The reasons why I ordered the hierarchy the way I did is as follows; firstly, I put the worker class to be the parent class of all of the classes because it holds most of the methods that the child classes inherit and override. Secondly, Worker also includes a lot of private integers that wouldn't have been able to be accessed by its current child classes if it was lower on the hierarchy. Also, if it was lower from the hierarchy, Worker object arrays including Janitors, Designers, and Coders wouldn't be able to exist because you cannot make a child array that includes parent objects because not all of the children are objects (not all rectangles are squares), so at that point you'd figuratively be comparing apples and oranges. Thirdly, another reason why Worker at the top of the hierarchy is because Child classes can't inherit from more than one parent class, and by putting it below Janitor, Coder, and Designer, it would only be able to 'connect' to one of them and that doesn't make any sense because Janitors, Coders, and Designers are ALL Workers.

The reason why Janitor and Coder are children of Worker now seem pretty obvious because of the aforementioned reasons, but why is SoundDesigner below Designer? Why is it not the parent of Designer? That is because of the simple concept that all SoundDesigners are Designers but not all Designers are SoundDesigners. This means SoundDesigner has an 'is a' relationship and should inherit from Designer. Still not convinced? As shown on the hierarchy, both of them override the toString method. This means that if SoundDesigner goes before Designer, the modifications to the toString method will be reversed and so will the order of the text.

In short, this is why my Game Design Firm Hierarchy is laid out the way that it is. The classes weren't laid out randomly; The inheritance between them is most definitely intentional and meaningful.