Overall, I feel a lot more comfortable with java syntax such as private, public, static, void, final, etc. This lab really gave me a better understanding of what each word does at different points in the program, and how it affects that files future use. I also became a little more familiar with the differences between interfaces and superclasses. Furthermore, the layout of the lab assignment made the overall project much more manageable for me. I learned how to read code and work backwards, and it also helped me learn how to better tackle a project like this (start small).

This design incorporated future growth by creating separate files with distinct classes, subclasses, and methods. Future changes can be made to one file without necessarily impacting the rest of the files, as long as that particular file compiles. This design also incorporates future growth by setting up certain classes (WoodBlock and StoneBlock) a certain way so that they can be used correctly later (I assume).

There are a few things I’d still like to have a better understanding of:

Why use try and except block for the block factory? It seemed this could better be accomplished by using an “If” statement, particular with the way the driver is setup (only create a block if not null). Additionally, the subtractResources method will not allow you to use a number that is larger than the resourceBin’s current weight (which also seemed redundant).

I’m still confused on how we were supposed to make the block parent/child relationship work and I’m unsure that what I coded was what you were looking for. To use a no arg constructor, you need to have a default constructor in the abstract/parent class. However, the default constructor doesn’t work because you need to create two different block objects that can only have two different sets of arguments. You could create a default constructor that defaults to stone, for example, and have the other use manual arguments that create wood, but that would still be taking at least one set of arguments.

Could we not define the interface methods at all in the interface body? Or just use a parent class? The methods are the same between WoodBlockFactory and StoneBlockFactory (except for produce()). I assume the reason we used interface for these is because later we will have other methods that behave differently depending on the object, but I’m not totally sure.