Abstract Classes

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.

THE COLLEGE OF HIGHER LEARNING.

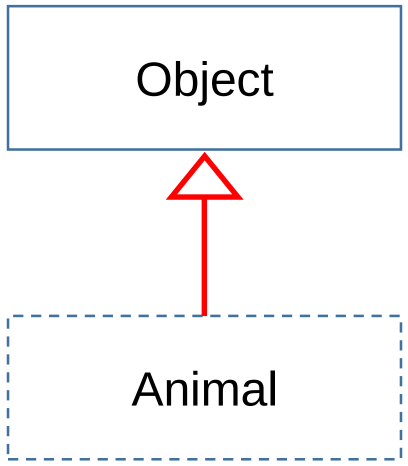
Overview

- Abstract Classes & Methods
  - Pros and cons
- Final Classes & Methods

Abstract Class

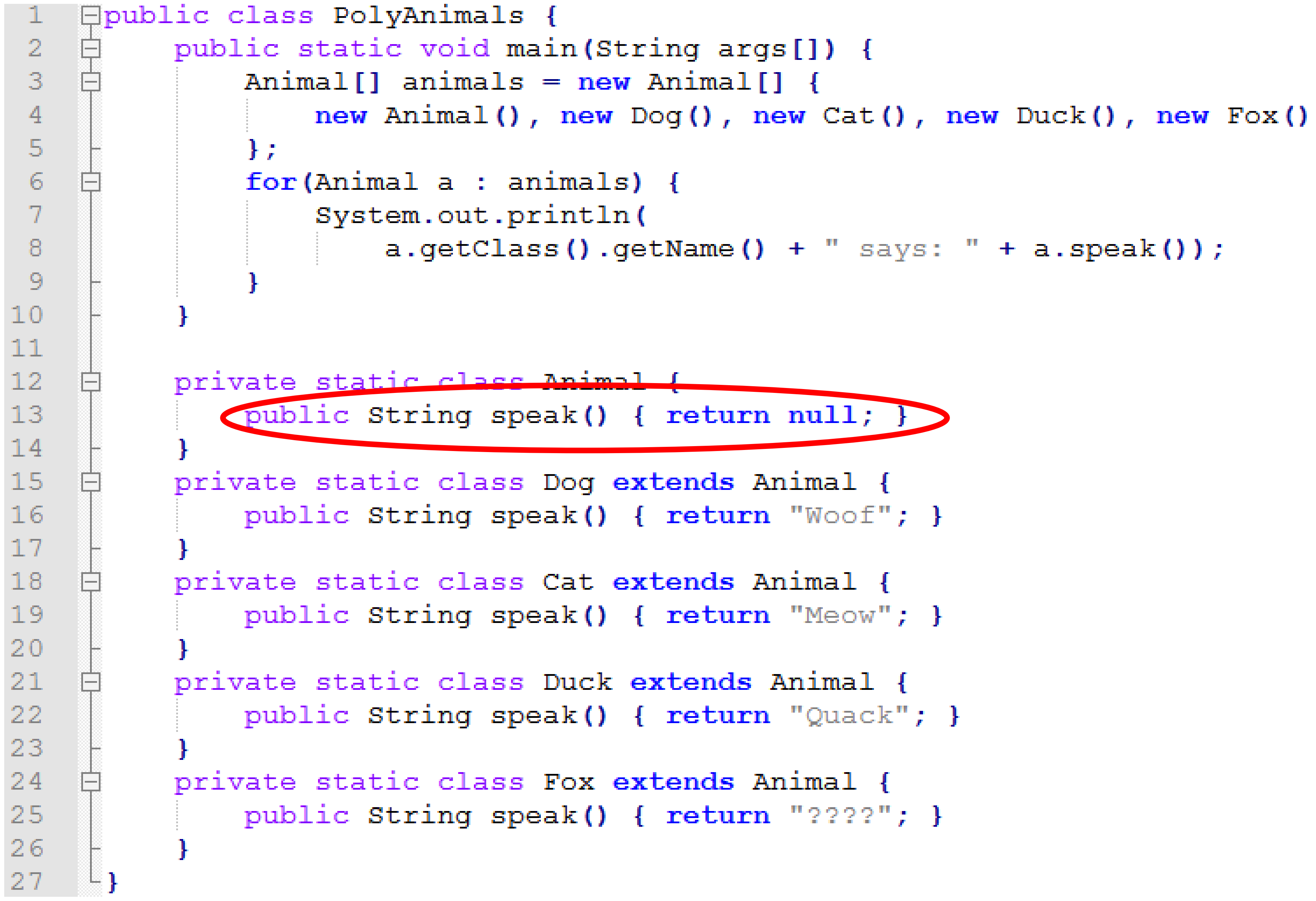
- An abstract class contains one or more abstract methods
- An abstract method is a method that is declared, but has no implementation

- Abstract classes can not be instantiated
- They require their subclasses to provide the implementation for the abstract methods
- Abstract classes get dotted lines in UML
  - Or stylize them with <<abstract>> below ClassName


Object
Animal

Ex: Polymorphism

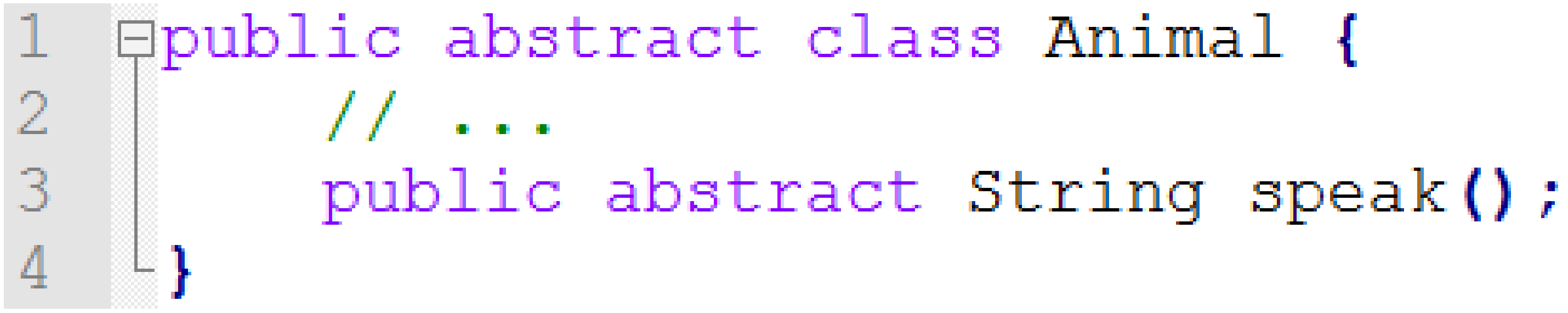
Meow!
Woof!
Quack!
Animal
Animal
Animal
Polymorphism
speak();
by Sinipull for codecall.net



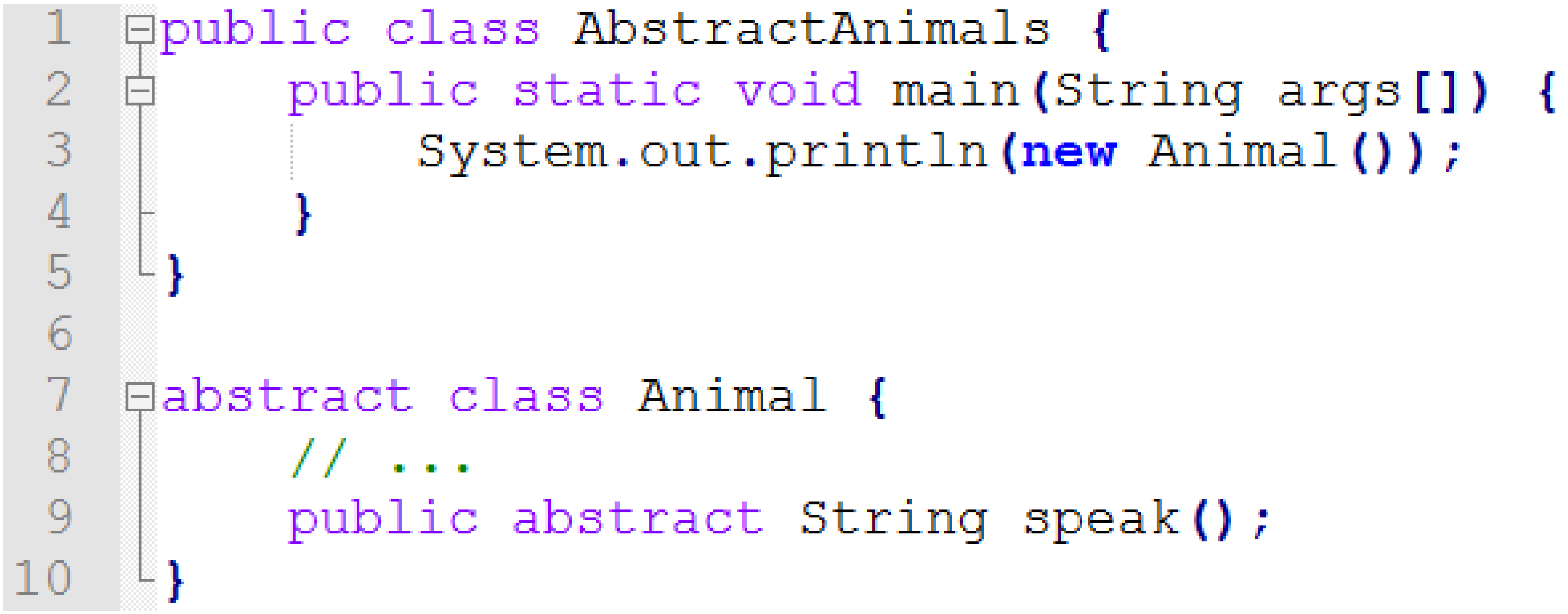
```java
public class PolyAnimals {
    public static void main(String args[]) {
        Animal[] animals = new Animal[] {
            new Animal(), new Dog(), new Cat(), new Duck(), new Fox()
        };
        for(Animal a : animals) {
            System.out.println(
                a.getClass().getName() + " says: " + a.speak());
        }
    }

    private static class Animal {
        public String speak() { return null; }
    }
    private static class Dog extends Animal {
        public String speak() { return "Woof"; }
    }
    private static class Cat extends Animal {
        public String speak() { return "Meow"; }
    }
    private static class Duck extends Animal {
        public String speak() { return "Quack"; }
    }
    private static class Fox extends Animal {
        public String speak() { return "????"; }
    }
}
```

Abstract Class: Animal

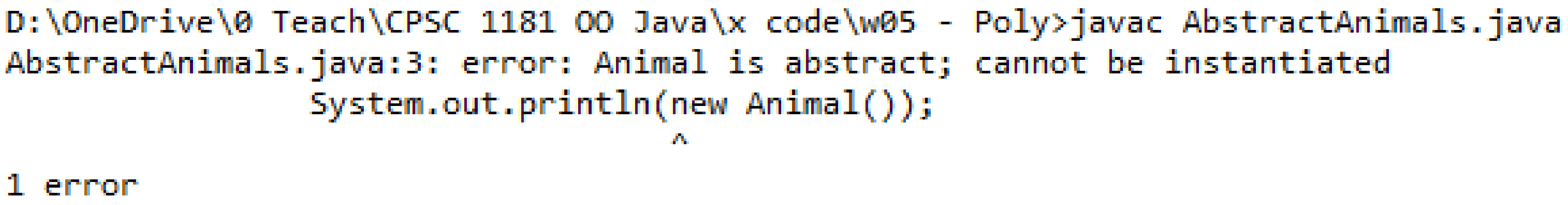
- What does it mean for an Animal to speak?



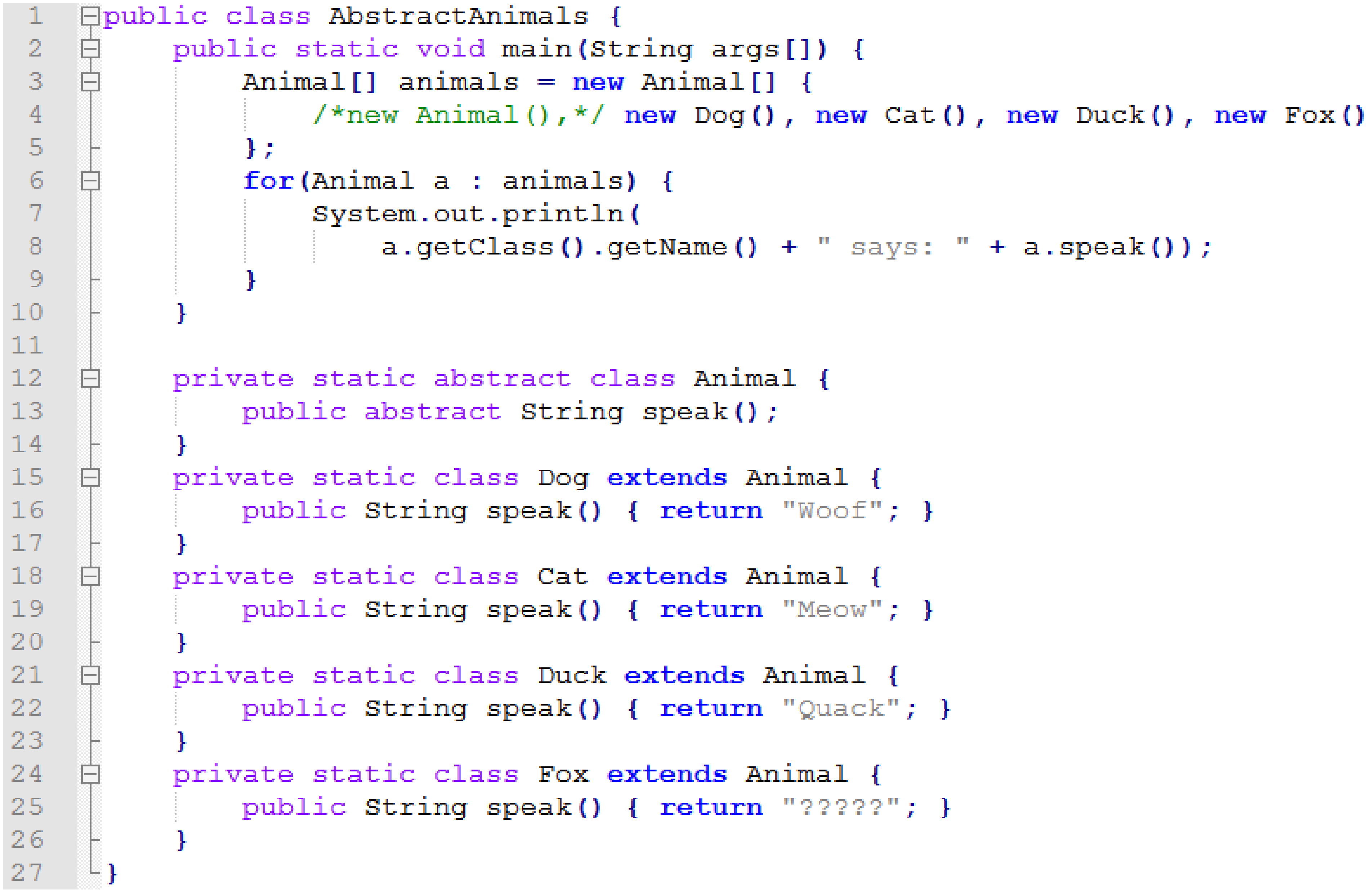
```
public abstract class Animal {
    // ...
    public abstract String speak();
}
```



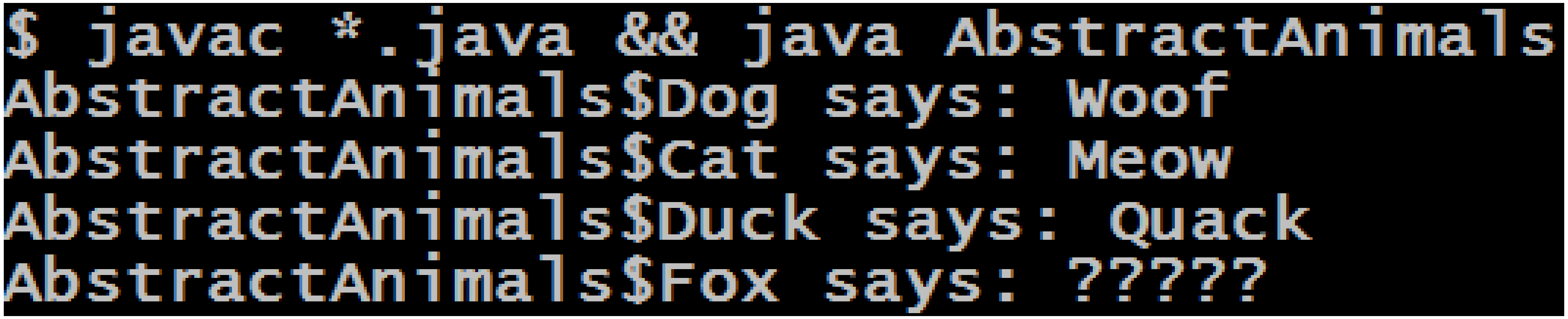
```java
public class AbstractAnimals {
	public static void main(String args[]) {
		System.out.println(new Animal());
	}
}

abstract class Animal {
	// ...
	public abstract String speak();
}
```



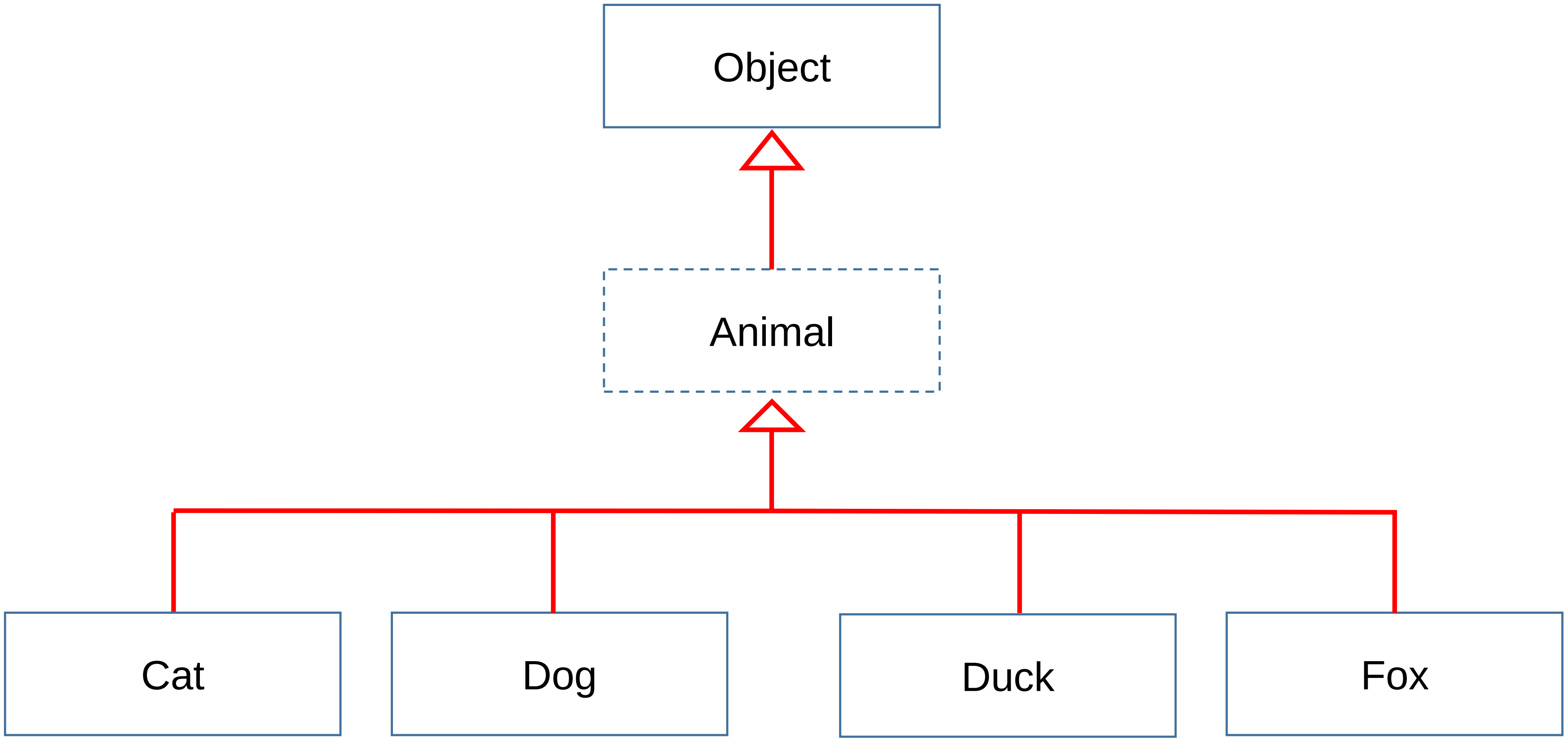
```
D:\OneDrive\0 Teach\CPSC 1181 OO Java\x code\w05 - Poly>javac AbstractAnimals.java
AbstractAnimals.java:3: error: Animal is abstract; cannot be instantiated
                System.out.println(new Animal());
                                   ^

1 error
```



```java
public class AbstractAnimals {
    public static void main(String args[]) {
        Animal[] animals = new Animal[] {
            /*new Animal(),*/ new Dog(), new Cat(), new Duck(), new Fox()
        };
        for(Animal a : animals) {
            System.out.println(
                a.getClass().getName() + " says: " + a.speak());
        }
    }

    private static abstract class Animal {
        public abstract String speak();
    }
    private static class Dog extends Animal {
        public String speak() { return "Woof"; }
    }
    private static class Cat extends Animal {
        public String speak() { return "Meow"; }
    }
    private static class Duck extends Animal {
        public String speak() { return "Quack"; }
    }
    private static class Fox extends Animal {
        public String speak() { return "?????"; }
    }
}
```

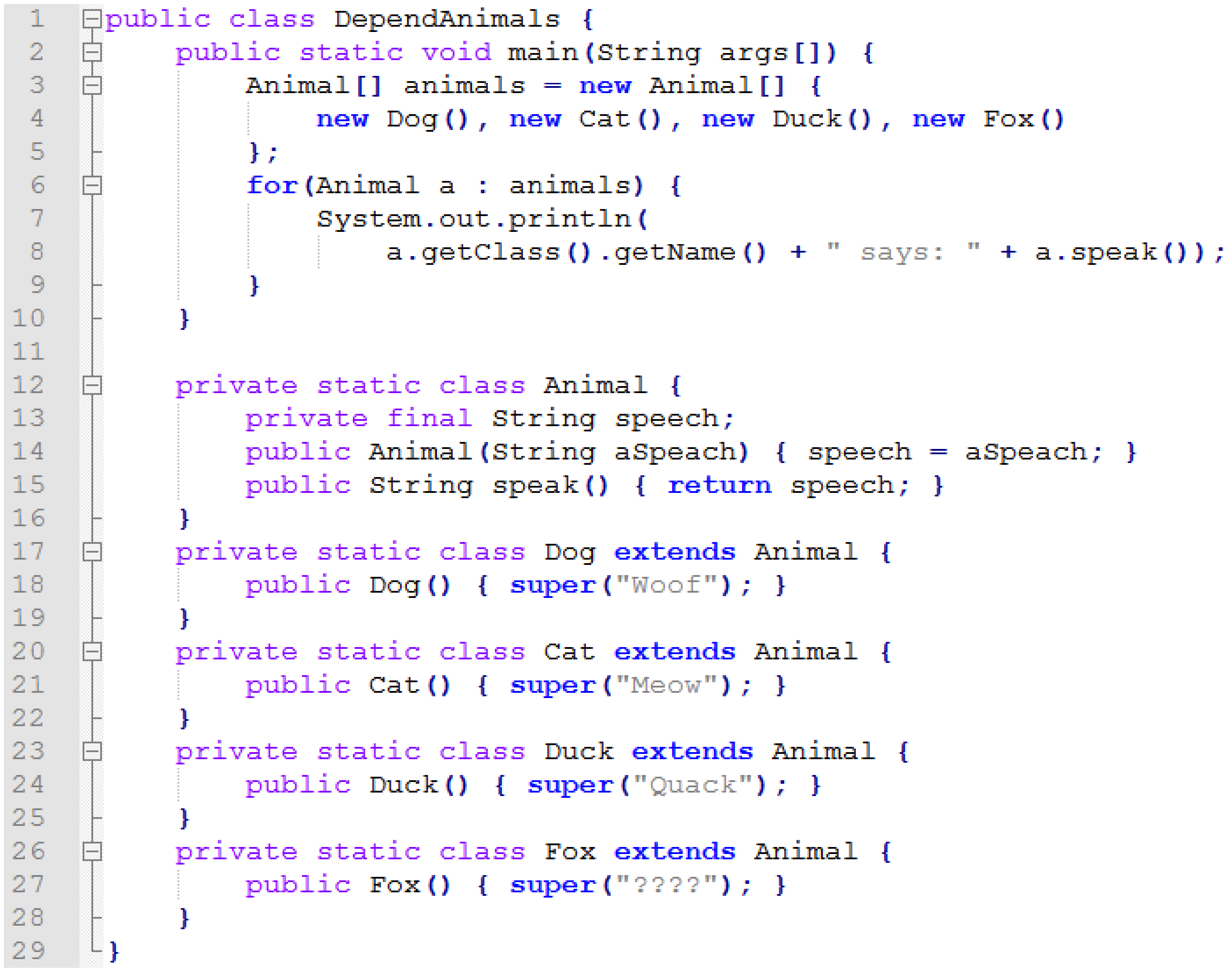


```
$ javac *.java && java AbstractAnimals
AbstractAnimals$Dog says: Woof
AbstractAnimals$Cat says: Meow
AbstractAnimals$Duck says: Quack
AbstractAnimals$Fox says: ?????
```

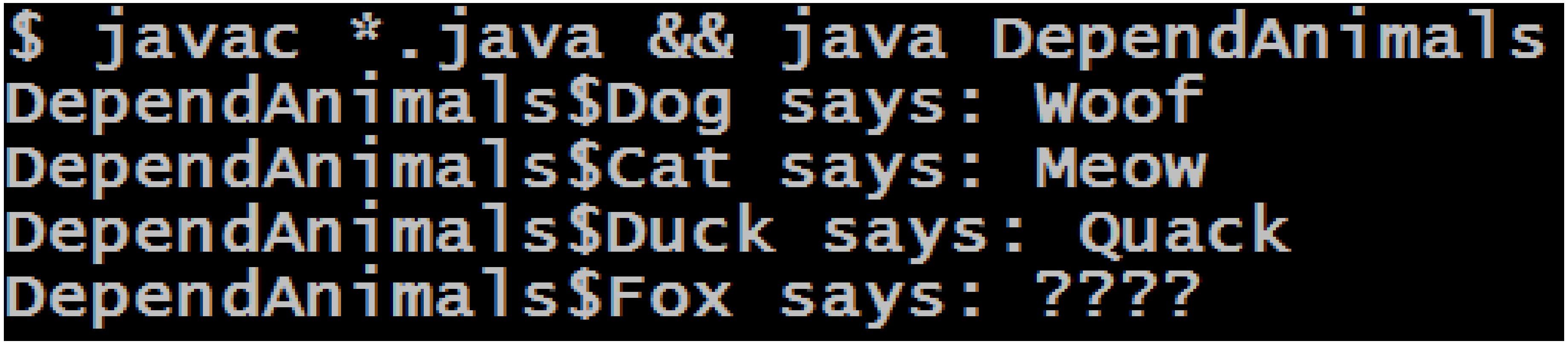

Object
Animal
Cat
Dog
Duck
Fox

Should You Use Them?

- Pro:
  - Code reuse
    - Animal has most of the implementation
      - Only have to fill in the details of abstract method
    - Lets you group related types together in your hierarchy
- Con:
  - Forces you to inherit when you may not want to
  - An abstract method indicates that you have some cross-cutting concern
    - Bad modularity, hard to test
    - Can be factored out with a dependency injection
      - Dependency Injection makes modularity (cohesion and coupling) better



```java
public class DependAnimals {
    public static void main(String args[]) {
        Animal[] animals = new Animal[] {
            new Dog(), new Cat(), new Duck(), new Fox()
        };
        for(Animal a : animals) {
            System.out.println(
                a.getClass().getName() + " says: " + a.speak());
        }
    }

    private static class Animal {
        private final String speech;
        public Animal(String aSpeach) { speech = aSpeach; }
        public String speak() { return speech; }
    }
    private static class Dog extends Animal {
        public Dog() { super("Woof"); }
    }
    private static class Cat extends Animal {
        public Cat() { super("Meow"); }
    }
    private static class Duck extends Animal {
        public Duck() { super("Quack"); }
    }
    private static class Fox extends Animal {
        public Fox() { super("????"); }
    }
}
```



```
$ javac *.java && java DependAnimals
DependAnimals$Dog says: Woof
DependAnimals$Cat says: Meow
DependAnimals$Duck says: Quack
DependAnimals$Fox says: ????
```

Not Abstract

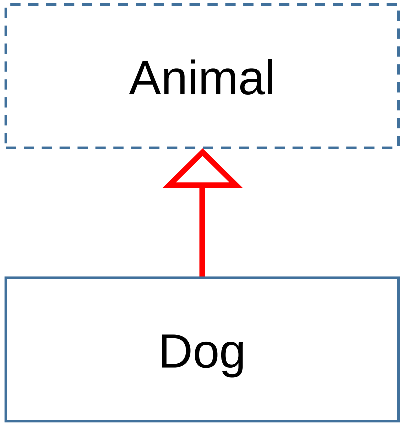
- Must define default implementation
  - May not make sense
- Forced implementation

Abstract

- Don’t have to define default implementation
  - Force subclasses to implement
- Inheritance dependency

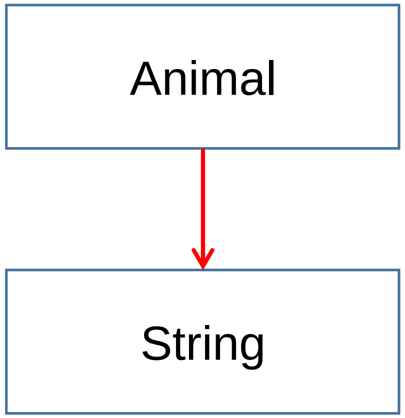
Abstract

- Inheritance dependency
  - “is-a”
  - (strong)



Dependency Injection

- Composition dependency
  - “has-a”
  - (weak)

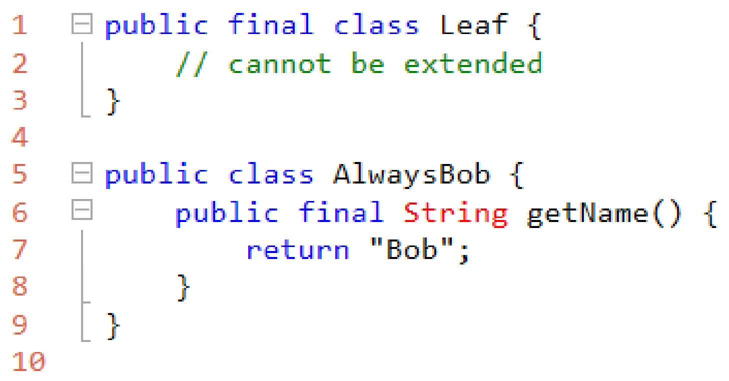


Opposite: final

- Abstract classes must be extended and have methods overridden

- A method declared final cannot be overridden
- A class declared final cannot be extended

Opposite: final



```java
public final class Leaf {
    // cannot be extended
}

public class AlwaysBob {
    public final String getName() {
        return "Bob";
    }
}
```

Recap

- Abstract Classes & Methods
    - Pros and cons
- Final Classes & Methods