# Overriding Methods & Accessing Inherited Code

**Big Ideas**

**Products can be designed for life cycle**

Software organizations are always updating and altering software source code. This is done in order to fix bugs, add features or improve performance. It is for this reason that it’s always important to remember that software can be designed for life cycle.

The original programmer can ensure that their code is well written and well documented. This can facilitate later updates and changes.

The original programmer can also think carefully about how additional features and updates might be implemented. This can impact the original design of objects or classes in order to ensure the ability to further add to and improve the design.

With your partner, complete the steps outlined on this sheet to modify your Class Poster from 6.1. You should use this worksheet to draft and correct your ideas; you can write a final, neat version on the paper that has been handed out to you.

## Step 1

Take a moment to brainstorm at least one method that will replace a method inherited from the superclass in a subclass. Before writing your code, in the space below, you should write pseudocode that explains what behaviors will be changing, and why.

Step 2

In the space below, write one or more methods in your superclass that allow your subclass to access a method from the superclass. Before writing your code, you should write pseudocode that explains what method you will be accessing, and why you will be using that information in your subclass method.

Step 3

In the space below, construct some client code that calls one of the methods you wrote in Step 3. To receive full credit, you must include pseudocode or comments that explain what the method does.

Step 4

Take a few minutes to check all of your code with your partner. Don’t forget to use your notes, 4 Commandments of Scope, error-checking guide, textbook, and classroom posters to help you check for mistakes or logic errors. At this point, you may trade your work with another group and look over each other’s answers.

Step 5

Using the supplies provided by your instructor, write a neat final version of code from Steps 1 – 3, and place the additional code in the proper location of your poster.

*Note: your poster might start to look a little crowded or messy. This is OK for now; physical cut and paste is not nearly as orderly as cutting and pasting using computer software. Before word processing (what we commonly call “Word” or “Docs”) existed, people used to type or write books, essays, and reports on a typewriter. If they wanted to move around text, they would actually cut the text out and paste it with glue in the correct position (like you’re doing on your poster).*

Step 6

Once you have placed all of your overridden methods and accessor code onto your poster, have your instructor come by to check your work. You should be ready to explain how your code works. Once you’ve received approval, begin Step 7.

Step 7

Using the string or yarn supplied by your teacher, illustrate the flow of control through the class hierarchy of the method you wrote in Step 3.

1. Tape, glue, or pin the first end of your string to the method within your client code
2. Stretch the string across your poster to the method it references in the subclass. Attach the string at that method.
3. Stretch the string to the next method that the subclass method calls.
4. You should end your flow-of-control string at the superclass method. (Do not show the flow of control returning back to the client code.

When you are finished, you should have a zig-zagging line of string that traces the execution of the methods through the class hierarchy.

Step 8

Once you have finished, walk around the room to others’ examples, and take a look at their code.

1. Take a minute at each poster to read the added code. If you do not understand something, make a note of it to ask later in class.
2. Trace the execution of methods through the class hierarchy by following the string on each Class Hierarchy Poster.
   1. Start by touching the string at the client code method.
   2. Read that method, then using your finger, trace the flow of control to the subclass method.
   3. Read that method and any other methods around it if you need context to understand the purpose or function of the code.
   4. Use your finger to trace the flow-of-control yarn/string to the superclass method.
   5. Read the superclass method, then try to predict the final output in the client code.
3. If you are confused, surprised, amused or impressed by any Class Poster, write down a note with your comments in your notebook to share during the class critique.