PROFESSOR: OK. You can probably figure out, I'm not Professor Guttag. He's away today. And so he's asked me to cover this lecture, which I'm happy to do. I'm asked to remind you that there is no recitation on Friday. I was expecting at least a smile or a small cheer. Early lead into spring break. This ought to be good.

So my understanding is that at the end of last class, Professor Guttag was walking you through a set of classes. And I want to remind you of where you were. I want to look at one of those classes in a bit more detail, because we're going to show you one new piece of the language. And then we're going to turn to a different topic, which is going to be the subject for the next several lectures.

So just to remind you where you were, Professor Guttag showed you a set of classes which started off with a person class. There's a definition I'll put up on the screen. This was just something that created a kind of object. I'm not going to go through the details, but you had the standard things you'd expect. You had a way of initializing it. So you'd make each instance. And it had a way of pulling out the name, so you could have a full name and a last name. You could get out pieces of it, it set birthday, it had age, it had a bunch of stuff. And that's fine.

The second thing that he did that we do in a lot of dealing with classes is he created specialization of that, a thing called an MIT person. And you all know that MIT people are special people, right?

And what did an MIT person do? Well, again, you can look at it. It had, again, an init method. But one of the things that was a little different here-- you may have seen it in some other places-- is that we can create, inside of that class definition, an internal variable. In this case, it was the ID number. So that every time we created

an instance of this class, we could give that instance a special number and a unique number.

And that was exactly what we did here, right? If you're going to create one of these things, you set the ID number to be the next number. And then you would increment that, so that you'd be able to, next time you created an instance, give it a unique number. And that let you do things like put things out in order, and things of that sort.

There was also a method in this class that we're going to use in a second, which is something that would say, is this person a student? I'll just highlight for you there the definition of it, which said you'd look at the type of the instance and say, is it undergrad or grad? And that was the other two specializations we had here. We had undergraduates and we had graduates.

And again, I'm not going to worry about the details. You saw this last time. Undergrads, they have some methods like you could create an instance of it -- you'd have its year. You could give back what the year was. And for the graduate students, we didn't even bother with putting details in. It could be anything you want.

Now these are things that you ought to be getting used to looking at. So this is creating some class definitions. I can make some instances. There are a couple of nice variations like, for the MIT people, each one has a unique identifier, which lets us keep track of them.

I think where Professor Guttag ended up last time was actually creating a course list. So a list of students in a class, like 6.00. So I want to show you that, and I want to show you some highlights of that. So there's the definition. It's on your handout as well.

So what does a course list do? Well, the idea should be, it's a way of collecting together all the students in 6.00. So if you look at our init function up here, when we create an instance of a course list, what does it do? Well, we give it a number, like 6.00. And it's going to bind that locally inside of a definition. And it's going to create

a collection, a list. It's going to be a list of all the students.

Now if I'm going to do that, I need to have a way of putting students into the class. And I know nobody drops 6.00, but just in case-- drop date's coming up-- we have to have some way of removing a student from a class. So I'm going to have a method right here-- oops, sorry, I'm going to get it from the right place, right there-- that would add a student in.

Now take a look at that for a second. It's not just putting the student into the list. The very bottom down here, that last line, self.students.append(). What's that doing? Well, self, remember, refers to the instance. .students will get me that list for that instance. And then .append says, add on to the end of that list the person that I just added in, the person that is bound to who.

The things above it are being careful. They're defensive programming. They're ways of saying, look, make sure that the person is actually a student. Huh. Wait a minute, where'd that come from? isStudent. Oh, yeah, that was up here in MIT person.

So we're making an assumption here that says, we're only adding in students. No post-docs are welcome. We're going to only add in students. So we're taking instances of one of these two classes, and checking using the inherited method to make sure that this person is actually a student. And we're doing this in a careful way, which is we're going to raise an error if, in fact, it's not a student.

And then the second thing, notice, that we're going to do here is we're going to say, gee, make sure that we're not adding you twice. So we're going to do what? We're going to say, get out the list of students right there. and I begin. self is pointing to the instance.students gives me the list of students currently in that instance. And I'm basically saying, is this person in that list? And if they are, again, I'm going to raise an error saying, wait a minute, you're already in the class.

So these are things I'm sure you've seen before. But they're ways of being a little careful about the programming.

The removeStudent is roughly the same. So let's just look at some examples here. I've created a set of people. And just to show you what they look like, let's go down here. So I've created m1. Let me do that again, sorry about that. I'm going to run this.

All right. So if I look at m1-- it's not liking me, is it? We'll try it one more time. I'm going to look at m1. It gives me back an instance of an object. [UNINTELLIGIBLE] if I actually say a print of m1, it'll print out the name. And the same thing for the undergraduates here. So I've made a couple of undergraduates, I've made a couple of graduate students. And notice what I've done down here. I've created a course, SixHundred, by calling CourseList. So CourseList is going to make an instance of that. I gave it the name.

So we can go back over here. Let's look at what SixHundred says. OK. I'm sure you're used to reading that gobbledygook. It says, ah, it's an instance, and it's an instance of a CourseList. And it gives me a pointer to where it is.

Now what if I want to see the students inside this course? I'd like to see, who are all the students there? So printing SixHundred-- let's see if that does the right thing. Oops, no. That kind of makes sense, right? SixHundred is just pointing to an instance.

So where are the students? They're sitting down inside of an internal variable, which is, in fact, in this case, the list of students. So one way I could do this is I could say, take SixHundred, which is an instance, and go into that frame that corresponds to that and look up the local variable, students. And if I do that, oh, I've got a bunch of students named Main.

Now this shouldn't surprise you, right? This is doing what? It's giving me back that instance of the list. If I actually want to print out the names of the students, I have to do something a little more careful. So here's one way I could do it. I could say, for every student in that list, let's print out their name. Oh, good.

You're wondering, why am I doing this? Well, I want to show you what's going on in

this class. And then I want to highlight something for you. So let's think about what I just said. I want to get the names of all the students in the class. So I took the instance, I went in and pulled out an internal variable, and then I did a for loop that walked along it, printing those out. Is that a good idea? Is that a not-so-good idea? Is that a really bad idea? Don't you hate professors who ask questions at 10 o'clock in the morning?

Well, I'm going to suggest that it's not a great idea. And here's why. I'm reaching into an instance and pulling out an internal variable and doing something with it. A much cleaner way of dealing with that is to build an interface, to build a function, a method, that belongs to that instance, that would give me back the pieces I want. And the reason I'd like that is then I don't have to worry about, if I change my mind in terms of how I represent things internally, all I have to do is think about that interface function. Right

So the last thing I want to show you here in this example is let's look, in fact, right here. I've added a method called allStudents. But it's a little different, I think, than things you've probably seen before.

So what's the idea? I want allStudents. To be a function, or a method. When I call that, I want it to give me back all of the students. Now, I could have had it just give me back the list, but I might want to have the method give me back not just all the students, but maybe some students, like only the undergraduates in the class. And in fact, right below there, I've got a method that does that.

Let's look at this one. It says, I want to get back the undergraduates in the class. And what's it do? It's going to do a little counting. It's going to set index to 0. And then it's going to run through a while loop where it basically takes each student and checks its type. This is something you've seen before, taking a loop and walking along, taking each element of that list and saying, if it's an undergraduate, I'm going to give you back that student, and I'm incrementing index at the bottom.

Here's the funky thing. I've used a word called yield. And that's different than you've seen before. I'm not directly accumulating a list. You don't see anywhere where I'm

building up a list here. So yield is an example of a form that you're going to want to use called a generator.

Now a generator is a little bit like a return, but with one big exception. If I had a return in that method somewhere, when it found the first thing, it's going to give me back the value and it's done. And so any information it stored about where it was in that computation-- the phrase we use is it gets popped off the stack frame.

Yield is a little different. And it's different because a generator-- I'm going to write this down carefully. A generator is a function that remembers the point in the body where it was. It remembers the point in the function body where it last returned, plus all the local variables.

So what does this mean? It says, it's going to keep track of what was the state when I did the computation, so that if I call it a second time, it goes right back to that state and continues the computation. If I called it a third time, it goes right back to that state and gives you a continuation on the computation.

So in fact, what yield does, what a generator does, is what you would do what you normally walk through a list. But it gives you some control. So it gives you a way of actually deciding, how do I want to create something I could use in a for list. And if we go back and look at it, you can see it right there.

The method definition for UGs, undergraduates-- if I call it, it is basically going to walk through, one at a time, each element in the underlying representation and generate for me the next element of the thing I want to use. So let me show you a couple of examples of using it. In this case, I've buried inside of a for or a while loop, but if I go over here, I should be able to do things like, are all the students in SixHundred? And notice the open/close paren, because that's a method. I need to call it.

Now you're going, whoop-de-do. That's just like the print thing I did before. But the difference here is I'm controlling now how I get access to those. And to show you just a slightly different way of doing this, I could now do things like say the following.

For all the students in SixHundred, and I'm going to take all the students again in SixHundred. So notice what I'm doing, two loops. And I'm going to create something that basically keeps track of where I am in each one of those loops using that yield function.

So take a look at what it printed out. That outer loop is using the generator to generate each element of the list. It hangs onto that, so that the inner loop can walk through again, a generator going through each element of that list. And I'm printing out students squared -- first student times all other students, second student times all other students, third student times all other students.

And of course the other thing I can do is I can do the same thing-- sorry about that-- for s in SixHundred undergrads. And in this case, it's walking through that internal representation, generating each element that satisfies the constraints.

OK. It's a little esoteric, but you're going to find it really useful. And it's the last piece, I think, of syntax you're going to see for a while in terms of the language. You could do this by just manipulating the list directly. But the idea of a generator you're going to find really useful when you want to control access to a collection. Really think of the generator as something that keeps track of the state of the computation, so it can go back and pick it up whenever it needs it.

OK. That wraps up that piece of what Professor Guttag was doing. So I want to now switch to a completely different topic, which we're going to do today, we're going to do Thursday, and it may well carry into after spring break.

The second topic deals with, how do we go about building computational models to solve real problems. And I want to start by putting up here a comparison. For much of the history of science, people focused on trying to find analytic methods. Big word, what does it mean?

Well, an analytic model is something that lets you precisely predict the behavior of a system, just based on some initial conditions and a set of parameters. So this is a mathematical function, if you like. You can predict behavior given some initial

conditions and some parameters. Newtonian physics, 8.01-- analytic models. Spring constants-- analytic models. They're things that let you predict what a system's going to do.

For, I don't know, several centuries, this was basically what science did. It was really useful. It lead to things like calculus and to things like probability theory. It lead to basically understanding the macroscopic physical world. So it's the stuff you saw in high school.

That's nice. But it doesn't always work. And indeed, as the 20th century progressed, one of the things we saw was that there are places where this doesn't really satisfy what you need, and you're better off with simulation methods, which is what we're going to talk about here.

So what do I mean by simulation methods? First of all, let me talk about why do we want to get a simulation method. The idea was that there are going to be some places where it's really hard to build a model. So in fact, places where this comes up-- sometimes we have systems that are not mathematically tractable.

Again, a highfalutin word. What does it mean? It says, there are some systems where it's really hard to build a physical model that exactly predicts what's going to happen. Think weather forecasting. It doesn't work well. Actually, in Boston, it's easy. Just assume it's going to snow, and you're fine. All right. That's one reason why we want to move towards simulation models.

Here's a second reason why we'd like to move towards simulation models. There's times when, in fact, as things get more complex, we're better off just successively refining a series of simulations. In other words, rather than spending the effort trying to build a very detailed predictive model, analytic model, just run a simulation that gives us an insight to what the problem's doing. We'll refine the simulation, and we can keep going on like that.

The third reason you might like to do it is it's often easier to extract useful intermediate results from a simulation than it is to try and build a detailed analytic

model.

And of course, the last one is pretty obvious. The advent of computers made this possible. I want to just give you an example of this. Put yourself back in the 1700s and think about how you would do a computation. You have two choices. You can crank it out by hand. Good luck. Imagine how detailed a model you could actually do that way. Or you could build really precise, detailed mechanical models.

And in fact, there are wonderful collections of these from people like Leonardo. For example, people that wanted to understand the solar system built what were called mechanical orreries, which were these very complex pieces of clockwork that allowed you to try and predict the motions of the planets.

That's a really expensive way to try and do a simulation. So computation suddenly makes things change a lot.

Now just to give you a sense of this, I did the following experiment this morning, to see how important simulation is. And I haven't really said what simulation means. So simulation means giving me an estimate, rather than a prediction. Giving me a sense of what might happen to a system under certain conditions and doing that multiple times, actually running, if you like, a model of the system rather than trying to predict exactly what's going to happen.

So I did the following experiment this morning. I went to Google-- I could have gone to Bing-- and I typed in "finance simulation." I got 5 million hits, most of them probably wrong by the way, but that's OK. I typed in "biology simulation." I got 11 million hits. I typed in "physics simulation." I got 15 million hits. And then finally, I typed in "game simulation." How many hits do you figure I got? North or south of 15 million? North. Yeah. 50 million hits.

OK, so why should you care? A, it's fun to do. B, it's really a common tool that we want to use here.

So let's think about what a simulation would look like, and then we're going to start building one. So the idea of a simulation, then, is-- what we're trying to do is we

want to build a model with the following property. It's going to give useful information about the behavior of a system. Boy, there's a statement that has no content to it, right? Let me tell you what I mean by that.

I'm going to compare this to an analytic model. An analytic model would exactly predict what this system's going to do. With a simulation, we want to build a model that says, if I give you some sense of the state of the system, it will give me some information about how that system is going to behave. It may not be exactly right, But it's going to give me some simulation of it.

Another way of thinking about it is it's going to give me an approximation to reality. And another way of saying it is simulation models are descriptive, not prescriptive.

So what does that mean? An analytic model is prescriptive. You're doing 8.01 problems. You type in the definition of the parameters of the problem, and it would tell you, at least to the accuracy of the computer, exactly what's going to happen.

With the simulation, it says, given a particular scenario, I can give you a good guess of what's going to happen. But it might actually be the case, by the way, that for exactly the same scenario, I run the simulation multiple times and I might get slightly different answers. Because maybe the world, as we said over here, is something that we can't model exactly mathematically. So we want to be able to have that ability, to sort of go back and forth.

Now probably the easiest way to think about this is let's look at a model and a simulation. So the idea here is I want to build simulations. I want to control the fact then I may not get a precise answer. I might not get the same answer every time. But if I do enough simulations of a circumstance, I can get some good sense which I can refine of how this object's actually going to behave.

So here's the example I'm going to start with. I'm going to go back, again, a couple hundred years. In 1827, a Scottish botanist named Robert Brown observed that a pollen particle suspended in water seemed to just float at random. You've probably heard this term. It's called Brownian motion, named after Robert Brown. He had no

plausible explanation for this, by the way. He just observed it. And he made no attempt to model it mathematically, which kind of makes sense. That's 1827.

The first really clear mathematical model of Brownian motion didn't come around until 1900. A guy named Louis Bachelier had a doctoral thesis called "The Theory of Speculation." Part of the problem for Bachelier, though, was that he didn't model Brownian in pollen particles. He did it in finance markets. And that was considered, at least at the time, completely disreputable. And so nobody paid any attention to his thesis.

That's not going to happen to your thesis. Your thesis is going to be reputable when you get done with this place.

So it took eighty years to get to that point. Unfortunately for poor Bachelier, five years later, another person came along and actually built the kind of model that introduced this sort of-- what's called stochastic thinking into the world of physics. This was a model that was almost exactly the same as Bachelier's, but it was used to confirm the existence of atoms.

Anybody know who did that model in 1905? No physics buffs here. If I told you he was born in Switzerland, would that help? A minor guy named Albert Einstein. So that was one of the first things Einstein did, was he actually built the first really good model of Brownian motion. And that allowed, in fact, this kind of thinking to go into real-world problems.

So Brownian motion is an example of a tool we're going to use a lot. I'm probably hiding that below the screen where you can't see it. It's an example of what we call a random walk. Random walks you're going to see all over the place. They're an incredibly useful way of building a simulation.

And the essential idea of a random walk is, if I've got a system of interacting objects-- could be pollen particles-- I want to model what happens in that system under the assumption that each one of those things is going to move at each time step under some random distribution. It's going to move in a particular direction. I

want to model what the overall system does.

OK, let me give you some examples of where this is really useful. It's really useful in modeling physical processes. Well, we're going to start with pollen particles in air. But you could think about any kind of particle in water. You could think about any kind of air particle in a larger fluid. Ah, weather. Modeling weather-- if we could really model the motion of all those molecules-- is just a really large random walk.

Random walks are really useful in understanding biological processes. For example, the kinetics of displacement of RNA from heteroduplexes of DNA is a great example of a random walk. And in fact, people interested in bioinformatics or computational biology will see random walks used all the time to try and to understand the displacements of things.

It's really useful in social processes. Movement of the stock market is definitely a random walk, except for the day when the markets are all crashing for unfortunate reasons. So it's something we want to use a lot.

In the example I'm going to use to motivate a random walk, we're going to build a simulation. Here is what was the traditional view of a random walk. So here's the motivation. Excuse me a second. Let's take a drunken university student. Not an Institute student, a University student. So this is a Harvard student, not an MIT student, because I know you're all well-behaved. At least smile at me when I make these bad jokes. Thank you.

All right. You've got a drunken student. They're out on a big field. And they start off in the middle of the field, and every second, this student can take one step in one of the four cardinal directions. So north, south, east, west. After 1,000 seconds-- after 1,000 steps-- how far away is that student from where he or she started out?

So I need some help here. Guesses. 1,000 steps. How far away after 1,000 steps is that student-- we'll make it a he. From where he started out. Yeah.

AUDIENCE: It depends on the probability of what direction you're going.

PROFESSOR: OK, so let's assume that the steps were equally likely. The four steps-- north, south, east, west-- they're all equally probable.

AUDIENCE: [INTERPOSING VOICES]

PROFESSOR: OK. So the suggestion over here is, where he started. So 0 distance away. That's not a bad guess, right? You're just going in different directions. It's equally likely you end up where you started. So one possibility is 0. You end up back where you started. Yeah.

AUDIENCE: Maybe 150 or so?

PROFESSOR: Well, I'm going to take bets here. I've got 150. It's an interesting number. Not bad. Any other guesses. This side of the crowd. I don't want to just do the right side. I'm very liberal. I want the left side of the crowd over here. Somebody over on this side. Help me out. Any other guesses? I've got 0, and I've got 150 steps.

Boy, I get no eye contact when I do this. This is great. Yes, please.

AUDIENCE: 500 root 2 steps?

PROFESSOR: 500 root 2. Can I drop the root 2 for second? The 500 is not a bad guess, right? Because you're about half the distance away. The root 2 we'll come back to in a second. Keep that in mind. All right. 500. So we've got 0, we've got 150, we've got 500.

Well, gee, that's the whole point of building the simulation. Let's see if we can figure out what might happen here. Ok. And I'm going to build the simulation in a second. But before I do that, one of the good things to do is to try and build a simple model that we could use to get an intuition about what's going to happen. So I'm going to start with a very simple model here.

There's my field. The guys who mow the lawn at Fenway have been over and very nicely put x- and y-axes onto this field. And let's assume the students starts off there. We'll just call that the origin. It doesn't matter what it is, but that's where the students starts off.

And what I want to do is I want to get a very rough sense of how far away the student might go after a few steps. We won't be able to go very far, but enough steps that we can start estimating this.

So they start there. After one step, there are the places the student could end up. Equally likely. So after one step, if I want to build a distance associated probability, there's probability 1 they're 1 unit away, no matter which direction they went.

OK, let's take this one here. Let's assume the student went east. In fact, they're all going to be roughly the same. And the second step, where can the student go? He can go there, he can go there, he can go there, or he could go there. Right?

So what are the distances here? Well, in one out of four cases, he's at a distance 0, which would be back to where we started, which is that original pretty good guess. If he goes there or he goes there, those are both what distance away? Root 2, right? So it could be root 2 away, 2 out of 4 times. And if we went east again, we're 2 units away. Again, with probability 1 in 4.

I'm cheating slightly here, in the sense that I should do the same things if the first step was north, south, or west, but they're all symmetric to this. So it'll all come out the same way. So there is what happens after two steps.

Now let's do one more step. It's going to get a little messy here. But suppose I want to go three steps. About three possibilities. Well, I've got three possibilities. If I'm in this case, which is the 0 case, I'm here, in which case no matter which step I take, I'm 1 unit away. So this will say, distance 1. And the probability of being here was 1/4, so the probability again is still 1/4 that one unit away.

Let's take one of these, the root 2 ones. That says I'm either here or I'm there. They're going to be symmetric. If I'm here, where can I go? I've got possibilities going to those. So these two are 1 unit away. So half the time I'm 1 unit away, and it was 1/2 that got me there. Did I do that right? Yes. That's also with 1/4. That was getting if I went here or I went there.

If I'm up here or I'm over there, that's root 5 away. So let me put that over here. So I've got a probability of 1/2 getting there, and a probability of 1/2 from here of getting that, so that's also with probability of 1/4.

And then what's the one case I have left? The case I have left is I went all the way over to here, two steps over, in which case I'm going to go what? There, there, there, or there. If I just pull these together-- I'm just going to do them for you. That says, I have a probability of being 1 away with probability 1/16, root 5 away with probability 1/8 -- and I'm running out of room here -- and 3 units away with probability of 1/16.

Don't sweat the details. What you can see is I'm just building this up. What I want you to see is what do the distances look like? After one step, I'm one unit away. After two steps, that was this. You can kind of do the math in your head. What's the expected distance away? What's the average distance I'd be away? Well, it's 0 times 1/4, plus root 2 times 1/2, plus 2 times 1/4. So that's 1.4, that's 2.8, and that's 4.8 divided by 4. So it's about 1.2.

Now I'm sure you can do the math for three steps in your head, right? Yeah, right. But if I do that, if I add those things up-- you can go do this yourself-- what you'll find is that this is about 1.4 or 1.5. Let me make it 1.4. It's probably a little bit closer.

All right. Why was I doing that? This is something you want to do as you start building a simulation, which is do a little bit of a calculation to get a sense of what you expect to happen here. So what conclusion could I draw from that?

Admittedly on very small numbers, it looks like the more steps the Harvard student takes, on average the further away they are from the starting point. So your 0 was a great guess, but it's probably not going to work here. And we want to see why.

On the other hand, 500 that looks-- well, maybe we're still OK. Three steps, half of 3 would be 1 and 1/2. So maybe the 500's OK.

Let's see what happens if we do this. So what I want to do is see how I could build a set of classes that would let me build this simulation. And part of the design process

here is I want to try and invent classes that correspond to the types of things I expect to see happening.

So in this case, what do I have? I need to model a drunk, I need to model a field, which is where the drunk's wandering around in. And I'll need a third thing, which may not be obvious immediately. But the third thing I'm going to need-- so let me just put the pieces up. I need a drunk, I need a field, and I need to keep track of where the drunk is in the field. So I'm going to pull out another class I'm going to call a location, that's going to tell me where the drunk actually is.

OK. With that in mind, I'm going to show you some code. I'm going to walk through it reasonably carefully, just to let you see what it looks like. So there is my class definition for a location. Now at this stage in the course, hopefully you can look at that and already see the main pieces of it.

So what do we have in here? Well, we going to have an initialization. You can kind of see that right there. When I create a Location, I'm giving it an x and y. I'm going to make an assumption, which is they're floats. I'll come back to that in a second. And that's just going to store them away in some internal variables. That's things you've done before. So self.x gets that value, self.y gets that value. And of course, I could get out the values right there.

I need to know where the drunk goes. So one of the things I'd like to do is to say, given the current location of the drunk, I might like to know how far away is that from some other location, like the place the drunk started. So that last method down here, distance from-- notice what it's going to do.

It says, if you give me another, which is another location-- remember, self will point to my instance-- what can I do? Well, I can just do Pythagoras's theorem to figure out how far away am I. I get the x- and y-coordinates of the other location. I get my own x- and y-coordinates. And then I just take those distances squared and take the square root of them. So it's just giving me distance away, which is pretty nice.

There's one other method here which is a little funky. This is a design choice. And

that method is the one up here called move. So the background to this is going to have a drunk at a location. one of the things I will want is to be able to change location.

So move says, if you give me a change in x and a change in y-- I'm going to assume they're floats-- I will give you back the new location, which is to go from my x and y by some amount. It could be in the east direction. It could be in the north direction. But notice what it returns. It returns a new instance of a Location.

OK. I made a couple of assumptions which are worth thinking about here. So one assumption is I'm assuming that this is a 2D world. You can't elevate. You can't rise up. There's no change in altitude. That kind of makes sense.

The second assumption I made was I said, I want to build it in so that things like delta-x and delta-y are floats. What is that doing? Well, that's saying, I don't want to restrict myself to just moves that are only in the cardinal directions. I'm going to start there, but initially, I'd like to have the ability that the move could be along a diagonal, or it could be some partial step. So notice I'm making a choice there that is kind of nice.

All right. That gives me locations. And that's a pretty straightforward class. So I've got locations. That's always a good start. I have this.

Now again, I want to model a drunk wandering around in a field. So the second thing I need is I need to say, what's a field going to be? And I want to show you that definition, which is right here. And a field is basically going to be something that maps drunks to locations. So a field is just going to be a collection of drunks, keeping track of where they are.

So let's look at what that definition says. When I create a field, right up here at the top it sets up a variable inside of that instance, just called drunks. And we're going to use a dictionary. We could use other choices, but the dictionary is going to be nice, because it's going to let us keep track of the drunks. I can add a drunk to the field.

So notice what it says. It says, given I've got a field, I'm assuming I have a drunk and a location, because the drunk's got to be in a particular location. Notice what it does. First of all, it checks to make sure that I don't already have this person in my collection. So that first check there says, if this particular drunk is already in my dictionary, I complain. You can be drunk enough to see double, but you can't be drunk enough to be double. Another bad joke right off the back wall. Great.

OK, so what else do I want? Otherwise, notice what I'm going to do here. I want to make sure you can see these things fairly cleanly. If I'm going to add a drunk, what do I do? Into the dictionary, under that label of drunk, I'm going to insert a value corresponding with that label. And the values going to be the location. So my representation here is now a dictionary of drunks and where they are. And that's going to be handy as I want to use this. And again, if I want to get out the location, well, I just return exactly that.

The final piece is, when I go to move the drunk, I've got some tests here to make sure that I actually have a drunk. But otherwise, I'm going to ask the drunk to take a step right there. We have to say what a drunk does. That's going to give me back a change in x and a change in y. And then notice the funky little call below here.

This says what? The right-hand side says, I get the dictionary of drunks. I go in and index into it to get out the actual drunk. That's this first part right here. That gives me back what? It gives me an instance. So the dot says, for that particular drunk, get its move method, which was that call, and have it move by that amount. And that returns for me then a location which I can store back into the list.

All right. I made some assumptions about Field. They're worth pointing out. No constraints on locations. How big is this field? As big as you want. So I didn't build in a real physical representation of the limits of the field with every location represented. I just said, it's just a collection of drunks. So there's no limits on how far the drunk can go.

The second assumption I made here is I can have multiple drunks. And we're going to have a simulation of that in a while. You can imagine it's going to be fun watching

them collide with each other. All right.

And the third thing is, it says nothing about the patterns in which the drunk moves. So this is a design to keep this clean. It says nothing about how the drunks move. It simply says, give me a change in x and y. I'll change the location.

The last piece is we're going to capture how the drunk moves in a drunk. And there's the definition of the drunk. What does that say to do?

Well, we're going to give it a name, to start it off. And the only method it has is that thing that's going to do a takeStep. And here's where we're going to build in the choices about how they can move, what you asked about earlier. Notice what I've done there.

In this case, I've said, I'm going to pick at random from this set, stepChoices. And that's where I'm building in the idea that it's only going to move plus or minus 1 in the x and y direction-- north, south, east, west. I could clearly change that. I'm also building in here that it's equally likely, because random picks one of those equally likely. I might decide to change that. And I can certainly do that if I want to move to other directions. But right now, that's where I'm going to build it in.

OK. So now we're ready to try the simulation. We've got what? We've got location. That's just representing a place. We've got fields. That just lets us map drunks to locations. And then we've got the ability to move the drunk around. So I can now think about making this actually move around.

So let me show you a quick example. Let's create a drunk. I'm picking the obvious person. I'm going to create a location. We'll call the origin just the spot at 0, 0. And I need to create a field which I can do. And then I'm going to add Homer to this field at the origin.

And now the last thing I want to do is I want to have Homer wander around. I want to see how far he goes. So I'm going to build a little function over here-- you can see right there-- called walk, which takes the field, takes a drunk in the field, and says how many steps do I want him to stagger around with?

And what's it going to do? I'm going to set the start location to be where the drunk was initially. It might be at the origin or it might be somewhere else. And then I'm just going to run through that number of steps. So range is just saying, take the number of steps, just moving the drunk. And then I'll return how far away he is.

So if I do that here, I can do walk with that field, with my drunk, Homer. And let's take 10 steps. Hmm. That's interesting. Let's try it again. I was getting worried there for second. So we're getting an interesting range of values.

So here's the last thing I'd like to do. Let's actually do a simulation of this. And I'm going to do this reasonably quickly. simWalks right here is simply going to do what I just did, but multiple times. That is, it's going to take a drunk, a field, some number of steps, and it's going to run a bunch of trials, however many trials I want to get.

So I do what I just did over here, and collect those together. And then I can just print out some statistics about how far away did the drunk get as I do that. And that's what drunk test is going to do. We're going to do it for 10 steps, 100 steps, 1,000 steps, and so on. So let me simply run that.

I run drunkTest. And we'll do it with 10 trials each. OK. Does that look right? Does that look good? You can tell this is a trick question.

So what should you be looking for there? Look at it. It says, if I take 10 steps, on average I'm about 2 and 1/2 steps away. If I take 100 steps, on average I'm less than two steps away. 1,000 steps, I'm about two steps away. 10,000 steps, I'm about two steps away. 100,000 steps, on average I'm about two steps away.

Wait a minute. Didn't we say that it looked like, over on that board there, that as I increased the number of steps, I got a little bit further away? Hmm.

So I've done this deliberately. This suggests what? The first key point, think about what your simulation should return. What are good examples to think about? Because this doesn't look right.

OK. So here's how I could test it. Why don't I try a simulation on values for which I know the answer? So I'm going to go back in here, and right here, I'm going to replace this with-- we're either going to take zero steps or one step. Let's run that.

Oh, don't do that to me. Oh, yes, I do have an error. Thank you. Get rid of it right there. Almost out of time, so this is perfect. And now let's try drunkStep again.

Wow. If I take a random walk taking no steps, on average I end up 2 and 1/2 units away. I teleported. That's pretty amazing. And with perfect timing, we'll figure out what's wrong with my simulation next time. Thanks.