REQ1 – Design Rationale – Assignment 1 – Dean Mascitti – 33110360

Design Goals

The goal of this requirement was to add functionality in that creates 2 different scraps being metal sheets and large bolts and make it so that the intern can pick up these scraps and drop them off when prompted.

The goals in designing this functionality were to ensure that the design was extensible for future extension in upcoming assignments, as well as repeated code was limited throughout the project, abiding by the do not repeat yourself (DRY) principle. The use of abstraction is a goal of this requirement to ensure that both these principles are abided by.

Design & What OOP Principles Were Applied

The main design idea for this requirement was to make use of a Scrap abstract class, this scrap abstract class can then be inherited by MetalSheet and LargeBolt classes. In the future if other scraps are added to the game if appropriate and depending on the context.

The Scrap abstract class also inherits from the Item abstract class in the engine code, ensuring it adopts the functionality made in the engine code for items that can be picked up and dropped off.

In terms of functionality the Items were placed in the game map in the Application class for the Intern to interact with.

In my design for this requirement the following OOP design principles were used: extensibility, open closed principle, abstraction and DRY.

How They Were Applied & Why in This Way

- The use of the abstraction of use of the Scrap class was implemented because if there is any similar functionality to be done by scraps in the future it can be written in the Scrap abstract class therefore applying extensibility. In addition, this ensures that any similar functionality is written within the Scrap abstract class and does not need to be repeated in other subclasses (scraps) applying the do not repeat yourself (DRY) principle.
- The open-closed principle is present in this design due to the fact that if we need to add any more scraps to the project, we can simply create new classes and make them inherit Scrap, therefore not needing to edit any existing code. This was done so in further requirements it is easy to add features and does not require refactoring.

Design Pros and Cons

Pros

- Use of abstraction
- Easily extensible for future applications and features
- No repeated code within the program

Cons

- Double abstraction (Scraps inherit from Scrap and Scrap inherits from Item)

- Other scraps may have completely different functionality to MetalSheet and LargeBolt and it may not make sense for them to both be subclasses of Scrap
- Most functionality is done in engine code (PickUp and Drop actions) so Scrap abstract class does not have a lot of responsibility at this point in time.

Alternative Designs

An alternative design would have just been to create a MetalSheet and LargeBolt class each and have them inherit from Item, however the Item class is very generic and there is expected to be many different types of items, and scraps are one of them. We can expect that scraps may potentially have similar functionality in the future, therefore the reason I chose to make this abstract class is because it is a place where the functionality can be written so that each scrap subclass does not need to repeat themselves.