Lab 5 Requirements

Create a new Eclipse workspace named "**OOP**" on the desktop of your computer. Create a Java Project named "Lab5". Download the sample code **lab5-src-for-student.zip** from iSpace and unzip it. Next to import the sample code(right click the src -> Import... -> File System -> locate to the unzip folder -> import all the questions to your project) . More details, refer to file *How to Use Eclipse and submit to Autolab for student.pdf* on iSpace. Note: you **must** use the sample code, so AutoLab can work correctly.
Answer all questions below. Note: a question may need answers of the previous question(s), just copy paste the previous code to your new package. At the end of the lab, create a ZIP archive of all the src. Refer to previous labs requirements if you forgot how to compress files.
Extract your zip file to confirm that your zip file contains folders. Rename your src.zip to "**Lab5_1234567890.zip**" (replace **1234567890** with your student ID number). Upload the ZIP file onto AutoLab.

question 1 (Reviewed)

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

```
+-------------------------------------------------------+
|                        Cat                            |
+-------------------------------------------------------+
| - name: String                                        |
| - weight: double                                      |
+-------------------------------------------------------+
| + Cat(String name, double weight)                     |
| + getName(): String                                   |
| + getWeight(): double                                 |
| + feed(): void                                        |
| + testCat(): void                                     |
+-------------------------------------------------------+
```

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() {

        System.out.println ("Test constructor");
        Cat c = new Cat("Meow", 2.0);
        System.out.println("name: " + (c.getName() == "Meow"));
        System.out.println("weight: " + (c.getWeight() == 2.0));

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

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

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

question 2 (Reviewed)

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

```
+--------------------------------------------------+
|                      Dog                         |
+--------------------------------------------------+
| - name: String                                   |
| - breed: String                                  |
| - weight: double                                 |
+--------------------------------------------------+
| + Dog(String name, double weight)                |
| + getName(): String                              |
| + getWeight(): double                            |
| + getBreed(): String                             |
| + setBreed(String): void                         |
| + feed(): void                                   |
| + testDog(): void                                |
+--------------------------------------------------+
```

Feeding a dog adds 2.0 to its weight. The breed of the dog is set through setBreed. The default breed is “unknown”

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

```java
public static void testDog() {
        System.out.println ("Test constructor");
        Dog d = new Dog("Woof", 2.0);
        System.out.println("name: " + (d.getName() == "Woof"));
        System.out.println("weight: " + (d.getWeight() == 2.0));
        System.out.println("breed: " + (d.getBreed() == "unknown"));

        System.out.println ("\nTest setBreed");
        d.setBreed("Poodle");
        System.out.println("name: " + (d.getName() == "Woof"));
        System.out.println("weight: " + (d.getWeight() == 2.0));
        System.out.println("breed: " + (d.getBreed() == "Poodle"));

        System.out.println ("\nTest feed");
        d.feed();
        System.out.println("name: " + (d.getName() == "Woof"));
        System.out.println("weight: " + (d.getWeight() == 4.0));
        System.out.println("breed: " + (d.getBreed() == "Poodle"));
}
```

Remember to modify the **main** method of the **Start** class to test the new **Dog** class too!

question 3 (Mandatory)

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

```
+---------------------------------------------------+
|                    Student                        |
+---------------------------------------------------+
| - name: String                                    |
| - pet: Cat                                        |
+---------------------------------------------------+
| + Student(String name, Cat pet)                   |
| + getName(): String                               |
| + getPet(): Cat                                   |
| + testStudent(): void                             |
+---------------------------------------------------+
```

Do not forget to write the **`testStudent`** method of the **`Student`** class to test the **`getName`** and **`getPet`** methods, and to modify the **`main`** method of the **`Start`** class to test the new **`Student`** class too!

Read the following sample test cases carefully and learn how to write unit testing code and how to write system testing code.

```
public static void testStudent() {
     Cat c = new Cat("Meow", 2.0);
     Student s = new Student("Philippe", c);

     System.out.println(s.getName() == "Philippe");
     System.out.println(s.getPet() == c);
}

public class Start {
      public static void main(String[] args) {
            // Unit tests
            Cat.testCat();
            Dog.testDog();
            Student.testStudent();

            // System tests, to test both classes together.
            Cat c = new Cat("Meow", 2.0);
            Student s = new Student("Philippe", c);

            System.out.println(s.getPet().getName() == "Meow");
            System.out.println(s.getPet().getWeight() == 2.0);
            // Feeding the student's pet cat:
            s.getPet().feed();
            System.out.println(s.getPet().getName() == "Meow");
            System.out.println(s.getPet().getWeight() == 3.0);
      }
}
```

question 4 (Mandatory)

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 the following UML diagram:

```
+-----------------------------------------------------------+
|                    Animal                                 |
+-----------------------------------------------------------+
| - name: String                                            |
| - weight: double                                          |
+-----------------------------------------------------------+
| + Animal(String name, double weight)                      |
| + getName(): String                                       |
| + getWeight(): double                                     |
| + setWeight(double weight): void                          |
| + testAnimal(): void                                      |
+-----------------------------------------------------------+
```

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).

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;

Test Cases:

question 1

```java
	public static void testCat() {

		System.out.println ("Test constructor");
		Cat c = new Cat("Meow", 2.0);
		System.out.println("name: " + (c.getName() == "Meow"));
		System.out.println("weight: " + (c.getWeight() == 2.0));

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

question 2

```java
	public static void testDog() {
		System.out.println ("Test constructor");
		Dog d = new Dog("Woof", 2.0);
		System.out.println("name: " + (d.getName() == "Woof"));
		System.out.println("weight: " + (d.getWeight() == 2.0));
		System.out.println("breed: " + (d.getBreed() == "unknown"));
```

```
        System.out.println ("\nTest setBreed");
        d.setBreed("Poodle");
        System.out.println("name: " + (d.getName() == "Woof"));
        System.out.println("weight: " + (d.getWeight() == 2.0));
        System.out.println("breed: " + (d.getBreed() == "Poodle"));

        System.out.println ("\nTest feed");
        d.feed();
        System.out.println("name: " + (d.getName() == "Woof"));
        System.out.println("weight: " + (d.getWeight() == 4.0));
        System.out.println("breed: " + (d.getBreed() == "Poodle"));
    }
```

question 3

```
    public static void testStudent() {
        System.out.println ("Test constructor");
        Cat c = new Cat("Meow", 2.0);
        Student s = new Student("Philippe", c);
        System.out.println("name: " + (s.getName() == "Philippe"));
        System.out.println("pet: " + (s.getPet() == c));
    }

public class Start {
     public static void main(String[] args) {
        System.out.println("Unit test");
        System.out.println("Test Animal");
        Animal.testAnimal();

        System.out.println("\nTest Cat");
        Cat.testCat();

        System.out.println("\nTest Dog");
        Dog.testDog();

        System.out.println("\nTest Student");
        Student.testStudent();

        System.out.println("\nSystem test");

         // Write the System tests by yourself, to test both classes together.

         //Create a cat named "Meow" with 2.0 pounds here
         //Create a student Philippe who has the above cat here

         //Test Philippe's pet's name is Meow, weight is 2.0. Then
         //feed the cat 1.0, and test Philippe's pet's name is still
         //Meow, but the weight becomes 3.0.
         //Use System.out.println and == to test, the output should all be true as we've
             been practicing.
         //Output for this part should be
         //   pet name: true
         //   pet weight: true
```

```
      }
}
```

question 4

```
    public static void testAnimal() {
          System.out.println ("Test constructor");
          Animal a = new Animal("Blob", 2.0);
          System.out.println("name: " + (a.getName() == "Blob"));
          System.out.println("weight: " + (a.getWeight() == 2.0));

          System.out.println ("\nTest setWeight");
          a.setWeight(3.0);
          System.out.println("name: " + (a.getName() == "Blob"));
          System.out.println("weight: " + (a.getWeight() == 3.0));
    }
    public static void testCat() {
          System.out.println ("Test constructor");
          Cat c = new Cat("Meow", 2.0);

          // The getName and getWeight methods are inherited from Animal.
          System.out.println("name: " + (c.getName() == "Meow"));
          System.out.println("weight: " + (c.getWeight() == 2.0));

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

          // The setWeight method is inherited too.
          System.out.println ("\nTest setWeight");
          c.setWeight(2.0);
          System.out.println("name: " + (c.getName() == "Meow"));
          System.out.println("weight: " + (c.getWeight() == 2.0));
    }


    public static void testDog() {
          System.out.println ("Test constructor");
          Dog d = new Dog("Woof", 2.0);
          System.out.println("name: " + (d.getName() == "Woof"));
          System.out.println("weight: " + (d.getWeight() == 2.0));
          System.out.println("breed: " + (d.getBreed() == "unknown"));

          System.out.println ("\nTest setBreed");
          d.setBreed("Poodle");
          System.out.println("name: " + (d.getName() == "Woof"));
          System.out.println("weight: " + (d.getWeight() == 2.0));
          System.out.println("breed: " + (d.getBreed() == "Poodle"));

          System.out.println ("\nTest feed");
          d.feed();
          System.out.println("name: " + (d.getName() == "Woof"));
          System.out.println("weight: " + (d.getWeight() == 4.0));
          System.out.println("breed: " + (d.getBreed() == "Poodle"));

          System.out.println ("\nTest setWeight");
          d.setWeight(2.0);
          System.out.println("name: " + (d.getName() == "Woof"));
          System.out.println("weight: " + (d.getWeight() == 2.0));
          System.out.println("breed: " + (d.getBreed() == "Poodle"));
    }
    public static void testStudent() {
```

```java
        System.out.println ("Test constructor");
        Cat c = new Cat("Meow", 2.0);
        Student s0 = new Student("Philippe", c);
        System.out.println("name: " + (s0.getName() == "Philippe"));
        System.out.println("pet: " + (s0.getPet() == c));

        System.out.println ("\nTest s1's pet");
        // Creating a new student with a dog as pet:
        Dog d = new Dog("Woof", 2.0);
        Student s1 = new Student("Mr. Li", d);
        System.out.println("name: " + (s1.getName() == "Mr. Li"));
        System.out.println("pet: " + (s1.getPet() == d));

        System.out.println ("\nTest s2's pet");
        // We can now also use an Animal object as a pet:
        Animal a = new Animal("Blob", 5.0);
        Student s2 = new Student("Ms. Chen", a);
        System.out.println("name: " + (s2.getName() == "Ms. Chen"));
        System.out.println("pet: " + (s2.getPet() == a));
    }


public class Start {
    public static void main(String[] args) {
        System.out.println("Unit test");
        System.out.println("Test Animal");
        Animal.testAnimal();

        System.out.println("\nTest Cat");
        Cat.testCat();

        System.out.println("\nTest Dog");
        Dog.testDog();

        System.out.println("\nTest Student");
        Student.testStudent();

        System.out.println("\nSystem test");

        // Creating a new student with a cat as pet:
        System.out.println("Student s0");
        Cat c = new Cat("Meow", 2.0);
        Student s0 = new Student("Philippe", c);
        s0.getPet().setWeight(3.0);
        System.out.println("name: " + (s0.getPet().getName() == "Meow"));
        System.out.println("weight: " + (s0.getPet().getWeight() == 3.0));

        // Creating a new student with a dog as pet:
        System.out.println("\nStudent s1");
        Dog d = new Dog("Woof", 2.0);
        Student s1 = new Student("Mr. Li", d);
        s1.getPet().setWeight(3.0);
        System.out.println("name: " + (s1.getPet().getName() == "Woof"));
        System.out.println("weight: " + (s1.getPet().getWeight() == 3.0));

        // We can now also use an Animal object as a pet:
        System.out.println("\nStudent s2");
        Animal a = new Animal("Blob", 5.0);
        Student s2 = new Student("Ms. Chen", a);
        System.out.println("name: " + (s2.getPet().getName() == "Blob"));
        System.out.println("weight: " + (s2.getPet().getWeight() == 5.0));
    }
```