Program:	**CPA2/3**
Course:	**INFO-5060**
Professor:	**Tony Haworth**
Project:	***Quiddler™ Scores***
Due Date:	**Friday, Feb 11, 2022 by 11:59 pm**
Last Update:	**Jan 17, 2022 @ 6 p.m. -** *Corrected Discard() in class diagram, page 2*

This project should be done in groups of two.
Students in different sections of the course may work together.

Description

You will create a .NET 5.0 (.NET Core) library assembly using C# that provides game services for the card game Quiddler™. You must also implement a .NET 5.0 client that demonstrates the services provided by the class library by managing user inputs and generating all output in the console.

Purpose

On completing this project you should be able to:

1. develop a .NET class library assembly and a console client
2. use the *interface* type in C# to define and publish component interfaces
3. integrate managed and unmanaged components into a .NET application

Background

You will **partially** implement the game *Quiddler*, a card game in which the players take turns making words using cards in their hand. A card contains either one or two letters along with its point value. The cards are dealt to the players from a deck and the number of cards in each player's hand can vary between 3 and 10. The goal is to earn points and to ultimately "go out" by using all the cards in the hand. Any subset of cards in a hand may be used to form a word. A card only be used once within a word. A word's score is the sum of the points for each card. Here are links to the game instructions and a video that explains it. **However**, you don't need to understand the game rules for this project since it's only about calculating how many points a word is worth.

The following tables describe the complete set of cards in the Quiddler deck:

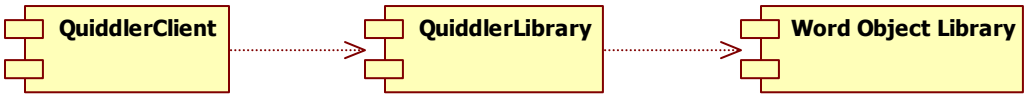
Number of Cards in the Deck

Cards	*Count*
b, c, f, h, j, k, m, p, q, v, w, x, z, cl, er, in, qu, th	2
d, g, l, s, y	4
n, r, t, u	6
i, o	8
a	10
e	12
Total	*118*

Card Point-Values

Cards	*Points*
a, e, i, o	2
l, s t	3
u, y	4
d, m, n r	5
f, g, p	6
h, er, in	7
b, c, k	8
qu, th	9
w, cl	10
v	11
x	12
J	13
z	14
q	15

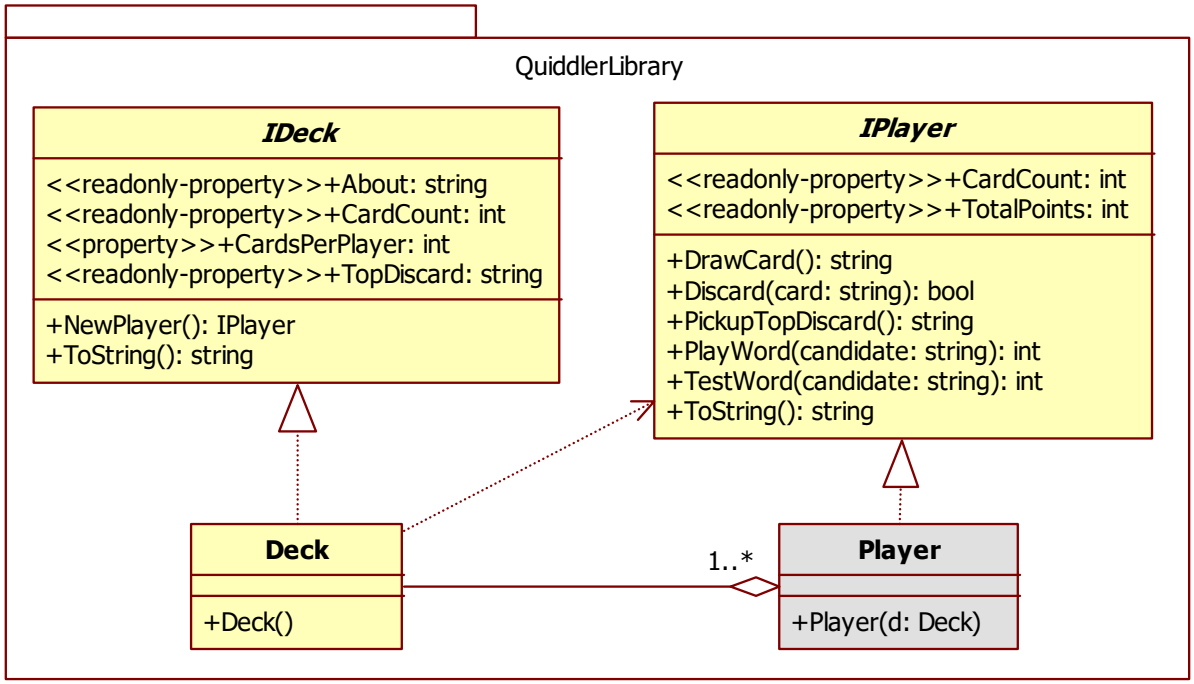
Component Architecture



The diagram above illustrates the software components involved in this project. You will be responsible for developing the *QuiddlerLibrary.dll* assembly and a client. The *Microsoft Word Object Library* is an existing unmanaged (COM) component that should already be installed and registered on any Windows computer that has desktop version of *Microsoft Office* installed and will be used validate words. Note that to use the Word Object Library your *QuiddlerLibrary* must also reference the *Microsoft Office Object Library*.

Exposed Types

QuiddlerLibrary



The *QuiddlerLibrary* component must expose *IDeck* and *IPlayer* interfaces as well as the *Deck* class shown in the diagram above and further explained on the next page. These types must belong to the *QuiddlerLibrary* namespace. **It must not expose any other types.** Note that the *Player* class should not be visible to any client of *QuiddlerLibrary.dll*.

For full marks, the interfaces must be defined exactly as shown in the diagram without any additional members. However, the interfaces only include methods and properties that must be exposed to an external client. You may include any other unexposed members in your *implementation classes* (*Deck* and *Player*) as needed to make your implementation functional. You may also add classes or other types as required to complete your implementation as long as these classes are not visible to the client.

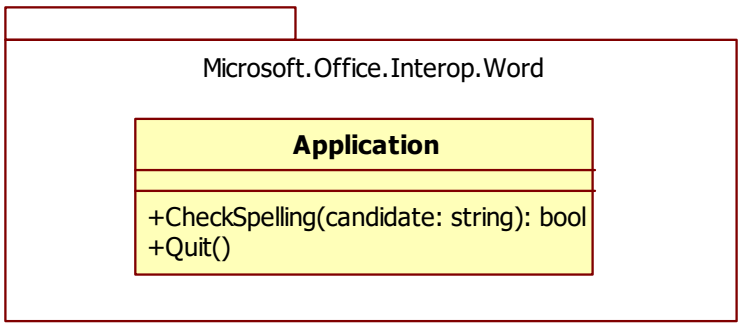
IDeck Interface Properties	
About	Read-only string that provides identifies the library and its developer *(see sample output)*
CardCount	Read-only integer property indicating how many undealt cards remain
CardsPerPlayer	Read-write integer property representing the number of cards initially dealt to each player and which throws an A*rgumentOutOfRangeException* if an attempt is made to assign a value outside the range 3-10 (inclusive)
TopDiscard	Read-only string property indicating the top card on the discard pile *(the top card on the discard pile is the only card in that pile we need to keep track of).* If *TopDiscard* is uninitialized (null) when read, then it should initialize itself by taking the next card from the deck before returning that card string.

IDeck Interface Methods	
NewPlayer	Creates a new player object, immediately populates it with *CardsPerPlayer* cards and returns an *IPlayer* interface reference to the player object.
ToString	Returns a formatted string describing the inventory of cards available in the deck in alphabetical order like "a(8) b(2) c(1) …" *(see sample output).*

IPlayer Interface Properties	
CardCount	Read-only integer property indicating how many cards are in the player's hand
TotalPoints	Read-only integer property indicating how many points in total have been scored by the player from playing words

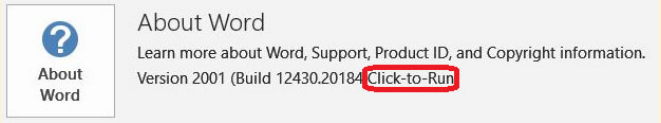
IPlayer Interface Methods	
DrawCard	If the deck is empty it throws an *InvalidOperationException, otherwise* it takes the top card from the deck, adds it to the player's hand and returns the card's letter(s) as a string.
Discard	Accepts a string representing a card and verifies that the card is in the player's hand. If it's not in the hand it returns false, otherwise it removes it from the hand making that card the top card on the deck's discard pile and returns true.
PickupTopDiscard	Takes the top card in the discard pile and adds it to the player's hand. It also returns the value of the card as a string. See the *Tips & Suggestions* section about an easy way to implement the "discard pile".
PlayWord	Accepts a string representing a word (with the card values delimited by spaces like "th er e") and then calls *TestWord*. If the candidate word is worth more than 0 points then all the cards in the word will be removed from the player's hand and the points will be added to the player's *TotalPoints* property. The points for the word are returned by the method.
TestWord	Accepts a string containing a word using the same space-delimited format used by PlayWord() and returns its point value based on three criteria: 1) the player has not used all their cards to form the word (need to discard) 2) the letters of the *candidate* string are a subset of the letters in the current rack object using spaces as delimiters, 3) the candidate provided (after removing the delimiters) is a valid word as tested using the *Application* object's *CheckSpelling()* method. If a candidate word fails to meet either criteria the method returns 0. Note that this method does not modify the player's *TotalPoints*, nor does it remove the cards from the player's hand.
ToString	Returns a string containing all the hand's card values (letters), separated by spaces. This method will override the object's inherited *ToString()* method.

Microsoft Word Object Library

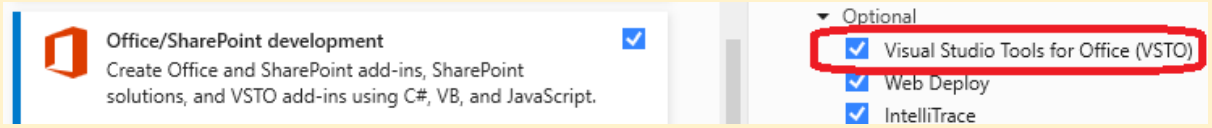


Prerequisites:

- You must have a desktop ("Click-to-Run") version of Microsoft Word installed



- You must have *Visual Studio Tools for Office* installed! This may already have been installed when you installed Visual Studio.



- Your QuiddlerLibrary must reference both of the following *unmanaged* components:
  - *Microsoft Word Object Library* and *Microsoft Office Object Library*

The class diagram above only shows the type and methods required for the project.

Application Class

- Including the namespace, this type is *Microsoft.Office.Interop.Word.Application*.
- The *CheckSpelling()* method also has several optional parameters but only an argument for the *word* parameter must be provided when calling it. Note that only words in **lowercase** format will be properly spellchecked.

Your *QuiddlerLibrary.dll* should be designed to "finalize" the *Application* object by calling the object's *Quit* method before it gets cleaned-up by the garbage collector. This will unload an instance of MS Word that was loaded for your library.

Other Requirements

QuiddlerLibrary

- Must be implemented as a .NET 5.0 (.NET Core) class library assembly using C#
- Must work with your client.
- Must recognize letters in **lowercase** format. May optionally recognize UPPERCASE letters as well.
- Must not directly produce any output, not even error messages
- Must properly unload the Microsoft Word Object Library when finished

Client

- Must be implemented as a .NET 5.0 (.NET Core) console application using C#.
- May only make calls on the *Deck* and *Player* objects via the *IDeck* and *IPlayer* interfaces and may only create Player objects via the *IDeck* interface's *NewPlayer()* method.
- Must demonstrate the features of the *QuiddlerLibrary* assembly by doing the following:
    - Create one *Deck* object.
    - Show QuiddlerLibrary author information using *IDeck.About.*
    - Allow the user to choose the number of cards to be dealt to each player (3 to 10)
    - Allow the user to choose the number of *Player* objects to be generated from the deck and create them using *IDeck.NewPlayer.*
    - Display a complete inventory of the cards in the deck via *IDeck.ToString()* and the total number of cards via *IDeck.CardCount* after the deck is populated and at the beginning of each player's turn.
    - Allow each player to take a turn, in rotation, demonstrating all methods and properties of the *IPlayer* interface and the displaying any values returned in each case.
    - After each player has taken a turn, give the user the choice to either quit, or allow each player to play another turn using the same player objects. However, if a player is out of cards at the end of their turn then the game is over after each of the other players has completed a turn in the rotation.
    - Report the total points scored by each player before exiting the program.
- Must handle exceptions for all calls on *QuiddlerLibrary* methods/properties.
- Must perform any additional steps required to ensure the *QuiddlerLibrary* objects are disposed of properly

Tips and Suggestions

- The interfaces *IDeck* and *IPlayer* may be implemented either implicitly or explicitly. For example:

```
public class Deck : IDeck
{
    public IPlayer NewPlayer()
    {
        // implicit
```

```
public class Deck : IDeck
{
    IPlayer IDeck.NewPlayer()
    {
        // explicit
```

- To award points for words in your *QuiddlerLibrary.dll* you'll need to ensure the word consists of cards that are a subset of the player's available cards. One technique is to create a data structure of pairs with the key value being the card's letter(s) and the data value being the frequency of the card in the hand. You could use a similar data structure for the cards used in the word. Next, loop through the second data structure comparing the frequency of each letter with the frequency of the same letter in the first data structure. One suitable data structure for this in .NET is the *Dictionary<>* type.
- The *Deck* object doesn't need to store ALL the cards in the discard pile, but only the top card. If a player picks-up this card it can be temporarily set to an empty string since the card below it in the pile will never need to be picked-up. This is true because a player must always discard a card at the end of their turn.
- You may find the performance of your client to be poor (i.e. sluggish) if your library creates a new *Application* object for every word it needs to validate. To avoid this, consider creating a single reusable instance of *Application* that will exist until the client application terminates.
- The next two pages show output from a sample console client:

Sample Output

```
Test Client for: Quiddler (TM) Library, © 2022 T. Haworth

Deck initialized with the following 118 cards...
a(10) b(2)  c(2)  d(4)  e(12) f(2)  g(4)  h(2)  i(8)  j(2)  k(2)  l(4)
m(2)  n(6)  o(8)  p(2)  q(2)  r(6)  s(4)  t(6)  u(6)  v(2)  w(2)  x(2)
y(4)  z(2)  cl(2) er(2) in(2) qu(2) th(2)

How many players are there? (1-8): 2
How many cards will be dealt to each player? (3-10): 6

Cards were dealt to 2 player(s).
The top card which was 'y' was moved to the discard pile.

--------------------------------------------------------------------------
Player 1  (0 points)
--------------------------------------------------------------------------

The deck now contains the following 105 cards...
a(9)  b(2)  c(2)  d(4)  e(9)  f(2)  g(4)  h(2)  i(6)  j(2)  k(2)  l(2)
m(2)  n(6)  o(7)  p(2)  q(1)  r(6)  s(4)  t(6)  u(5)  v(2)  w(2)  x(2)
y(3)  z(2)  cl(2) er(2) in(2) qu(2) th(1)

Your cards are [e l i e q th].
Do you want the top card in the discard pile which is 'y'? (y/n): n
The dealer dealt 'a' to you from the deck.
The deck contains 104 cards.
Your cards are [e l i e q th a].
Test a word for its points value? (y/n): y
Enter a word using [e l i e q th a] leaving a space between cards: l i t h e
The word [l i t h e] is worth 0 points.
Test a word for its points value? (y/n): y
Enter a word using [e l i e q th a] leaving a space between cards: l i th e
The word [l i th e] is worth 16 points.
Do you want to play the word [l i th e]? (y/n): y
Your cards are [e q a] and you have 16 points.
Enter a card from your hand to drop on the discard pile: q
Your cards are [e a].

--------------------------------------------------------------------------
Player 2  (0 points)
--------------------------------------------------------------------------

The deck now contains the following 104 cards...
a(8)  b(2)  c(2)  d(4)  e(9)  f(2)  g(4)  h(2)  i(6)  j(2)  k(2)  l(2)
m(2)  n(6)  o(7)  p(2)  q(1)  r(6)  s(4)  t(6)  u(5)  v(2)  w(2)  x(2)
y(3)  z(2)  cl(2) er(2) in(2) qu(2) th(1)

Your cards are [u o i l e a].
Do you want the top card in the discard pile which is 'q'? (y/n): y
Your cards are [u o i l e a q].
Test a word for its points value? (y/n): y
Enter a word using [u o i l e a q] leaving a space between cards: q u a i l
The word [q u a i l] is worth 26 points.
Do you want to play the word [q u a i l]? (y/n): y
Your cards are [o e] and you have 26 points.
Enter a card from your hand to drop on the discard pile: e
Your cards are [o].

Would you like each player to take another turn? (y/n): y

--------------------------------------------------------------------------
Player 1  (16 points)
--------------------------------------------------------------------------
```

```
The deck now contains the following 104 cards...
a(8)  b(2)  c(2)  d(4)  e(9)  f(2)  g(4)  h(2)  i(6)  j(2)  k(2)  l(2)
m(2)  n(6)  o(7)  p(2)  q(1)  r(6)  s(4)  t(6)  u(5)  v(2)  w(2)  x(2)
y(3)  z(2)  cl(2) er(2) in(2) qu(2) th(1)

Your cards are [e a].
Do you want the top card in the discard pile which is 'e'? (y/n): n
The dealer dealt 'e' to you from the deck.
The deck contains 103 cards.
Your cards are [e a e].
Test a word for its points value? (y/n): n
Enter a card from your hand to drop on the discard pile: e
Your cards are [a e].

------------------------------------------------------------------------
Player 2  (26 points)
------------------------------------------------------------------------

The deck now contains the following 103 cards...
a(8)  b(2)  c(2)  d(4)  e(8)  f(2)  g(4)  h(2)  i(6)  j(2)  k(2)  l(2)
m(2)  n(6)  o(7)  p(2)  q(1)  r(6)  s(4)  t(6)  u(5)  v(2)  w(2)  x(2)
y(3)  z(2)  cl(2) er(2) in(2) qu(2) th(1)

Your cards are [o].
Do you want the top card in the discard pile which is 'e'? (y/n): n
The dealer dealt 't' to you from the deck.
The deck contains 102 cards.
Your cards are [o t].
Test a word for its points value? (y/n): y
Enter a word using [o t] leaving a space between cards: t o
The word [t o] is worth 0 points.
Test a word for its points value? (y/n): n
Enter a card from your hand to drop on the discard pile: o
Your cards are [t].

Would you like each player to take another turn? (y/n): n

Retiring the game.

The final scores are...
------------------------------------------------------------------------
Player 1: 16 points
Player 2: 26 points
```

Grading Scheme

1. The *IT Division Policy on Missed Evaluations and Evaluation Deadlines* will be applied to this project.

2. Project submitted by the posted submission deadline will be evaluated as follows:

Description	Marks
QuiddlerLibrary is a .NET 5.0 class library project and correctly defines, implements and exposes the published interfaces *IDeck* and *IPlayer* and the public class *Deck*, but hides all other types.	10
These features of the *IDeck* and *IPlayer* interfaces in *QuiddlerLibrary* work according to the requirements provided: 1. IDeck : About, CardCount, CardsPerPlayer, TopDiscard, NewPlayer(), ToString() 2. IPlayer : CardCount, TotalPoints, DrawCard(), Discard(), PickupTopDiscard(), PlayWord(), TestWord(),ToString()	20
QuiddlerLibrary uses the *Microsoft Word Object Library* component's *Application* class effectively to validate candidate words. It also invokes the component's *Quit()* method before the client application terminates.	5
Client submitted effectively demonstrates all *QuiddlerLibrary* features exclusively via the *IDeck* and *IPlayer* interfaces and handles exceptions thrown by the *QuiddlerLibrary*	15
Maximum deduction for generating output from *QuiddlerLibrary*	-10
Maximum deduction for poor coding style (unclear and inconsistent or undescriptive naming of code elements, poor code alignment, missing and/or inadequate inline and header comments)	-5
Maximum deduction for an incomplete submission (missing source code or required VS project files)	-50
Maximum Total	**50**

Submit

If working with another student only one of you needs to submit your solution, but you will both be responsible for ensuring you submit it on time. Make sure to include a submission note (in the drop box) that identifies both students. Use the INFO-5060 *Project 1* dropbox in *FanshaweOnline* to submit an archive (zip or rar file) containing your complete, cleaned Visual Studio solution(s) containing all your source code. I should only have to rebuild your project before testing without changing any project settings or source code!

Code submissions to the FOL drop box should be made on time! Late projects will be subject to late penalties as follows:

- A penalty of 20% will be deducted if your submission is received < 24 hours late.
- Your project will receive a mark of 0 (zero) if it is more than 1 day late.

Project Corrections

If any corrections or changes are necessary they will be posted on the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project.