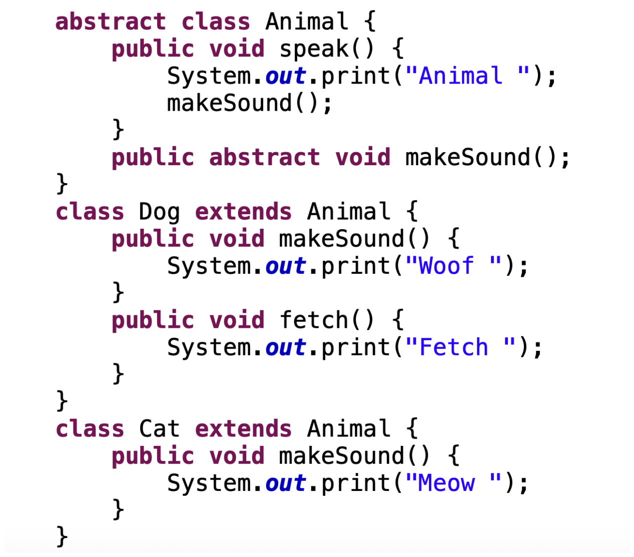
CSSE 220 – Object-Oriented Software Development
Rose-Hulman Institute of Technology

Worksheet Polymorphic Practice

1.
```java
abstract class Pet {
    public void speak() {
        System.out.print("Pet ");
        sound();
    }
    public abstract void sound();
}
class Dog extends Pet {
    public void sound() {
        System.out.print("Woof ");
    }
}
class Cat extends Pet {
    public void sound() {
        System.out.print("Meow ");
    }
}
```

Pet a = new Dog();
Pet b = new Cat();
Dog c = new Dog();

a.speak(); Pet Woof
b.speak(); Pet Meow
c.speak(); Pet Woof
a.sound(); Woof
b.sound(); Meow

2. 

```java
abstract class Animal {
    public void speak() {
        System.out.print("Animal ");
        makeSound();
    }
    public abstract void makeSound();
}
class Dog extends Animal {
    public void makeSound() {
        System.out.print("Woof ");
    }
    public void fetch() {
        System.out.print("Fetch ");
    }
}
class Cat extends Animal {
    public void makeSound() {
        System.out.print("Meow ");
    }
}
```

Animal a = new Dog();
Animal b = new Cat();
Dog c = new Dog();

a.speak(); Animal Woof → speak() from Animal, makeSound() from Dog

b.speak(); Animal Meow → same logic, runtime = Cat

a.makeSound(); Woof → dynamic dispatch still applies

c.fetch(); Fetch → c is a Dog, method exists

a.fetch(); compile error → Animal does not define fetch()

((Dog) a).fetch(); Fetch → explicit downcast, runtime object is Dog

((Dog) b).fetch(); runtime error → compiles, but Cat is not Dog → ClassCastException

Animal d = c; OK (no output) → implicit upcasting is always safe

Dog e = a; compile error → implicit downcasting is NOT allowed → Java refuses without a cast