people. The ACM is the Association of Computing Machinery. We talked about them at the very beginning of the class when we talked about these libraries. They put together some nice libraries of stuff that are really useful for teaching, which is why we use them.

Today, what I'm going to do is lift a little bit underneath the hood and talk about standard Java, which is what you would get if you didn't use the ACM libraries and you just used the standard Java libraries. Now, there's no reason why you can't continue to use the ACM libraries after this class. They're just another set of libraries that were written by a group of people that you're certainly welcome to use.

So there's no reason why you should stop using them, but there were a couple important issues related to standard Java. Now it's time for you to know. So the first thing that's related to thinking about standard Java is when you're running your programs, when you go into Eclipse and you click on the little running guy to compile your programs. It give you a list of what classes you actually might want to run.

If you only have one project, you may only get one choice, but one of the things you kind of think about is in the name surfer program, I actually have four or five different classes. How come it always knew which class to run? How come it always knew the name surfer class was the class that I actually should run? Anyone want to venture a guess?

Student: [Inaudible].

Instructor (Mehran Sahami): It's the only class with [inaudible] which is very related to an underlying issue. It's the only class that actually was extending programs. So one of the extended programs, what actually was happening in these ACM libraries is you were getting a method called main. Main is actually the place – you're old enough to see main. Main is actually the method at which Java classes actually start running.

So one of the things you should think now, you never wrote a method called main. I never saw a method called main, and you're telling me that's where Java programs actually start running. Yeah, in fact it is. It's because programs provided this main method for you. What this main method did in the program was essentially get the rest of you program running by getting a few things set up and then kicking off your run method. So you didn't actually need to worry about this.

But now you're sort of old enough to actually see what that main method is all about. So if we think about what this main method does, the header for the main method is also kind of weird. This is part of the reason why we never showed you the main method before. The header for the main method is actually public static void main, but we're not

done yet. Main actually has some arguments. It has an array of strings called args and arguments, and then something in here happens inside of me.

If we showed this to you on the first day, we would've had to go through and explain what all these words meant before we explained what main even was, before we explained how you write your first program. That would've been a pain. Now we can just tell you. Public mains is a public method. You know that. You probably recall the other public methods you've written. Static means that this is actually a method that belongs to the class – it's not something that you would actually call on a particular object.

So you never have some object – like, here's my object X, and I call X .main. Main is just something that gets called. It's a class method as opposed to being a method that gets called on an instance. Void means it just returns nothing.

What is getting passed in here is an array of strings. Where is that array of strings coming from? This actually harks back to when computers weren't all nice and graphical and everything. When people wrote programs, the wrote program and were typing on what's called a command line. They wrong the name of the program out. They actually typed it, and then they typed a bunch of things that they wanted to be passed into the program such as initial information to start that program. That was the initial thing, so if you had some program like name surfer, you might actually start off by giving the name of the program.

Then after name surfer, you might give it the name of the data file, like data dot text. You might've given it some other things as well that were separated by spaces. This list of stuff is essentially what gets passed in here as arguments. They're strings, and this is how the program would actually know what came in on the command line when the program was kicked off. Java's not that old of a language. It sort of came around and gaining popularity in 1995. People weren't doing a lot of this in 1995. I already had my mouse and my folders and all this other stuff, even if you were six years old. You probably did.

You're like, I never typed this stuff, so why do I care about it? The reason why Java's derived from another language called C, and there's a variation called C++ that was created when people were writing programs in the days of yore. The whole notion of main and having some arguments to get passed to main kind of came along with the baggage of actually having a program language that matches the same style programming languages when they did do this.

So a lot of the times in real Java programs these days, there aren't really any arguments. If there are arguments, there's some system parameters or something like that. We don't usually worry about them. So when you go and look at some other Java program that isn't using the ACM libraries and you see this main thing, and you're wondering what it's all about, you can think of main analogously to run. It's just where the whole time you've been thinking of run as where you're execution starts, main is really where execution started.

If you think about execution actually started in main, so how did this thing actually kick off my run method? Now you're sort of old enough to see that, too. So what it actually did – let's say this was the main method is something like name surfer. So somewhere inside of a program, inside of the ACM libraries for program, we had this main method that figured out what the name of your class was. Essentially, it had a one-liner in it that would've been equivalent to this.

New name surfer dot start cards. So it's a one liner. Now you know what this means. What was it actually doing? When main started, no objects exist in the world. It's a static method. So there's no object that you're giving the main message to. Main just wakes up and says, hey, I'm main. What am I going to do? Why don't I create some object of this particular type, name surfer, which happens to scan an object, which is your program. Remember your program, as we kind of talked about, implements the run method, and actually a program underneath the hood implements the runable interface that we talked about last time, with threads. We talked about the runable interface.

All programs implement the runable interface. How do you kick off something that's runable? You say start. So what it basically did was created and object of your class, which was name surfer, and told it to start. It happened to start pass along these arguments, but you never needed to see those arguments. As a matter of fact, you never did see them because when your object was in stand shaded, it didn't expect any arguments. So the arguments actually got passed to this thing called start, which just ignored them, basically, and then started your run methods to kick everything off.

Last time when we talked about start, we talked about this in the context of threads. So we said, oh, you create a new thread, and the object that's you're going to start running, we put inside a thread. We kick the thread off with start. In this case, we're not actually creating a new thread. We're just saying we want to start executing this object I just created. It implements runable, so you'll start running from the method run, but I'm not creating it in thread. So this thing is going to execute in the same thread of execution as the entire class.

So I don't suddenly kick off something that's running in parallel with this guy. It's actually going to sequentially start name surfer, and that's the last thing this guy does. Run your whole program. Any questions about that? Kind of a funky concept, but that's basically what's going on. You should see it. We were creating an instance of your program and then just kicking it off. That's why, this whole time, we had this thing called the run method that had to be public because it was implementing the runable interface. But now you've seen main.

We could've just had main in your program to begin with and included all this code. The only reason we didn't put it in there before is because we didn't want to explain any of this stuff. In week two of the class, right after [inaudible] all holding hands and singing Kum Ba Ya. We're like, oh, it's [inaudible] the robot. Let's give Java, public static void main. Straightening args. You're like, what is going on?

We hadn't done arrays. We hadn't done classes worrying about static. We certainly hadn't done methods. We hadn't even done parameter passing. So we just waited until the end. Now you see it.

Now that we have that idea, now we can think about, okay, if this is kind of what the standard Java world is, let me think about taking this idea and using it to help me take my existing programs that I've written and pack them up into a form so I can share them with family and friends. So that's what we're going to do next. The basic concept of doing this is something that's called a jar file. You've actually seen jar files before because you've been working with something this whole time in your projects called ACM dot jar. This was just a jar file that contained all the ACM libraries.

Basically, all a jar file is, where does it get its name. It's not a big mason jar, although you can think of it like that. It stands for Java archive. That's where the name comes from. The basic idea behind the jar file is this can contain a whole bunch of different stuff. Most of the time, what it contains, is a bunch of classes in Java. You can think of this as the complied version of the classes.

You can actually put source code inside the jar if you want, but most of the time when you get a jar, it doesn't have the source code associated with it. I just has the actual dot class files, which are the compiled version of the files. So you could put source files in here if you wanted. You could put data files in here if you wanted. You could put a bunch of things in here if you want, but we're really going to focus on the case of putting classes in here.

So one of these already existed for you. It was ACM dot jar, and we're going to figure out how to actually create some of these ourselves and use them because you can actually think of them as something that's executable.

So if we come to the computer, here's name surfer. This is an actual working version of name surfer, so I'm not going to show you the rest of the files in case you're taking [inaudible]. All the code's in here, so the first thing I did in the name surfer program is I thought about, hey, I want to think of this in the standard Java world now, even though you're still using the ACM libraries. When I want to build this jar, I want to build it in a way that sort of makes it maximally portable. I can give it to someone who's over here on this PC and someone who's over here on this Mac. They don't need to have a Eclipse or anything like that. They can just run it. That's the whole point.

So the first thing I'm going to do is to introduce my friend, the main method. So basically I put in the code you just saw. I add a method, public static void main has this array of strings called args that sets the parameters. That's just the way main is always defined to be. What it's going to do is create a new name surfer object and kick it off. That's the only thing I need to add to my program. So anywhere you had some class that extended program, you would add these three lines of code to get it compliant with standard Java.

Once we do that, we need to create the jar file, which is the thing we're actually going to be able to execute. One thing you might be saying to yourself is that the ACM libraries already provide this. So why am I putting it explicitly in there? As a matter of fact, it's not super required that it be in there. The real reason why we put it in there is to try and maximize portability. Even though Java's supposed to have this property where we write the Java code once and it can run on PCs and it can run on Macs and it can run on Vista and Tiger and all this other stuff, in reality, there's some little differences between these operating systems. So little problems creep up here than there, every once in a while. You actually saw a few of those in class on occasion. Some of you experienced them in the Lair.

By putting in this line explicitly rather than kind of relying on the code that's in the ACM libraries, this is actually doing a bunch more complicated stuff. It needs to infer what the name of your class was. You never told it explicitly, hey, ACM library, I need a new one of these guys. Java has a facility called reflection where it can go and say, oh, let me take a moment to reflect. I'm going to go and look at the names of your classes and then do something to actually generate some code based on the names of your classes, which is what it was doing.

But that kind of stuff can get a little bit frosty, so we're just going to say, hey, ACM, don't worry about it. I'm just giving it to you. I'm a nice guy. So just put it in to make it explicit. It maximizes portability is the main reason we put it in.

So how do we create the jar file? What we're going to do – these steps are all in excruciating detail in your handout, so you don't need to worry about scribbling down notes quickly. What we're going to do is you first click – that's the most difficult part of the whole thing. You select the project that you want to create the jar file from. So name surfer. Then we go to the file menu, and we pick export. So this whole time you were doing imports when you were bringing stuff in, now it's finally time to give something back in the form of an export.

So what you want to do when you click export is it brings up a little dialogue box. Sometimes this is closed. Sometimes it's open. It's not a big deal if it's closed. Open it. Inside Java, you click on jar file. That's what you want to create. It has this nice little jar icon there to remind you it's a jar file. You click next. I'm just going to take you through all the steps.

We need to specify a couple things here. First we need to specify what's in the jar file. So I'm going to open up name surfer, and what I'm going to put inside the jar file – I don't want to have all of name surfer in the jar file. In terms of if you throw everything in there including this stuff called dot project and dot class pass, it actually gets a little bit confused. Some of those things are not germane to what we want to pack up in our jar. They're just kind of other administrative information.

What we really want to have is everything inside the default package. So I just click on default package. I can double click on this to make sure that all of my Java files were

checked. That's what I want in my jar. I want basically all of my Java files or the compiled version of all of my Java files. I make sure that this is clicked. It's clicked by default. Just make export generated class files and resources. So what it's going to do is compile those Java files into their corresponding class files. That's what it's going to put in the jar.

There are other options, like I can export the Java source files if I wanted to. Then I give someone the actual source code. Most of the time, you don't want to do this if you don't want peoples sniffing through your source code. You're like, here, take the compiled – the ACM libraries, we're not going to give you the source code. Just take the ACM libraries. They're good for you.

So once we have this, we need to specify where we want to save this jar file. So that's what select export destination. They should've just said, where do you want to save it. That's the export destination. We can browse around. Basically, I've already created a folder that has all my name surfer code in it over here. I'm going to save the jar file in that same folder. So this is just in the same folder for my project called name surfer that has all of my class files in it. You can put it wherever you want, just don't forget. That's kind of the key.

I'm going to save it here. You don't need to worry about the options down here. They're just fine in the defaults that they're at. You click next. Then you come to the screen that seems like, what's going on. Export files with compile warnings. Yeah, we just want to export everything. So we just click next.

This is the most interesting part of the whole thing. What we want to do when we actually create this jar file, and this is something we only need to do when we have other jar files like the ACM libraries, we need to generate a manifest file. Basically, all a manifest file is, it's a complicated name. It sort of sounds formal. It sounds like they're on a boat, and they're like, oh, where's the passenger manifest? Really, all a manifest is – I'll show you the manifest file. It's two lines long, and then you add one line to it. It's basically just a little bit of administrative information that's kept around with your jar file so it knows what kind of stuff you're using with this jar file. That's all it is.

So what we want to do is generate the manifest file. Make sure you save the manifest in the workspace is checked, which it should be. Then you need to specify where you want to save the manifest file. The place I want to save it is basically in the same folder for my name surfer project in the name manifest. You could browse around if you wanted to, but usually the name you give it is the name of the folder that all your project stuff's in, and then the name manifest, which is going to be the actual name of the file. Any questions about that? Manifest file?

The other thing you need to do when you're specifying a jar is a jar's just a bunch of classes. It needs to know in some sense if someone ends up running this jar – you'll see that in just a second. You can run a jar – where should it start? Which class should I call its main method. That's what you specify here. Select the class of the application entry

point. It's kind of a very formal way of saying, where should I start running? So the place I want to start running, it lists to you all the classes that have a main method. Those are all the ones that can start running. Name surfer is the only one.

So I click okay. So it says the main class is name surfer. Now there's no more next buttons. Now all I can do is click finish, and I just created a jar file. You're like, oh, [inaudible] double click. Double click. We're not there yet.

There's two things that you'll notice if you look over here in the package explorer. You'll notice now we have something called a manifest file because that's where I saved it. I also have name surfer dot jar, the jar file I just created. They were both put in the same folder with my other files. That's were I wanted to save them so they happened to show up here in the package explorer. Here's the funky thing. Even though we created this manifest, and we created this jar, they don't have quite the right information that we want. So what we do is we double click on the manifest file to open it up. Here's the whole manifest file.

It's got a version, which is 1.0, and it's got the main class, which is name surfer. Why does it know the main class is name surfer? Because I told it. That's the application entry point. Just for letting me know that I told it, that's where the application starts.

There's one other thing I need. What I need to do is say, you know what, that's a good time, but you also need to use the ACM libraries. It says, you didn't tell me about the ACM libraries. You say, well, now I'm going to add the coveted third line that was talked about in the manifest file. So I'm actually going to modify the manifest file and add something called a class path.

All a class path is, it actually looks just like this, C-P. Class path just tells basically the application what other stuff were you using. Are there other jar files that you're using? So what you specify here is the name of any jar files that you're going to be using as part of your program, separated by spaces.

So I'm going to use ACM dot jar. Here's ACM dot jar over here. I'm going to use the ACM libraries, and those are all in some jar file. I'm also going to use the jar file I just created, name surfer dot jar. If I had three or more jar files, I'd list them all on the same line. So if you go and write some application some day where you're like, hey, I got this jar file for my friend. Here's this jar file I downloaded from the web, which I wouldn't encourage you to do, and there's this other job file from somewhere else. You just list them all here with space in between.

Then we save the file. That's probably the most difficult thing that people forget. So you save the file. You save the manifest file. What do you do after you do all this? Create a jar all over again. Why? Just trust me.

Here's how it's going to work. I'm going to do it a little bit different this time. Rather than going to the file menu and picking export, I'm going to take the advanced course. I'm

going to right click on the project name and click export. It brings up exactly the same window. I'm going to export a jar file. Come over here. What do I want to export? Oh, yeah. It's all this [inaudible] again. What I want to export is not all this stuff. I just want to export everything in my default package. Where do I want to save it? I want to save it in the same place I had before. You might say, but aren't you going to replace the one that already exists? Yeah, I need to replace the one that already exists because I updated my manifest.

I need to say it's not just about you anymore. Now it also involved ACM jar. You can replace the old one. So I'm going to put it in the places before. Then I click next. Here's the thing about export warnings, export affairs. I don't care. I'm just going to export. Here is the only place where things are different. The second time I go through this whole thing, I don't want to generate another manifest file. I generated a manifest file the first time, and then I modified it. What I want to do is use that modified manifest file. So I say use existing manifest from workspace, which is again, just a formal way of saying use the manifest file that's already there.

It asks me, where is the manifest file? Strangely enough, if you look, this is exactly the same text as here. No coincidence. You just saved it. That's the one you want to use. It's going to be the same name. Now notice which class of the entry point is grayed out. It's grayed out because I told it before, your entry point is name surfer. That's in the manifest file. If that's already in a manifest file, I'm using it, it doesn't need to know it again. So it doesn't even let me specify it again. Just say that's the difference. Then I click finish.

They give me one last one. Are you sure you want to override that last jar file? He was your friend. I say yes. We didn't know each other that well. Now I've created this jar file. What do I do now? Well, I kind of come over here, and I say, here's the jar file I just made. Now one thing I can do with the jar file is I can double click it. If I double click on the jar file, it just starts kicking off the name surfer application. So I don't need Eclipse anymore to go and run and have a little running dude. I have a little application.

I'm like, oh, yeah. How popular was that? Bob's kind of fallen off in the meantime. There are some other interesting ones. I was looking at these the other night. Verna. After the '60s, done deal. Thanks for playing. So name your children Verna. Make a comeback.

So name surfer's just kind of running here. It's fun. It's just a little stand-alone application. If you wanted to pack this up and send it to a friend of yours, you wouldn't just send name surfer dot jar. What you would do is say, hey, you know what I want to do? I want to create – I'll just do it in here – some new folder. So I'm going to create some new folder, and I'll call this my program or whatever you want to call it. I'm just not going to call it name surfer again because I already have a folder called name surfer. My program. What I'm going to put in my program, that folder, is the jar of my program. I'm also going to put in the jar of the ACM library because I also need that jar. That's separate.

I also need in here any data files like names dot data. That's all the data that gets read when the program starts. It's not like magically that's just going to be there now. It's still

going to go and try to find that file. So I still need to put all that stuff here. Now this folder is something I can zip up and send it to a friend of mind. When my friend gets it, they would be like, oh, name surfer dot jar. Then they get going.

So now you can package up any program that you've written in this class. Remember to put in main. You got to go through this two-part [inaudible] the jar, export it, process, modify the manifest, create another jar, put it out there again. But you're good to go. Any questions about that? Now you can package up and share with friends.

What's even cooler than sharing with friends that you can email stuff to is sharing with friends on the web. You just have millions of friends on the web. Most of them you just don't know about, but they're probably on in the late hours of the night, looking at your web pages and things. You can actually take any files that you create here, like jar files, and make them available in a web browser.

There's one other thing I should mention, and this is kind of a little thing. Before you go and create all this stuff and send it off to your mom and dad and be like, mom, dad, breakout. Go play. It's a good time. In order for them to run your jar files, they need to have the Java run time environment installed. Remember on the second week of class where we said go to the CS106 web site. You need to download a clip. There's this thing called the JRE you also need to download.

If you have a make, the JRE in most cases is already installed. If you have Windows, it's not installed. They need to go down, and you can send them to the CS106 page and say download the JRE. Here's a copy of handout No. 5. Go ahead and install it, and then they can run your programs. Your program can't run if the computer doesn't have Java, right? It's a bunch of stuff that's going to execute Java byte code, and your computer's like, what's Java byte code? I don't know what to do with it, so it won't run. They need the Java Runtime Environment.

Now assuming someone has the JRE, they can go to a web page. You might put some page on the web that allows you to load your applet. Your applet is just a webified version of your program. So here look. It's running inside a web browser. This isn't the actual application. This is my web browser. I could go to some search engine from here. I'm sitting in the web browser. I'm not just running a regular application on my desktop. This guy's actually running in my web browser.

Here's how I make it happen. I create a web page. If you don't know about HTML and creating web pages, unfortunately I can't explain that to you in five minutes. What I can show you is basically what this file's going to look like. So if you know a little bit of HTML or you just want to essentially copy and paste this idea, this will work for you. So all you do is say – the entire page that allows your applet to run. You say this page is HTML. There's these little things called tags.

The name of that page is name surfer. I want to create a little table. I'm going to create a little border around my application. What's my application? My whole application is right

here. I have an applet. What's the name of the archive, which is a Java archive that contains my applet? It's name surfer dot jar. Code is what's the entry point. This guy no long has a manifest file available to us. What's my entry point? Name surfer dot class. That's where you start running. So it says I'm going to go name surfer dot class, find its name. That's where I start running.

The space I'm going to give you to run on the screen is 700 by 500. That little snippet of code is what goes on whatever web server this is serving on. It looks for those jar files, slaps them into the page and then they're good to go to run more java program inside their web browser. How many people know HTML? You folks. There's some number of folks that this might be a reasonable thing to do. If not, you just don't need to worry about it. It's not a big deal.

The one other thing you should know if you create a web page is that when something's running in the web page, it doesn't have access to the rest of the file system. What that means is if I actually happen to be over here – notice I have index dot HTML. I have name surfer dot jar, and I have ACM dot jar. What happens in my names dash data file? It doesn't exist. Why doesn't it exist? Because I couldn't read it anyway. Once I'm running on the web browser, for security reasons, it doesn't let you go in and pull stuff out of your file system.

If it did, people could do really bad things to your computer. So what you need to do if you want to find that data, you let us run name surfer on the web. What did you do? Here's the dirty little secret. I actually created a giant array that had all the data in it and made it part of the program. So sometimes you can do stuff like that if you don't actually want to read from a file. The other thing you can do is you can take those files and include them in the jar file because the jar file cannot only have compiled [inaudible]. It can also have data files. That's another way of doing it if you want to do it that way.

We don't have time to go into all the details. It's kind of the same process. When you're exporting stuff to the jar file, you include some data files in there. So what's creating the executable. Now that we know all this funky stuff about executables put on a web page if I want. I'm feeling happy. I'm good to go. It's time to come back to our friend, standard Java.

Standard Java's what allowed us to think about doing this stuff because we learned about main. I want to show you a couple examples of programs that actually don't use the ACM libraries at all to show you why we use the ACM libraries. One of the things you might be wondering is why weren't we just doing the standard Java thing the whole time? Part of the reason is things are just so much easier and cooler when you have the ACM libraries.

So if you go back over to Eclipse, we're kind of done with name surfer and all this manifest stuff. Here's a program that's written in standard Java that writes out hello world on the screen. This is something you kind of did in the first class by saying public class

hello world extends console program in printlin. Hello world out to the screen, and you would've gotten it in the console.

So what's different here? What's different is we have public class hello world, and it doesn't extend anything. It doesn't extend program. It doesn't extend console program. There are no imports for the ACM libraries. We're not using any of the ACM stuff. So we don't have a console program. We don't have a nice console that we write stuff out to that displays in a nice little window. What we do have is something called the system output console. If we want to put stuff on that, it looks real similar to what you had before. We used printlin, which is why we made the method that you used called printlin to match this. But we need to say system dot out dot prinlin and then the text we want to print out.

Again, here I have a main method. My main method can have whatever I want in it. This is just where execution starts. So if I compile this and run it – let me show you why. This world is not all that cool. Here's hello world. It just ran. You're like, I don't see anything. You don't get a cool window that comes up and is like, hello world. You have the system console. The system console if you happen to be using a development environment like Eclipse or something else, it's basically just a window in that development environment that shows messages that you print out. If you happen to be in the bad old days that I think I erased over here where you have command line text where you actually type stuff in, the console would just be that same window where the text would appear where you type stuff.

If you don't get it in a separate window, it's not that cool. A lot of times, this window's close anyway. If you wanted to kick it up a notch, why use the console? Why not do something graphical. So here's graphical hello world. What we want to do is create a window that is going to have some title associated with it. We're going to put the text hello world in that window. What do we need to do? Again, execution starts at main. We need to create something called a J frame, which you never had to worry about before. What's a J frame? It's actually a frame that's going to hold the window.

We're going to start running our application. It's going to create a little window for us that we're going to display stuff in. What are we going to put in that window? We're going to put a label. J labels you've seen before. This is just like you've seen. We're going to create a J label that's called hello world, and we want this label to be center justified as opposed to left justified or right justified.

So I need to say J label dot center to center justify it. If I don't give it a justification, it'll by default by left justified and look kind of ugly because we want it center. Then I add this label to my frame. So similar to the idea of having a canvas and adding stuff to the canvas, it's exactly analogous. We want to make it just as easy. So when you saw standard Java, all the same concepts would apply. Here, we're just adding the J label, which as you know now, is a J component in the big Java hierarchy. J components can be added to J frames.

So we have a J label that gets added to a J frame. I set the size for that J frame, which is 500 by 300. That tells me how big the window's going to be when it starts. Then there's this other stuff that I need to do. You would think why do I need to do that? It doesn't make any sense that I would not otherwise want to have it this way. Here's what I need to do.

If someone clicks close on the window, I need to say, hey, if someone clicked that, then you need to close yourself. Otherwise, the window goes away and the application keeps running. It seems odd, but we need to have that there for this application to stop running. Then we say window, I know I created you and everything. You need to make yourself visible. Otherwise, no one will be able to see you.

You're like, why would I create a window that I wasn't going to make visible? Yeah, that's why we use the ACM libraries. So we need to make sure that this guy's visibility's true. If we run this, here's graphical hello. So after all this stuff, here's graphical hello.

Yeah, would you want all that explained to you so you can get hello world? You're like, I'm writing a social network. I don't need to worry about hello world in the middle of my screen. That's because you have the ACM libraries. So you're like, okay, let's kick it up even one more notch. You're like, [inaudible] all this mouse stuff and interaction, right? That should be something I get some benefit from having standard Java. So we'll have an interactive version of hello.

Center the interactive version of hello. Now some of this stuff should begin to get a little more repetitive in the sense that you have your main. You have your J frame. The frame is called interactive hello. That's the title of that window. What we're going to add to this J frame is a new class that we're going to create called a moving label. I'll show you what a moving label down in just a second, but we need to set the size of the window, set default close operation exit on close. Set visibility to true.

Basically all this does is create this window of a particular size. It's going to add this thing called a moving label to it. Moving label's just another class I create. What's a moving label? A moving label is a J component. It needs to be a J component because I'm going to add it to a frame. To display something on a frame, I can only display components on the frame. This is going to extend component.

It's going to implement our friend, mouse listener. It's going to listen for the mouse. You're like, oh, I remember mouse listener. That where the mouse got clicked and dragged. That stuff's exactly the same. So I have my constructor. What my constructor has, it has some starting text for this label at its starting X and Y location. That should look kind of familiar to you. Basically I just store off the text. I store it at X and Y. This guy wants to listen for mouse events.

So it says add a mouse listener, and it needs to say if you get some mouse event, send them to this. So this is a little bit different than you've written in your programs before

where you just said add mouse listener. You just had an open [inaudible]. That's because we kind of took care of the rest of this stuff for you.

Here's where things get a little bit funky. The difference between this and thinking about having some sort of label that you just display on a canvas, some text that just sits there, is that this guy now has to worry about what's known as painting himself. That means it needs to draw itself on the screen.

Well, when I had labels before, they just knew how to draw themselves. Yeah, that's because we gave you a label that was a little bit smarter and knew that it was a label that should draw itself. This guy needs to be told to draw himself. There's a method called paint component that gets called whenever this guys gets displayed on the screen. There's some other thread that's going to call it for you automatically. Here's the graphics context in which your going to call yourself.

Within that graphics context, I'm going to draw some string. The actual method name and everything is not important here. It just shows you that there is extra stuff you need to worry about, which is why we didn't want to do all this stuff to begin with. This stuff you've seen before. What happens if a mouse is clicked? I get the new XY location. I repaint. What does repaint mean? It means redraw yourself. I'm going to redraw myself at this new XY location. When I call repaint, someone comes along and says, hey, to repaint this area, I'm going to call your paint component method so you can repaint yourself.

Whoa, this is really weird. I'm over here. I get a mouse click, and I know that I want to redraw myself. But rather than telling myself directly to redraw myself, I go and tell the system, I need to get repainted. Then the system says, that's the start of Jabba the Hutt kind of sitting there fat and happy. Some day, if you actually – we won't get into it.

You go and say I need to repaint myself. It says, okay, you need to repaint yourself. Well, when I'm ready for you to repaint yourself, I'll call your paint component method. Until then, you don't repaint yourself. So it delays other stuff in the system to worry about, and then oh, yeah. There was you. You asked to get repainted. I'll call your paint component. You say, okay, well, now I'm going to draw myself at the new XY location.

So it kind of convoluted the whole notion of you're doing something here, and you're asking someone else to do something for you. Then they're going to call you back to do what you originally intended to do. Now it makes a little bit more sense after we talked about threads and after you've seen all this stuff. Third day of class, not so hot.

So if we run this, this is called interactive below. Basically, what it does is it just brings up our particular message, CS106A rocks in the middle of the screen. Now every time I click the mouse button because this is the mouse click event over here. Every time I click the mouse button, the XY location of the mouse becomes the new base point for the text that gets drawn. So it just moves around the screen.

Any questions about that? So this is standard Java. You've seen all the concepts using the ACM libraries. The notion of adding things, having mouse listeners. As a matter of fact, the whole mouse listeners concept, we just took the standard Java ideas and used them sort of in conjunction with the ACM libraries. But there's a lot of things in the ACM libraries that just made it so much easier to, for example, to graphics contest entries or to write a social network.

So you're welcome to continue to use the ACM libraries after this class. Some people were wondering why we use these libraries, and this is the reason why. There's a whole lot of stuff you'd have to worry about otherwise. Questions?

Student: [Inaudible].

Instructor (Mehran Sahami): You'd be using your favorite word processor, like notepad. There's actually some places which I won't name, but it's actually reasonable that some schools in your first programming class, what you do is they say, you need to have notepad, or you need to have some text editor. Then we're going to do command line stuff. You're going to type in name surfer, names dash data dot text to run your program. Then everything is going to be [inaudible].

So one thing I want to leave you with now, in our final few moments together, is sort of a notion – we're still going to meet next week, talk about life after this class, but if you want to go on in terms of learning more about Java, especially standard Java, we sort of just started things off by giving you this book, which talks all about the ACM libraries. This, I think, is a great book to actually learn everything with and use the ACM libraries. But if you want to go on, what are some other resources you can use?

Student: [Inaudible].

Instructor (Mehran Sahami): It's not pretty. If you go to the Java section of any bookstore, just be prepared to stay a while. One book that I would recommend, not that I get any kickbacks from these folks, called Learning Java. It's actually a pretty good time. Some of the examples you actually saw here were based on this book. It's a little big. That's why we don't use it as a textbook in this class. Will we break the 1,000-page mark. It's so close. No, 950. Sorry, close.

There are some other books. Actually, the original specification of the Java programming language. This is an older version of the book. This was when I was a wee tyke in the days of yore and got the old version. I forget what version they're up to now. This was second edition. I think now they're up to three or four. Something like that. The books are a little bit bigger, but for a book that specifies the language, Java, it's actually very well-written as a reference. So I'd recommend this as well.

If you're into oh, I want it all, Big Java. It's big. I think it might actually cross the 1,000 page – oh, yeah. It's like 1,200. If you really want to get hardcore and you're all about web-based Java, Java Server Programming. Everything you would want and a lot of

things you don't want to know. Everything you want to know and more. That's just a small set.

So these are just a few books I'd recommend if you want to go on beyond this class. But you can go into any bookstore, and you get inundated with that kind of stuff. Now you have a context for putting all the pieces together because you've seen all the things that you actually need to know to be able to work with the huge set of tools that Java actually has. Any questions about any of this stuff?

You're good to go? All right. I'll let you go a couple minutes early because most of the time, I let you go a couple minutes late. Have a good weekend.

[End of Audio]

Duration: 45 minutes