**Instructions**

You will build a simple app to show that you understand classes, objects, subclassing, and the *Factory Method* pattern. You will design and implement five classes:

* One base class that is a generic version of some kind of noun or item.
* Three subclasses that are more specific versions of the noun or item in the base class.
* One factory that can generate objects for the three subclasses depending on which is requested.

You will wrap all of this in a simple, single-screen app that uses labels to show the values of the created objects.

**The Object Model**

Your classes must include the following functionality:

* At least one numeric data member in the base class.
* At least one string data member in the base class.
* Accessor and mutator methods in the base class to get and set the values of the data members. These methods **must not** be overridden in the subclasses.
* A method **i**n the base class that performs some kind of calculation or manipulation based on the data members,**returning some kind of result**. (Not void) Even though the calculation method in the base class will be overriden, it should still exist, and it should still do a calculation!
* Overridden versions of the previous method in the subclasses. **Each class must do something slightly different**, such as altering the method of calculation, to show how functionality can change in subclasses.
* At least one additional data member (of any type) and associated accessor and mutator methods in each of the subclasses. This data member should make sense as being unique to the specific subclass.
* The factory should include a class (aka static) method to instantiate and return objects of the subclasses, given an argument that specifies which type.

Your classes should represent any noun or item of your choosing as long as you don't recreate any examples given in the course material.

**Visual Features**

Your app should use the factory to instantiate one of each of the three subclasses and add their string and numeric member values to three labels placed on the primary view. When adding the values to the labels, be sure to use the accessor methods provided by the base class to get the members, and **do not** just hard-code the values in the labels. For example:

turkeylabel.text = [NSString stringWithFormat:@"The %@ are combined and you must %@.", [thanksgivingMeal ingredients], [thanksgivingMeal instructions]];

Your app should also call the custom mutator method (such as setPounds on the turkey recipe class) to set the unique data member for each class, and then use the corresponding accessor to show the value in another set of labels:

turkeycustom.text = [NSString stringWithFormat:@"The %d pound turkey needs to cook %d minutes.", [thanksgivingMeal pounds], [thanksgivingMeal calculateCookingTimeMinutes]];