**Inheritance**

inheritance is used when a class is built upon another class—in terms of data or behavior—and it adheres to the rules of substitutability—namely, that the derived class can be substituted for the base class. An example of this would be if you were writing a hierarchy of database classes. Let's say you wanted to have a class for handling Microsoft SQL Server databases and Oracle databases. Since these databases differ in some respects, you'd want to have a class for each database. However, both databases do share enough functionality that you'd want to put common functionality into a base class, derive the other two classes from the base class, and override or modify the inherited base class behavior at times.

To inherit one class from another, you would use the following syntax:

class *<derivedClass>*: *<baseClass>*

Here's what this database example would look like:

using System;

class Database

{

public Database()

{

CommonField = 42;

}

public int CommonField;

public void CommonMethod()

{

Console.WriteLine("Database.Common Method");

}

}

**class SQLServer : Database**

{

public void SomeMethodSpecificToSQLServer()

{

Console.WriteLine("SQLServer.SomeMethodSpecificToSQLServer");

}

}

**class Oracle : Database**

{

public void SomeMethodSpecificToOracle()

{

Console.WriteLine("Oracle.SomeMethodSpecificToOracle");

}

}

class InheritanceApp

{

public static void Main()

{

SQLServer sqlserver = new SQLServer();

sqlserver.SomeMethodSpecificToSQLServer();

sqlserver.CommonMethod();

Console.WriteLine("Inherited common field = {0}",

sqlserver.CommonField);

}

}

Compiling and executing this application results in the following output:

SQLServer.SomeMethodSpecificToSQLServer

Database.Common Method

Inherited common field = 42

Notice that the *Database.CommonMethod* and *Database.CommonField* methods are now a part of the *SQLServer* class's definition. Because the *SQLServer* and *Oracle* classes are derived from the base *Database* class, they both inherit almost all of its members that are defined as *public*, *protected*, or *internal*. The only exception to this is the constructor, which cannot be inherited. Each class must implement its own constructor irrespective of its base class.

C# *does not* support multiple inheritance through derivation. You can, however, aggregate the behavioral characteristics of multiple programmatic entities by implementing multiple interfaces.