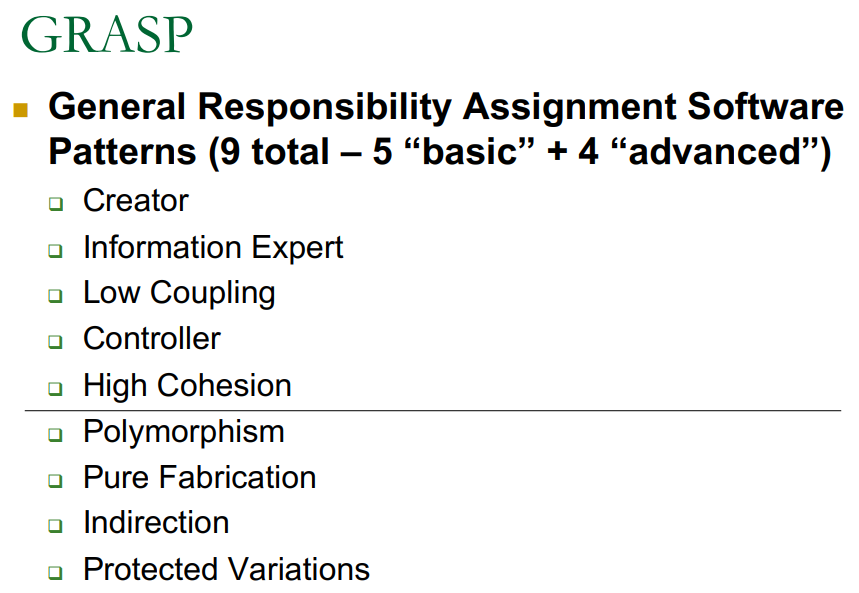
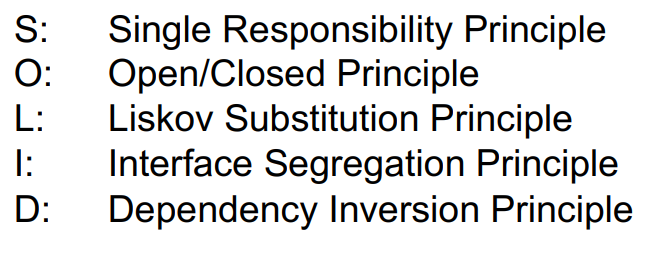
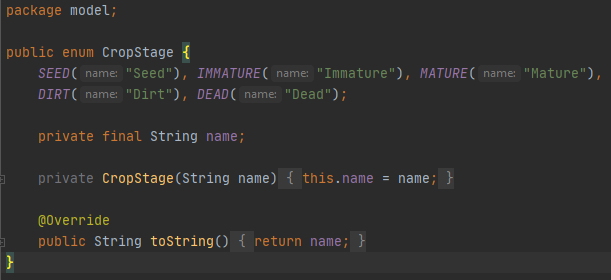
SOLID and GRASP Principles



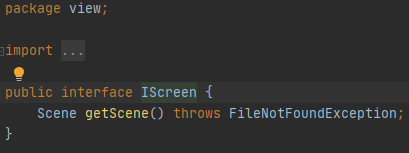
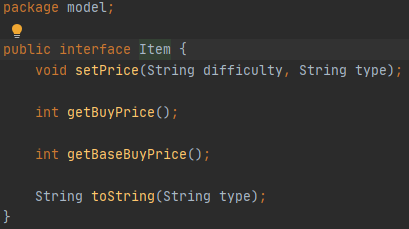
**SOLID Principles (3):**

Single Responsibility Principle (SRP):



This class enumeration for crop stage follows the SOLID principle of SRP. The class is only responsible for holding the stages of the crop’s life cycle. It is precise and contains short naming conventions that are self-explanatory. It is easy to understand and alter if more crop stages need to be added or removed.

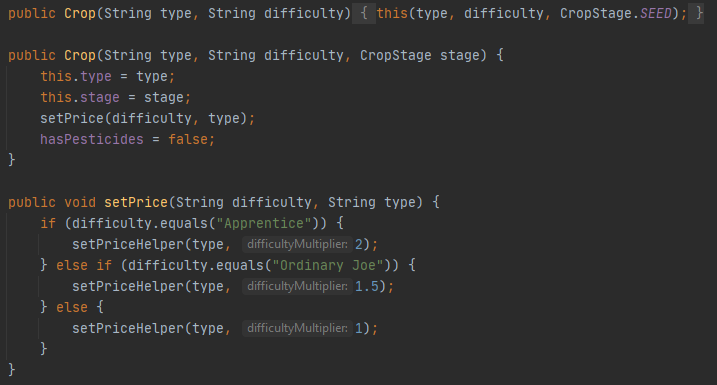
Interface Segregation Principle (ISP):



Both classes are related to the functionality of the Inventory class in our code. Instead of making a larger interface including both, two smaller more focuses interfaces are created. The interfaces are made as small as possible further focusing on the ISP.

Open / Closed Principle (OCP):

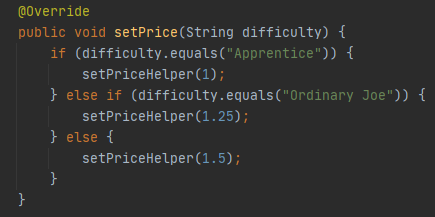


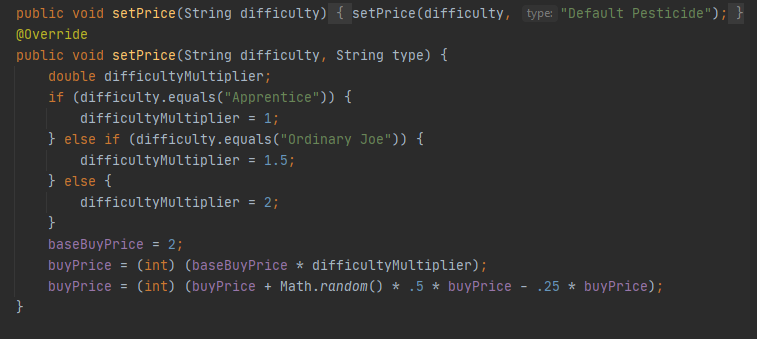


This code above follows the Open / Closed Principle by having a price set in each of the child classes. By placing the variable in each of the classes if follows the principles rule of being open for extension but closed for modification. The functionality is extended by adding code instead of changing the existing code. This makes the code less likely to break the core of the system.

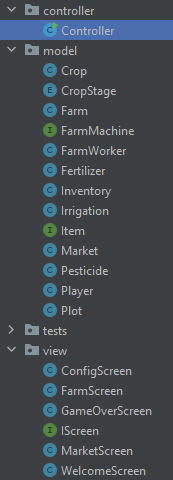
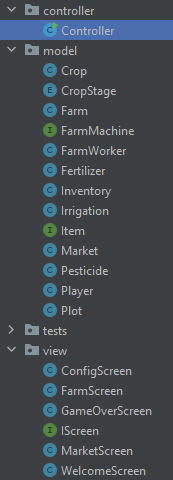
**GRASP Principles (5):**

Polymorphism (advanced):



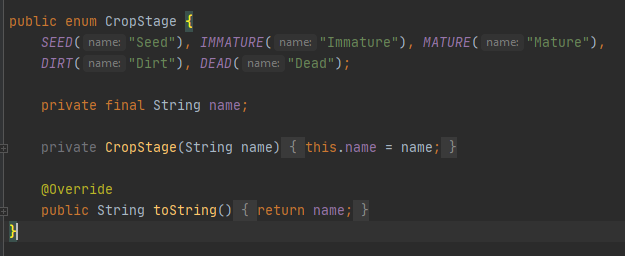


The setPrice method is used in multiple instances throughout the code. The parameters of them are different to account for a default case and a case where the type is specified. Also, the methods are spread across different classes. The name of the method is simple and reusable which allows for polymorphism between changing and adding parameters. This allows for added parameters to make a new method in a different class with the same name that will add specificity, if needed, to alter the pricing.

Controller (basic):

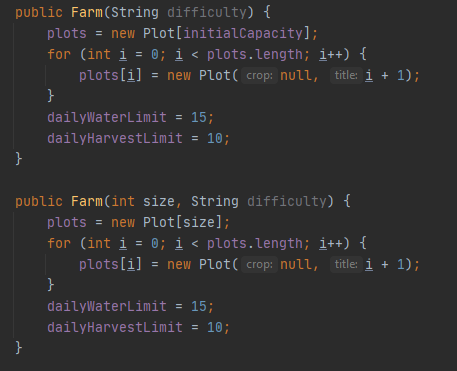
The code for the controller utilized only a small portion of the code, as a controller should. It demonstrates the controller principle of GRASP by having multiple methods other than the controller (Screen methods) to control the code. It delegates to other objects work to be done as well as coordinates and controls high level activity.

Pure Fabrication (advanced):



In attempting to add a crop stage variable that would keep track of a crops various growth stages we had no current method that could be used to hold this information without convoluting the method. We decided to create a new class that is solely responsible for holding the crop Stages. This follows the GRASP principle of pure fabrication since there was not an appropriate class to house the different crop stages, resulting in a new fabricated class called CropStage.

Creator (basic):



There are many examples of the creator principle of GRASP in our project’s code. One simplistic example of this is the Farm class. The Farm class creates a Plot array and functions as a Creator for those plots. Farm contains instances of the class Plot thus the class Farm functions as a creator for the Plot[].

Information Expert (basic):



The FarmScreen class serves as an Information Expert for most of the code in each milestone. It contains several of the child classes and most other classes will branch off this overarching class. It follows the GRASP information expert principle since many other classes are referencing based of the information stored in this class. In other words it contains a significant responsibility of the code since it also contains such a high amount of information stored from the code.