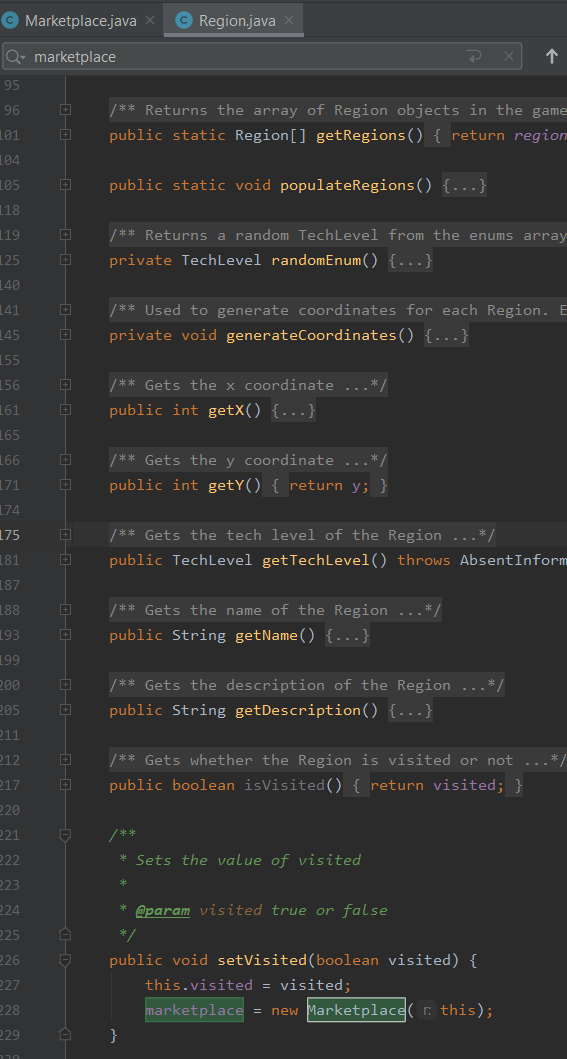
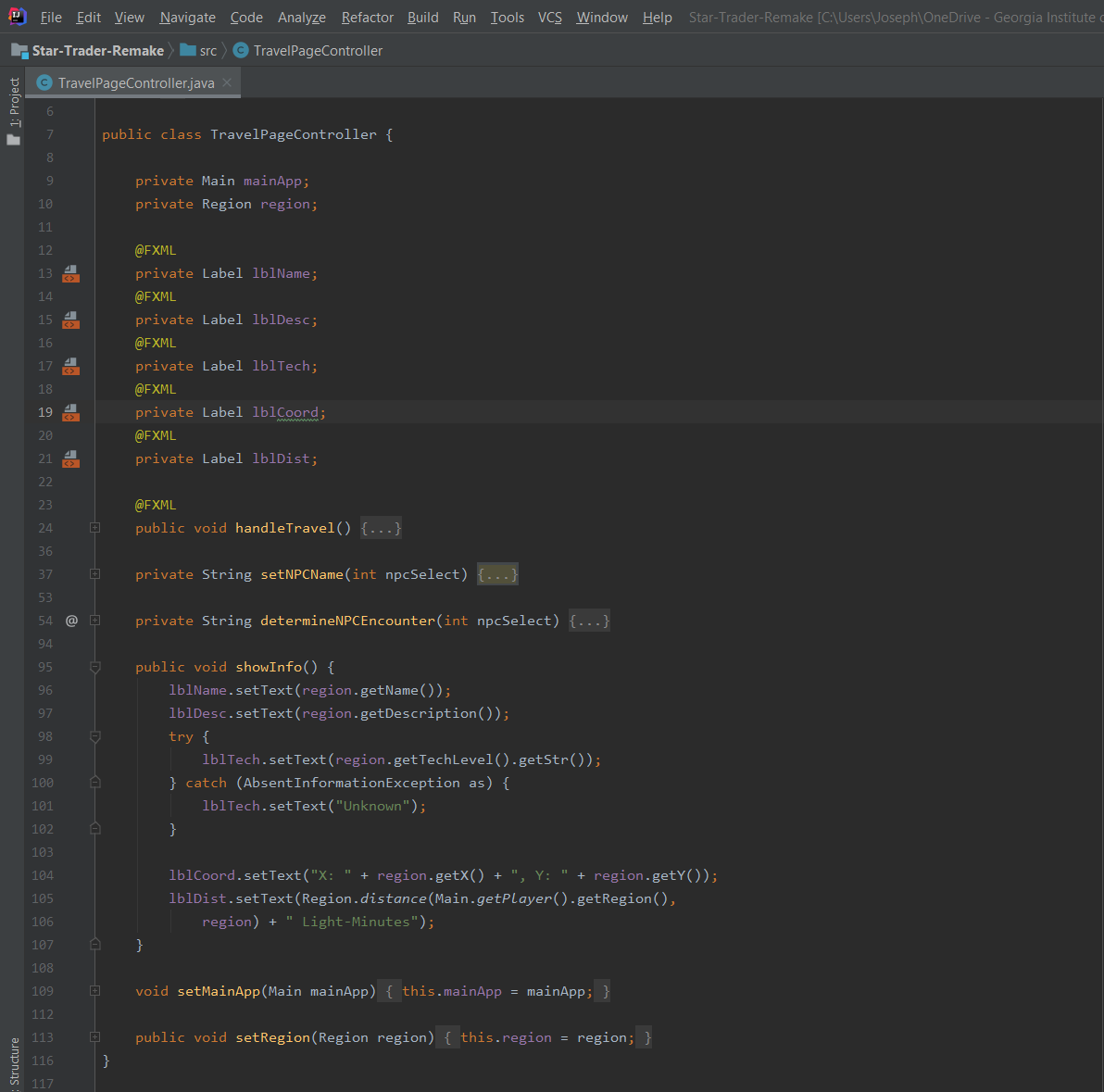
GRASP Principle #1: Creator



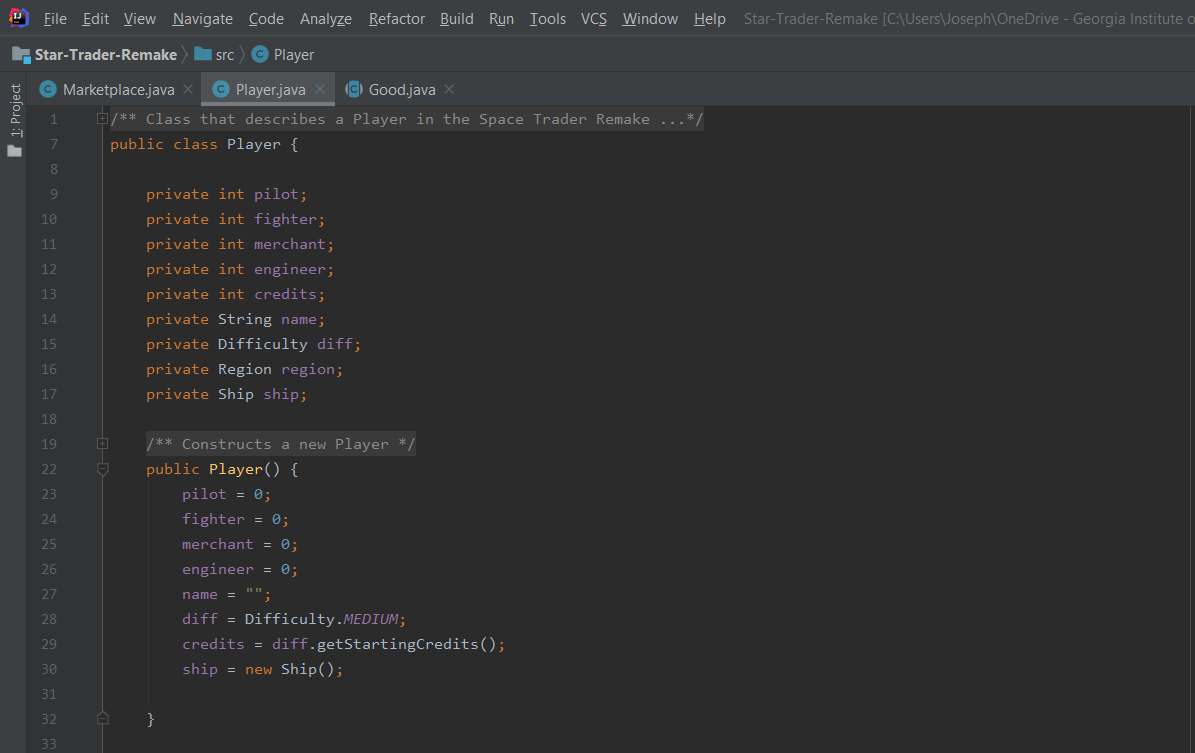
This is a screenshot of the Region class for our game. As shown, a Region object creates/initializes a Marketplace object. This is the Creator GRASP pattern because the Region object contains a Marketplace object.

GRASP Principle #2: Controller



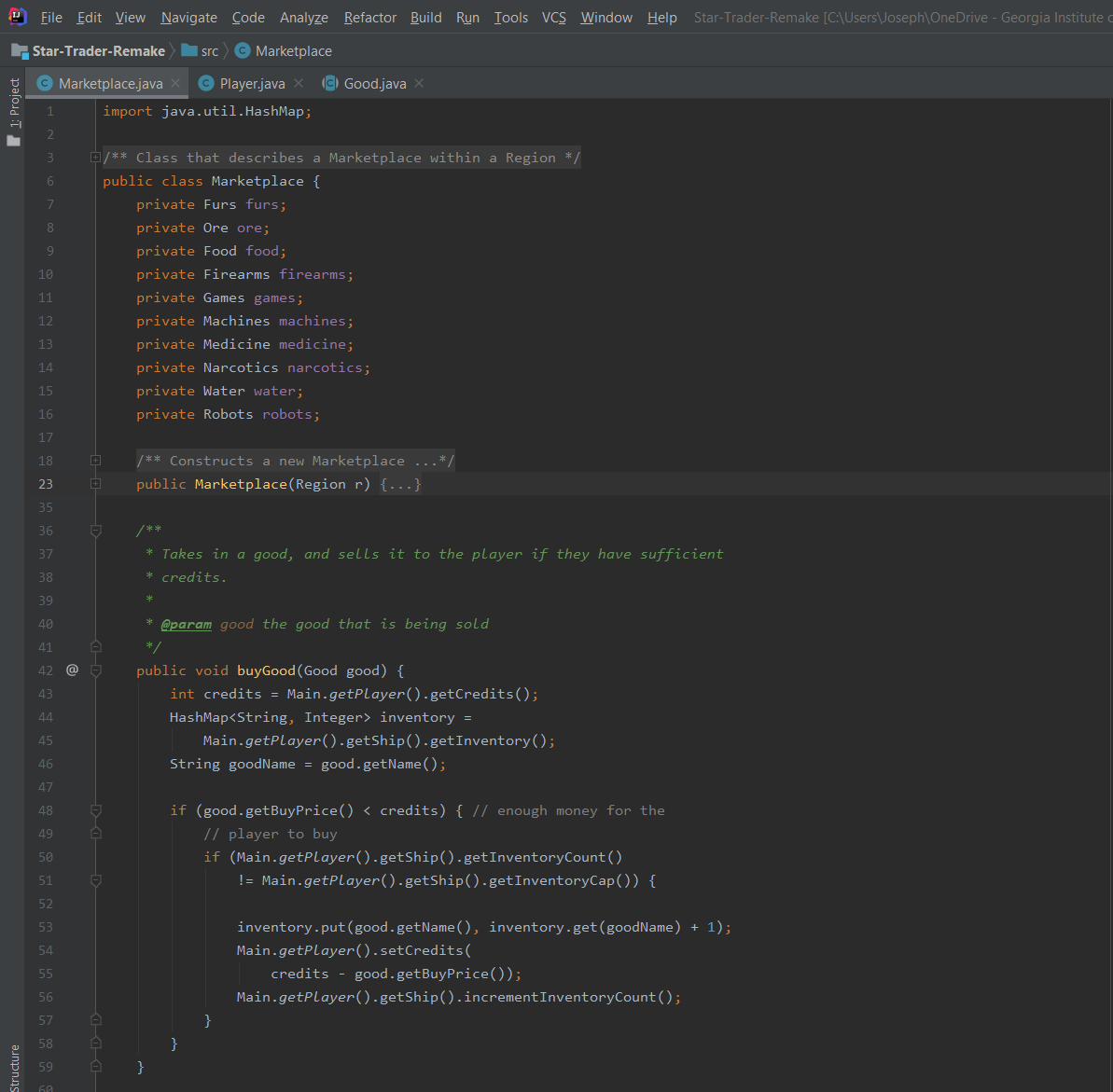
This is a screenshot of the TravelPageController class. As can be seen, this is a class that handles UI logic. Because it is used to pass on information and coordinate input to the model classes, it fulfills the controller pattern.

GRASP Principle #3: Information Expert



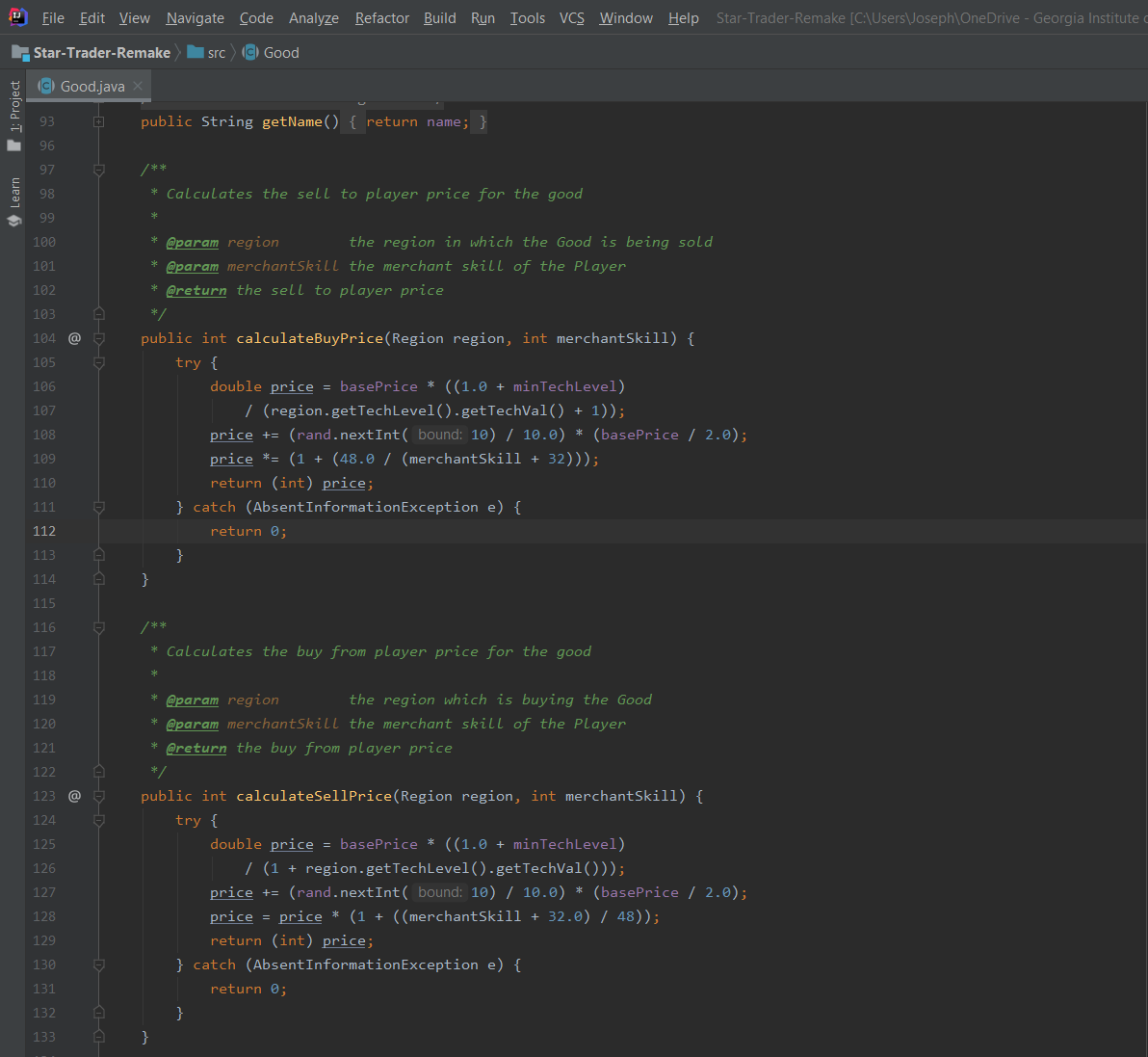
This screenshot shows the Player class. This class has all the information necessary to contain a Ship class. In fact, it contains the only Ship object in the entire game. Thus, it fulfills the information expert pattern.

GRASP Principle #4: Polymorphism



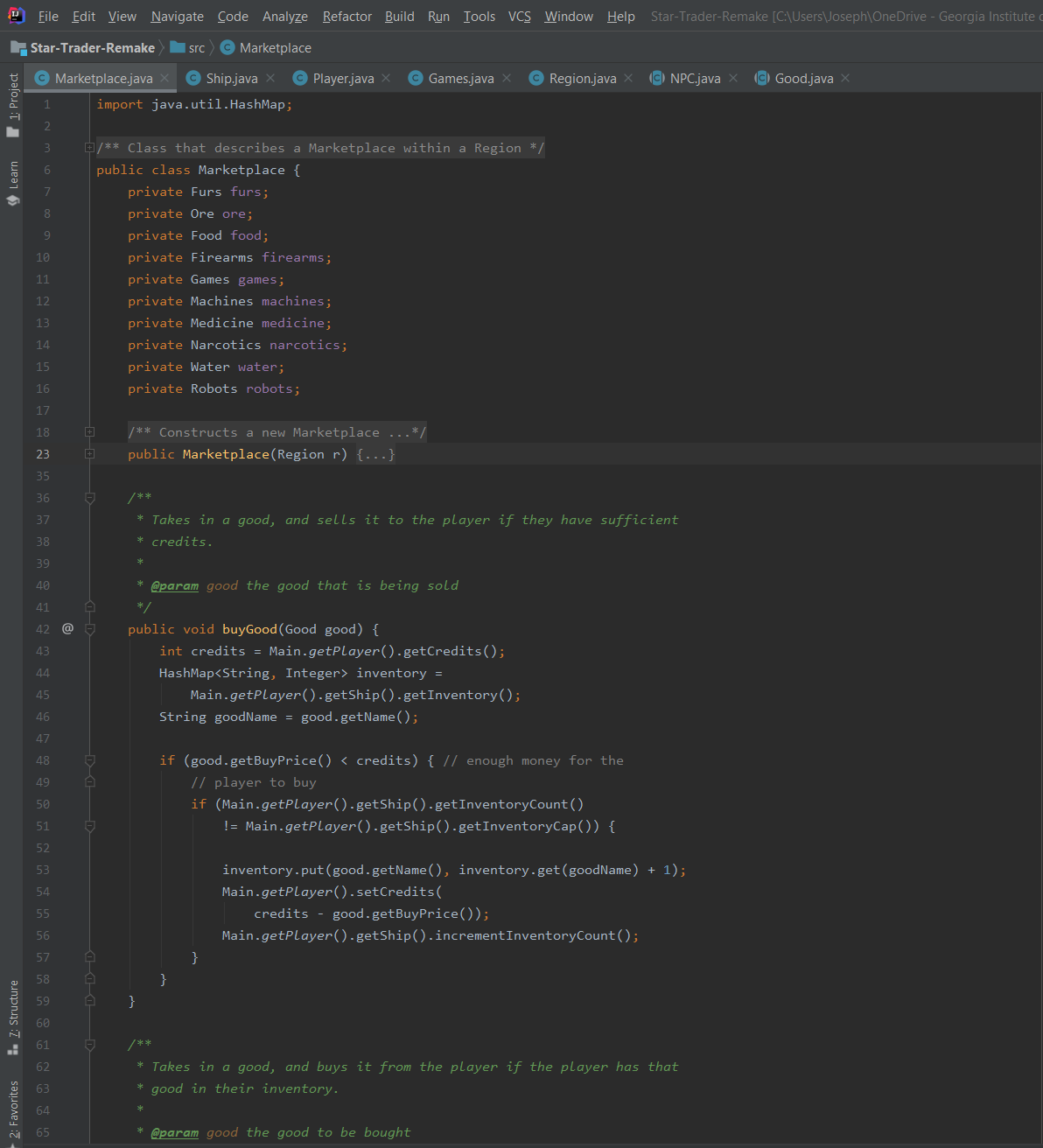
This screenshot shows the Marketplace class. Inside this class, there are multiple methods that take in a Good, which is an abstract class. Because the actual instances are more specific classes that extend Good, this displays polymorphism.

GRASP Principle #5: Low Coupling



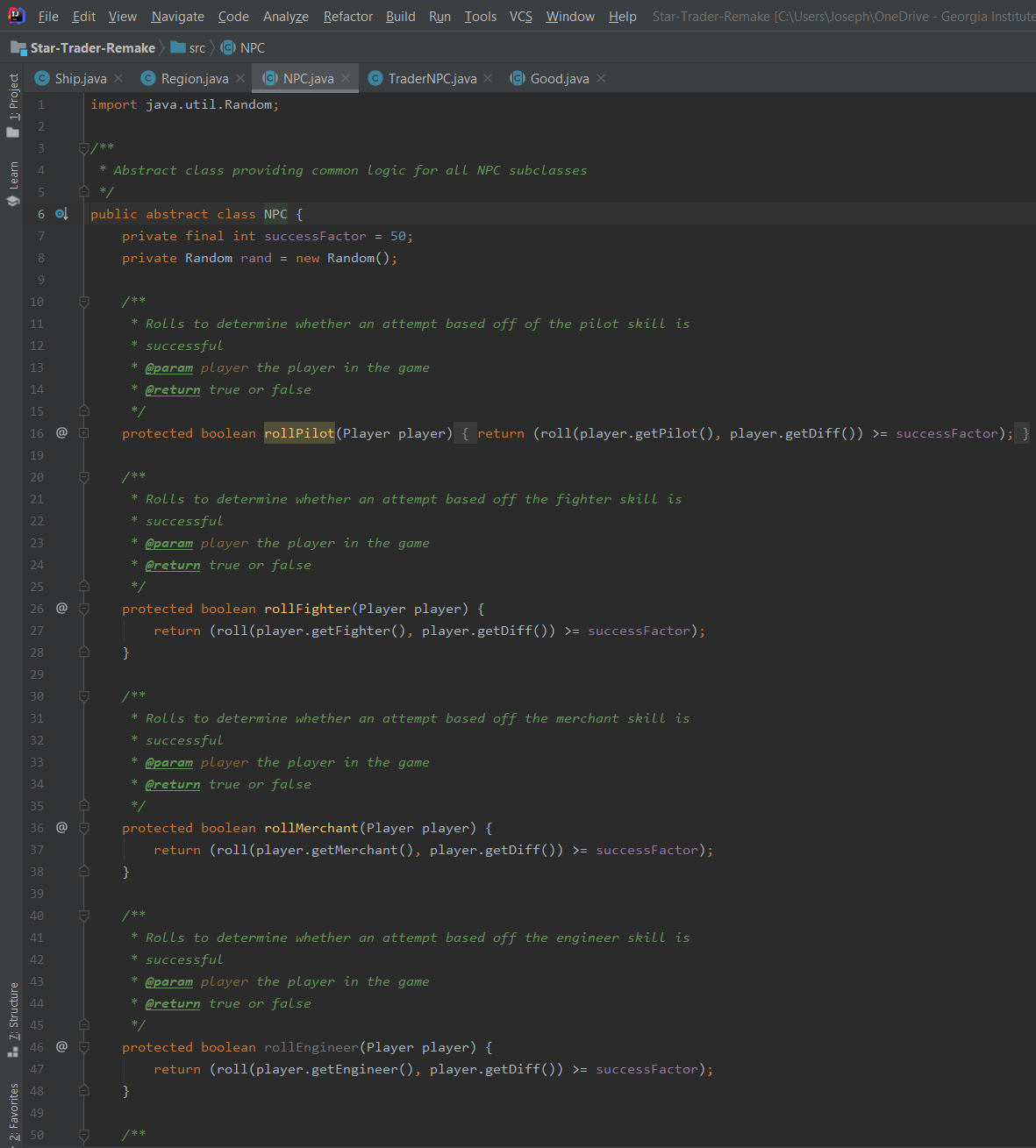
This screenshot shows the Good class. In this class, there are two methods used to calculate the buying and selling price for the good. The alternative way of handling this problem would’ve been to have the buying and selling price calculated inside the Marketplace class. However, the solution presented demonstrates low coupling because it prevents fields of the good class from being populated by the Marketplace class.

SOLID Principle #1: SRP



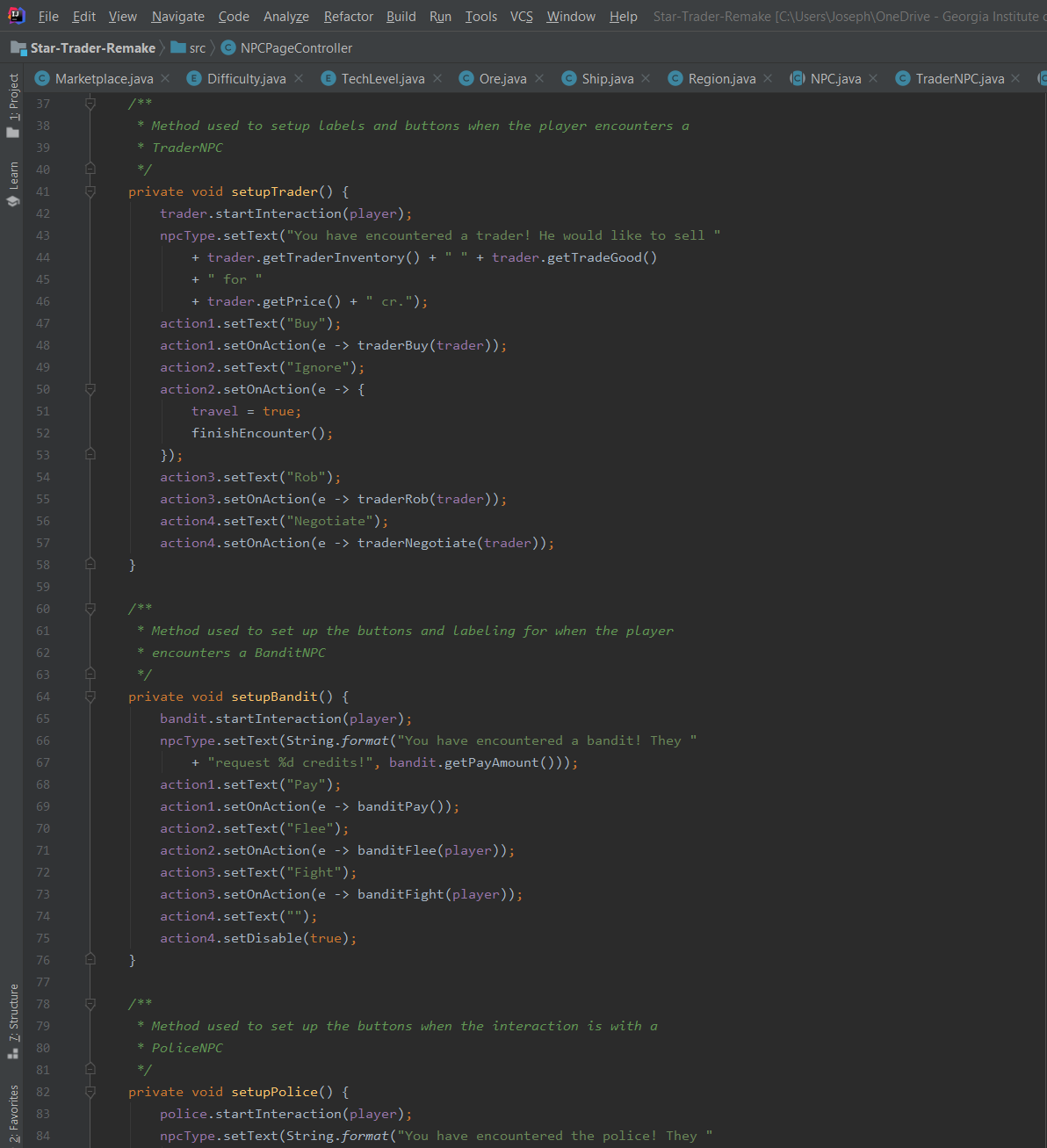
This screenshot shows the Marketplace class. Its responsibility is to handle buying and selling of any Goods as well as selling fuel/repairs for a Ship. Because its responsibilities do not include anything extraneous, this class demonstrates the Single Responsibility Principle.

SOLID Principle #2: OCP



This picture is a screenshot of the NPC abstract class. It provides common logic for all NPC types. This demonstrates the Open/Closed principle because the NPC class itself should not need to be changed when a new NPC is added.

SOLID Principle #3: DIP



This screenshot shows the NPCPageController. In each of its methods, it forwards the handling of actual logic to the appropriate NPC class. However, because the controller work is handled by the controller class, it does not depend heavily on the NPC subclasses.