## Polymorphism

Polymorphism is a generic term that means 'many shapes'.

In C++ the simplest form of Polymorphism is overloading of functions, for instance several functions called SortArray( arraytype ) where sortarray might be an array of ints, or doubles.

There are two kind of polymorphism.

1. Runtime Polymorphism
2. Design time Polymorphism

## Operational description of runtime polymorphism

Here is an operational description of runtime polymorphism as implemented in C# through inheritance and method overriding:

* Assume that a class named SuperClass defines a method named Method.
* Assume that a class named SubClass extends SuperClass and overrides the method named Method.
* Assume that a reference to an object of the class named SubClass is assigned to a reference variable named ref of type SuperClass.
* Assume that the method named method is then invoked on the reference variable using the following syntax: ref.method()

**Result**: The version of the method named method that will actually be executed is the overridden version in the class named SubClass, and is not the version that is defined in the class named SuperClass.

This is runtime polymorphism.

Runtime polymorphism is very powerful. As you gain more experience with C#, you will learn that much of the power of OOP using C# is centered on runtime polymorphism using class inheritance, interfaces, and method overriding. (The use of interfaces for polymorphism will be discussed in a subsequent lesson.)

## An important attribute of runtime polymorphism

The decision as to which version of the method to execute is based on the actual type of object whose reference is stored in the reference variable, and not on the type of the reference variable on which the method is invoked.

## Why is it called runtime polymorphism?

The reason that this type of polymorphism is often referred to as runtime polymorphism is because the decision as to which version of the method to execute cannot be made until runtime. The decision cannot be made at compile time (as is the case with overloaded methods).

