Even if you didn't understand inheritance, you might be able to deduce some things about the hierarchy from the classes' names and the relationships between employees in the real world. So let's take this sort of exercise one step further. We'll make the names of the classes into nonsensical phrases that bear no resemblance to real-world entities. Admittedly this is not something you'd do when really programming, but our goal here is to examine the mechanics of inheritance and polymorphism on their own.

Assume that the following classes have been defined:

```java
public class A {
    public void method1() {
        System.out.println("A 1");
    }

    public void method2() {
        System.out.println("A 2");
    }

    public String toString() {
        return "A";
    }
}
```

```java
public class B extends A {
    public void method2() {
        System.out.println("B 2");
    }
}
```

```java
public class C extends A {
    public void method1() {
        System.out.println("C 1");
    }

    public String toString() {
        return "C";
    }
}
```

```java
public class D extends C {
    public void method2() {
        System.out.println("D 2");
    }
}
```

Consider the following client code, which uses the above classes. We declare an array of variables of a superclass type and fill it with objects of the various subclass types. When we call methods on the elements of our array, we should observe polymorphic behavior.