**Unit Test #2**

**Due Wednesday 14-Oct-2020 11:59pm**

* What is wrong with this code, and rewrite it to fix the issues, if there are any:

public class Zoo

{

private string name;

public string Name

{

get

{

// this.name = value; <-- Incorrect

return this.name; <-- Correct

}

set

{

// return this.name; <-- Incorrect

this.name = value; <-- Correct

}

}

}

The get function is suppose to return this.name since it is "getting" the name field. It is not suppose to set it equal to a value in get. Set should be this.name = value since it is the job of the set function to set the variable that was returned equal to the public variable.

* What is wrong with this code, and rewrite it to fix the issues, if there are any:

public sealed class Circus

{

public string name;

}

static class Program

{

static void Main()

{

Circus myCircus = new Circus();

}

}

- This code works fine in this example. If we wanted to inherit from Circus, we couldn't because it is a sealed class. We can however created objects from a sealed class so there is nothing wrong with this code.



3. Meet the Tuple! C# allows you to define collections of objects by using the Tuple class. If you want a collection of a string, int and double to store a country's average rainfall, you can define:

Tuple<string,int,double> avgRainfall = new Tuple<string,int,double>("USA",2017,32.21);

There is a shorthand for doing this using only parentheses:

(string,int,double) avgRainfall = ("USA", 2017, 32.21);

We know that a SortedList<> of doubles can be indexed on strings by defining it as:

SortedList<string, double> sortedList = new SortedList<string,double>();

Using a tuple, you can have a sorted list of average rainfall indexed on country and year, where average rainfall is a double, country is a string and year is an int (try it!):

SortedList<(string,int), double> avgRainfall = new SortedList<(string,int), double>();

// where the index key fields are string = country name and int = year

// add a row for USA/2017

avgRainfall[ ("USA",2017) ] = 32.21;

// access the SortedList via the tuple

Console.WriteLine(avgRainfall[ ("USA",2017) ]);

Given the formula z = 4y3 + 2x2 - 8w + 7 write the code for a SortedList<> based on the tuple (double,double,double) representing w, x and y. Use the necessary for() loops to store the values of z (which would also be a double) for the following ranges of w, x and y using the tuple as the index key:

* -2 <= w <= 0 in 0.2 increments
* -1 <= y <= 1 in 0.1 increments
* 0 <= x <= 4 in 0.1 increments.

* Convert this schUML to C#. Note that capitalized field names are properties, which may have a read-only tag (:r). All member methods should be empty code blocks, if applicable.



* Using the code from #4: Overload the boolean operators ==, !=, <, >, <= and >= for the Tardis class such that whichDrWho is used to compare Tardis objects. But write the logic so that the 10th Doctor (whichDrWho == 10) is greater than the rest, otherwise they should compare based on their value.
* Using the classes from #5: Write code which creates a Tardis object and a PhoneBooth object and passes each to a UsePhone(object obj) method using the interface to call MakeCall() and HangUp(). UsePhone() should have the following signature:

static void UsePhone(object obj)

* Using the code from #6: Without changing the signature of UsePhone(), add the code to the UsePhone() method to call OpenDoor() if it's a PhoneBooth object and TimeTravel() if it's a Tardis object.
* Create a schUML for a data model for one of your favorite hobbies. There should be at least 1 abstract class, 1 property, 1 abstract method, 1 virtual method, 2 derived child classes and 2 interfaces.
* Convert the schUML from #8 into C#.
* Write a console app that includes your classes in #9, creates objects of your 2 child classes and calls a method that demonstrates polymorphism.

Your method should have an object as the parameter such as:

static void MyMethod(object obj)

You should call myMethod() with each of the objects and use the interfaces and parent or derived classes to call the supported methods, based on the object type.

* Given the following code:



* List the order of the line numbers that are executed when myObj is created at line #29.

1. int nVal is set through the constructor. Since the parent class has nVal added to this.myInt, the derived class also adds nVal to its myInt field.

2. this.myInt is set equal to the equation on the right which first adds this.myInt and two, then multiplies the result of that by four.

* What is the value of myObj.myInt at line #30?  
  myInt is equal to 176.
* Write the code to output the contents of the following SortedList<> using Console.WriteLine(); for each list element.

SortedList<string,DateTime> friendBirthdays = new SortedList<string,DateTime>();

where string = friend's name and DateTime = their birthdate (use the format string "MM/dd/yyyy" in your output)

// Of course there has to be actual dates added to the list to display anything. For example you might want to write

friendBirthdays.Add("Date:" , new DateTime(2020, 10, 14));

foreach ( var dates in friendBirthdays)

{

Console.WriteLine(dates);

}

* Write the code to make a shallow copy of objectA into objectB:

MyClass objectA = new MyClass();

MyClass objectB;

MyClass objectB = objectA;

* Write a console application which converts the following code from using struct Friend to public class Friend and generates the same output:

using System;

namespace StructToClass

{

struct Friend

{

public string name;

public string greeting;

public DateTime birthdate;

public string address;

}

class Program

{

static void Main(string[] args)

{

Friend friend;

Friend enemy;

// create my friend Charlie Sheen

friend.name = "Charlie Sheen";

friend.greeting = "Dear Charlie";

friend.birthdate = DateTime.Parse("1967-12-25");

friend.address = "123 Any Street, NY NY 12202";

// now he has become my enemy

enemy = friend;

// set the enemy greeting and address without changing the friend variable

enemy.greeting = "Sorry Charlie";

enemy.address = "Return to sender. Address unknown.";

Console.WriteLine($"friend.greeting => enemy.greeting: {friend.greeting} => {enemy.greeting}");

Console.WriteLine($"friend.address => enemy.address: {friend.address} => {enemy.address}");

}

}

}

***Submission***

Upload this completed document, the schUML for #8 (.svg file) and the GitHub URL's for the project folders for #10 and #14 to the corresponding myCourses dropbox. For #4, 5, 6, and 7, only submit the GitHub URL for the final .cs file for #7.

URL for questions 4, 5, 6, and 7:

<https://github.com/jag8932/myIGME-201/tree/master/Test2_Questions_4_5_6>

URL for 9 and 10:

<https://github.com/jag8932/myIGME-201/tree/master/Questions_9_10_Test2>

I was a bit confused by the submission request for the final question so I am submitting both the .cs and URL.

<https://github.com/jag8932/myIGME-201/tree/master/FinalQuestion_Midterm>

Student you = new Student();

you.Party();