csTask49 Assignment

The activities in this assignment require you to create and modify Java programs using BlueJ. The BlueJ project csTask49 already contains various classes for you to work on.

Follow the directions given for each part. At the end of class, all modified programs (class files) must be in your project, and the project must be copied to your shared Dropbox folder.

When questions are posed, write your responses as full sentences in the Readme.txt file in your project.

There are two parts to this assignment:

I. Exploring Inheritance
II. A Sorted Integer List

I. Exploring Inheritance

Open the BlueJ project csTask49 and study the Dog class — notice what instance variables and methods are provided. Use BlueJ to run the Dog constructor to instantiate a Dog on the workbench. Run each method to see what it does.

Study the code for the Labrador and Yorkshire classes that extend Dog. Don't worry about constructing objects of these classes quite yet.

DogTest is a simple driver program that creates a Dog and makes it speak. Study the DogTest class by compiling and running it to see what it does.

Now modify these classes as follows:

1. Add statements in DogTest (after the statements that create and make the dog speak) to create a Yorkshire and a Labrador and make each of them speak. Note that the Labrador constructor takes two parameters: the name and color of the Labrador, both strings. *Don't change any files besides DogTest.*

   (a) When you compile DogTest with your changes, you should get an error saying something like

   ```
   Constructor Dog in class Dog cannot be applied to given types
   ```

   If you are not already displaying line numbers in BlueJ, select Options/Preferences/ then check the "Display line numbers" box. If you look at line 17-20 of the Labrador class, the constructor sets the color (the instance variable added by the Labrador subclass) only. Why is this error occurring? (*Hint: What call must be made in the constructor of a subclass?*)

   (b) Fix the problem (which really is in Labrador) so that DogTest creates and makes the Dog, Labrador, and the Yorkshire each speak.

2. Add code to DogTest to print the average breed weight for both your Labrador and your Yorkshire. Use the avgBreedWeight() method for both. What error do you get? Why? Fix the problem by adding the needed code to the Yorkshire class. Yorkshires are small dogs, so make the average breed weight lower than for the Labrador class. Why does it make sense that the breedWeight variable is static?

3. If we want every subclass of Dog to have this avgBreedWeight method (seems like a good idea…), then we should add an abstract method `int avgBreedWeight()` to the Dog class that returns the average breed weight. Remember that in an abstract method the word `abstract` appears in the method header after `public`, and that the method does not have a body (just a semicolon after the parameter list). It makes sense for this to be abstract, since Dog has no idea what breed it is. Now, because there is an abstract method in the Dog class, every subclass of Dog must have an avgBreedWeight method; since both Yorkshire and Labrador do now (you added one to the Yorkshire class in step 2, above), you <u>should be</u> all set, but…

   (a) When you compile your changes, you should get an error in Dog (unless you made more changes than described above). This is because only abstract classes can have abstract methods, so change your Dog <u>class</u> to be abstract by adding the reserved word "abstract" into the class header after "public". Now recompile DogTest, and…

   (b) You should get another error, this time in DogTest. Read the error message carefully; it tells you exactly what the problem is. This problem can be fixed by either (i) eliminating the instantiation of dog1 in DogTest, or (ii) creating a new non-abstract class RealDog (ha ha) that extends Dog with its own constructor and overrides avgBreedWeight, the only abstract method of Dog. Which of these two approaches makes more sense to you in this case? Explain your answer,

II. A Sorted Integer List

The IntList class contains code for an integer list class. Study it; notice that the only things you can do are: create a list of a fixed size and add an element to a list. If the list is already full, a message will be printed. ListTest is a driver class that creates an IntList, puts some values in it, and prints it. Compile and run it to see how it works.

Now write a class SortedIntList that extends IntList. SortedIntList should be just like IntList except that its elements should always be in sorted order from smallest to largest. This can be done efficiently by inserting elements into their proper place when they are added to the list, rather than just putting them at the end of the array as in IntList.

To do this you'll need to do two things when you add a new element:

• Traverse the array until you find the place where the new element should go. Since the list is already sorted you can just keep looking at elements until you find one that is at least as big as the one to be inserted.
• Move down each element that will go after the new element, that is, everything from the one you stop on to the end.

This creates a slot in which you can put the new element. Be careful about the order in which you move them or you'll overwrite your data! Now you can insert the new element in the location you originally stopped on. All of this will go into your add method, which will override the add method for the IntList class. (Be sure to also check to see if you need to expand the array, just as in the IntList add method.) What other methods, if any, do you need to override?

To test your class, modify ListTest so that after it creates and prints the IntList, it creates and prints a SortedIntList containing the same elements (inserted in the same order).