Programming Methodology-Lecture04

Instructor (Mehran Sahami):So welcome back. Now we're officially started with the class. I hope you had a good weekend. I was just asking people before class what kind of stuff they did this weekend. So if anyone wants to, come early. We'll just engage in random conversation.

So a couple of quick announcements: There's one handout, which is your section handout for this week. Make sure you get that. What I'll do every week that we have a section, is that week, you'll get a handout on Monday, which is a problem that you're going to go over in section.

This gives you a chance to think about it before section. It's not required that you do it. It's not extra homework; it's just something and if you want to read through it to get familiar with what you want to do with section or possibly do it yourself if you want more practice to be prepared for section, you can do that and then you'll go to section; you'll cover the problem and actually in section, you'll get solutions to the problem, as well, so you'll get examples of code for all the section problems.

Section assignments – hopefully you should have signed up. The deadline for signing up was Sunday at 5:00 p.m. Section assignments will be mailed out to you tomorrow night so you'll know which section you've been assigned to and you can go to sections this week, get your section leader's email address, complete Assignment 1, at least the email part by sending email, and I don't need to mention the sections actually start this week, so you should go this week once you get your section assignment.

If for whatever reason you didn't manage to sign up for a section, you should remedy the problem by emailing cs198@cs and let them know you forgot sign-up for a section for CS106a and they'll let you know what times are available and you can sign up.

Don't email me because I have nothing to do with the sections, other than actually assigning the problems for the section. Don't email Ben because he's not involved in administration for the sections either. He also worries about the content of the sections. But the 198 folks handle all the administration. Uh-huh?

Student:[Inaudible]

Instructor (Mehran Sahami):Yeah, you should submit it again after you've been assigned to a section just to clarify. All right, but if you've already submitted it, we can sort figure it out eventually, but it's in your best interest to re-submit it after you've been assigned to a section. So any questions about anything else before we start? Uh-huh.

Student:[Inaudible].

Instructor (Mehran Sahami):Yeah, the most submission is the one that's counted if you submit multiple times. It just lets you kind of submit more if you'd like, if you submit

something and you're like, "Oh, I made a mistake." Then you can go back and submit again. We don't feel like, "Oh, every time I like save, I need to submit." Don't worry; just submit when you feel like you're done and if you need to go back and make changes, that's fine. Uh-huh?

Student:[Inaudible].

Instructor (Mehran Sahami):If it's really necessary and the main thing is if there's space in other sections. Yeah, you would email cs198@cs and let them know. All righty.

So let's go ahead and get started with the actual content. And so today is the day where we make a little bit of a transition. It's like one of our transition points in life. It's kind of like coming to college. And the transition point we make is you now come into the Java world, and you're like, "But I'm still doing Karel." Yeah, in lectures, we're going to start to wave bye bye to Karel and Karel's like, "Yeah, I'll be there in your dreams when you're actually programming, right."

So Karel's not really gone until you're done with the Assignment No. 1. But now, we take – we sort of graduate into the world of Java and you'll actually see that a lot of things that exists in Karel's world will come back because we give them to in a gentle form in Karel's world and then the Java world, they come full steam.

So before we dive into that, I want you to get a little notion of the history of computing. Okay, and we'll go over this very briefly, but computing actually has a very long history and it dates back about 4,000 years, okay. So roughly 4,000 years ago, the first one we think of is computing device was made available, but then you want to know what that was? Advocates.

The first computing device allowed you to do some fairly rudimentary arithmetic on it, but it actually was something that allowed people to compute a lot faster than they could by keeping track of stuff in their head. The real sort of advances in what we think of as computing actually came around in the 1800's. That was someone by the name Charles Babbage. Anyone heard of Charles Babbage, by the way, a just quick show of hands. We can just call him Chuck.

He was actually a very well learned and well-known person in his day. He had the Lucasian chair, which is the same chair at university that actually Steven Hawking has now that Sir Isaac Newton had at one point. So, pretty smart dude, and he came up with this notion of something called the Difference Engine, which is a way of being able to automatically.

And in those days, right, they didn't have silicon. Well, they had silicon, but it was in the form of sand, and they didn't have computers in the way we think of computers, so he wanted to build a mechanical device like with real, actual, you know, mechanics and pieces of woods that's been around, and in his time, he thought is would be powered by

steam, right, because this is sort of pre thinking about electronic computers that would do automated calculation.

And he designed something called the Difference Engine and then he designed something called the Analytic Engine, which was supposed to be even more powerful, and interestingly enough neither one of them was actually built during his lifetime.

It turns out that years later, the difference engine was actually built from a bunch of wheels and rotors and stuff and now the things, you know, that he would have liked to think of as the Analytic Engine is actually these puppies. It actually turned in the 1800's; he had a lot of the same ideas that are in our laptops or in our computers today strangely enough.

But, so that's kind of where our computing we think of the notion of computing is really coming from, and it turns out the first programmer was a woman by the name of Ada Byron, actually Augusta Ada Byron and if the Byron looks familiar, too, is because she was the daughter of Lord Byron, the English poet and she was actually very taken by the designs of Charles Babbage's machines and actually wrote programs.

The machines were actually written to have sort of cards, sort of like the Jacard Loom if you've ever heard of the Jacard Loom, if you haven't, it's not important, that actually would have programs for the kind of computations that it would do and see, she actually devised various programs for Charles Babbage's Analytical Engine.

And you might be sitting there and you might think, "But Marion, you've just told me that the Analytical Engine was never actually built in their lifetime, so what's she doing writing programs for a computer that doesn't exist," and that's actually something that happens today. People write programs for computers that don't exist and you might wonder why. That's not a very good use of your time.

Well, guess what, if the next generation of computers is going to come out while it's still being designed before it's actually manufactured and built, someone needs to figure out what kind of programs you want to run on that machine, so there are actually programmers today who actually write programs for machines that aren't built. And they simulate them by hand and they go through and try to figure out what they would do and it's perfectly reasonably thing to do.

But as a result, Ada Byron is in some sense, the first programmer, right, because she was actually writing programs for in some sense, a general purpose computational device over 100 years ago, which is kind of an astounding thing if you think about it.

Now it turns out computing actually got its first real, you know, what we think more closely of as computing devices in the 1930's. I should put 19 here just so we're clear – 1930's and 40's, they were sort of the first prototypes of electro mechanical computers that were actually built at the time.

There was a computer that was built at Iowa State by Atanasoff and Berry. Their names are actually not – well, they're important if you're a lawyer because it turned out there were actually lawsuits at the time by who patented it in the computer, but we won't talk about that.

Started sort of in this era with sort of prototypes and then the one machine you may have actually heard of by 1946, there was a machine called ENIAC, which was actually built by Elkhart and McCauley at U Penn which was standard for, if I can remember this, for electronic, numerical integrator and calculator.

And basically with one of these big things, you know, sat in a big warehouse, a few tons, but it actually did computational, you know, the kinds of things we would think of as computation and it was really in some sense the first large-scale electro magnetic computer that we think about, again, you know, modular lawyers.

And then really it was in 1971, that the first microprocessor came around, so we come to a fairly modern time, right, so it's not all that long ago, like we're talking 36 years, the microprocessors have actually been around.

And so the first microprocessor and one of the folks who was on that team who built the first microprocessor, Ted Hawk, is actually Stanford alum, interestingly enough, it was built at Intel. The Intel 4004 was the name of that microprocessor and at the time, no one actually thought that having a single chip microprocessor was going to be that interesting.

He sort of designed this thing and it was originally going to be going over for use in some machine in Japan and they were just going to give them some design and the patent for it and the folks were getting like, "Well, we're not really interested. Yeah, sure it's an interesting chip, but we're not really interested in, you know, owning the patent. We don't think there's anything interesting necessarily here," and so they kind of ran with it and the rest is history, as they say.

And as you can imagine in the last 35 years or so, there's just been an astounding amount of development in computing and computer science, right, so really this field is like hundreds or thousands years old, if you think about it, but really it's kind of in your lifetime that all the interesting stuff is really happening, which is also why it's exciting to be alive now and be a computer scientist because you're in this acceleration phase, right.

Think about what's going to happen, like all the stuff that happened in the last 36 years, you're gonna be around to see the next 36 years and you're gonna be doing the next 36 years, so that's just an exciting thing that I think about.

But another thing that comes up when you think about this, what is computer science, right, like why do we – we've been talking about Karel programming and a lot of this class we're gonna learn about software development and Java programming and so a lot of people tend to equate computer science with programming and they think, "Well, why

isn’t a major just called, you know, computer programming and why is there really a science to it in some sense.”

That’s something that people actually would use to debate. They don’t really debate it so much anymore. Is there a real science to computing or is it just programming and in fact, the difference between the two is that computer science or CS as we’ll just affectionately refer to it, really is the science or in some sense, the study of problem solving with computers, and you don’t need to write his all down; it’s just for your edification, with computers or I should say, computational devices.

Computational – I’ll even say methods because some people actually look at, you know, theoretically what’s possible without even necessarily thinking that it’s real life in hardware, right.

Turns out it actually surprises people there are some functions that you can prove are not computable. I'm like, “Really, you can?” Yeah, and it’s good to know about them before you go and try to write a program to actually compute them because in theory they’re not computable and you can prove that, and if you’re actually interested in that, take CS 103 or CS 154 and we’ll talk about it all the time.

But that’s really what the science is, is thinking about problem-solving and the approaches to problem-solving, how you analyze how efficient a solution is to a problem, how solvable it actually is, the different approaches you take to it.

So, notice the word programming doesn’t show up here. Programming is an artifact. It’s something we do to realize that particular problem-solving technique in a computer or some computational method. It’s just something we do as part of the process, right. It’s not what the whole thing is really about.

In some sense, there is actually this famous quote that computer science is just as much about programming as astronomy is about building telescopes. Right, astronomy is about far more than building telescopes. It’s really sort of a science of celestial bodies. Telescopes is the mechanism by which you actually look at them. Uh-huh, question in the back?

Student:[Inaudible].

Instructor (Mehran Sahami):Well, if you think about computer engineering, part of it is just kind of a definition that a particular school wants to have for it, right. And some places actually – our computer science program is in the School of Engineering. Some places it’s actually in humanities and sciences. It depends on the view and it also depends on the particular program that’s there, so some places may actually –there’s no way I'm gonna hit you in the back – may just be training programmers, right.

And the science of computing is not what they’re actually interested in, and so it depends on the program, but we really think of the computer science as a science, and it’s an

interesting philosophical argument. We can certainly talk about it more in class, but I just wanted to kind of push on.

All right, so with that said, that's kind of the quick tour of a couple hundred years of computer science history, it turns out when we get into modern day, when we actually think about computing, what does a computer really understand and what are the programs that you write on it and how do they turn into something that the computer understands.

So it turns out that the computer actually understands zeros and ones. And so we like to think of that as binary, right. Binary is just a number system that only has zeros and ones. It's base two numbers basically and that's all that computers understands, zeros and ones on and off.

So how do we take these things like move or turn left and turn them into ones and zeros, and so there's something called machine language that is what a computer understands or in some sense what the microprocessor understands, right. The machine languages is defined by what chip you have inside your computer. That's eventually going to boil down into a bunch of ones and zeros.

People are real bad at writing ones and zeros in a form that computers can understand. So we program what's known as a high-level language, okay. And there's all kinds of high-level languages, so one of the ones that you're gonna learn in this class is Java and there's some other ones you may have heard like C or C++ or Basic or Fortran or whatever the case may be.

These are all high-level languages, because they're at a higher level than what the machine understands, and so there's this question that's actually part of the science of computing is how do you go from this high-level language into something the machine understands, right. What is this translation process because there actually is some translation that needs to happen and this process going form the high-level language to the machine language is something we refer to as compilation, right.

So compilation is something, strangely enough, that's done by a compiler and Eclipse which you've been using this whole time is in fact a compiler. It takes the instructions that you write in some high-level language and converts them into some form that the machine understands.

Now it turns out that the process for doing this in some languages is actually slightly different than other languages, so I want to show you a quick little overview of how that looks in Java just so you can get a real sense of what's going on between the time you write a program and the time the machine executes it so you sort of understand it at a low level.

As programmers, you write what's called source code. Know it, learn it, live it, love it. Source code – that's what you're gonna write. When you were writing Karel programs,

you were writing source code. When you write Java, you're going to write source code. What the machine understands, these zeros and ones here, it's something that's called objects code.

An object code is essentially just the low-level instructions, the machine instructions that the computer understands and so going from source code to object code is basically what the compiler does. It's the translation from this to this.

Now in regular languages, that might look, if we got the overhead. Is shouldn't say regular language. In some classical languages, it might look something like this, you write some source code in what we refer to as the source file.

The source file is just a file that contains source code. So you write your program that goes into some compiler, say Eclipse, which is the compiler you're using and what might come out of it is an object file because what is contained in that file is just essentially object code or in this case a bunch of zeros and ones that the computer understands.

And there might have been other programs that someone wrote along the way like some libraries you might make use of like you did with Karel, right, all the basic Karel functionality might be in some other library and so there's other files that contain object code and all of this stuff gets linked together into one big set of object code which we refer to as the executable or the application.

That's the thing that, say, when you double click on your word processor on your computer and it runs, you're running some executable file, which is just basically a file of a bunch of ones and zeros that eventually or as some time ago, someone wrote and source code and it got compiled down in this executable file.

And for a lot of languages like C or C++, this is the process that actually happens. Now the people who did Java thought of things slightly differently and here is kind of where things get funky in Java, okay?

In Java's world, part of what's going on is actually run on a virtual machine. What does that mean; it means you write some source file in Java, okay, that goes through some compiler and what comes out of it is not an object file, but something called a class file.

It takes all of the high-level stuff that you write in Java or in this case, you could say Karel because it's the same thing, and turns it into some set of numerical instructions that are not yet ones and zeros that the computer understands but are some intermediate language, which is just a numerical language.

And sometimes you refer to this as Java bite code, but the name is actually not important. It's some intermediate language, and guess what, there's other classes just like there were before that contain instructions in this intermediate language and those all get linked together in some big file that we call a jar archive which just stands for jar just means Java archive, strangely enough.

So it’s actually redundant to say jar archive. People just do, but it means Java archive. All right, and then this whole thing is now instead of this intermediate language, it’s something that neither the human really understands – okay, there’s a few humans in the world that might and they’re a little weird and we won’t talk about them. Most people don’t understand this and the computer doesn’t understand it directly either.

So you say, “Well, that’s the most useless thing ever. Why would I ever do that?” These instructions go to something called the Java Virtual Machine, the JVM, and what the Java Virtual Machine says, it says, “Hey, guess what? I’m going to be pretending like I'm a machine that understands this as my object code. I understand this stuff as the basic language I speak, and I take that and when I run it, I do something on your computer.” And you might say, “Why would you want to have this extra process?”

And the reason why you want to have this extra process is the fact that guess what, in the world there’s things like Macs and there’s PCs and there’s Linux computers and there’s all kinds of machines out there, which means if you sort of do things in either the good old way or the bad old way, the compiler needs to understand what are the ones and zeros that your computer speaks and the ones and zeros that a Macintosh speaks are different than the ones and zeros that a PC speaks.

And they’re different than the ones and zeros that the Linux machine speaks. So the compiler needs to know all that if it’s actually gonna generate this kind of code for all of those different sorts of machines and most compilers don’t. Most of the time you’ve got a compiler that says, “Hey I'm a PC compiler. I'm just gonna generate stuff for the PC and that’s all I do.”

In Java’s world, things are a little bit more interesting because the compiler doesn’t need to know what kind of computer you’re running on. It says, “Hey, I'm producing this intermediate language and the intermediate language is the same for all computers.” The only thing that has to be different on your computer is you need to have this little thing called the JVM that knows how to translate this intermediate language into what’s going on on your computer.

So if you have a JVM for your Mac or a JVM for your PC or a JVM for your Linux machine, they can all run this same low-level intermediate code and do the right thing on those computers.

Okay, and so remember when you set up Eclipse or if you haven’t set up Eclipse yet, you’ll get there soon enough, but when you’re wetting up Eclipse, we asked you to download and install something called the Java run-time environment and how many people remember that? Raise your hand if you remember that? You folks – guess what the Java runtime environment provided? It provided this thing for your computer, okay.

So you write the program once; you compile it once and now that class information that you have, that class file can be run on any computer that has this Java Virtual Machine

and Java Virtual Machines are sort of ubiquitous. They exist for a lot of different computing platforms, okay.

So that's why they actually do it that way, and it's kind of funky, but it's a little bit different. Now, with that kind of said, that's the low-level stuff. That's kind of what's happening at the low level and you don't really need to know that low level intimately. I just want you to understand what's happening from the programs you write to what happens when they get executed on the computer.

Now what you do need to know, is how do we begin to write programs in Java. In Java, you'll see some examples of it today, is what we refer to as an object-oriented language, okay. So Java is object oriented and not all languages are object oriented. What is this whole object oriented thing about, so you should know what this whole notion is about before you start programming in the language that's kind of seeped in the idea.

The idea basically is that a program is written as a bunch of classes, right, so if you think about, let's back up for a second, if you think about Karel, right. When you are writing Karel programs, of you saw Karel programs, you would actually create some Karel program by saying, "Public, class Karel program extends Super Karel." Let's just say we are Karel model.

And then you put some stuff in here and you have like your run method in here, maybe some other methods and what you did was you were creating a class that was basically the information for your program, what instructions you were actually going to execute as part of that program.

So what a Java program is is just a set of classes. You may have more than one class. Like in Karel right now you always just have one class and inside that class you may have a bunch of different methods, but you always have one class. A Java program can actually have multiple classes associated with it.

And when we start in this class, we'll start with some simple ones that are all just one class. What you have to think about is that you can have more than one class, okay? And what these classes are, what you can think of a class as, okay, a class is just basically an encapsulation of some behavior that the program does, just like in Karel's world, you had some behavior of a bunch of different methods you define for Karel to actually to be able to execute, right, those are just different behaviors.

But you also have some data, so it's behavior and data, because in the Java world you are now actually gonna keep around a track of information that are not just beepers in some world, but they're actually just numbers that you're actually gonna store somewhere and that's your data.

And so what we do is we think of having the behaviors, all the kind of manipulations you might want to do with data, along with the data and put that all into this thing called the class.

So just like in Karel where you had a class an you had behaviors, you can just imagine, yeah, we're gonna have some behaviors, which means in Java we're gonna have some class and define a bunch of methods in that class, but we're also gonna have some data associated with it and that whole think together is gonna be a class.

Now, it turns out, the thing that's interesting about classes. Is there any questions about the notion of a class? It's just kind of this abstract concept right now. The thing that makes classes really powerful is the classes actually get organized into hierarchies, I mean, just draw it here, okay.

And you've actually already seen this in a very small way with Karel, so all the things that you've seen in Karel are gonna translate over. Remember we talked about the basic Karel model and when you are writing a program you could say extends Karel and when you did the basic Karel model, you got like move and turn left and pick beeper and put beeper and that's all you got.

And then we said, "Hey, you really want to turn right and turn around, well there's this thing called Super Karel and Super Karel is just a souped up version of Karel." Well, guess what, Karel was just a class, and Super Karel was also a class, but Super Karel is a class that extends the functionality of the basic Karel class, okay.

So we say that Super Karel, the way we would draw things, if we were gonna put boxes around them and draw like the book is say that Super Karel is a sub class of Karel and Karel is the super class, this is where it's gonna get confusing, but super class are the things in some sense that are more general and the sub class are the things that take the basic idea of some class and extend it.

So Super Karel does everything that Karel does, but it also does turn right and turn around, okay. So Super Karel is a sub class of Karel or alternatively Karel is a super class of Super Karel, which sounds a little bit backward, but the super here and the super here mean two different things. Super here means sort of above, and super here means cooler than, okay.

So don't want to think of them as they mean the same thing, it's just an overloaded term, all right? So you might say, "Okay, Marion, this is getting a whole little abstract for me." Well, let me put it in terms that you understand. You're a human being, right? At this point, you should say right. And if you're not, come talk to me. All right, it wouldn't be the first time.

So we have humans, okay. Humans are just a class, okay, and all humans are primates. Primates are a super class of humans, because all humans are primates, all primates are mammals, and all mammals are animals. So there's actually a much deeper and richer hierarchy say in biology then there is with Karel and when you actually see Java, Java will eventually have sort of a richer hierarchy associated with it.

Now, the interesting thing about this, this is not only humans primates, but monkeys are also primates, okay. Humans are not monkeys and monkeys are not humans, so there's not a strict – you might say, "Well, we don't know about that." But there isn't a strict relationship between humans and monkeys other than they're both primates, and that makes them mammals, and that makes them animals, and the whole notion of having this hierarchy of classes, right, so you could think of being animal as just a class and it has some behaviors. What does it mean to be an animal?

Well, to be an animal means you digest food, right, it's not something like something like [Inaudible] actually, and one of the technical terms – I'm not a biologist, but just what my friends tell me, embryos pass through the blastula stage. Any biologists in here? Is that right?

Student: Yeah.

Instructor (Mehran Sahami):Okay, good, good – because I was like asking a friend of mine, like, "What differentiates animals form plants," and he was like, "Oh, of course, the embryo was passed through the blastula stage and I was like, "Oh, right on." I had no idea, and I was like, "How do you spell blastula?"

So I'll just put blastula down here, right. So that means, guess what, all mammals inherit those same properties. Mammals also have some internal digestion of food and also pass through this blastula stage, but mammals additionally we know are warm blooded in general and they have mammary glands which is where the name mammal comes from.

So mammary glands or just kind of a generalization of sweat glands, but that's what mammals have, and then all primates are mammals, which means all primates have all of these same properties plus they have more and so primates interestingly enough have five fingers. I didn't know all primates have all five fingers, but evidently I'm told that they do. They have opposable thumbs. And they also have fingernails, but fingernails we just won't put up there. It's not a big deal, okay?

And then we get down to humans and monkey, right, and humans it turns out in theory anyway were supposed to have highly developed brains. I'll just put brains in quotes here because other animals have brains, too, but we're supposed to have highly developed brains and we have an erect body carriage.

That's just what we are. We're wrecked brains. And monkeys have mother things that are going on, so evidently their brains are not as highly developed and they're sort of like their knuckles drag on the ground. Sometimes I do that in the morning, but that's not important.

But that's the whole point. This is the class hierarchy. These are all classes and things that are sub classes of some other class means they inherit all of the behaviors of the things above them, all the way up the chain plus they may have their own additional

behaviors and that's the key concept here of organizing things in classes and you'll see that in more specific instance when you get into Java.

You saw it already with Karel in just really simple example, right. There was four commands for Karel. Super Karel gave you those four plus an additional two; that was it; okay? Now, besides this idea, there's another key idea that comes up besides this notion of classes, okay? And that's the notion of the instance of a class, okay.

And what the instance of a class is, is humans are a class. There is no one person that is, "Oh, you are humanity, right?" You are a person, right, so [Inaudible] what's your name?

Student: Eduardo.

Instructor (Mehran Sahami):Huh?

Student: Eduardo.

Instructor (Mehran Sahami):Eduardo, all right. So I hope I'm spelling this right, Eduardo; is that right? Is an instance of human, I would hope, right. So what that means is you're going to have things in your Java programs that you create which are objects and what an object is to differentiate it from a class, so Eduardo is an object. I'm sorry; sorry to break it to you. That's just – you're an object.

An object is an instance of a class; it is a particular example of a class. There will be multiple objects or multiple instances potentially of some class, and that instance of a class has all of the characteristics of the class and all of the other classes above it in the class hierarchy.

So by looking at this, I could say, "Hey, Eduardo, guess what, you went through the blastula phase when you were an embryo." Good times; because you're an animal, right and I know that and this is inherited all the way up.

So that's the important thing to keep in mind is you're going to be writing classes; you're going to be writing things that define some set of behavior and along with that, you'll be creating instances so that this is an instance of the class and the instances are what we refer to as objects so all of your instances in the world are just things that we think of as objects and your classes are in some sense the templates for those objects, okay.

So any questions about the general concepts; I know they're a little bit high level, but it's important for you to kind of understand. Okay. So with that said, let's actually begin to look at this notion in Java, okay? And so the way that this is gonna work – uh-huh?

Student:[Inaudible] in Karel, what was the object?

Instructor (Mehran Sahami):So in Karel, it turned out what you're actually just creating was a single Karel object was being created that contained your program and

that's what was actually run. So you created the class Karel, and when that puppy actually sort of fired up and you saw little Karel running around in the world, that was a Karel object; that was an instance of your class.

Now, you actually didn't see multiple objects or multiple instances there. There was only one Karel instance. So there it's very – it's a good question because it's easy to get confused between the instance and the class in that case because it was only one.

But in here you'll actually see we'll actually create classes and we'll have multiple instances of them, which are running Karel with basically the instance of that object. Okay, question?

Student: So, basically like you can have, like several Karels running around in like one program?

Instructor (Mehran Sahami):You could if we sort of set it up that way. In the Karel, the way it's kind of set up is it only allows you, it only sort of behind the scenes creates for you this one instance of Karel, but in some, you know, alternative Karel universe, we could have actually have just taken her class and said, "Hey, guess what, we're gonna just create multiple Karel instances from your class and have them all run around in the same room."

We could have, and guess what, that's what seemed to have happened in this room, right. Somewhere along the way, there was this class human and someone came along and created multiple instances of you. Well, I wouldn't just say just one person, but now there's multiple instances of you, right, and you're all running around in the world doing your thing and you interact with each other and guess what objects actually do; they interact with each other, okay?

So with that said, there is gonna be some functionality we're gonna use that was written by someone else. So just like Karel, the basic version of Karel was written by someone else. When we start with Java, we're gonna start with some set of scaffolding or libraries that were written by someone else that are gonna allow us to do a bunch of powerful things in Java very early on, okay? And this is something that is called the ACM program hierarchy. So it sounds all complicated.

But all this is, right, and much the same way that I drew that picture over there with animals, mammals, primates, humans and monkeys, this is a hierarchy of classes that exist in Java's world or at least are provided by the ACM and you might say, "Who is the ACM?" Anybody know who the ACM is? They're big; they're bad; they're nationwide.

They're the Association for Computing Machinery. It's the oldest computing society. They've been around for actually for about 60 years or so. And you're like, "But Marion, I thought you said 100 years. Yeah, before then, people called it math.

So Association for Computing Machinery has this program hierarchy and what that means is the kind of programs that we write in this class, just like when you wrote your Karel programs you extend Karel or you extend Super Karel, you're going to be extending different kinds of programs, either a console program which is something that produces textural output, a dialog program which brings up little dialogue boxes that ask for information or a graphics program that actually draws some pretty funky stuff on the screen.

All of those classes are classes that are inheriting from a class, a super class called program. So all of the things that you write as a dialogue program or a graphics program are all something that are programs. And all programs are something uptight J-Applet, which just means Java Applet and all Java Applets, are of some type called Applet.

Anyone ever heard of an Applet? An Applet is something you can run on your web page, interestingly enough. That's kind of where the term comes from. It's like a lightweight application, and applet – application, but small.

And so it turns out since all of the programs that you're writing are actually inherit the properties of being an applet, they actually will be applets, which means you can put them on your web page and run your programs on your web page if you want and later on in the class we'll talk about how to do that, okay?

But in this class, as least, you're not gonna write anything that directly extends like J-Applet or Applet or program. Everything you write is gonna extend down to this level, but it's important to know that there's different kinds of programs that you can actually write at that level.

So let's look at an example of a Java program, and I'll show you your first Java program today. So we can feel good about Java – no it's not that one. It's this one. It's small; it's tiny; it's fun; it's Java. All right, so let me expand this whole thing out so you can see the whole program still fits on one screen, okay.

And you might suddenly notice there's a bunch of things in this program that look very similar to Karel. That's because Karel was implemented in Java. So the first thing you want to look at here is we have a file called helloprogram.java, just like your Karel programs are written with a .Java file, you are creating a .java file here. This is a source file because it has source code in it.

Up at the top, you have a comment, "Gee, it actually uses the exactly same structure as Karel comments." Yeah, because Karel comments were actually Java comments. Comments in Karel and Java are exactly the same. Just like you imported stanford.karel.star, now you're gonna be using the stuff that the lovely folks at the ACM have provided for you, so you're going to import acm.graphics.star and acm.program.star.

What are these things? These are just – well, I won't back up because we're not on the slide. These are just – remember I said you write your classes and someone else may have written some classes and they all get linked together before they're executed. These are just some other classes that someone else wrote that are going to get linked into what you do.

Right, they provide you the definition for things like what a graphics program actually is. So now, what you're gonna do, as we talked about, all Java programs are just the collection of classes, so just like in Karel, we have public class and some name for your program. Here we'll call it Hello program.

And a Hello program is a particular kind of program. It doesn't extend Super Karel because it's not a Super Karel program anymore. It's gonna be a graphics program. It's actually gonna draw some stuff on the screen, so it's gonna extend the graphics program.

But all the boilerplates should look the same, right, that's what we had you do Karel because all the stuff from Karel just translates directly over to Java, except now we're sort of cooking with gas; now we're doing the real thing, okay.

So it extends graphics program and guess what, inside here we have a run method just like Karel and that's where the program begins executing. Now, where things get funky is you're like, "Oh, may what happened to pick beeper and plug beeper and like life was so good; it was easy and you know, turn right and it was like the extent of it. What is it like add nugi label, hello world, like what is this all about, okay?"

We'll go through this step by step. All this is saying is when you create a graphics program, we'll go through the details in just a second, you're gonna get a blank screen, you're going to get an empty canvas in some sense.

You're gonna be an artist and you're gonna draw stuff on that canvas, and the thing that you're gonna draw stuff on that canvas and the thing that you're gonna draw on that canvas is some label and all that label is is just basically words and the words are gonna be Hello World, because you wanna write Hello World on the screen and you wanna write it at a particular location on the screen.

The location on the screen you wanna write it at is 100,75 and I'll tell you where that is on the screen in just a second. And then once you create this little label, you're gonna add it to your canvas. You're gonna say, "I have some canvas, plop that puppy onto my canvas," okay?

So let's run the program and I'll just plop it onto the canvas or you'll see what's going on. Life will be good. So we run this program, we drop the microphone. Dropped the microphone again, all right. So we want to do Hello program. We're getting excited. We're running; we're running. The disc is just turning away and there's your first Java program.

You're all Java programmers now. What did you do; you created a graphics program, which brought up this big window and said, "I'm blank canvas. Draw on me." And you said, "All right, Hello World – woohoo and I'm done," because that's all you did. You said here's Hello World, put it up on the canvas for me at a location 100,75 and thanks for playing; that's all I'm gonna do.

But now you've actually gone through the whole process of compiling your Java program and turning it into this intermediate code and this intermediate code gets executed and you actually did something and that's like half the battle right there. The next nine weeks is the other half of the battle, but half the battle is getting this up. I kid you not, really. If you can make that happen, you're just most of the way there.

So let's do something a little more exciting than this. Let's actually do some interaction with the user. Let me show you another program. Remember this one is a graphics program. I told you there was a bunch of different kinds of programs you can have when I showed you the picture. I said, "We have a graphics program." Let's look at something called the console program over here. Uh-huh.

Student:[Inaudible].

Instructor (Mehran Sahami):No, not in Java. You're always extending one kind of program and a graphics program will actually, as you saw you can put up text. A console program is just meant to be a textural program, so you're not gonna do any drawing.

But here's another program, right, again, a lot of the same things is before. You have some comment up at the top. You have some libraries that you're gonna import. In this case, you're not doing anything with graphics, so you don't need to import acm.graphics.star, you just need o import acm.program.star which just says, "Get me all the standard stuff for a program."

If you're doing graphics, you also need to have a second line for acm.graphics.star. If you're not doing graphics, you don't. So again, public class. This one called add-to integers because guess what, it's gonna ask the user [Inaudible] programs, put tow integers and add them together, but in fact, it's gonna do something interactive with the user which is pretty exciting in itself. This is gonna extend the console program because it's not gonna have any graphics.

Aw, yeah I know, it's sad times, but sometimes text is very powerful. What's it gonna do when it runs? It's gonna write out to the screen and this time we're gonna use something called print lin, we just like to drop some vowels here and there, which stands for print line, and if print a line, this program adds two numbers, then it's gonna read an integer from the user and we'll go through all this stuff in more detail. Don't worry; this is jus a high-level overview so you get some basic idea of what Java looks like.

It asks the user for N1, right, because you have to be very scientific, so rather than give me the first number, you say enter N1, right, and suddenly we're much more formal.

Enter N2, right, and so what this is actually doing it's asking the user for a number and whatever the number the user gives us, we stick in this location called N1.

And whatever the number the user give us here we're gonna stick in a location called N2 and we'll actually talk about what these locations are and how they get set up and everything next time, but I just want you to see something before we can kind of go into all the details because we gotta start somewhere.

Then we're gonna add N1 and N2 together and store them in a place called total and then we're gonna write out the total is, whatever the total is, and then a period. Right, so this is in some sense the world's most expensive calculator that adds two numbers.

And we'll go ahead and run it, add two integers. We're compiling; we're feeling good and that's the program we want to run, and now this comes along and notice, now we have some window that rather than putting graphics all over it and telling it where to display some words like Hello World, we just said, "Write some line," and so what it's gonna do is just sequentially write lines on the screen because this is what we think of as a console.

All a console really is, the way you can think about it is it's a window that contains texts and can potentially interact with the user, so there's places where we might ask the user for input. So we said if this program adds two numbers, I hope you can see this in the back. It's tiny, tiny font. Enter N1 and it's sitting there with a blinking cursor, and any time you see the blinking cursor, that means, "Hey, I'm expecting some value." So give me something of value. What's some integer?

Student:[Inaudible].

Instructor (Mehran Sahami):Twelve; someone said 12; someone said one, so I'll take 12 as N1 and then it asks for N2. I'll give it one and low and behold, it says the total is 13 period, end of program, okay? At this point, there's a blinking cursor, but the program is done. But now you've just seen a real Java program that actually takes in values from the user, computes some, does some computation on them and displays output, okay?

And this is a console program as opposed to a graphics program because all the stuff we're doing here is textural. Okay, so any questions about the basic idea of classes or objects or these different programs we're writing? The programs that are running, when they run, they are being, they're objects. They're instances that are being run for you. What they're doing as instances, they're objects that are inheriting all the behavior from the class template and then when we run, an instance is actually created in this run. Uh-huh?

Student:[Inaudible].

Instructor (Mehran Sahami):Because if you say acm.star, strangely enough, you won't necessarily get everything underneath ACM as well. You'll get everything at that level.

So there's actually a package for ACM and there's a package ACM graphics, and there's a package ACM program and what you really want is everything inside ACM program and ACM graphics, not just the things that are in ACM graphics by itself.

Yeah, it's an interesting technical question, and you don't need to worry about the details. Just add both of them and you'll be fine.

Student:[Inaudible].

Instructor (Mehran Sahami):No, the console program you just print out. That's one of the differences with the graphics [Inaudible]. With the graphics program, you have to tell it where stuff goes. Console program is just writing out line by line. Uh-huh?

Student:[Inaudible].

Instructor (Mehran Sahami):Sorry.

Student:[Inaudible].

Instructor (Mehran Sahami):Yeah, every time you run a Karel program, you can think of what it's doing is creating an instance of Karel which is that little guy you see on the screen and he's going around and doing stuff, so in Karel there was only one instance of the object.

Here you'll actually see we'll create some objects where we can have multiple incidents of an object in the same roll and I'll show you some examples of that in just a second, but we just haven't gotten there yet, okay?

So what we're gonna do now, is we're gonna think a little bit about graphics that we can actually draw. Because the graphics world it's easiest to see multiple objects. We're gonna put multiple stuff up inside our little graphics canvas.

So what's this whole graphics thing about? Let me tell you a little bit about the graphics window and then we'll put a couple different objects up in the graphics window and you can see what I'm talking about. So in terms of graphics, all graphics that we're gonna do are modeled called a collage model and if you remember in the days of yore when you were a week tyke, did anyone have like a little felt board with little felt animals that you put on it?

Yeah, that was a collage, or if you had a big piece of paper and you had some like crazy, not crazy glue, what was that, glue stick. You don't smell that stuff too much – like, but you would take some piece of paper and you're like, "Oh, here's a picture of a cat," and you would like put some glue on the back of it and put it up on there, and then you'd say, "Oh, here's a picture of a dog," and put it up on there. That's the model for graphics in Java.

We're gonna have these little pictures or little objects and we're gonna say, "Show up here and show up over there." So what you have is a collage which starts off as a blank canvas and you're going to put objects onto that canvas, okay?

And so what we can do is create various kinds of shapes or pieces of texts and put them up in various places on the canvas and you just saw an example of that with Hello World, but I'll show you some of the other things that you can create. Two different kinds of objects that exist in the graphics world, one you just saw. It's called a GLabel for a graphic label. This is just text in some sense.

It's text that you can put up in the world. There's also a GRect, which is a rectangle, oddly enough, a GOval, which is an oval or if you actually make the height and width the same, it becomes a circle, right. So that's why we don't have circle; we have oval and same thing with rect versus square because all squares are rectangle, and GLine, which just means a graphic line.

So these are the different kinds of objects we can have. These are classes. They are templates for objects. We are going to create particular instances of these objects and put them up in the collage, but these just tell us the kinds of things we can have in the general class.

So if we come back, if we close this guy, and we come back over here to fund the below program, what we've done is we've said, "I want a GLabel. GLabel is the class. When I say new GLabel, what that actually does is says, "Give me a new instance of that class. Give me an object that is a GLabel," and the things that you're going to specify about that particular instance or it has some text associated with it, which is Hello World.

We've put all the texts inside the double quotes, okay. So Hello World without the double quotes is actually the text associated with this label and we give it some location, 100,75.

So now this little object, okay, which is an instance of a label says, "Hey, I'm one particular label in the world and the kind of label I am, I say Hello World, and I'm at location 100,75," and it's like, "Okay you have this label, what are you gonna do with it."

And I'm like, "Put it on the canvas," and the way you put it on the canvas is you say, add. So all graphics programs, when you say add, there is something that you will specify to add, and the things that you were specifying to add are instances. They are individual objects that you're going to put on the canvas and the way you get those individual objects is you create a new version of some particular class, okay?

So this creates that new label, location 175 and when it gets added, it gets put up there. Now, there is other kinds of things we could do with this, so if we're kind of return over here, this is just the add-to integers program, let's do add-to integers real quickly – 17, 25, 42, writes up same thing you just saw except now in excruciating detail.

Let's say we want to do something that makes a GLabel more funky than we originally had, okay? So again, we're gonna have our Hello Program that's gonna extend the graphics program. Then we're gonna do things slightly differently. What we're gonna do here is we're going to create a new instance of a GLabel object.

So if we say new GLabel and again, we're gonna name is Hello World and 100,75, so I have some label that's Hello World and location 100,75, and I say, "Hey, you know what label, I want you to look different than you actually are, okay?

So now what do I have this object, I have this little label, you can think of it as a little object that sits somewhere in the world that's not being displayed yet because I haven't added it yet to my canvas. It's being stored somewhere which I have named label, okay, and we'll talk about this notion of GLabel, label next time, but it's something I've created called Label which is just a new instance of this Hello World label.

The first thing I want to say is, "Hey, you know, I want to change what the font looks like." Anyone know what font is. All right, you've probably used different fonts in your word processors. It's just the way characters actually appear on the screen. It's a thing that comes from type setting.

I want the font to be san seraph 36, so what happens when I start off, is I say, "Create for me a new label that's name is Hello World," and so it says, "Okay, I have some label object, the label with the words associated with that label Hello World."

Notice there's still nothing on the screen. Here's my screen. Here's my canvas. This is all in the computer's memory somewhere. It's saying, "Yeah, you've created some label. You haven't displayed it yet. You've just created it," and you say, "Hey I want to tell that label to change it's font to sans seraph 36."

The way I do that is I have an object that I've created. The object is named label. I use the name of the object followed by a dot, followed by a method that I'm going to call on that object, remember in Karel's world, we had methods. The way we called Karel's methods was we just gave the name of the commands to go execute that method.

Now, all methods are associated with objects, at least for the time being. So when I have label and I say, "Set font," what that actually means is set the fonts for this particular label object to be sans seraph 36, and so when I execute that, what happens is it becomes 36 point font in sans seraph which means seraphs are just little things of if you can see the font up there has you know, swishes at the end of letters, those are seraphs, so a font like this doesn't have any of the squishes at the end of the letter. San seraph, sans means not, Okay?

Then we say, "You know what, I'd really like you to be red because black is just so blasé and red is really the new black, so set your color to be the red color." Like, okay, and so I execute that and it turns itself red. There's still nothing on the screen.

This is all like of with the label and now that I've taken this label and I've taken this text and made it big and turned it red I finally say, "Hey, graphics program, add this label, add this object to your collage," and it says, "Okay," and it adds it at the location at which this label was created which was 100,75.

So then it finally shows up on the screen. It only shows up on the screen when I actually do the add. Up to that point, all I've done is modify some of the properties of this thing, okay.

So starting next time, what we're gonna actually get into is thinking about what is this GLabel thing about, what does it mean when we actually specify a method for an object and go into all that in a little bit more detail, but hopefully seeing an example of what your first Java program looks like.

All right, I'll see you all on Wednesday.

[End of Audio]

Duration: 51 minutes