E Gurnee

Object-Oriented Design Principles/Plans/Practices

-so you at least survive your classes

The big ones to know NOW:

One Responsibility Rule – each project and system you work on will have many things going on. Do not set out to write one class/main method to handle everything; even if you could, there is a better way. A class should only ever have one responsibility. Allowing for classes to bloat will create unneeded complexity which at the very least obscures other potential problems. Each facet or responsibility of your project should be handled by the minimum amount of code necessary for handling, extra code just gets in the way; i.e. don't just suppress the "unused" warnings in eclipse.

Draw A Picture – if you do not understand the assignment enough to express it visually, go bother the people in your class, your professor, tutors, random people on the street, really anybody who will listen and let you talk through the problem. Being able to talk about the project is good, being about to draw something about it is even better; drawing about a project allows for organization and focusing of thoughts, not to mention the benefits of thinking about the problem in a new fashion. If time is only spent thinking about the code that needs to be written for a given project, you can lose sight of the big picture and miss out of ways to make the project easier. Flowcharts, ER-diagrams, UML, Use-cases, in-system diagrams are all great ways to get a new visual of the project, but seriously just drawing out all of your incoming data as invading mushroom people that your WizardController needs to enchant and send to various other locations in your KingdomProject will be helpful, so do that! (And then e-mail it to me at eddie.gurnee@gmail.com!)

Model-View-Controller – Model-View-Controller is a design pattern that encourages the separation of a system into: the data logic (Model), what the user interacts with (View), and a go-between handler (Controller). All user interface (UI) should be removed from any class who's responsibility is **not specifically handling UI input**. If user input is needed, then there needs to exist a class to handle the user input. sysout should not appear in classes that aren't explicitly designated.

You Aren't Going To Need It – don't write out a whole bunch of code for something you aren't actively using. Eclipse can be great at encouraging the YAGNI behavior: start referring to things that don't exist and eclipse will very helpfully suggest that you create them, it'll even give you a stub to get started. You can create complete classes with enough stubs to get you started without too much work. On the other hand if you do go and write out a full implementation to a class before you need it, you could end up with a situation where you're trying to shoehorn code that wasn't called for into a project that doesn't need it.

tl;dr

- you classes should do one thing, if you need to do two things, make two classes
- if you can't explain your project with a picture, you're gonna have trouble coding it
- user interface belongs only in a UI class, don't mix logic and view
- don't start writing the implementation of every single possible method that a class could even need, your program probably won't need it and can only be made worse

oneliners you should hear anyway:

- write to the Interface not implementation
- abstract code to the general problem, not just the specific problem
- objects only needs to know about what they need to do at their level, they trust the implementation of any levels below them
- remember your assembly, just do the simplest positive thing immediately in front of you