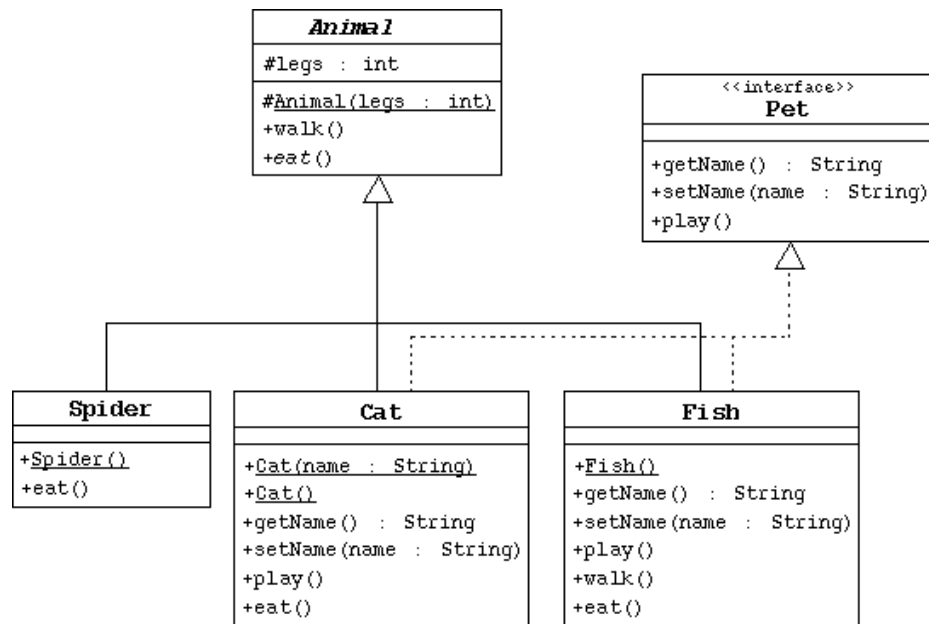


Task 3:

In this exercise you will create a hierarchy of animals that is rooted in an abstract class `Animal`. Several of the animal classes will implement an interface called `Pet`. You will experiment with variations of these animals, their methods, and polymorphism.



1. Create the `Animal` class, which is the abstract superclass of all animals.
 - a) Declare a protected integer attribute called `legs`, which records the number of legs for this animal.
 - b) Define a protected constructor that initializes the `legs` attribute.
 - c) Declare an abstract method `eat`.
 - d) Declare a concrete method `walk` that prints out something about how the animals walks (include the number of legs).
2. Create the `Spider` class.
 - a) The `Spider` class extends the `Animal` class.
 - b) Define a default constructor that calls the superclass constructor to specify that all spiders have eight legs.
 - c) Implement the `eat` method.
3. Create the `Pet` interface specified by the UML diagram.
4. Create the `Cat` class that extends `Animal` and implements `Pet`.
 - a) This class must include a `String` attribute to store the name of the pet.

- b) Define a constructor that takes one `String` parameter that specifies the cat's name. This constructor must also call the superclass constructor to specify that all cats have four legs.
 - c) Define another constructor that takes no parameters. Have this constructor call the previous constructor (using the `this` keyword) and pass an empty string as the argument.
 - d) Implement the `Pet` interface methods.
 - e) Implement the `eat` method.
5. Create the `Fish` class. Override the `Animal` methods to specify that fish can't walk and don't have legs.
6. Create a `TestAnimals` program. Have the `main` method create and manipulate instances of the classes you created above. Start with:
- ```
Fish d = new Fish();
Cat c = new Cat("Fluffy");
Animal a = new Fish();
Animal e = new Spider();
Pet p = new Cat();
```

Experiment by: a) calling the methods in each object, b) exhibiting polymorphism using interfaces and inheritance, and c) using `super` to call super class method