# Artisanal Classes & Objects

Today you will hand-craft a custom-made class, several instances of that class (objects), and accessor and mutator methods of your choice. The group with the most creative, complete, and correct code will win a prize as described by your instructor. You may refer to section 8.1 - 8.3 of you textbook, to your notes, and to any classroom posters or examples. Your group must work independently on this project (no collaborating with other groups).

Check to make sure your packet of supplies contains the following:

* A template for your work
* A blank sheet of paper for client code
* A colored marker
* Tape or glue
* 1 large sticky note
* 1 medium sticky note
* 2 small sticky notes
* 2 dialogue bubbles

Before you begin constructing your custom-made class and objects, use the space below to brainstorm some ideas with your group. A good example will (1) clearly show the relationship between the class and instances of that class, (2) will require use of 3 – 5 fields, (3) will use instance methods that serve as good examples of accessor and mutator methods.

## Step 1

Choose a descriptive class name. Using proper syntax and naming conventions, fill in your public class header at the top of the template sheet. (Make sure you select a group member with neat handwriting to do this part!)

## Step 2

What characteristics define the different objects in your class? On your template sheet, declare between 3 -5 fields that you want every object of this class to have.

## Step 3

Because you are custom-designing your own class of objects, you can make it possible to create an object in any location by writing a constructor. What information should your objects always contain when being constructed? Discuss this with your group, then write your constructor on your template sheet.

## Step 4

What are some common behaviors for each of the objects in your class? As a group, decide on an appropriate instance method that will provide information about the object, but not change the object. Write this instance method on your template sheet.

## Step 5

On your template sheet, write an instance method that changes the state of the object (changes its data).

## Step 6

Using the index card with a dialogue bubble, write out what the accessor code is telling the object to do. For example, s.length tells the string object “Return the length of String s.” For extra credit, your group may opt to use the second index card to write out what the mutator code is telling the object to do.

## Step 7

Underline the accessor method using the colored marker provided. Circle the mutator method.

## Step 8

Using the colored marker provided, draw a star next to the implicit parameters in your instance methods.

## Step 9

Using two of the small sticky notes provided, re-name your mutator method using the “set” convention, and re-name your accessor method using the “get” or “is” convention. If you forget these naming conventions, check section 8.2 in your textbook.

## Step 10

Using one of the medium-sized sticky notes provided, insert a comment explaining what your second instance method does.

## Step 11

If you don’t write a toString method in your class, your class will use a default version that returns the class name followed by an @ sign and its location in memory (represented by a bunch of numbers). This could be confusing if you are trying to print out your object to the console with system.out.println. Since this isn’t what you usually want when you convert your object to a String, you should always write your own toString method in your class.

Discuss with your group what your object should look like when you call the toString method, then write your custom toString method on the large sticky note provided to you. You must include a comment explaining what the code does. Insert the toString method in the proper location on your template sheet.

## Step 12

On the blank sheet of paper given to you, write some client code that will interact with the objects you created. You must include comments explaining what your code does.