ProgrammingAbstractions-Lecture17

**Instructor (Julie Zelenski)**:Hey there. All right. So thank you very much for your duty to the CS106B midterm exam last night. An unusual event by all accounts, but we did all manage to get it done and have that past us. We'll be grading that this week, but given the staff member's scheduling, we're actually going to do most of the grading this weekend. So it should be a little while before you hear back from us on the grades for that, but never fear. We've got them and we're looking at them.

I look through them last night and am already getting good sense that there's a lot of things you guys know how to do well, which is very pleasing. So we'll have more reports on that later.

Assignment five, which we gave out as your parting gift for coming to the midterm – if you didn't pick it up, there's copies of the handout in the back, and the sources are up on the web.

Just to give you a little sense of the assignment, it's actually a smaller assignment, right? It's one that I think most of you will be able to complete in less than ten hours. And by design, kinda giving you something to kind of practice a little bit with sorting but give you a little bit of a break when everything else is getting crazy in the rest of your classes. There are two parts to it. One is kind of identifying some mystery source by doing some experimentation and kind of a scientific method on that. And then the second one is running your own template sort based on a new sorting algorithm that you go and explore and learn something about that. And that has a pretty different setup for what we're expecting to do, which is actually we are encouraging you to go look at other books and web resources and talk to people and think about things and come up with an algorithm that you'd like to learn a little bit more about and implement. You can make this task extremely easy. I don't recommend it, but it turns out you can go and find somebody who's actually almost done the job for you and just copy and paste their code. That is allowable by the bounds of the assignment to be clear, but we are hoping that you actually put a little more thought and effort and learn a little bit more from the process of exploring and thinking about a goal – like, having a goal. I'd like the kinda sort that does this well or would be useful in this situation and learn more about it. But that said, you are able to use a lot of resources. You need to be clear about citing your sources, right? That's the most important thing in terms of academic integrity – always – in this class and everything you do, both in the academic and the professional world – is that when you are making use of someone else's work as part of something you're producing – is to be clear that you give credit where credit is due. So you can make it into a bigger project if you've got the time and energy. You can also take it a little bit easy gearing up. There will be two more assignments to go out after this one that both are pretty substantial. So – in terms of planning this – maybe the time to take it easy and then get ready for the things coming. Today, I'm gonna finish the little bit about template functions that I had left hanging on Friday's lecture, and then we're gonna start talking about OOP and class design implementation. Pretty much from here to the end, we're gonna do just a lot of class implementation. We'll be talking about, "Well, how is it that vector works? How is

it that stack works? And queue works? And map works? And set works? How is it that we build classes up from the inside that have useful properties from the outside?" And there'll be a lot of neat data structures to look at and big old trade-offs to talk about pointers to be used, link lists and more dynamic structures and a great opportunity to kind of put into practice a lot of the hardcore stuff. The reading that we're at is chapter eight. This is when we talk about objects in classes today – kind of the general background material for that, and then we'll talk about class templates starting on Wednesday. Wednesday – Friday – whenever our next lecture is. And then we'll do a lot of class templates from here for several weeks before we're done. All right. Anything on the administration doc? Question?

**Student:**Just a quick thing. Could you give us, like, a general timeline as to when the next [inaudible]. [Crosstalk]

**Instructor (Julie Zelenski)**:So typically – the point is that one's gonna go out Monday and be due on Wednesday, and then one's gonna go out Wednesday and be due Friday. And there's a week and a few days after it. So the size of boggles space for doing the work. It's not Friday. It's the Friday of dead week.

Anything else you wanna know?

Okay. So what did you think of the midterm? Thumbs up? Thumbs down? Loved it, hated it – two thumbs up. Oh, a thumb sideways. All right. Thumbs up, you get to talk first. What was great about it?

**Student:**I just loved that big O.

**Instructor (Julie Zelenski)**:You loved that big O. You couldn't have enough of that big O. Okay. All right. Two thumbs up for the big O.

Anybody else who said thumbs up who wants to engage us with their positive memories of the exciting event that was the midterm? There's only one of those. Who wants to say?

**Student:**No link lists.

**Instructor (Julie Zelenski)**:No link lists. Okay. Okay. You can call that a positive. That seemed to please a certain number of people. And those of you who were kind of a little more iffy? Where was the iffyness came in? Where did the iffyness come in?

**Student:**[Inaudible].

**Instructor (Julie Zelenski)**:[Inaudible] the time. Within looking at them, everybody did have – not everybody. That's certainly no true. But almost all the ones that I was getting through had kind of stuff to say about every problem. But that is an indication you had all the time to say what you wanted to say. But there was at least some sense that I felt like,

from what I could tell, people had been able to allocate their time enough to get to a little bit of everything.

Somebody with their thumb way down? Wanna tell me what was trouble about that? Anything that surprised you or –?

**Student:**[Inaudible].

**Instructor (Julie Zelenski)**:I'm a very kinda formulaic exam writer on purpose. So in fact – hopefully you felt like the comparison to the practice was pretty expected – that you're like, "Oh. These are either questions that looked a lot like this on the practice."

We moved stuff around and twisted up a little bit, but there shouldn't be a lot of surprises in terms of what I told you to expect and what there is to expect. In fact, the match to the assignments, also, I think is an important thing I try to maintain.

So that said, hopefully it wasn't too unexpected. Okay.

Well, good. I'll finish – this is the last slide that we had shown on Friday. And I had just blasted it up there and then didn't really talk about it. So I'm gonna talk about it now and then just finish the final adjustment we're gonna make to this, right? So this is – the selection sort code that we have worked to templatize and do a template function, and now it will sort vectors of unknown type. And the change we just made – the one that's highlighted here in blue – was rather than directly taking V sub J, V sub X and comparing them using a less than – is that we're using a call back function. Call back functions – the parameter CMP, or compare there, takes two parameters that are of the client's type, returns an N – which is their ordering – zero, negative or positive, and then, instead of making a direct expression involving less than, we're evoking the client's call back on the two things that we need to compare and saying if that result is less than zero – so V sub J precedes the V of the main index. So it's actually smaller than the main element we've seen so far. Then we update our main index to record J as the place we've seen it. And then it goes to the rest of the code normally swapping that small thing out to the front and then going back around for multiple iterations to keep doing that. With this change, right? We have now kind of generalized this fully in a way that you can sort vectors or strings or vectors of coordinate structures or students or vectors of sets of things as long as the client supplies the appropriate comparison function that explains how do you want those things ordered? Which ones go in the front? Which ones go in the back? It's up to the client to say by giving us that call back. And so a client could use this, for example, to sort coordinates – something that has kind of an X and a Y field by deciding that, "Well, if – first, from the base of the X coordinate, the first one's X coordinate precedes the other, than it's considered less than." You have to make up an ordering, right? That represents what you want, and in this case, I'm gonna say that, "Well, first sort on the X dimension, moving smaller values of X to the front. And where they are tied in the X dimension, then use the Y to break ties here – looking at the Y fields – if it has gotten through those first cases of X compared between C1 and C2. And if all of those things have kind of not been through, then we know that, at this point, X

and Y are exactly equal. And so we have two elements that are the same." And so, given a vector of chord, right? I would invoke sort passing that vector and then the matching comparison function that lets the sorting routine know how to take two chords and decide which goes in front. Yes, sir.

**Student:**Where we use the type-name template –

**Instructor (Julie Zelenski)**:Um hm.

**Student:**Is type name a special word?

**Instructor (Julie Zelenski)**:It is. So the type name is a C++ key word that says, "This template depends on a type name."

There are actually other things you can templatize on. We won't see that this quarter, but I'll just kind of leave it as a – things – that there are other things that you may actually have as kind of parameters to the pattern that you're using. The one we're gonna see is when you are actually using a type as a placeholder. And so type name means there's a type in the body that follows, right? There will be usage of the key word of a – our chosen word, type, as a placeholder for something that is supposed to be a type name.

The other word there – they type, right? That was our choice. That could be T or E or F or my type or whatever it is I wanted. That was just a name we got to pick. That my core compare – and the last thing I'm gonna do with this is just make it a little bit more convenient that – given the form I have right now, it says that really, "Oh, there's gonna be two parameters. You're gonna give it a vector, and you're gonna give it a comparison call back function." Before I went through all this trouble to make it have a comparison function, it used to actually just work by default for things like ents and doubles and strings that actually already operated correctly with less than. And so what I'm gonna do is I'm gonna – having made it generalizable, I'm also gonna go back and add in a default behavior which says, "If you don't care, otherwise, and the default less than would work for you, then let's go ahead and use that unless otherwise indicated." So that, kinda, we got back the original behavior we wanted, which is you get to say, "Sort a vector of numbers and have it do the right thing without having to go to the trouble of building a compare ent function to pass to it." So what we will do is we have an operator compare – I'll show you what it looks like here – that there is another header file in the 106B collection called CMPfunction.H. It looks like this. It is a template function itself. It's called operator compare that takes two type things, and then it uses equals, equals and less than to decide whether to return a zero, a negative one or a one. And so operator compare will only work when instantiated for types for which the built in equals, equals and less than are defined and make sense. Now – and that includes all the built – the regular prototypes and double and whatnot, and also includes the string class and then potentially other classes, right? That you may know of that actually have behaviors that have implemented those operators. It will not work on things like struts or vectors or other things that don't have the behavior for this. And it's not intended for those. It's kinda just a pattern for which you can build this – the comparison function needed for the

form by sort. And so then I change the prototype here of sort. It takes a vector of that type. It takes a comparison function. It takes these two things, and then I said, "A deep-hole argument for that is to use operator compare." And so when somebody invokes sort not passing the second argument, then what that will cause the code to do is say, "Okay. Well, they didn't pass it. We need to use the default expression over here, and then operator compare will be instantiated for whatever the type is." So if it was string, then it will build operator compare that operates on strings. If they were ent or double or anything that has a meaningful way to apply the built-in operators if we use that – if I still called sort in this case, past let's say, vector of chords – if I went back to this picture – and I forgot this argument, right? I will get a compiler. And a complier will come from trying to instantiate an operator compare that works on chord where it will get to that line about equals, equals and less than, and say, "You're comparing two structure types, and that doesn't make sense to me." So in this case, the client has the choice of specifying it when they intend it to control that behavior or they need to control that behavior. They can also leave it off in the cases where that default of operator compare will do what they wanted anyway. So I have an example here at the end that says, "If I wanted to sort an array of numbers, I won't need to pass a second argument in the case where I just want to sort it in ordinary increasing order." If I wanted, for example, to sort it in inverted order – I want the largest value to come to the front – that I don't need to run a new sorting routine that then goes and looks for the max and pulls it to the front. I just need to trick selection sort here into believing that the larger numbers go in front of the smaller ones. So I basically wrote an inverted compare function that if A is less than B, it returns one, which basically says, "Well, B goes before A in the ordering I want, which is the larger one should precede it in the output, and similarly for the other case, and then returning zero when they're the same." So this gives us control when we want it in the case of the primitives as well as control when we need it for those types for which the built-ins don't have defined behavior for it. So this makes this, like, the end-all, be-all sorting template. Everything you could want in one package. You will be running something that looks just like this for the second part of the assignment this week, and then what will differ is – well how does it do its work? What strategy is it using to get things in order? But the overall design of it is to build this general purpose, could be used for everything, has a client call back option, also has a default used that makes for very nice, convenient as a client. Any questions about that? It is your life.

**Student:**I have a question on the [inaudible].

**Instructor (Julie Zelenski)**:Yeah.

**Student:**Right below the void line –

**Instructor (Julie Zelenski)**:Uh huh.

**Student:**So what does that do?

**Instructor (Julie Zelenski)**:So the line that's underneath this one?

**Student:**Yeah.

**Instructor (Julie Zelenski)**:This one here?

So this is the one that says, “The second parameter – the first parameter’s a vector of type by reference. The second – it says the second parameter is a comparison function that takes two type arguments and returns it in, and the default assignment to that parameter, if you have not specified it, will be to use operator compare.”

That’s got a lot of things mashed in there. The syntax for the passing a function as a parameter is a little bit goopy in C++. It kinda includes what looks like a full prototype because that describes for the compiler, “Well, what kind of functions are they?”

And they have to have this parameters in this order – this return type. So we actually kind of have to give this full information about what the shape of such a function is so that it can match it. It can’t just take any function, right? It can’t pass it get line or convert to lower case – like, not just any function is good. I need to give a description of what kind of functions are right, and those are the kind of functions that take two type arguments and return an ent. And that’s what – we’re helping the compiler out there with all that syntax.

**Student:**How will it know if you have a comparison function based on that call?

**Instructor (Julie Zelenski)**:So this isn’t the call, right? This is the function I defined. So when I make a call to sort – so if I look at this one – it’s all a matter of, “Do I pass one argument or two?”

**Student:**All right.

**Instructor (Julie Zelenski)**:It’s just a default argument. If I don’t specify it, then it will use the default value, just like any of the default argument things we know about, like find on a string, which if you don’t say, it will start at position zero doing a search. If you do specify, then it uses your index instead.

All right?

Yeah. Let me move on.

So in your 106A class, right? You did a lot of things in the genre of object-oriented programming. This class is not really an object-oriented programming class. It’s a class of – that’s, in many ways, very procedural, but we use a lot of objects. And what I want to do, just for a minute here, is just kind of reiterate the advantages of using objects in your code – what it is that they provide in terms of structuring and engineering and working to a solution. And then we’re gonna go on and start talking about, “Well, we used a lot of objects. What does the other side look like? What is it like to implement an object starting from scratch? What pieces need to be done, and how does that work?

And so the idea of just object-oriented programming is actually – it's a relatively new concept in computer programming, right? It only dates back to the '80s, and the realization, right? That programs at that time – we were operating on a lot of data and moving around employees, or you're managing this calendar, which has these events that are at certain times. And that you had a lot of operations that were, obviously, intended to work on that data – like, move this event to another time or calculate whether this time perceives another – conflicts with this.

And that operations, though – the functions that manipulate that data and the data itself – were not really very tightly coupled in the kinda past. And the idea of [inaudible] used to really couple those things together – to say that, "If you have a data type like a time or a stack or a vector, that the operations that manipulate a vector are best made as part of the package of vector itself – that rather than there being a print vector function that operates somewhere else, having vector-heavy print method."

Putting something onto a stack or asking it for it's size is really an operation that should be owned by the stack – that a stack variable should be able to respond to requests to do things on your behalf through messaging. And so this idea has become kind of permeated the language design in the last decade or so to where pretty much every modern language has some facility for object orientation. Even older languages that didn't before have been updated and brought forward, right? To visual basic, right? Basic, which has been around for half a century, is now objectified, right? In these latest versions.

So [inaudible] that the leverage to the real world is actually an important part of the advantage that, when you talk about a time or a stack or an event or a message if you're doing a mail program – those things have real-world meaning. And so the idea of what they do and how they act actually has – there's a lot of leverage of what you already know to be true about those things in the real world. The notion of kind of dividing it up and saying, "Here's this abstraction of what a stack is," – so we've used stacks all along. Push and pop, right? Push and pop. What does it really do? How does it really manage stuff? Where's the memory coming from? What internal structures? It's not actually something we have to worry about. We're using it abstractly. We're saying, "It's a stack. It has push and pop behaviors."

All the other details of how it works are not something we have to worry about. So let's just focus on some of the other problems – more interesting problems than the kind of things that happen behind the scenes. They're also very tightly encapsulated. So we're not mucking with the stack. It's actually acting force as a black box with – it's kinda like one of those microwaves where it says on the back, "If you take this panel off, right? You void your warrantee." Like, I don't know how my microwave works. I never take that panel off. I hit the panels on the front, food gets hot – I'm very happy. And then things that happen in the large – as you get to building larger and larger projects, right? The idea that multiple engineers are working together and designing on different timeframes – that having things divided in these very strongly capsulated bundles makes it possible to design and test independently of your partners and then join back together and have a

much better eventual result that, if everybody's trying to write all the same code in the same places on top of each other without real discipline to it. Nice packaging for reuse

Something we've been seeing a lot if, right? Is the things that – Lexicon or stack or hue we've used in a lot of different ways – sort of a multi-purpose object. Let me tell you about what – who does what in here and what new rules we're moving into, right? We have done a lot of client use of objects. So a program like the maze program or the random-writer program is some code file containing actual code – .cpp. It messages to objects. It creates objects. It asks them to do things on its behalf. Every class that it is using, it includes the class.h – so stack.h or que.h or vector.h based on what it needs. And so this is the only role we have played so far. We've been client. We've used them, but we don't know anything about these other two things that need to get done.

In the middle, between the client and the implementation here, is something called the interface. We have looked at these header files like stack.h and que.h, and that tells you about what the class provides – what abstraction it is, what it's member functions are, what they're names are, what they're parameters are, what the usage – the correct usage of it is. And so this serves as information to the client about what you can do and what you can't do, what's legal, what's available – and it doesn't contain any inner information about how does it really work? What does it do behind the scenes? What kind of thing can you expect other than kinda what's the correct behavior expected? What we're moving into, right? Is looking at this role and this role over here on the right, which is what is that implementation side work look like? So if we have described what Lexicon is – it's a word list where you can check for the existence of a word or the match of a prefix – and it's our job now to implement that, it's like, "Well, what are we gonna do?" Now this is where we get down and dirty, right? Now it's not about the pretty abstraction. It's about making it work – making it work. Well, making it work efficiently, using appropriate data structures, making it robust, making it handle kinda all kinds of things you throw at it. So for example, the vector class – if you ask it to get something that's off the index, right? It's gonna tell you about it, right? So taking care to make sure the thing just works in all situations, is very bullet proof, is very sturdy, is very informative, right? And well designed, sort of clean, easy to use, does the right things, has the right functionalities, isn't missing anything and that implements it well.

And so in this file, the stack.cpp or the scanner.cpp, right? We have all the [inaudible] functions, so we make the things really happen. We include the class interface, too, because the interface is actually used by both people. The interface on this side tells the client what's there – what you can do. The interface tells the implementer what needs to work, right? You have offered up a method called contains prefix or contains word that is the implementation's job to make that thing do the right thing and return the right answer. So let's look at a simple class. The class I'm gonna talk you through is one of creating a time – sort of a moment in time – 2:15 – that potentially, right? You might be using, let's say, a calendaring program to decide what events you have scheduled for a particular day. You'd like to say, "What's at this time?" And then the idea of time and the things that manipulate time – it actually makes a very good object. It has a very real world analogue about what a time is. And there's a lot of behavior that seems to go with a time.

So in the .h file, which is the interface for this, will be the listing of the class. So the outer structure for this – our class time and that open [inaudible], closed [inaudible] and then a closing semicolon. This kind of models the same syntax that C++ uses for struts. When you say strut something, and then you have these fields, and you also have this closing semicolon. That semicolon, right? Is not present in Java, and it's a pretty easy thing to leave off and then get a lot of [inaudible] error messages that are a little bit goofy from it. So one thing you must wanna register as something to be attentive to. What the interface declares – so this is separate in Java – not separate in Java. I'm saying in C++ it is – is there actually really is a file that says what the class provides, and there's another class that says how it works. In Java, those are kind of one and the same – that the definition of the class and the kind of interface of it were not separated and maintained separately. There are advantages to this and disadvantages. It means there's two separate files. That means that what you give to somebody who's using your code doesn't contain any of the things you don't want them to see – how it works and how it's internally configured. But it also means that you have to keep the two in sync – just the same way any kind of separate prototyping does – is if you change the name in one place, you have to change it in both places to kind of make sure that they're always in match.

So the two main things that get declared within the class declaration or its interface are the data members – so the fields that are a part of a particular time object. And these are declared like ordinary variables in hour, minute. They can be variables of other types, string, vector and things like that as long as anything I'm using, I would have to [inaudible] include at the top. And in this case, I have introduced them under a private section. So unlike Java where every single field has it's own private or public modifier, in C++, you actually introduce a section with private [inaudible], and then everything from there on down is private until it sees something that changes that back to public. So typically, right? You'll have one big, public section, one smaller, private section – they can be in the other order – either way. I typically use this form where the public things sit at the top and the private beneath.

The other things that get declared as part of your interface is what member functions are available. What are the operations that you can manipulate a time with? And so maybe right here, I have something that allows you to change the hour or get the current hour from a time or move it forward by some amount of hours and minutes. I wanna push back that meaning by an hour and 50 minutes. I can call shift by 1:45 to do that. Something that converts time into a string format may be suitable for printing or using in a display is a two-string member function I could add in the time class as part of its public interface. And so the whole list of what I have here – and so kind of as a rough rule, right? Most of your functions will end up public because they're operations you're offering. Your data is almost always private. You don't wanna make that accessible outside of the class. And then occasionally, right? There are reasons, actually, to have some member functions that are private. They're used internally as helpers, and they're part of the strategy, but they're not something you want a client to be able to directly call. So you actually keep them in the private section to indicate they're not part of the interface a client needs to know about. You see it all the time. Any questions about what it looks like there?

So the time [inaudible] I have declared here has two data members – an integer hour and an integer minute. The mechanics of how this works is that the object is basically about the same size as the comparable strut. So it has fields for hour and minutes. It doesn't actually have a bunch of storage for the member functions. Since those are shared across all times, there's no reason that every time carry around a duplicate of that. So what it really carries is its own field – the data members that were declared in the class. And so when I say time T, what I'm getting space for – it looks a lot like a strut – something that has an hour and a minute field. By default, the initialization of this field is as it would be if they were to be declared on the stack. So if I said N to hour, N to minute, I would just get junk contest. So it doesn't set them to zero or do anything clever on our behalf. With these primitive types, it just lets them stay uninitialized. And we'll see about how to fix that in a minute, but just to know.

And then when we talk about time objects, right? Every single time object has its own hour and minute. So this time is 2:15, this time is 7:00 p.m., and the numbers that they are storing and maintaining, right? Are different for their hour and minute. So it's kinda like across all the different time objects, there is individual hour and minute fields associated with each particular time. It all kinda seems to make sense back to the 106A days where you did a lot of this. But if you have a time object – and you've already seen this, but just to mention that – yeah, you access its member functions and its field using dot notation. If you happen to have a pointer, you can use the arrow, which combines the star and the dot, setting – calling set hour, calling get hour, trying to access a field that both the member function use and the field use is dependent on if the – the feature that I'm trying to access being declared public. It was in a private section, and if you don't specify – if you forget to put any of the [inaudible] specifiers on it – by default, they were all private. And so my attempt to access any of these things that were private will result in a compiler error. So it won't let me get past that kind of mistake.

Now the message, we call that the receiver. And we talked about that before – T being the receiver to the set hour, and the assumption being that what I'm trying to do is tell this time object, right? To set the hour it is to 3:00, overriding whatever value was there before. Now what does it look like on the other side? If you are the implementer of time, what kind of things do you have to do to make time behave the way you said it would? You have a file timed out CPP – that's the class implementation file for it. The first thing it will always do is include the class file that it's working on. That's because if I start defining the features of class, I need to know what they are so the compiler can check and make sure that I've – I'm telling the truth, right? That the functions that I'm trying to implement and their parameters and the instance – the data members I'm trying to use match the description I earlier gave about what time was. So both the client needs to see it to use it. The implementer needs to see it to implement the right things that have the right names or the right prototypes.

This is where all the code for the member function goes – is in this file. And there's a little bit of a syntax that you'll need to know about this, which is when I'm ready to implement the set hour member function, that the name that it goes by – its full name – its kinda real name – is time colon colon set hour – that everything that was defined in the

time class is considered to be within this scope, and C++ is not the word for this. The scope time and – that the way to access something from within a scope is to use the name of the scope colon colon and then the thing you wanted to get out – you were trying to find, trying to use. So when we're defining all these member functions, we're saying, "It's the times set hour member function that I'm writing, not just a function called set hour." If I leave this off, I have a function that says, "Boy, its set hour N to new value."

What the compiler thinks I'm writing there is just an ordinary global function. It thinks there's a global function, just name this set hour. It takes one primary to reach its end, and it returns to void. It doesn't think it has anything to do with the time class, and the next thing it will notice is in here, when I'm trying to access features that are relevant to time, it will start giving me compiler errors. It'll say, "Hour? Where did hour come from? I know about new value. It's a [inaudible]. But hour just came out of nowhere. That was completely undeclared to me. Something must be wrong." So if we – that's a mistake you will certainly make at least once, and we'll want to commit to being attentive about. It's like it is the time set hour. It is the times two string. And that is distinguished from other functions of that same name.

That actually is kind of a neat little feature – the fact that that worked that way [inaudible]. The member function side shows up in a lot of our classes on vector, on stack, on string, on map – that the idea that all of those things can be called size is actually really handy because who wants to remember that one of them's length and one of them's size and one of them's numb entries and one of them's depth and one of them's length or something. That – having them all be size means that you have this one name you use when you want to get the information about how big a collection is. And the fact that the compiler can keep them all straight is based on this scoping mechanism. There's a vector size, which is different than stack size, which is different than map size, and it doesn't confuse them up because it has this scoping to keep them all straight.

So when we're writing the implementation of our member functions, there's a couple things, right? That we need to know about how to make them work the way they're supposed to. So in the time member function like shift by, that if we just refer to the field hour or minute – one of the data members of a time object, it is assumed that the hour or minute we're talking about is of the receiver. So shift by is never called without a receiver. There is not mechanism that lets you just call shift by without something in front of it. So shift by got called – there's some time object somewhere where it says t.shiftby an hour and 15 minutes that moves that – the T time forward by that amount of hour and minute change. So in the context of shift by, you can count on that hour and minute refer to the fields of a receiving time object that currently has some value in the hour and minute that we're changing by some delta to move it forward. So any reference to the hour and minute fields with no other qualifying marks means my hour, my minute – the one that got the message.

In the body of the member function, we also – so we can directly access these fields. We can also make further calls to other member functions. If there is a set hour and a set minute setter that are available on the time object, then we can say, "Set hour, hour plus

D – hour – the minute plus D minute, and then that will go through our own setter to update the hour and minute field to the new values. So this is a case where you're seeing a call to a member function without an explicit receiver, and in this case, the receiver is assumed to be the same receiver who originally got the shift by message. So this really isn't dropping the receiver or losing it, it's actually just assuming that, without any other explicit receiver, that it means me. So the time object that was asked to shift tells itself to change the hour and change the minute.

We're gonna talk a little bit about why that might be an important strategy relative to the alternative here, but just to know that mechanically, they both work. And then the last thing [inaudible] is this secret variable – this idea that it knows who the receiver is and that it's actually using it as part of the access for the hour and the minute and the caller – that there actually is a way to explicitly access that syntax if you wanna know. There is a special variable called this that is only valid within the member function of some class – any class. This is a pointer toward the receiver object. So this, in a time member function, is a time pointer. Now in a stack, it would be a stack pointer, and so on.

And I can use the longhand form of this arrow hour to mean take the time this – do you reference it to get to the time object and access its hour field. [Inaudible] this arrow is doing the dereference and then sending itself the member function. This is equivalent, right? To dropping that entirely. So you will rarely see a C++ program where you use this longhand form. But it is the mechanism behind it – you can see explicitly in situations where you might wanna be very clear about it. Sometimes you'll use that. For example, when you have two time objects, and you're trying to compare them, you wanna be distinguishing the parameter from this. You might explicitly say, "This hour – if this hour equals other hour." You might wanna make that clear which one you're talking about rather than dropping it in that situation. Any questions about a little bit mechanics? So I said that I would talk a little bit about, "Well, why might you want to have your member functions just make calls to other member functions rather than directly modify those fields?"

And one of the advantages, right? Of that object sorting programming is that you can actually really be very tightly managing the object's state and trying to make sure that the object never gets inconsistent. For example, if you're a stack, you wanna be sure that if you think the depth of the stack is ten, that there are ten valid elements. You never wanna get in a situation where somehow you think there are eight, but really there are ten or vice versa. It's your job to make sure that it makes sense. In the case of time, there's actually kind of a lot of invalid or illegal time values that you can imagine sneaking into your system where suddenly, you have the time – 55 negative 14. Right? You wanna be sure that the hour and minute really make sense for what you know to be the domain of valid times. By having the instance variables be private – the data members – nobody can muck with them other than you and the implementation. And so if you actually are very disciplined about making sure they can't ever get wrong – that no matter what operations you offer – shifting it forward, shifting it back, resetting it to new values – you make sure that there's no way a bad value could creep in.

And probably the easiest way to do that is to build a gatekeeper – is to have there be one central point where you change hour, and everything goes through that one central point. And that central point is designed to make sure that nothing can sneak past it. So if somebody tries to change the hour to something that's not between 1 and 12, you can just decide to bound it. This may not be the right thing to do. Maybe I should raise an error, but at the very least, I need to do something if they give me negative 14 or 82. And so right now, I choose to say, "Well, I'll just bring it to the closest value that's in range." An error would be a completely valid alternative.

Similarly, something about the minute – if they give me a minute that's not something between 0 and 59, that doesn't make sense. And so I just [inaudible] it – just to take the lower order part of it. If – these are the only places – let's say, imagine all of the time class where I ever assign to hour and minute. And everywhere else where I wanna change it, I say, "Well, here. Change it by this much and stick it through. Here's the value I'd like to set it to. Let me ask set hour to do it on my behalf – that if I've made some error in the calculation or I've been asked to do something kind of ridiculous that it will cause the right eventual assignment to be made." And so that's why I say, "What's this advantage? Why would we wanna do this?"

It's about control, right? Kind of centralizing the access and the robustness of your interface. Just say, "Well, here at the one place wherever change it. Let's make sure everything goes through that same interface. And that way, there's no way you can sneak a – through some back door – a bad value in." So a couple other things you need to know about class mechanics. There are a few special member functions. One is the constructor. So the constructor is a part – it's just automatically tied in with allocation – that when you allocate an object – either with new or on the stack – the constructor is automatically invoked as part of that process. And so that gives you your opportunity, if you define a constructor, to set up your data members the way you want. If you don't specify a constructor, you get the default constructor. The default constructor takes no arguments and basically does nothing to your arguments. So it leaves them uninitialized.

So for most things, that's not gonna be appropriate. It's probably the first thing you wanna do when you make a class is to write a constructor that sets yourself into a known, good state. It has a special prototype. It has to have exactly the same name as the class. It has no return type – not void, not anything. So it looks a little strange when you first see it. And then it can have parameters if you need them. It also can be overloaded. If you want more than one constructor, you can have multiple of them. So if I were to add one to my time class, I put in a function you notice has no return type, right? So the first thing you're seeing is the name of the constructor, which has to exactly match the case and name of the class itself – time. And in this case, choosing to make a constructor that takes two integer arguments – the starting hour and minute that I want.

In the CPP, right? The implementation of the constructor – time – within the scope of the time class – time colon colon time – again, no return type here – taking the hour and minute and setting it. I can actually make calls to my member functions, and given my earlier little speech about this, it seems like it would be even better, for example, to be

calling set hour of HR and set minute of MIN to just make sure that even if they give me initial values that are total garbage, right? That I don't let the time start off with some negative 45 – 100 – kind of state that doesn't make sense.

Once this constructor is in place, right? Then the calls to create a time will show two arguments are required to construct a time that's – to specify the hour and minutes is no longer optional, whereas prior to having a constructor that was a default constructor – the default constructor took no arguments. Once you declare any constructors, the compiler stops giving you anything for free. And so if you want both the default constructor and another several argument constructor, you'll just make several of them. You can have a time here that just says, "Time, open [inaudible] close [inaudible]," and then have a time colon colon time open [inaudible] close [inaudible] and set the hour and minute to sub default of 12:00 or whatever you want it. You can have as many constructors as you need, but typically, I would say most classes have one or two if not that many starting configurations you need to support typically.

So something that's new to you, if you're coming from the Java world, the idea that there's a corresponding parallel function that is tied into deallocation that when an object is going away – the two times when an object is going away. One is when it's leaving scope for a stack allocated object. So if you have time T declared in some foreloop, and when you exit the foreloop, it automatically deallocates that object or destructs it. [Inaudible]. If you have new – that object new time out in the heap that when you delete that time object as part of the deallocation of memory, it will destruct it. So what it does is it calls the destructor when that happens. The destructor is your hook for, "I need to clean up when this object is going away." It has the same name as the class, but prefix with a tilde. So tilde time is the destructor for the time class. It can never have any parameters, and it never has any return type. So time is the constructor name. Tilde time is the destructor's name.

You don't always need a destructor. It's actually a little bit more unusual with some simple classes that – what you would need it for is if there were something that you did as part of constructing or building the object that you needed to clean up. And the most common need for that is you have dynamically allocated some members. So you have an object that holds onto a link list and has just a pointer to the head note and the bunch of cells that follow it that when you were destructing, you would wanna go trace that link list down and delete all those cells if you were gonna clean up after yourself. If you don't do that, you don't have a destructor, right? Then the pointers just get [inaudible], and all that memory is just hanging out there, clouding your heap.

Sometimes there are things about opening files – like maybe you have a bunch of buffered output that you wanna be sure got committed to disc or a network connection you wanna close or something like that – so things that are open that you need to release. So typically, it's like resources that you need to clean up around that goes in the destructor. For the time class, there'd be no need, right? We have just simple integer variables. There's nothing special about them. So we don't actually need to go build a destructor. But some of the classes we'll see later will have a reason for that.

So this idea of object design is sort of one of the more important points to think about when you're setting up and building an object, right? Is it's your responsibility to make an object that kind of does what it says it's gonna do and tries it's best to be robust in the face of all situations. Even in this case that, maybe, the client uses it incorrectly, that the point of this sort of programming is – it's not to think that we're probing among enemies or we're malicious, but it is people make mistakes, right? People accidentally have the time, and they changed it to something that doesn't make sense, or they do the calculation incorrectly or they access off the end of the vector – that having the response to that by your object be to crash or to just produce incorrect information or to return an erroneous result – is really not doing the best you can. So part of your goal in designing this object is just to build something that really is a lovely work of art that does exactly what you want and catches all the mistakes, and even things that are really just the client's fault, right? Asking you to do irrational things are handled gracefully.

And so part of that is really taking kind of a big picture view of all the operations on the class and making sure that you have published the operations that are appropriate, right? That they respond to all forms of input gracefully, and that they kind of manage the object's state as it moves through the different operations in a way that's consistent. And so part of that, right? Is thinking about, "Well, who accesses the data where?" And making sure that those accesses are really understood and controlled. Never having any public data members – once you have a public data member, all bets are off. If hour and minute are public, anybody, anywhere, anytime can just reach in and change it to whatever they want. All guarantees are lost. So you would never want to do that, right? You are exposing part of your internals, right? For everyone to see. It's like having the door on the back of the microwave – you just open it up and you can start, like, rewiring. It's like – that's not gonna end well. You correctly initialized all your members in your constructor, and then you consider where and when to allow access to directly modify those things if – like through a setter – if it's needed and if it is properly constrained.

The other thing that's an important part of kind of designing the interface is making sure that the object [inaudible] is fully formed. That it does all the things you'd like it to do and kind of simplifies the client's use of it as opposed to kind of going half the job and then leading them to have to do a bunch of other things. It has a complete kind of coherent set of applications. So if you needed to print a time that one way, you could supply those. You can say, "Well, you can get the hour. You can get the minute. And you can convert them to string. And you can put a colon in the middle of them. And then you can print it. Why bother me about it? Like, you wanna print time? That's your business." On the other hand, right? Printing seems like a pretty fundamental operation that would just be nice for time to support. Maybe it supports it by having just a print method. Maybe it supports it by giving you a string form like two string where it does those calculations – it converts it and gives it back to you, and then you can decide to just output it as a string.

But having that be a part of the class really makes the client's job just easier. They're likely to need something like that. Why make everybody who uses your time class have to reproduce this functionality that really feels like something the time class itself should

support? And so when you think about your set of operations, you have some lists you might need in the near future. You're also trying to think a little bit more big picture about what – well, in all the situations you use times, what kind of operations come out as being useful. While I'm implementing time, why don't I just kinda build the full-fledge version that has the functionality for the things I imagine to be likely to be needed.

So comparing, right? Two times – is this is one before this one? Are they equal? Right? Moving them around – just kinda giving you the whole package. So here's a little thought about what are the advantages, right? Of the object to design or object in capsulation means that you have described what time does. You say it allows you to set the hour. You have described that it'll compare them to see if one's less than another. You have a two string or a shift by, but you have made no guarantees to them about what you really do – how you really internally work it. The time interface might describe things, like, in terms of hour and minute. "Oh, it's an hour and a minute in time, and you can shift it by hours and minutes. You can ask me for the hour and the minute."

But internally, there's no rule that says – because our interface is hour and minute – that we have to have fields called hour and fields called minute. That in fact manipulating it, right? As hour and minute is messy. And if you throw in a.m./p.m., it's even worse. That all the things, like, people decide if one time is less than another – it's like, "Well, if this one's a.m. and that one's p.m., then it's less than." Right? And if they're both a.m. or p.m., right? Then we start looking at the hour. And if the hour's the same, we start looking at the minute. Like, it just doesn't – it gets goopy. Right? We can make it work, but why do it if we don't have to? So if we just, instead, pick a different representation – it's our internals. Nobody needs to know what we're doing. It's like, "I don't wanna keep track of hour and minute. Hour and minute is some other artificial sort of human representation."

Computers are really good at much more simple things, like what if I just said, "Look. Every day – every minute within the day is assigned a number: zero is midnight, right? And then 720, right? Is noon." And just whatever. You just – you said, "Minutes since midnight." Two minutes into midnight is 12:02. Right? Twelve minutes is 12:10. Sixty minutes is 1:00 a.m. Right? I just mark everything down there. Then it turns out, right? A lot of my operations become really easy. I wanna see if this time is less than the other? It's like, "Okay. Just compare their two minutes since midnight."

I wanna move a time 60 minutes forward? Well, I just add 60 minutes to whatever number I have. You can do the wraparound really easily. So like, if you were close to the end of the day at 11:30, and you add an hour. Right? It's very easy to just use [inaudible] to wrap you back around to the 12:30. But if you're doing it with hours and minutes and a.m./p.m., all those things are very ugly. So the thing is we can still describe our interfaces hours and minutes because that's how clients want to see it. But internally, we get to make all the decisions we want, and we might as well make the ones that make our lives easier – make it easier to get the code right. It's actually – it takes less time to do the operations right. It'll run more efficiently. It uses less space, in fact. And it just made everything easier from our point of view.

And it turns out we can still have operators – like set hour and get hour and set minute and get minute, right? They just internally kinda convert to the internal representation. But they provide an interface, right? That seems appropriate for the client. It doesn't mean it has to match what we do inside. So that's actually kind of a really neat – a neat thing to keep in mind, right? Is that that's part of what we bought – is that they don't actually have any guarantee about how we implemented it, and we're not gonna tell them. So I will leave you with this thought about abstraction because that is sort of a key advantage of working this way, right? Is there really is a wall between the client and the implementer that is an important part of – it feels like a restriction. It feels like a jail. It feels like a prison if you think about it that way. But in fact, it's liberating. It means we're not in each other's hair. We're not dependent on each other in any meaningful way. We have this little, tiny conduit – that chink in the wall we call the interface – that describes what things we both can depend on.

But then everything else is at the liberty of the person who's on their side of the wall to do what they want without regard to what the other person's expectations are. So we'll think about that in terms of, let's say, Lexicon and scanner and vector and all these kinda neat things on Friday to kinda dig around the backside and see what we've been missing out on. I will see you guys then.

[End of Audio]

Duration: 50 minutes