CS 240: Code Layout Transcript

This video shows a split screen of Professor Wilkerson on the right and a PowerPoint screen on the left. Any text displayed or action performed that is not verbalized will be included in italics as visual descriptions.

[00:00:00] **JEROD WILKERSON:** In the first video where I talked about code quality, I showed an example of bad code.

[00:00:05] The main thing that was noticeable in that code is it was laid out poorly.

[00:00:09] One of the best things you can do to write quality code is to have a good layout for your code.

[00:00:15] There's a lot that can be said about that, so I'll give you quite a few details about code layout.

Begin visual description. The unread bullet points on the current slide are the following: The physical layout of the code strongly affects readability. (Imagine a program with no newlines. Imagine a program with no indentations.) Good layout makes the logical structure of a program clear to the reader. End visual description.

[00:00:23] A good layout makes it really easy to understand the code and it helps avoid bugs.

[00:00:29] It helps us avoid introducing bugs when we change the code.

[00:00:32] If you can remember back to the original example that I showed, it would've been really hard to modify that code without introducing bugs.

[00:00:39] In fact, you probably would have needed to reformat the code before you could really make any changes to it.

[00:00:45] One of the important things with code layout is that you pick a style for the layout and you use it consistently, so you don’t want to be inconsistent in different methods or different classes.

[00:00:55] It’s also really important that you follow the conventions that you should follow.

[00:01:01] For any technology, there will be a set of conventions; any programming language, you will have a set of conventions, and you really should follow those.

[00:01:08] If you’re working for a particular company or organization, you should follow that organization’s standards.

[00:01:14] Even if you don’t like it, it’s more important that you follow the standards than that you use a coding style that you like.

[00:01:22] It’s rare for an organization’s coding standards to conflict with the technology standards.

[00:01:27] For example, there are standards for Java.

[00:01:30] It’s rare for a company that does a lot of Java development to use standards that don’t match the Java standards.

[00:01:35] If they do, you should use the company standards and maybe work to get the company standards to be in line with the technology standards.

[00:01:46] One important thing is to use whitespace properly.

[00:01:51] Spaces, tabs, and line breaks are really important in making our code easier to understand.

[00:01:56] We want to organize our methods into paragraphs.

[00:02:01] What I mean by that is take lines of code that are really closely related, maybe the creation of a variable and the use of that variable in a loop, and put those together in a paragraph which is denoted by whitespace before and after it or new lines before and after it.

[00:02:19] Then of course, we want to be careful with indenting.

[00:02:22] We want to indent consistently.

[00:02:25] In the same way, it’s common to use either a tab or four spaces.

[00:02:30] It’s actually more common to use four spaces for indenting than a tab.

[00:02:36] For expressions, there’s a lot we can do with expressions to make them easy to read, and they can be hard to understand if we’re not careful.

[00:02:45] One of the things we need to be careful with is parenthesizing.

[00:02:48] You might understand the rules of precedence so well with your language that you think you don’t need to use parentheses in a complex expression, but you should use them even if you don’t need them to understand it because you likely won’t be the only person that looks at your code.

[00:03:01] Maybe the next person doesn’t know those rules as well as you.

[00:03:04] You’re not infallible, you might make mistakes in what you think the precedence rules are or when you’re changing the code later, you might be confused by it.

[00:03:13] Here we have an example that is not really using parentheses and it’s not using spaces between the operands and the operators.

[00:03:22] This code is a little hard to understand.

[00:03:25] The code below it is doing exactly the same thing, but we have more parentheses and we have spaces around the operators, and that makes it a lot more clear, a lot easier to understand.

[00:03:35] We want to separate conditions on separate lines.

The following is the first piece of code:

```
if ((‘0’ <= inChar && inChar <= ‘9’) || (‘a’ <= inChar && inChar <= ‘z’) || (‘A’ <=
inChar && inChar <= ‘Z’)) {}
```

End of first piece of code.

The following is the second piece of code:

```
if ((‘0’ <= inChar && inChar <= ‘9’) ||
   (‘a’ <= inChar && inChar <= ‘z’) ||
   (‘A’ <= inChar && inChar <= ‘Z’)) {}
```

End of second piece of code.

[00:03:40] If we look at these two pieces of code, they both do the exact same thing and the only difference is new lines, but I think we would all agree that the bottom code is lot easier to understand.

[00:03:50] In fact, if I didn’t show you the bottom code and I asked you to tell me what the top code does, you could figure it out, but it would take you a little bit of looking at it to understand that this is checking to see if something is alphanumeric.

[00:04:04] If we look at the bottom, it’s pretty clear.

[00:04:07] We can see that we’re checking first to see if something is a digit.

[00:04:11] Next, we’re checking to see if it’s an uppercase character.

[00:04:15] Then we’re checking to see if it’s a lowercase character.

[00:04:18] Then we’re checking to see if it’s an uppercase character.

[00:04:21] If it’s any of those, we’re going to go into our if statement.

[00:04:24] The bottom one is really clear that it’s alphanumeric that we’re checking for.

[00:04:27] Top one, not as clear.

[00:04:31] We can make this even more clear though, by creating well-named submethods.

[00:04:38] Here we’ve taken the same code, and we have created some methods we can call isDigit, isLowerAlpha, isUpperAlpha, so that’s better than the previous code.

Start of code.

```
if (isDigit(inCHar) || isLowerAlpha(inChar) || isUpperAlpha(inChar))
```

End of code.

[00:04:49] It’s pretty clear, but we can do even better than that.

[00:04:52] We can write a method that is called isAlphaNumeric, then that method can check to see if it’s a lower alphabetic or upper alphabetic.

Start of code.

```
if (isAlphaNumeric(inChar)) {

}

boolean isAlphaNumeric(char c) {

        return (isDigit(c) || isLowerAlpha(c) || isUpperAlpha(c));

}
```

End of code.

[00:05:05] Again, there’s a lot that you can do to make your code more clear.

[00:05:09] This example is more clear by following that concept of algorithm decomposition, breaking things down and giving them clear names.

[00:05:19] Here’s a concept that can inspire a lot of religious debates within software development.

[00:05:26] Where do we put the curly braces? Here are four examples that you see in code.

Start code of first example.

```
for (int i=0; i < MAX; ++i) {

        values[i] = 0;

}
```

End code of first example.

Start code of second example.

```
for (int i=0; i < MAX; ++i)

{

        values[i] = 0;

}
```

End code of second example.

Start code of third example.

```
for (int i=0; i < MAX; ++i)

{       values[i] = 0;

}
```

End code of third example.

Start code of fourth example.

```
for (int i=0; i < MAX; ++i)

        {

        values[i] = 0;

        }
```

End code of fourth example.

[00:05:31] The top two I would call reasonable examples.

[00:05:34] They are good examples of how to use curly braces.

[00:05:37] Never do the bottom two.

[00:05:39] For the top two, the difference is where the opening curly brace goes.

[00:05:44] Does it go on the same line as the thing that it’s a curly brace for? Or do we always start opening curly braces on a new line? I used to feel really strongly that this *(second of the four curly brace options)* was the best way and it was the clearest.

[00:05:56] I would sometimes even get into discussions about that with other programmers.

[00:06:02] Unfortunately, for me, this *(first of the four curly brace options)* is the Java standard.

[00:06:05] The Java standard was to have the opening curly brace on the same line as the thing that it’s associated with.

[00:06:11] I didn’t use to like that.

[00:06:13] Even though I’ve been doing Java development for a long time, I would write my Java code this way and I would just go against the Java standard because I didn’t like it, but then I was hired for a consulting project on a company where they were really strict about enforcing that curly brace standard, where you follow the Java-accepted standard.

[00:06:31] I was on that project for several months, and at the end of those months, I had gotten so used to putting the curly braces here that I liked it better and it seemed more clear to me.

[00:06:41] The point is it’s not so important which standard you follow or which way you do it.

[00:06:47] The important thing is that you do it the standard way and that makes it so everybody who works within that codebase will understand it.

[00:06:55] People at a company should understand your code because you all write it the same way.

[00:06:59] Java developers should all understand everybody’s Java code because we write it the same way.

[00:07:04] This *(first of the four curly brace options)* is the standard for Java code, and this *(second of the four curly brace options)* is a standard for C# code.

[00:07:10] If you’re doing C# development, you put opening curly braces on a line by itself.

[00:07:15] Never follow this standard *(third of the four curly brace options)* where a curly brace can have other code after it on the same line.

[00:07:22] In this example *(fourth of the four curly brace options)*, I guess they're trying to make things clear by lining up the curly braces with the parentheses, but that's not clear, never follow that standard.

[00:07:32] Let's look at this statement.

Start code.

```
for (int i=0; i < MAX; ++i)
        values[i] = 0;
```

End code.

[00:07:35] What do you think about that? A for loop that doesn't have curly braces.

[00:07:39] Some people like to do that if they have a really simple loop or really simple if statement, they like to write it without curly braces.

[00:07:46] The rule is that a control statement applies only to the next line unless that next line of code is wrapped in curly braces.

[00:07:55] That's how you make it be multiple lines.

[00:07:57] Almost any coding standard book that you can find will tell you that this is a bad idea.

[00:08:02] What we should do instead is have curly braces wrapped around it, even though the language doesn't require it.

[00:08:08] That'll make your code more clear, makes it visually easier to understand.

[00:08:12] It also can eliminate questions.

[00:08:14] If I have this *(first)* line of code with this one *(second line of code)* and then another one *(another line of code)* after it *(second line of code)*, if that's formatted poorly, it's going to be really hard to know.

[00:08:24] In fact, even if it's not, it can be hard to know.

[00:08:27] Is this line of code *(hypothetical third line of code)* supposed to be part of the for and they just wrote a bug or is it really supposed to be separate? I can have that question.

[00:08:35] But if I use curly braces around it, it's going to be really clear. Just get in that habit.

[00:08:40] Always write curly braces around your if statements, for loops, or any kind of control structure.

[00:08:45] Even if it's only one statement, your code will be more clear, it'll be less ambiguous, and it will be easier to read and understand.

[00:08:54] Let's think about method parameters.

[00:08:57] We want to use spaces at least between individual method parameters to make them more clear.

[00:09:03] If you look in this example, definitely don't do it the first way where we just have commas and no spaces between the variables.

Start line of code.

```
WebCrawler.crawl(rootURL,outputDir,stopWordsFile);
```

End line of code.

[00:09:10] This next way is one acceptable way where we have spaces between all the variables, and it also has a space after the opening and before the closing parentheses.

Start line of code.

WebCrawler.crawl(rootURL, outputDir, stopWordsFile);

End line of code.

[00:09:20] That's a standard that a lot of people like to follow.

[00:09:23] This is also a reasonable standard where we don't have spaces before and after the parentheses, but we do have them in between the variables.

Start line of code.

WebCrawler.crawl(rootURL, outputDir, stopWordsFile);

End line of code.

[00:09:32] When choosing between these two standards, what I would say is follow the standard of your organization.

[00:09:37] If your organization doesn't have a standard, I would encourage them to create one.

[00:09:41] The code will be easier for everybody to understand.

[00:09:44] You can get used to either one of these two, but it needs to be standard.

[00:09:49] Another thing you want to do is make sure you have only one statement per line.

[00:09:54] If we look at this example, this is a problem.

[00:09:58] This is C++ code, and probably what the programmer intended to do was to create two int pointers, one called p and one called q, but that’s not what they did.

Start line of code.

```
int * p, q;
```

End line of code.

[00:10:07] This line of code has one int pointer and one int q is just an int.

[00:10:13] If they had put those two statements on separate lines, they wouldn’t have made that mistake.

[00:10:17] This is what we should do.

Start code.

```
int * p;

int * q;
```

End code.

[00:10:21] These are subexpressions, so when you’re initializing a variable, don’t do it as a comma-separated list, do it this way.

[00:10:27] Then when you have statements, don’t put them on the same line.

[00:10:30] If you look on the left here, this is not as clear as this with these two statements on separate lines.

Start unclear code.

```
x = 0; y=0;
```

End unclear code.

Start clear code.

x = 0;

y = 0;

End clear code.

[00:10:36] Always put separate statements on separate lines.

[00:10:39] We don’t want to have our lines get too long, so we need to wrap our lines at some point.

[00:10:45] The question is, where do we wrap them? It used to be pretty standard when we all used a standard size monitor and their resolutions were not that high.

[00:10:54] We could say that 80 characters was the right place to break a line, but now that’s not necessarily true.

[00:11:00] A lot of us have wide format screens, we have high resolution, so we can fit a lot more on a screen, so 80 is not necessarily the right length.

[00:11:09] But we want to have a common length and we don’t want to go too long.

[00:11:13] It’s probably 80 or 100 or 120, something like that.

[00:11:17] Intelligence has a built-in default, which is a pretty good standard to follow, and it has a little line on the end so you can see when you’re going past it.

[00:11:25] We also want to think about how to align continuation lines.

[00:11:30] When I say continuation lines, if I have a statement that’s too long and I need to break it up, how do I align that next line with the one above it? Here’s some examples of that.

[00:11:43] Sometimes we end up breaking a method declaration up.

[00:11:47] We have several parameters and they won’t all fit.

[00:11:51] It’ll be too long of a line, so we’re going to break it up on some of the parameters.

[00:11:54] First of all, don’t break it up in between the datatype and the name; you want to always break it up on a comma.

[00:12:03] Then the question is, how do we align the next line? This is one way to do it.

[00:12:06] This is a line by tabbing, which is not quite as clear as this, so it’s better to align it so the variables start in the same place.

[00:12:14] It’s just easier to read that way.

[00:12:16] Then if you have an expression where you’re not breaking up the parameters of a method, you’re just breaking up some other part.

[00:12:22] Like in this example, we will typically just tab it in or space it in four spaces from the previous line.

Start code.

```
DailySchedule newDailySchedule =

        new DailySchedule(getNextSchedulableDay(today));
```

End code.

[00:12:30] Here, if we have something like this where we’re doing a return or we have something with multiple expressions as part of a statement, we want to line them up together and it’s easier to read that way.

Start code.

```
return (date.get(Calendar.DAY_OF_WEEK) == dayOfWeek &&
        date.get(Calendar.MONTH) == month &&
        date.get(Calendar.DAY_OF_WEEK_IN_MONTH) == n);
```

End code.

[00:12:42] Those are just some of the details of how to format your code in a way that makes it clear and easy to understand, easy to maintain, easy to change.