/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Part 11: DelegatesAndEvents

\*

\* Topics: 1. Applies covariance in the use of a delegate.

\* 2. Applies contravariance in the use of a delegate.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CovarianceAndContravarianceDemo

{

class Animal

{ }

class Mammal : Animal

{ }

class Tiger : Mammal

{

public void TigerName()

{

Console.WriteLine(

"\nHello, my name is Tiger Lady.");

}

}

class Giraff : Mammal

{

public void GiraffName()

{

Console.WriteLine(

"\nHello, my name is Mr. Giraff.");

}

}

class Program

{

// Covariance means only one delegate is needed to cover

// all zoo animals instead of a separate delegate for

// each type of animal when the RETURN TYPE is a base

// of the return types returned from methods called by

// the delegate.

delegate Animal NewAnimalDelegate();

// Contravariance means a need to declare a separate

// delegate for each PARAMETER TYPE. However, only

// one method is needed that has a parameter of a base

// type to the parameter types in each of the delegates.

delegate void ShowTigerDelegate(Tiger t);

delegate void ShowGiraffDelegate(Giraff g);

static Tiger NewTiger()

{

return new Tiger();

}

static Giraff NewGiraff()

{

return new Giraff();

}

// Method called only once, but executed multiple times

// for each delegate in the delegate collection.

static void AddZooAnimal(NewAnimalDelegate newZooAnimals)

{

foreach (NewAnimalDelegate nad in newZooAnimals.GetInvocationList())

{

Tiger tiger = new Tiger();

Giraff giraff = new Giraff();

Animal zooAnimal = nad();

// No explicit casting is needed here as zooAnimals collection is Animal type.

zooAnimals.Add(zooAnimal);

// However, what if we wanted to introduce each of these animal types by

// calling their individual methods. We will then need to explicitly cast.

Type animalType = zooAnimal.GetType();

if (animalType == tiger.GetType())

{

tiger = (Tiger)zooAnimal;

tiger.TigerName();

}

else if (animalType == giraff.GetType())

{

giraff = (Giraff)zooAnimal;

giraff.GiraffName();

}

else

{

Console.WriteLine("\nNo code for this animal type.");

}

}

}

// Contravariance means only one method is needed to show all

// zoo animals.

static void ShowZooAnimal(Animal a)

{

Console.WriteLine("\n" + a.ToString());

}

static List<Animal> zooAnimals;

static void Main()

{

zooAnimals = new List<Animal>();

// Covariance being applied by adding methods to a delegate with

// different return types.

NewAnimalDelegate zooAnimal = NewTiger;

zooAnimal += NewGiraff;

// Covariance adds efficiency by needing to call method below

// only once instead of multiple times for each zoo animal.

AddZooAnimal(zooAnimal);

// Contravariance means only one method needed to be coded to

// be referenced by any delegate with a parameter type that is a

// derivation of the parameter in the method being referenced.

ShowTigerDelegate ShowTiger = ShowZooAnimal;

ShowTiger(NewTiger());

ShowGiraffDelegate ShowGiraff = ShowZooAnimal;

ShowGiraff(NewGiraff());

Console.Write("\nPress any key to end.");

Console.ReadLine();

}

}

}