CS 430 Virtual Tables

Consider the following Java-like code snippet (all non-primitive variables are reference types).

```
abstract class Animal
{
    boolean sleeping = false;
    void sleepToggle() { sleeping = !sleeping; }
    abstract void speak();
}

class Dog extends Animal
{
    void speak() { println(“Bark! Bark!”);
}

class Cat extends Animal
{
    int lives = 9;
    void speak() { println(“I CAN HAZ CHEEZBURGER?”); }
}

class Bulldog extends Dog
{
    void cheer() { println(“START WEARING PURPLE!”);
    void sleepToggle() { sleeping = false; }
}

void poke(Animal animal)
{
    animal.speak(); // location A
}

void main()
{
    Animal lassie = new Dog();
    Animal garfield = new Cat();
    Dog dukeDog = new Bulldog();
    poke(dukeDog);
}
```

1. Draw a diagram of the layout of the stack, heap, and static sections of memory when execution reaches location A (prior to the call on that line) as demonstrated in class. You should include an activation record for every active subprogram, a class invocation record for each instance/object, a virtual method table for each non-abstract class, and a box for each concrete method implementation, all clearly labeled as appropriate and placed in the correct sections of memory. Clearly specify the types and values of all variables, and use arrows to indicate the targets of pointers and references.

2. Explain how the compiler implements the call to `animal.speak()` at comment A and how this allows for polymorphic behavior (i.e., dynamic dispatch). Which version of "`speak()`" is called in this instance, and how?