

1. Write a quick class "Puppy" with a couple of instance variables that make sense to you, a constructor of some kind, and a toString method. Also implement the following method:

```
public int compareTo(Puppy other)
```

This method will compare the current object with the parameter and return either a negative value, 0, or a positive value.

The compareTo method will be used to put the puppies in order according to their "size". How the comparison is actually performed is up to you, but a.compareTo(b) should be negative if a is "smaller" than b, positive if a is "larger" than b, and 0 if a is the same size as b.

2. Write a class called "Litter" that represents a SORTED list of Puppies. (We will not use any sorting algorithm; rather, we will insert puppies carefully so that they are always ordered in the list.) The order of the list will be determined by the compareTo method that you wrote in the Puppy class.

- Use a private array of type Puppy as the only instance variable.
- Implement these methods for the Litter class:

- \* default constructor -- creates an array of size 0.

- \* add -- takes a Puppy parameter and inserts a reference to the Puppy into the array. This method MUST maintain the order of the elements in the array based on the results of the compareTo method. So the parameter might need to be inserted in the beginning, at the end, or somewhere in the middle. (Note: The array will have to be re-sized in order to accommodate the new Puppy, and this will involve making a bigger array, copying most of the data into the bigger array, etc.)

- \* getDeepCopy -- returns deep copy of the array of Puppies

- \* getShallowCopy -- returns shallow copy of the array

- \* getReferenceCopy -- returns reference copy of the array

- \* remove -- takes a Puppy parameter and removes it from the array. (The array will have to be resized.) If the Puppy is not there, then do nothing.