# Interfaces and Abstract Classes

Lab Exercise 5: Use Interfaces and Abstract Classes

Objective

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.

Directions
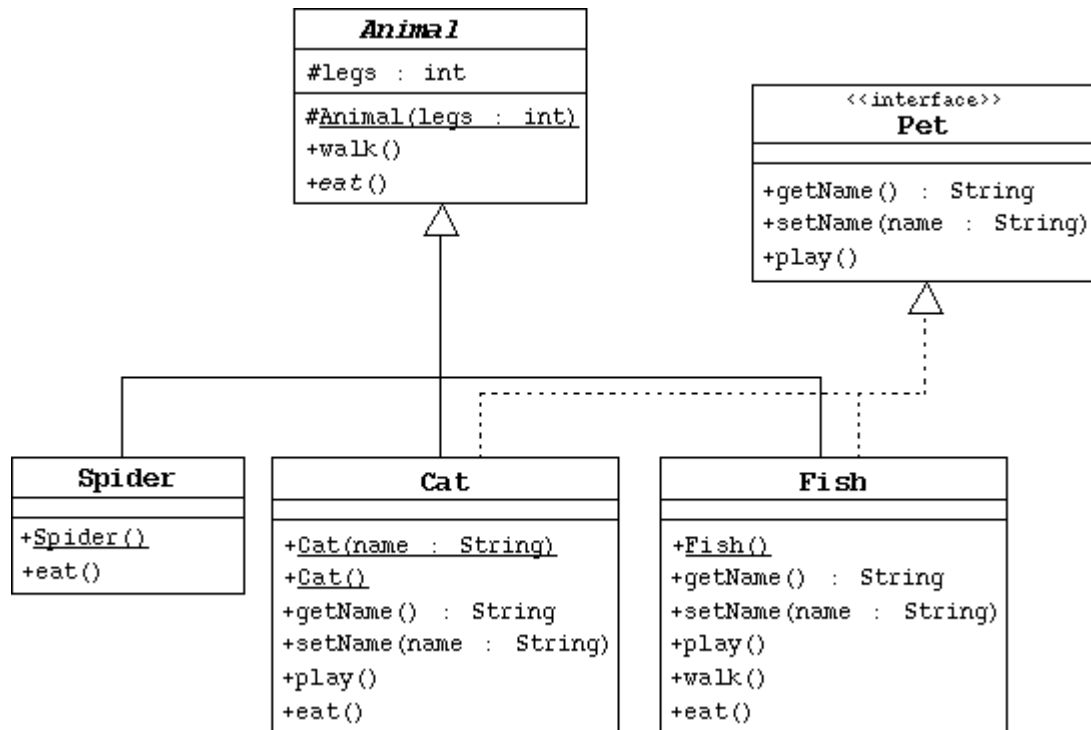
1. Create the `Animal` class, which is the <u>abstract</u> superclass of all animals.
   1. Declare a protected integer attribute called `legs`, which records the number of legs for this animal.
   2. Define a protected constructor that initializes the `legs` attribute.
   3. Declare an abstract method `eat`.
   4. Declare a concrete method `walk` that prints out something about how the animals walks (include the number of legs).
2. Create the `Spider` class.
   1. The `Spider` class extends the `Animal` class.
   2. Define a default constructor that calls the superclass constructor to specify that all spiders have eight legs.
   3. 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`.
   1. This class must include a `String` attribute to store the name of the pet.
   2. 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.

3. 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.
4. Implement the `Pet` interface methods.
5. 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 an `TestAnimals` program. Have the `main` method create and manipulate instances of the classes you created above. Start with:
7. Fish d = new Fish();
8. Cat c = new Cat("Fluffy");
9. Animal a = new Fish();
10. Animal e = new Spider();
    Pet p = new Cat();

Experiment by: a) calling the methods in each object, b) casting objects, c) using polymorphism, and d) using `super` to call super class methods.

Solution

```
public abstract class Animal {

  protected int legs;

  protected Animal(int legs)
  {
    this.legs=legs;
  }

  public void walk()
  {
    System.out.println("This animal walk by "+legs+" legs.");
  }

  public abstract void eat();
}
```

```
public  class Cat extends Animal implements Pet {
  private String catName;
  public Cat()
  {
    this("");
  }
  public Cat(String name)
  {
    super(4);
    catName=name;
  }
 // @Override
  public void eat() {
    System.out.println("Cat eat fishes");
  }
```

```java
    public String getName() {
        return catName;
    }

    public void setName(String name) {
        catName=name;
    }

    public void play() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

```java
public  class Fish extends Animal implements Pet {

    public Fish() {
        super(0);
    }

    private String fishName;

    //Override
    public void eat() {
        System.out.println("Fish eat plants");
    }

    public String getName() {
        return fishName;
    }

    public void setName(String name) {
        fishName = name;
    }

    public void play() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
    //@Override
    public void walk()
    {
        System.out.println("Fish has no legs.");
    }

}
```

```java
public  class Spider extends Animal {
    public Spider()
    {
        super(8);
    }
    public void eat()
```

```java
        {
            System.out.println("Spider eat insects");
        }
}
```

```java
public  interface Pet {

    public String getName();

    public void setName(String name);

    public void play();
}
```

```java
public class TestAnimals {

    public static void main(String[] args) {
        Fish d = new Fish();
        Cat c = new Cat("Fluffy");
        Animal a = new Fish();
        Animal e = new Spider();
        Pet p = new Cat();

        d.setName("MyFish");
        d.eat();
        System.out.println("This fish's name is"+d.getName());
        d.walk();

        c.eat();
        System.out.println("This fish's name is"+c.getName());
        c.walk();

        ((Fish)a).setName("MyFish2");
        ((Fish)a).eat();
        System.out.println("This fish's name is"+((Fish)a).getName());
        ((Fish)a).walk();

        ((Spider)e).eat();
        ((Spider)e).walk();

        ((Cat)p).setName("MyCat");
        ((Cat)p).eat();
        System.out.println("This fish's name is"+((Cat)p).getName());
        ((Cat)p).walk();
    }
}
```