1 Introduction

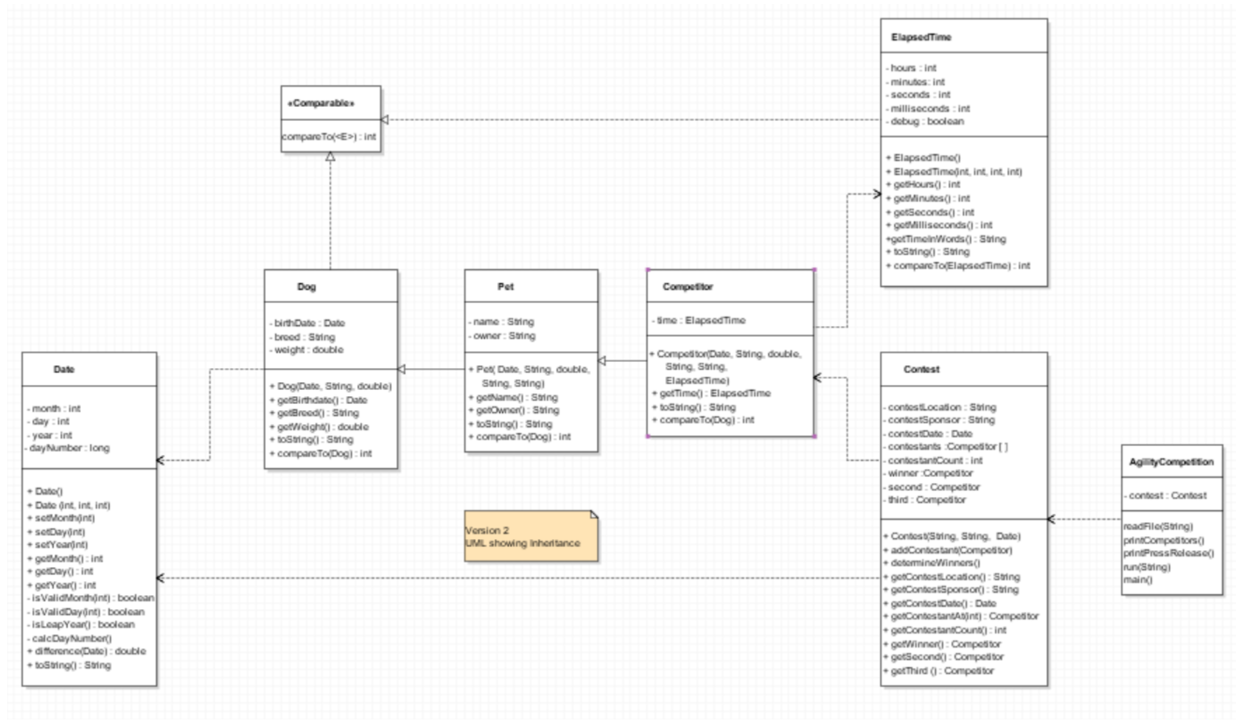
Last week we took another look at the Agility Contest from CSCI 110. I gave you a completed UML and you coded the classes using it. This week you will modify that to use inheritance. You can copy your lab from last week, or download the one I have provided.

2 Copying Last Week's Lab

In Windows explorer, right-clock the AgilityCompetitionV1 folder and choose copy. Then move your mouse pointer to an empty part of the window and right-click and choose paste. You should now have a folder called AgilityCompetitionV1(copy). Rename it AgilityCompetitionV2. Start Violet UML editor and open the AgilityCompetitionV1.class.violet.html that is in the AgilityCompetitionV2 project folder.

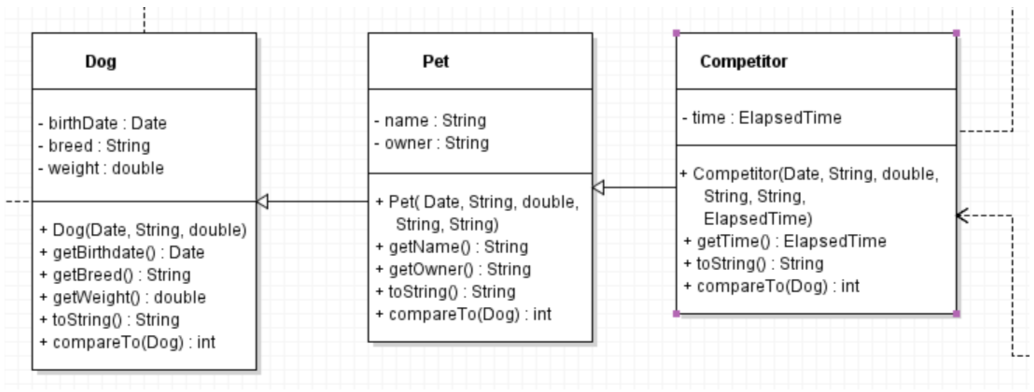
3 Modifying the UML

All dogs do not have the attributes that we included in Dog class. So we are going to move some to new classes. All dogs have a birth date, weight, and breed. Only a Pet has an owner and a name, and only a Competitor has an elapsed time on the agility course. We are going to separate this stuff out into different classes. Here is what the new layout will look like.



You need to add 2 classes between Dog and Contest. From the menu on the right make sure that Select is highlighted. Delete the connection between Dog and Contest by clicking on it and pressing the [Del] key. Click on the connection between Dog and ElapsedTime. Move the ElapsedTime class diagram over to the right. Competitor will depend on it instead of Dog. Click on the Class icon on the right and add two more classes between them. Click Select on the menu. You will want to move things to the right to make room. Double click on the empty class to the right of Dog and type Pet in the top part of the box and click OK. Double click on the one to the right of Pet and type Competitor in the top box.

Now you can redraw the “Contest depends on Competitor”, “Competitor inherits from Pet” , “Pet inherits from Dog”, and “Competitor depends on ElapsedTime” relationships. Just choose the correct connector from the palette, click on the first class and hold the mouse button down, move the mouse to the second class and release the button. Now let’s redistribute the properties, AND change the way the Constructors all look. Competitor will have to receive a value for its property, but it has to receive values for its superclass as well. Pet will pass the first 3 properties to Dog. Here is what the UML for those 3 classes should look like when you are done. Pet and Competitor inherit the fact that Dog implements Comparable, so they will have their own compareTo(Dog) methods that Override the one in Dog. To override them they must have the same signature that the original class had.



Now we need to make one more change and that is to the property and associated method parameters in Contest. Everywhere that you see Dog in the Contest class you need to replace it with Competitor. Save the new UML as AgilityCompetitionV2.class.violet.html

4 Implementing the new Classes

Start NetBeans. If there is an AgilityCompetition currently open, Close it by right-clicking on the project name and choose Close from the context menu. Now open your new Agility-CompetitionV2. Do not have both projects open at the same time. First we will add the two classes. Right-click on the package and choose Add =>Java Class. Name the first one Pet then do it again and name the second new class Competitor. Now follow these steps carefully.

1. Add ”extends Dog” to the class declaration of Pet.

```
public class Pet extends Dog {
```

2. Add ”extends Pet” to the class declaration of Competitor.
3. In Dog, select and Cut the properties name and owner, click on the Pet tab and Paste them.
4. In Dog, select and Cut the property time, click on the Competitor tab and Paste it.
5. In Dog, select and Copy the constructor and the JavaDoc for it.
6. In Pet paste the entire constructor. Change Dog to Pet.
7. In Competitor, do the same and change Dog to Competitor.
8. Go back to Dog and edit the constructor and comments so that it matches the following:

```
/**
 * Parameterized Constructor for the Dog class.
 * @param birthDate the dog’s date of birth (Date)
 * @param breed the dog’s breed
 * @param weight the dog’s weight
 */
public Dog(Date birthDate, String breed, double weight){
   this.birthDate = birthDate;
   this.breed = breed;
   this.weight = weight;
}
```

9. Go to Pet and edit the constructor and comments so that it matches the following:

```
/**
 * Parameterized Constructor for Pet.
 * @param birthDate the dog’s date of birth (Date)
 * @param breed the dog’s breed
 * @param weight the dog’s weight
 * @param name the pet’s name
 * @param owner the pet’s owner’s name
 */
public Pet(Date birthDate, String breed, double weight,
        String name, String owner){
    super(birthDate, breed, weight);  // call Dog’s constructor
    this.name = name;
    this.owner = owner;
}
```

10. Go to Competitor and edit the constructor there so that it matches this:

```
/**
 * Parameterized Constructor for Competitor
 * @param birthDate the dog’s date of birth (Date)
 * @param breed the dog’s breed
 * @param weight the dog’s weight
 * @param name the pet’s name
 * @param owner the pet’s owner’s name
 * @param time the time that the competitor ran the course in
 */
public Competitor(Date birthDate, String breed, double weight,
                  String name, String owner, ElapsedTime time) {
    // call Pet’s constructor
    super(birthDate, breed, weight, name, owner);
    this.time = time;
}
```

11. Cut and paste the accessors for the properties that we moved from Dog into the other classes. Cut and Paste getTime() from Dog to Competitor. Cut and paste getName() and getOwner() from Dog to Pet.

12. Now copy and paste the toString(), compareTo() and main() from Dog to the other two classes.

13. In Dog, edit the toString() to match the following:

```
/**
 * toString allows an object to be directly printed by returning a
 * String that can be printed to the console or to a file.
 * This one is  written in the input file format.
 *
 * @return a formatted string representing the values of the attributes
 * for a dog object.
 */
 @Override
 public String toString() {
     StringBuilder str = new StringBuilder();
     String eol = System.lineSeparator(); // end of line for this OS
     str.append(String.format("%4d %4d %4d %-25s%s%7.2f",
                birthDate.getDay(), birthDate.getMonth(),
                birthDate.getYear(), breed, eol, weight));
     return str.toString();
    }
```

A Dog can only send out its own properties, so everything else must go.

14. In Dog, edit the last else in the compareTo(Object) to match the following: (leave the top of the method as it is.)

```
} else {
   // if they finished in the same time
   // then we will sort by dog's name within the time comparison
   // this uses the String compareTo() method
   if (comparison == 0) {
        comparison =
           this.birthDate.compareTo(((Dog)that).getBirthDate());
   }
}
```

15. Edit the declarations in the unit test in Dog to match the following:

```
Dog dog1 = new Dog(date1, "Toy Poodle", 10.2);
Dog dog2 = new Dog(date2, "Bulldog", 20.4);
```

    Right-click to run the unit test in this file (Run File) Does the output make sense?

16. In Pet, edit the toString to match the following:

```
/**
 * toString allows an object to be directly printed by returning a
 * String that can be printed to the console or to a file.
 * This one is  written in the input file format.
 *
 * @return a formatted string representing the values of the attributes
 * for a dog object.
 */
 @Override
 public String toString() {
    StringBuilder str = new StringBuilder();
    str.append(String.format("%s %-15s%-12s",
            super.toString(), name, owner));
    return str.toString();
}
```

    This toString() calls it's superclass's toString() so the output from Dog will be first here and then the output from Pet is added to that.

17. In Pet, the compareTo(Object) method is written to override the one in Dog so the objects are technically Dogs and not Pets. While “this“ object knows that it is a Pet “that“ object doesn’t so we have to cast that Dog to be a Pet. Edit the last else in compareTo(Object) in Pet to match the following: (leave the top of the method as it is.)

```
} else {
    /* We will sort by owners’s name. Since owner is a String,
       this uses the String’s compareTo() method */
    comparison = this.owner.compareTo(((Pet)that).getOwner());
    /* if they have the same owner, we will sort by dog’s name within
       that. So all of the Pets that belong to a single owner will be
       together */
    if (comparison == 0) {
         comparison = this.name.compareTo(((Pet)that).getName());
    }
}
```

18. Edit the declarations in the unit test in Pet to match the following:

```
Pet dog1 = new Pet(date1, "Toy Poodle", 10.2, "Eudora", "Anne");
Pet dog2 = new Pet(date2, "Bulldog", 20.4, "Buster", "Jesse");
```

Right-click to run the unit test in this file (Run File). Does the output make sense?

19. In Competitor, edit the toString() to match the following:

```
/**
 * toString allows an object to be directly printed by returning a
 * String that can be printed to the console or to a file.
 * This one is  written in the input file format.
 *
 * @return a formatted string representing the values of the attributes
 * for a dog object.
 */
 @Override
 public String toString() {
    StringBuilder str = new StringBuilder();
    str.append(String.format("%s %d %d %d %d",
            super.toString(), time.getHours(),
            time.getMinutes(), time.getSeconds(),
            time.getMilliseconds()));
    return str.toString();
}
```

Again, the toString() here is receiving a string from its super class, Pet, adding on to it and returning that String to the calling module.

20. In Competitor, edit the compareTo(Object) to match the following: (leave the top of the method as it is.)

```
} else {
     /* we sort by time. There is a compareTo()
        in the ElapsedTime class that sorts on the 4 fields. */
     comparison = this.time.compareTo(((Competitor)that).getTime());
}
return comparison;
```

21. Edit the declarations in the unit test in Competitor to match this:

```
Competitor dog1 = new Competitor(date1, "Toy Poodle", 10.2, "Eudora",
        "Anne", new ElapsedTime(0, 2, 35, 40));
Competitor dog2 = new Competitor(date2, "Bulldog", 20.4, "Bubba",
        "Jesse", new ElapsedTime(0, 5, 24, 50));
```

Right-click to run the unit test in this file (Run File). Does the output make sense?

5 Change Contest and the Driver to use Competitor instead of Dog

The easiest way to do this is to use Edit = >Replace. Put Dog in the find box and Competitor in the replace with box. It will even fix the comments. Open the Contest.java file and replace all of those, then open the AgilityContestDriver and replace all of those the same way.

6 Time to Run and Test the Whole Application

Time to click the green arrow on the tool bar and See if Plato wins! Well? Did he?

7 What did you learn?

If you downloaded a Lab Project for today's lab, you will find a LabWriteUp.txt document inside it. If the lab instructions asked you to create a project for the lab, download the writeup document from Brightspace and save it to your project folder for today.

Open it in a text editor like notepad (do not use Word!) and answer the questions so that what you learned from the lab is clear. You MUST understand what you have done and why. If you don't understand, ask! Close all open projects in NetBeans.

In Explorer or Finder Right-click on the project folder and choose Send To Compressed file. Name it with your last name and ONLY your last name. Upload that to Brightspace.

There are Videos on how to compress folders on both Windows and Mac in the How Tos in the Welcome Content area. Do NOT use the zip utility in NetBeans. The Archive MUST be a .zip file.

8 How Labs are Graded

Labs are graded as Pass/No-Pass and can be resubmitted one time.

Pass	Projects are done according to the instructions and the Writeup shows a complete understanding of today's lab concepts
No-Pass	Projects are not done according to the instructions and/or Writeup shows very little understanding of the lab concepts