# Software Development (CS2500)

Lecture 30: **Polymorphism**

M.R.C. van Dongen

January 10, 2010

## Contents

## 1 Outline

These “notes” correspond to the last part of Chapter 7. The lecture slides have more content, but this is not suitable for a lecture notes format.

## 2 Polymorphism

When you define a supertype for a group of classes: Any subclass of the supertype can be substituted where the supertype is expected.

So let’s assume we have a `Animal` class with a `Dog` subclass. Then we can use a `Dog` reference where an `Animal` reference is expected. For example:

```java
Animal animal = new Dog( );
```

The term *polymorphism* means ‘having many forms.’

A polymorphic reference variable can refer to different types of objects over time [Lewis and Loftus, 2009].

Without polymorphism the type of reference variable and object are the same:

```java
Dog dog = new Dog( );
```

With polymorphism the type of reference variable and object may be different:

```java
Animal dog = new Dog( );
```

With polymorphism the reference type can be a superclass of the actual object type. A reference variable can refer to any object created from any class which is a subclass by transitivity.

The type of the object, not the type of the reference, is used to determine which method to invoke [Lewis and Loftus, 2009].

```java
Animals[] animals = new Animals[ 2 ];
animals[ 0 ] = new Dog( );
animals[ 1 ] = new Cat( );
animals[ 0 ].roam( ); // Roams in pack.
animals[ 1 ].roam( ); // Roams alone.
```

With polymorphism the formal parameters and the return types of methods can be polymorphic. So if the formal parameter is `Animal` the actual parameter may be `Dog`. Likewise, the return type may be `Animal` but a `Cat` may be returned.

### 2.1 Advantages

With polymorphism, you can write code that doesn't have to change when you introduce new subclass types to your program. For example, you can write a polymorphic sorting algorithm in the superclass. It also works for arrays containing subclass object references. Clearly, this is much better than having to implement the algorithm in each subclass. This works because the subclass inherits all public methods that belong to the superclass. So if a polymorphic variable requires an instance method/attribute, this is also supported if the variable references a subclass instance.

## 3 Quicksort

In Lecture 27 we implemented the `quicksort` algorithm for `int`s. In this section we shall implement a polymorphic version of the algorithm. We shall assume that out objects are `Comparable`, which means they hava an instance method `int compareTo( Object that )`, which compares `this` object and `that`.[1] The method `compareTo` returns a negative number if `this` is less than `that`, returns a positive number if

[1]For simplicity, it is assumed that `Comparable` is an actual class, which it isn't. However, for simplicity we shall assume that it is class.

`this` is greater than `that`, and zero otherwise.

Java
```java
public static
void quicksort( Comparable[] items ) {
    qsort( items, 0, items.length - 1 );
}
```

Java
```java
// Sorts items[ lo .. hi ] in non-descending order.
private static
void qsort( Comparable[] items, int lo, int hi ) {
   if (hi - lo >= 1) {
      int pivotPosition = partition( items, lo, hi );
      qsort( items, lo, pivotPosition - 1 );
      qsort( items, pivotPosition + 1, hi );
   }
}
```

Java
```java
private static
int partition( Comparable[] items, int lo, int hi ) {
   int destination = lo;
   swop( items, (hi + lo) >>> 1, hi );
   // The pivot is now stored in items[ hi ].
   for (int index = lo; index != hi; index ++) {
      int criterion = items[ index ].compareTo( hi );
      if (criterion <= 0) {
         // Move current item to start.
         swop( items, destination, index );
         destination ++;
      }
   }
   swop( items, destination, hi );
   return destination;
}
```

## 4 For Wednesday

- Study Chapter 7.

## 5 Bibliography

## References

[Lewis and Loftus, 2009] John Lewis and William Loftus. *Java Software Solutions* Foundations of Program Design. Pearson International, 2009.