Polymorphism

"Polymorphism": Occurring in several forms.

```
A x = new A();
B y = x;            // Okay if A inherits B
y.foo();            // Okay if B has method foo
```

Rationale:
~ A objects have inherited from B
~ So, A objects have all B methods
~ So, restricting an A object to B methods is safe

Polymorphism

Why do this? Say Dog, Cat and Pig all inherit Animal.

```
public class Main {
    public static void main(String[] args) {
        Animal[] alist = new Animal[3];
        alist[0] = new Dog();           // Okay if Dog inherits Animal
        alist[1] = new Cat();           // Okay if Cat inherits Animal
        alist[2] = new Pig();           // Okay if Pig inherits Animal
        for (int i=0; i<alist.length; i++) {
            System.out.println(alist[i]);  // Okay if Animal has toString
        }
    }
}
```

Inheritance & Polymorphism compile errors

1) `x=y` okay if `y` declared type inherits `x` declared type.

```
// Let's say B extends A (ie, B inherits from A)
A w;
B y;
w=new A(); // OK - same type
w=new B(); // OK - assign to type above in inheritance hierarchy
y=new A(); // NO - cannot assign to type below in inheritance hierarchy
```

2) `x.foo()` okay if `x` declared type has/inherits `foo`.

3) `((A)x).foo()` okay if `A` defines/inherits `foo`.

Inheritance & Polymorphism runtime errors

1) `x.foo()` follows reference in `x` and runs the target object's `foo`.

2) `((C)x).foo()` follows reference in `x` and runs the target object's `foo`. Okay if target object's is/inherits `C`.

```
// Let's say B and C both extend A
B w = new B();
A x = w;        // OK - assign to type above in inheritance hierarchy
w.foo();        // Invokes the foo defined in B because w is a reference to a B
x.foo();        // Invokes the foo defined in B because x is a reference to a B
((C)x).foo();   // ClassCastException because B does not inherit C
```

Polymorphism Exercise

```java
public class Main {
    public static void main(String[] args) {
        First var1 = new Second();
        First var2 = new Third();
        First var3 = new Fourth();
        Second var4 = new Third();
        Object var5 = new Fourth();
        Object var6 = new Second();

        var1.method2();
        var2.method2();
        var3.method2();
        var4.method2();
        //var5.method2();
        //var6.method2();
        var1.method3();
        ((Third)var4).method1();
        /*
        var2.method3();
        var3.method3();
        var4.method3();
        var5.method3();
        var6.method3();
        ((Third)var4).method1();
        ((Third)var4).method1();
        ((Second)var5).method2();
        ((First)var5).method3();
        ((Third)var5).method1();
        ((First)var6).method3();
        ((Second)var6).method1();
        ((Second)var6).method3();
        ((Third)var6).method2();
 */
    }
}

class First extends Object{
    public void method2() {
        System.out.println("First2");
    }
    public void method3() {
        method2();
    }
}

class Second extends First {
    public void method2() {
        System.out.println("Second2");
    }
}

class Third extends Second {
    public void method1() {
        System.out.println("Third1");
        super.method2();
    }
    public void method2() {
        System.out.println("Third2");
    }
}

class Fourth extends First {
    public void method1() {
        System.out.println("Fourth1");
    }
    public void method2() {
        System.out.println("Fourth2");
    }
}
```