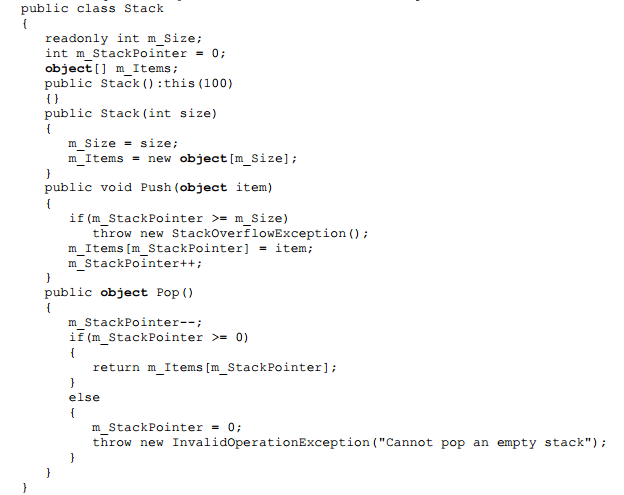
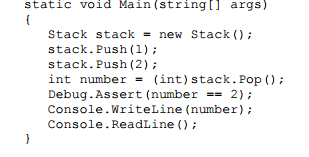
Exercise 1 –Generic Stack In this exercise, you will develop a generic stack, and experience the benefits of generics.

Task 1 – Develop a Generic Stack Open the Generics.sln solution in the Generics lab folder. The solution is a simple console application that uses a stack. Open the ObjectStack.cs file. It contains an object-based stack:

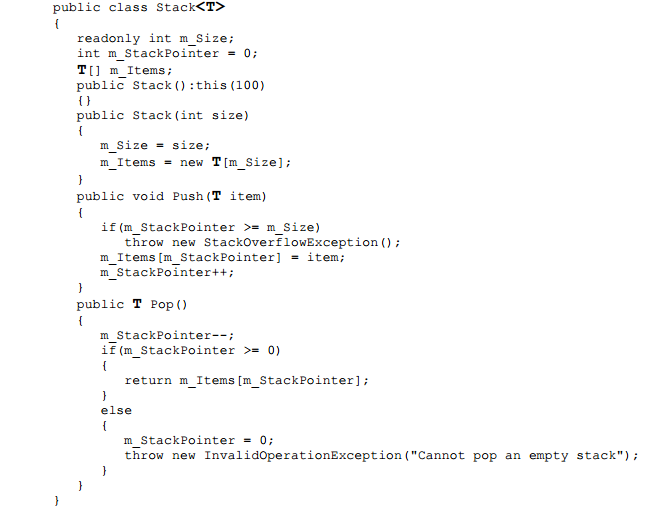


The readonly keyword is a modifier that you can use on fields. When a field declaration includes a readonly modifier, assignments to the fields introduced by the declaration can only occur as part of the declaration or in a constructor in the same class.

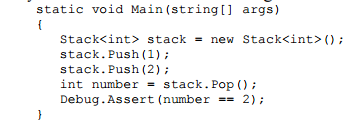
The Main() method uses the object-based stack:



Modify the object-based stack to a generic stack. First, save the file under the name GenericStack.cs. Next, change the object-based stack to a generic stack, by adding to the class definition, and by replacing the use of an object with a type parameter T:



Modify the Main() method to use the generic stack:

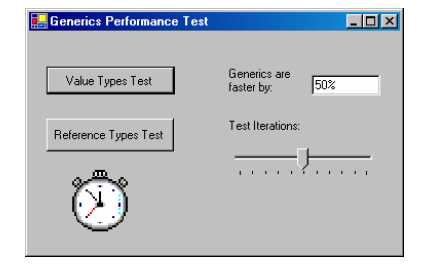


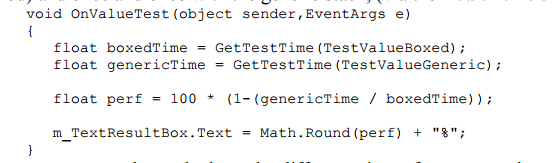
Build and test to make sure all is well. Try using Stack with integers and strings, to experience first-hand the productivity benefits of generics.

Task 2 – Testing the Generic Stack Next, you will test the performance advantage of the generic stack you built in the previous step. Open the solution GenericsPerfs.sln. The solution is a micro-benchmark application, which you will use to execute a stack in a tight loop. It will lets you experiment with value and reference types on an Objectbased stack and a generic stack, as well as changing the number of loop iterations to see the effect generics have on performance

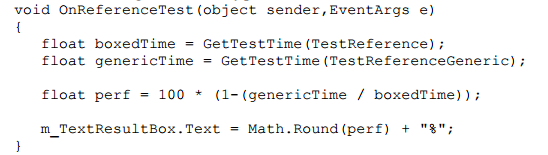
Copy the files ObjectStack.cs and GenericStack.cs to the GenericsPerfs solution folder by adding the files to the project: right-click on the GenericsPerfs solution in Microsoft® Visual Studio® 2005, and select Add|Add Existing Item… to add ObjectStack.cs and GenericStack.cs to the solution. Build the solution to make sure they were added properly. The performance tester uses a delegate to invoke different tests, using the delegate TestMethod, defined as: delegate void TestMethod(); The test routine GetTestTime() uses the Stopwatch class to measure the

test time. It invokes the delegate, and returns the test duration in milliseconds:

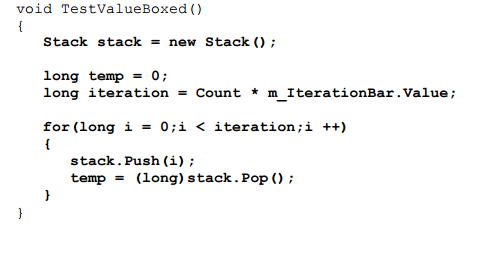


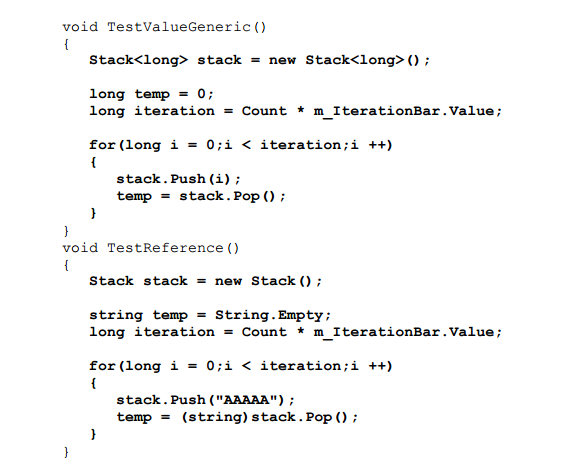
The method OnValueTest() handles the click event for the Value Type Test button. It calls GetTestTime(), invoking it once with the object-based stack (via the TestValueBoxed() method) and once and once with the generic stack, (via the TestValueGeneric() method): 

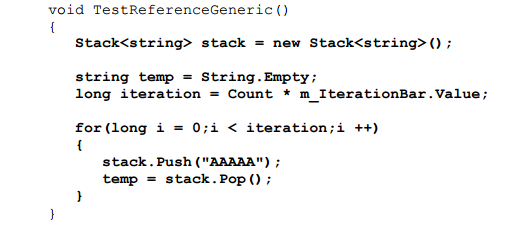
OnValueTest() then calculates the difference in performance and updates the display. In a similar manner, the OnReferenceTest() method handles the Click event for the Reference Type Test button, using reference types:



Next, you need to add the code for the actual four test methods. Add the following code to the test methods:







Build and run the performance test client. Experiment with different number of test iteration to see the effect of generics both on value types (about 50% performance improvement ) and on reference types (10% performance improvement). 