Refactoring Code Smells

1. **Code Smell #1:** Duplicated Code
   Pre-Refactor: Within the GetGoals class, the calorieGoal method would return an object which contained the progress of the user's caloric goals. This method was specific for caloric purposes and the calculations within it required other business classes pertaining to specifically calories. With this iteration's implementation of nutrient tracking (carbs, fats, etc), we would have to have a similar method for each of those nutrients. However, having 8 copies of similar code produces duplicated code smells.
   Refactor Applied: Change the method name and its functionality with a new parameter to handle any of the new nutrients as well as calories.
   Post-Refactor: We renamed the class to nutrientGoal which accepted an extra int parameter to determine if we were dealing with calories or a certain nutrient. Depending on this int, different calls to different methods were made to track the user's progress for that particular goal by using a switch statement.

2. **Code Smell #3:** Primitive Obsession
   Pre-Refactor: Ambiguous code numbers were used to determine the type of nutrient being used (ie: Fat = 0, Carbohydrates = 1, etc) with no other way to determine which nutrient was being used. This was inefficient and required multiple parameters for method calls to anything pertaining to nutrients, including the mandatory parameter of a code number.
   Refactor Applied: Create business objects to encapsulate relative primitive data types.
   Post-Refactor: We created a Nutrient class which was a parent class to all of this iteration's new nutrients. These subclasses contained necessary information pertaining to its respective type such as its name, the goal the user wishes to achieve, and the daily recommended amount based on the user's goal.

3. **Code Smell #10:** Switch Statements
   Pre-Refactor: Some methods relied on switch statements to distinguish the types of objects they are. To illustrate easier, we'll use the aforementioned nutrientGoal method (in GetGoals class) as an example. This method used a switch statement to determine the type of nutrient that was being dealt with and used different method calls depending on the nutrient. As mentioned in class and in our notes, switch statements are rare and should not be used to distinguish classes or objects.
   Refactor Applied: Replace switch statements with subclass method call(s) which automatically distinguishes subclasses from one another due to polymorphism.
   Post-Refactor: We utilized the subclasses of the Nutrient class and polymorphism. We did this by replacing the code number with a Nutrient object for the parameters in nutrientGoal. Calls to the Nutrient object's methods would always work since all subclasses shared those methods, and each method would work differently depending on the subclass. This eliminated the long switch statement which contained over 8 cases of varying code and replaced it with a few of lines of code which accomplished the same task.