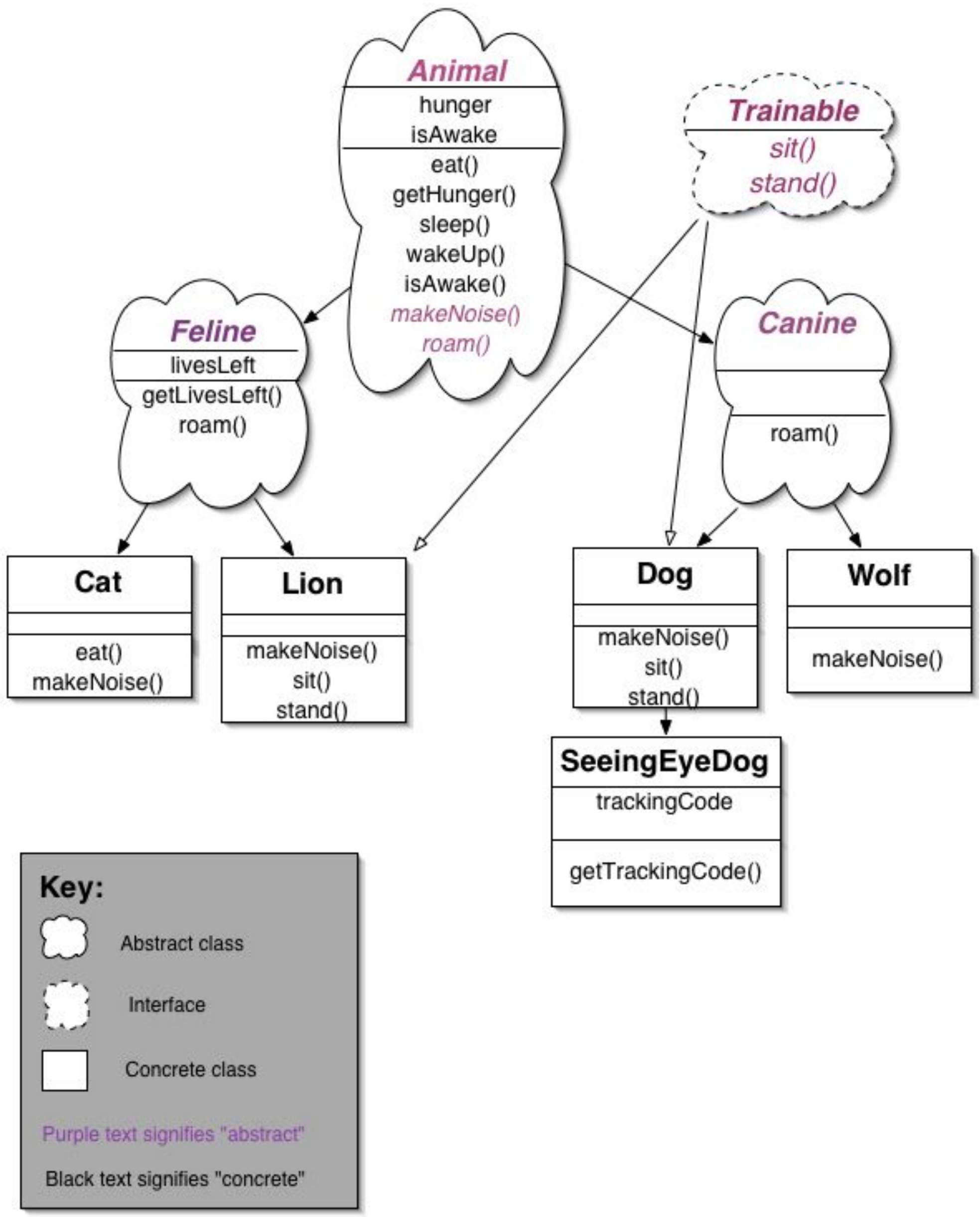
Constructor, Inheritance, Abstract Class, Interface

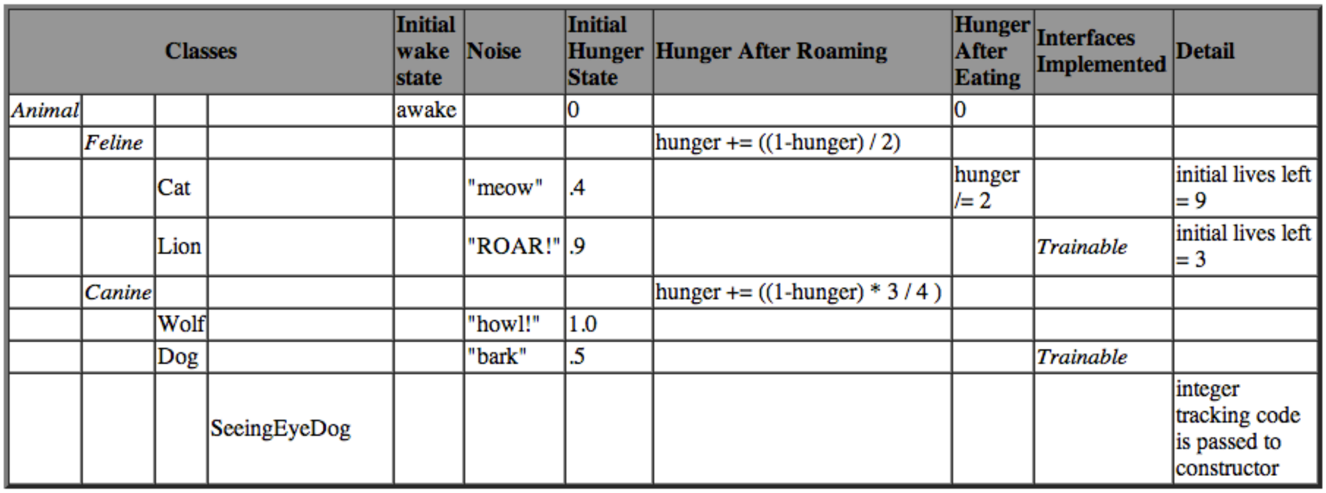
Goals

- Given an inheritance tree and a state chart, use concrete classes, interfaces, and abstract classes to write code for Animals.

Inheritance Tree



State Chart



Classes				Initial wake state	Noise	Initial Hunger State	Hunger After Roaming	Hunger After Eating	Interfaces Implemented	Detail
Animal				awake		0		0		
	Feline						hunger += ((1-hunger) / 2)			
		Cat			"meow"	.4		hunger /= 2		initial lives left = 9
		Lion			"ROAR!"	.9			*Trainable*	initial lives left = 3
	Canine						hunger += ((1-hunger) * 3 / 4)			
		Wolf			"howl!"	1.0				
		Dog			"bark"	.5			*Trainable*	
			SeeingEyeDog							integer tracking code is passed to constructor

Note

- That all Animals are born awake. An Animal's initial hunger state is 0 unless specified otherwise in a progeny's constructor.
- A SeeingEyeDog's initial hunger state is .5 because it inherits from Dog.
- +=, -=, /= are short cut operators
    - E.g. hunger += ((1 - hunger)*3/4) is equivalent to hunger = hunger + ((1 - hunger)*3/4).

Tips

- We provide you with code below for Animal.java, Canine.java, Dog.java, Trainable.java
- You need to write the following classes based on the Inheritance tree: Feline.java, Cat.java, Lion.java, Wolf.java, SeeingEyeDog.java
- **Make the instance variables in superclasses *protected* instead of *private*** . This way, you will be able to access the variables of superclasses. Remember, *private* means that the variable can only be accessed by the class it is in. *protected* variables can be accessed outside the class in a more secure fassion than *public* variables. See example in Animal.java
- In the interest of keeping this project as uncomplicated as possible, the **sit() and stand() methods** should print **"AnimalType sit" and "AnimalType stand" respectively** . See example in Dog.java

Constructor and Method Signatures

```
Constructors:
     Cat()
     Lion()
     Wolf()
     SeeingEyeDog(int trackingCode)

Methods:
In class Cat, Lion, Wolf
     String makeNoise() //concrete method

In class Lion
     void sit()
     void stand()

In class Cat
     void eat() //override from Animal class

In class Feline
     int getLivesLeft();
     void roam() //concrete method

In class SeeingEyeDog
     int getTrackingCode();
```

Interactions (You can put these into main method to test your classes and methods.)

Some sample interactions are provided to get you started.

```
---------------Abstract Class and Inheritance-------------
> Animal a = new Animal(); //cannot instantiate abstract class
InstantiationException:  > Animal a = new Dog();//variable of supertype
can hold a subtype
> a.getHunger() //return Dog's hunger state
0.5
> a.makeNoise() //Dog's makeNoise (concrete method) is executed
"bark!"
> a.isAwake()
true
> a.eat(); //eat is inherited by Dog and makes hunger 0
> a.getHunger()
0.0
> a instanceof Dog //Reference variable a points to dog obeject
true
> a instanceof Canine
true //Dog is subtype of Canine
> a instanceof Animal
true //Animal super super type is Dog
> a instanceof Object
true //Animal is super super type of Dog, and Animal is subtype of
Object (implicit or implied)
 -----------Interfaces--------------------------
> Lion l = new Lion()
> l.stand()  //stand() is implemented in Lion class
Lion stand
> Trainable beast = new Lion();
> beast.sit(); //calls sit method of the approriate object beast
variable is pointing to
Lion sit
> beast.stand();
Lion stand
> beast = new Dog();
> beast.sit();
Dog sit
> beast.stand();
Dog stand
-----------------Inheritance with Concrete Classes-------------
> SeeingEyeDog d = new SeeingEyeDog(5)
> d.getTrackingCode()
5
> d instanceof SeeingEyeDog
true
> d instanceof Dog
true
> d instanceof Canine
true
> d instanceof Animal
true
> d instanceof Object
true
 -----------------Some more interactions-------------------------
> Lion l = new Lion(); > l.isAwake()
true
> l.getHunger() //returns hunger state of Lion
0.9
> l.eat();
> l.getHunger()
0.0 > Cat c= new Cat();
> c.getHunger()
0.4
> c.getLivesLeft()
9 > c.makeNoise()
"meow"
```

Animal.java

```
public abstract class Animal{
        protected double hunger;
        protected boolean isAwake = true;

        public double getHunger() { return hunger; }
        public void eat(){ hunger = 0; }
        public boolean isAwake() { return isAwake; }
        public void sleep(){ isAwake = false; }
        public void wakeUp(){ isAwake = true; }
        // ~~~~~~ abstract methods ~~~~~~~~
        public abstract String makeNoise();
        public abstract void roam();
}
```

Canine.java

```
public abstract class Canine extends Animal{
        public void roam(){
        // exhibit pack behavior
        hunger += (1 - hunger) * 3.0 / 4.0;
        }
}
```

Dog.java

```
public class Dog extends Canine implements Trainable{
        public Dog(){
                hunger = .5;
        }
        public String makeNoise(){
                return "bark!";
        }
        public void sit(){
                System.out.println("Dog sit");
        }
        public void stand() {
                System.out.println("Dog stand");
        }
}
```

Trainable.java

```
public interface Trainable{
        public void sit();
        public void stand();
}
```