CIS 400: Object-Oriented Design, Development, and Testing

Fall 2023

Exam 1 – 100 points

This test is closed-notes and closed-computers.

There are 10 questions.

Name: _______Solution_______________________________

Score: ____________

1. (3 pts) Inheritance allows a class to:
    a) Implement multiple interfaces
    b) Receive the properties and behaviors of another class
    c) Create static members
    d) Define multiple constructors

2. (3 pts) What is the key difference between a static and a non-static member in C#?
    a) Static members cannot be inherited, while non-static members can
    b) Static members can be modified at runtime, while non-static members cannot
    c) Static members can be accessed using the name of a class, while non-static members cannot (accepted a also)
    d) Static members have access to private fields of other classes

3. (3 pts) What does encapsulation mean?
    a) The process of creating objects from classes
    b) The practice of bundling related data and operations
    c) The inheritance of properties and behaviors from a base class
    d) The use of interfaces to define requirements for objects

4. (3 pts) In object-oriented programming, polymorphism allows:
    a) Objects of different classes to be treated as if they are objects of the same class
    b) Objects of the same class to have different methods
    c) Objects to inherit properties from multiple base classes
    d) Objects to access private fields of other objects

5. (3 pts) When should you use a property instead of a field in a C# class?
    a) When you want to directly access the data from other classes
    b) When you need to define a method for manipulating the data
    c) When you need a simple data storage location without additional logic
    d) When you want to expose read-only or calculated values while hiding the underlying data

6. (22 pts) Consider the *IElection* interface and the *Candidate* class on the last page. Finish the class *VotingMachine* below, which should implement the *IElection* interface.

```
/// <summary> A class representing a voting machine </summary>
public class VotingMachine : IElection
{
   /// <summary> The first candidate </summary>
   public Candidate Cand1 { get; init; }
   /// <summary> The second candidate </summary>
   public Candidate Cand2 { get; init; }

   /// <summary> Constructs a new VotingMachine </summary>
   /// <param name = “name1”>The name of the first candidate </param>
   /// <param name = “name2”>The name of the second candidate </param>
   public VotingMachine(string name1, string name2)
   {
          Cand1 = new Candidate(name1);
          Cand2 = new Candidate(name2);
   }

   public void Vote(string name)
   {
          if (name == Cand1.Name) Cand1.Votes++;
          else if (name == Cand2.Name) Cand2.Votes++;
   }

   public string Winner
   {
          get
          {
                if (Cand1.Votes >= Cand2.Votes) return Cand1.Name;
                else return Cand2.Name;
          }
   }
}
```

7. (21 pts) Draw a UML diagram of *IElection*, *Candidate*, and *VotingMachine*

See PDF

8. (15 pts) Using the *Vehicle*, *Car*, and *Motorcycle* classes from the last page, what prints in the code below?

```
List<Vehicle> list = new List<Vehicle>();
list.Add(new Motorcycle());
list.Add(new Car());
list.Add(new Slingshot());
list.Add(new Vehicle());
foreach (Vehicle v in list)
{
      Console.WriteLine($”Wheels: {v.Wheels}”);
      foreach (string s in v.Description)
      {
            Console.WriteLine(s);
      }
      Console.WriteLine();
}
```

```
Wheels: 4
Vehicle

Wheels: 4
Vehicle
Car

Wheels: 3
Vehicle
Car
Slingshot

Wheels: 4
Vehicle
```

9. (13 pts) Add the property *Drivers* as if it were inside the *Vehicle* class from #8. It should have both get and set access and should have an initial value of 1. If an attempt is made to set *Drivers* to something outside the range of 1-4, you should leave *Drivers* unchanged. (You may also want to add a field.)

```
private int _drivers = 1;
public int Drivers
{
      get => _drivers;
      set
      {
            if (value >= 1 && value <= 4)
            {
                  _drivers = value;
            }
      }
}
```

10. (14 pts) Complete the following unit tests for the *Vehicle* class from #8-9:

```
public class VehicleTests
{
      [Theory]
      [InlineData(1)]
      [InlineData(2)]
      [InlineData(3)]
      [InlineData(4)]
      public void CanSetDriversToValidValue(int drivers)
      {
            Vehicle v = new Vehicle();
            v.Drivers = drivers;
            Assert.Equal(drivers, v.Drivers);









      }

      [Fact]
      public void DefaultDriversIs1Test()
      {
            Vehicle v = new Vehicle();
            Assert.Equal(1, v.Drivers);









      }
}
```

Feel free to remove this portion to make it easier to work.

```
//The following two items are needed for #6-7
/// <summary> Represents an election </summary>
public interface IElection
{
    /// <summary> Casts a vote for the candidate with the given name</summary>
    void Vote(string name);

    /// <summary> Gets the name of the candidate with the most votes </summary>
    string Winner { get; }
}

/// <summary> Represents a candidate in an election </summary>
public class Candidate
{
    /// <summary> The name of this candidate </summary>
    public string Name { get; init; }

    /// <summary> The number of votes this candidate has received</summary>
    public int Votes { get; set; } = 0;

    /// <summary> Constructs a new candidate with the given name </summary>
    public Candidate(string n)
    {
        Name = n;
    }
}
```

(Vehicle classes are on the back)

```
//The following items are needed for #8-10
public class Vehicle
{
      public virtual int Wheels { get; } = 4;

      protected List<string> _description = new List<string>();
      public List<string> Description => _description;

      public Vehicle()
      {
            _description.Add(“Vehicle”);
      }
}

public class Car : Vehicle
{
      public Car()
      {
            _description.Add(“Car”);
      }
}

public class Motorcycle : Vehicle
{
      public int Wheels { get; } = 2;
}

public class Slingshot : Car
{
      public override int Wheels { get; } = 3;
      public Slingshot()
      {
            _description.Add(“Slingshot”);
      }
}
```