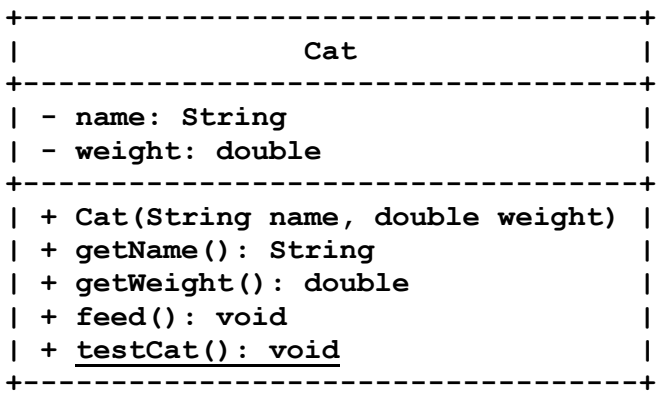
Lab 5 Requirements

Create a new Eclipse workspace named "**Lab5_1234567890**" on the desktop of your computer (replace **1234567890** with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**Question1**", "**Question2**", etc. If you do not remember how to create a workspace or projects, read the "*Introduction to Eclipse*" document which is on iSpace. Answer all the questions below. At the end of the lab, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**Lab5_1234567890.zip**" (replace **1234567890** with your student ID number). Upload the ZIP file on iSpace.

Question 1

Create a class a class **Cat** with the following UML diagram:



Feeding a cat adds 1.0 to its weight.

The **testCat** method is static and is used for testing the **Cat** class. Here is the code for this **testCat** method:

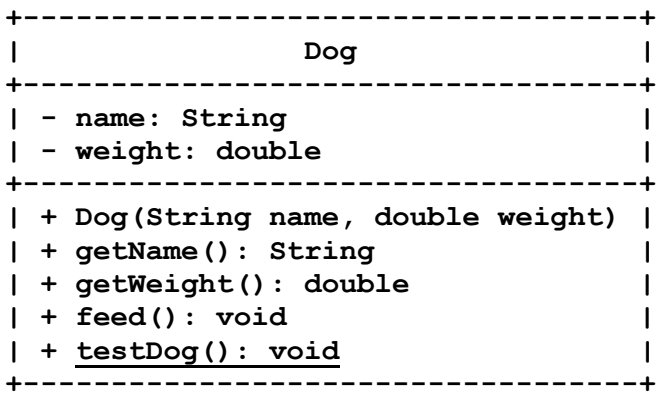
```
public static void testCat() {
	Cat c = new Cat("Meow", 2.0);

	System.out.println(c.getName() == "Meow");
	System.out.println(c.getWeight() == 2.0);
	c.feed();
	// The name is still the same but the weight increased by 1.0:
	System.out.println(c.getName() == "Meow");
	System.out.println(c.getWeight() == 3.0);
}
```

And here is the **Start** class to test the **Cat** class:

```
public class Start {
	public static void main(String[] args) {
		Cat.testCat();
	}
}
```

Question 2

Add to your program a class **Dog** with the following UML diagram:



Feeding a dog adds 2.0 to its weight.

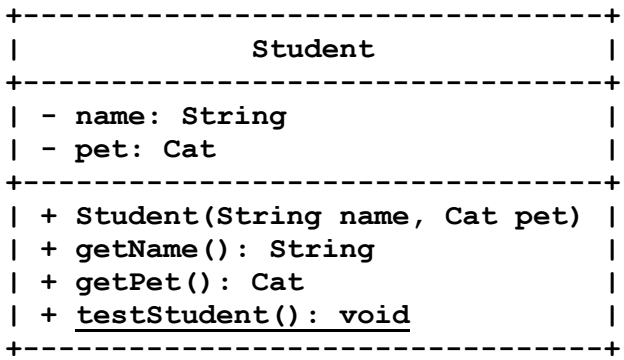
The **testDog** method is static and is used for testing the **Dog** class. Here is the code for this **testDog** method:

```
public static void testDog() {
      Dog d = new Dog("Woof", 2.0);

      System.out.println(d.getName() == "Woof");
      System.out.println(d.getWeight() == 2.0);
      d.feed();
      // The name is still the same but the weight increased by 2.0:
      System.out.println(d.getName() == "Woof");
      System.out.println(d.getWeight() == 4.0);
}
```

Do not forget to modify the **main** method of the **Start** class to test the new **Dog** class too!

Question 3

Add a **Student** class so that a student has a cat as a pet:



Do not forget to write the **testStudent** method of the **Student** class to test the **getName** and **getPet** methods!

Also add to the **main** method of the **Start** class some system tests that test the **Cat** and **Student** classes together.

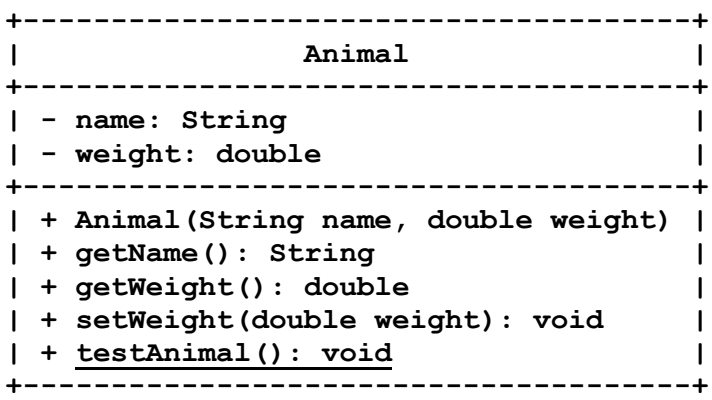
Question 4

If you look at the code of the **Cat** and **Dog** classes above, you will see that they are almost the same. The only differences are that:

- the names of the classes are different;
- the names of the constructors are different (since the names of the classes are different);
- the **feed** methods add a different amount of weight;
- the **testCat** and **testDog** methods are different (since the names of the classes are different).

Everything else (the **name** and **weight** instance variables, the **getName** and **getWeight** methods) is the same. This means that there is a lot of code duplication between the two classes **Cat** and **Dog**. Therefore it is a good idea to create a new class **Animal** that will contain only one copy of that code, and have the **Cat** and **Dog** classes then inherit the code from the **Animal** class.

Add a class **Animal** to the program above, with has the following UML diagram:



The **Dog** and **Cat** classes should then be derived classes from the **Animal** base class (in other words, the **Dog** and **Cat** classes should inherit from the **Animal** class).

Which instance variables, constructors, and methods from the **Dog** and **Cat** classes can be moved to the **Animal** class? Which instance variables, constructors, and methods of the **Dog** and **Cat** classes must stay in these classes? How should they be modified?

After adding the **Animal** class, modify the **Student** class so that the student has an animal as a pet, not a cat.

Do not forget to:

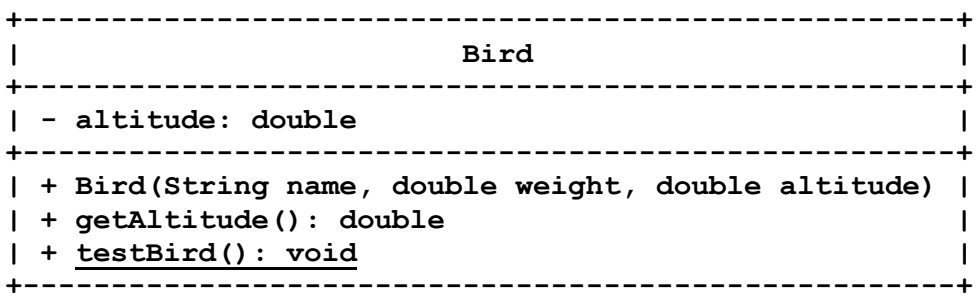
- change the **main** method of the **Start** class to run the unit tests of the new **Animal** class;
- add new tests to the **Student** class to test that you can now use an **Animal** object as the pet of a student;
- add new system tests in the **main** method of the **Start** class that use an **Animal** object as the pet of a student.

Now that the **Student** class uses an animal as pet, can you still use a cat object as the pet of a student? Why or why not?

Can you now use a dog object as the pet of a student? Why or why not?

Question 5

Add a **`Bird`** class to your program. A bird is an animal, therefore your **`Bird`** class must be a class derived from the **`Animal`** class. The UML diagram of the **`Bird`** class is as follows:



The **`altitude`** instance variable represents the altitude at which the bird is flying. Cats and dogs do not fly so they do not have an altitude.

The constructor for the **`Bird`** class takes three arguments: the name of the bird, the weight of the bird, and the altitude at which the bird is flying. The **`altitude`** argument of the constructor is stored into the altitude instance variable of the **`Bird`** class. Where are the **`name`** and **`weight`** of the bird stored? How?

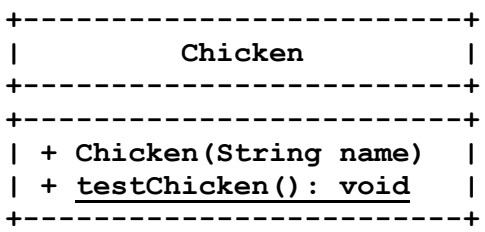
Do not forget to:

- change the **`main`** method of the **`Start`** class to run the unit tests of the new **`Bird`** class;
- add new tests to the **`Student`** class to test that you can now use a **`Bird`** object as the pet of a student;
- add new system tests in the **`main`** method of the **`Start`** class that use a **`Bird`** object as the pet of a student.

Suppose a student has a bird as a pet. Can the student get the altitude of his pet?

Question 6

Add a **`Chicken`** class to your program. A chicken is a bird, therefore your **`Chicken`** class must be a class derived from the **`Bird`** class. The UML diagram of the **`Chicken`** class is as follows:



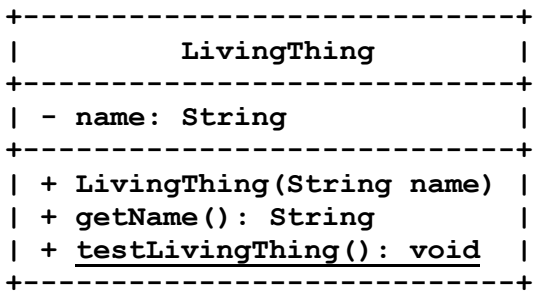
A chicken always has a weight of `5.0` and an altitude of `0.0` (chickens spend all their time on the ground).

Do not forget to:

- change the **`main`** method of the **`Start`** class to run the unit tests of the new **`Chicken`** class;
- add new tests to the **`Student`** class to test that you can now use a **`Chicken`** object as the pet of a student;
- add new system tests in the **`main`** method of the **`Start`** class that use a **`Chicken`** object as the pet of a student.

Question 7

Both the **Student** class and the **Animal** class have a **name** instance variable and a **getName** method, which leads to code duplication between these two classes. To solve this problem, add a new **LivingThing** class to your program, which becomes the superclass for the **Student** and **Animal** classes, and which contains only one copy of the code for the **name** instance variable and the **getName** method. The **LivingThing** class has the following UML diagram:



Change the **Student** class and **Animal** class so that both classes are now derived from the **LivingThing** class. Then remove the **name** instance variables and the **getName** methods from the **Animal** and **Student** classes. The other classes do not change.

Check that all your tests still work.

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **LivingThing** class. Note that there are no system tests using the **LivingThing** class, because the **LivingThing** class does not use any other class (it only uses strings for the **name** instance variable).

LivingThing is now the top-most class. It has two derived classes: **Animal** and **Student**. **Animal** has three derived classes: **Cat**, **Dog**, and **Bird**. **Bird** has one derived class: **Chicken**. **Student**, **Cat**, **Dog**, and **Chicken** do not have derived classes.