CIT195 LB [META 2302]

Started on	Monday, 10 April 2023, 8:14 PM
State	Finished
Completed on	Monday, 10 April 2023, 8:36 PM
Time taken	22 mins 15 secs
Grade	**49.00** out of 50.00 (**98**%)

5.00

Configure the query to join the two lists below on StudentID and display the Student's Name, Age, Major, and GPA.

```
        // Student collection
        IList < Student > studentList = new List < Student >() {
                new Student() { StudentID = 1, StudentName = "Frank Furter", Age = 55,
Major="Hospitality"} ,
                new Student() { StudentID = 2, StudentName = "Gina Host", Age = 41,
Major="Hospitality"} ,
                new Student() { StudentID = 3, StudentName = "Cookie Crumb",  Age =
41, Major="CIT" } ,
                new Student() { StudentID = 4, StudentName = "Ima Script",  Age = 18,
Major="CIT" } ,
                new Student() { StudentID = 5, StudentName = "Cora Coder",  Age = 35,
Major="CIT" } ,
                new Student() { StudentID = 6, StudentName = "Ura Goodchild" , Age =
20, Major="Marketing"} ,
                new Student() { StudentID = 7, StudentName = "Take Mewith" , Age = 19,
Major="Aerospace Engineering" }
            };
        // Student GPA Collection
        IList < StudentGPA > studentGPAList = new List < StudentGPA > () {
                new StudentGPA() { StudentID = 1,  GPA=4.0} ,
                new StudentGPA() { StudentID = 2,  GPA=3.5} ,
                new StudentGPA() { StudentID = 3,  GPA=2.0 } ,
                new StudentGPA() { StudentID = 4,  GPA=1.5 } ,
                new StudentGPA() { StudentID = 5,  GPA=4.0 } ,
                new StudentGPA() { StudentID = 6,  GPA=2.5} ,
                new StudentGPA() { StudentID = 7,  GPA=1.0 }
            };
```

```
gpa => gpa.StudentID,
(student, gpa) => new
{
    StudentName = student.StudentName,
    Age = student.Age,
    Major = student.Major,
    GPA = gpa.GPA
});
```

Your answer is correct.

5.00

Evaluate each statement about delegates and select either true or false.

A delegate can be passed as a parameter to methods. True ✔

The maximum number of methods referenced by a single delegate is one. False ✔

To invoke a delegate, use the delegate object and pass any parameters needed by the method it references. True ✔

In the code shown below, line #1 is declaring the delegate. False ✔

01 Calculator c = new Calculator(Program.Addition);
02 Console.WriteLine($"Addition of 5 and 10 is : {c(5,10)}");

In the code shown below, a Calc delegate can be used for either method when it is instantiated. False ✔

```
class Numbers
{
    public delegate void Calc(int x);

    public static int num { get; set; } = 0;

    public static void Addition(int a)
    {
         num += a;
    }
    public static int Multiplication(int a, int b)
    {
        int num = a * b;
        return num;
    }

}
```

Your answer is correct.

5.00

Evaluate each statement about properties and select either true or false.

Properties include accessor methods. True ✔

Properties cannot be immutable False ✔

Properties support encapsulation by hiding the private field. True ✔

Auto-implemented properties generate get and set methods for you. True ✔

Properties are not used in interfaces. False ✔

Your answer is correct.

4.00

Evaluate each statement about records and select either true or false.

The record declaration below allows you to directly access and change any value using the set accessor. False ✔

```
public record Person(string FirstName, string LastName);
```

You can print the record object to the console using the object name to see the properties and their values. True ✔

Using a record object, you can directly access and display any value using the get accessor. True ✔

Example:

```
public record Person(string FirstName, string LastName);


        public static void Main()
        {
            Person person = new("Scooby", "Doo");
            Console.WriteLine($"First name={person.FirstName}");
            Console.WriteLine($"Last name={person.LastName}");
        }
```

The record declaration below is non-positional. True ✔

```
        public record Person()
        {
            public string FirstName{ get; init; }
            public string LastName { get; init; }
            public string City { get; init; }
            public string State { get; init; }
            public string PostalCode { get; init; }


        }
```

Question 5

Correct

Mark 3.00 out of 3.00

Select the code from the dropdown lists to create a generic method.

```
static void printArray < T ✔ > ( T[] ✔ array)
        {
            foreach ( T ✔ a in array)
            {
                Console.WriteLine(a);
            }
        }
```

Your answer is correct.

5.00

Select the correct type of collection based on the description given.

A collection of values that can be identified and retrieved by using keys rather than indexes.	Dictionary ✔
A first-in, last-out (FILO) data structure with methods to add an item onto the top of the structure, remove an item from the top of the structure, and examine the item at the top of the structure without removing it.	Stack ✔
A double-ended ordered list, optimized to support insertion and removal at either end. This collection can be accessed FIFO or FILO, and it supports random access.	Linked List ✔
A first-in, first-out data structure, with methods to add an item to one end of the structure, remove an item from the other end, and examine an item without removing it.	Queue ✔
An unordered set of values that is optimized for fast retrieval of data. It provides set-oriented methods for comparing, combining, or intersecting data sets.	Hash Set ✔

Your answer is correct.

1.00

Given the class shown below, how would you instantiate a class object for an integer array with a size of 10?

```
public class MyGenericArray < T >
{
    private T[] array;

    public MyGenericArray(int size)
    {
        array = new T[size + 1];
    }
    public T getItem(int index)
    {
        return array[index];
    }
    public void setItem(int index, T value)
    {
        array[index] = value;
    }
}
```

a. `MyGenericArray < int > intArray = new MyGenericArray < int >(10);` ✔

b. `int[] intArray = new int[10];`

c. `MyGenericArray < int > intArray = new MyGenericArray;`

d. `MyGenericArray  intArray = new MyGenericArray[10];`

Your answer is correct.

1.00

Given the code shown below, which line declares the delegate?

```
 namespace test2
{
    public delegate void Notify();

    public class myBusiness
    {
        public event Notify Done;
        public void StartProcess()
        {
            Console.WriteLine("A long day of meetings has begun.");
            OnProcessCompleted();
        }
        protected virtual void OnProcessCompleted()
        {
            Done?.Invoke();
        }
    }
    class Program
    {
        public static void Main()
        {
            myBusiness meeting = new myBusiness();
            meeting.Done += EndOfDay;
            meeting.StartProcess();
        }
        public static void EndOfDay()
        {
            Console.WriteLine("Done for the day!");
        }
    }
}
```

a. protected virtual void OnProcessCompleted() { Done?.Invoke(); }

b. meeting.Done += EndOfDay;

c. public delegate void Notify(); ✔

d. public event Notify Done;

Your answer is correct.

1.00

Given the code shown below, which line registers the event?

```
 namespace test2
{
    public delegate void Notify();

    public class myBusiness
    {
        public event Notify Done;
        public void StartProcess()
        {
            Console.WriteLine("A long day of meetings has begun.");
            OnProcessCompleted();
        }
        protected virtual void OnProcessCompleted()
        {
            Done?.Invoke();
        }
    }
    class Program
    {
        public static void Main()
        {
            myBusiness meeting = new myBusiness();
            meeting.Done += EndOfDay;
            meeting.StartProcess();
        }
        public static void EndOfDay()
        {
            Console.WriteLine("Done for the day!");
        }
    }
}
```

a. protected virtual void OnProcessCompleted() { Done?.Invoke(); } ✖

b. public event Notify Done;

c. meeting.Done += EndOfDay;

d. public delegate void Notify();

Your answer is incorrect.

1.00

Given the studentList shown below, which query counts students who are over 18 years old?

```
IList < Student > studentList = new List < Student >() {
      new Student() { StudentID = 1, StudentName = "Frank Furter", Age = 55,
Major="Hospitality", Tuition=3500.00} ,
      new Student() { StudentID = 1, StudentName = "Gina Host", Age = 21,
Major="Hospitality", Tuition=4500.00 } ,
      new Student() { StudentID = 2, StudentName = "Cookie Crumb",  Age = 21,
Major="CIT", Tuition=2500.00 } ,
      new Student() { StudentID = 3, StudentName = "Ima Script",  Age = 48,
Major="CIT", Tuition=5500.00 } ,
      new Student() { StudentID = 3, StudentName = "Cora Coder",  Age = 35,
Major="CIT", Tuition=1500.00 } ,
      new Student() { StudentID = 4, StudentName = "Ura Goodchild" , Age = 40,
Major="Marketing", Tuition=500.00} ,
      new Student() { StudentID = 5, StudentName = "Take Mewith" , Age = 29,
Major="Aerospace Engineering", Tuition=5500.00 }
};
```

a.
```
countAge = studentList.Count();
```

b.
```
countAge = from s in studentList
           where s.Age>18
           s.Count();
```

c. ✔
```
countAge = studentList.Count(s=>s.Age>18);
```

d.
```
countAge = studentList.Where(s=>s.Age>18).Count(s => s.Age);
```

Your answer is correct.

1.00

Select the statement that creates the indexer in the following example

```
class Gym
 {
     private string[] members = new string[100];
     public string type {get;set;}
     public double price {get;set;}
     public string this[int i]
     {
         get
         {
             return names[i];
         }
         set
         {
             names[i] = value;
         }
     }
 }
```

a. `public string type {get;set;}`

b. `private string[] members = new string[100];` ✔

c. `class Gym`

d.

```
        {
            return names[i];
        }
        set
        {
            names[i] = value;
        }
    }
```

e.
```
public double price {get;set;}
```

Your answer is correct.

Question 12

Correct

Mark 1.00 out of 1.00

What type of builtin delegate should you use for the method below?

```
static int Sum(int x, int y)
{
    return x + y;
}
```

a. Action

b. Custom delegate

c. Func ✔

d. Event delegate

Your answer is correct.

2.00

Which 2 queries will retrieve C# Tutorials from the list below? Choose 2.

```
IList < string > stringList = new List < string > ()
{
"C# Tutorials",
"Advanced C# Tutorials",
"LINQ Query Tutorials",
"Learn C++",
"MVC Tutorial" ,
"MVC C# Tutorials",
"Beginning RazorPages"
};
```

a.
```
var result = stringList.Where(s => s.Contains("C# Tutorials"));
```

b.
```
var result = stringList.Where(s => "C# Tutorials");
```

c.
```
var result = from s in stringList
             where s.Contains("C# Tutorials")
             select s;
```

d.
```
var result = from s in stringList
             .Where(s => s.Contains("C# Tutorials"))
              select s;
```

Your answer is correct.

1.00

Which query groups the list below by Age?

```
IList studentList = new List() {
        new Student() { StudentID = 1, StudentName = "Frank Furter", Age = 55,
Major="Hospitality"} ,
        new Student() { StudentID = 1, StudentName = "Gina Host", Age = 41,
Major="Hospitality"} ,
        new Student() { StudentID = 2, StudentName = "Cookie Crumb",  Age = 21,
Major="CIT" } ,
        new Student() { StudentID = 3, StudentName = "Ima Script",  Age = 38,
Major="CIT" } ,
        new Student() { StudentID = 3, StudentName = "Cora Coder",  Age = 35,
Major="CIT" } ,
        new Student() { StudentID = 4, StudentName = "Ura Goodchild" , Age = 20,
Major="Marketing"} ,
        new Student() { StudentID = 5, StudentName = "Take Mewith" , Age = 19,
Major="Aerospace Engineering" }
};
```

a.
```
var groupedAge = from s in studentList
                 orderby s.Age
                 group s by s.Major;
```

b.
```
var groupedAge = from s in studentList
                 orderby s.Age
                 group s by s.ID;
```

c.
```
var groupedAge = from s in studentList
                 orderby s.StudentName
                 group s by s.Major;
```

d.
```
var groupedAge = studentList.OrderBy(o=>o.Age).GroupBy(s => s.Major);
```

Your answer is correct.

2.00

Move the code to create a query that will retrieve the average age of students over 18 and below 55 for the list shown below:

```
IList < Student > studentList = new List < Student >() {
      new Student() { StudentID = 1, StudentName = "Frank Furter", Age = 17,
Major="Hospitality", Tuition=3500.00} ,
      new Student() { StudentID = 1, StudentName = "Gina Host", Age = 21,
Major="Hospitality", Tuition=4500.00 } ,
      new Student() { StudentID = 2, StudentName = "Cookie Crumb",  Age = 21,
Major="CIT", Tuition=2500.00 } ,
      new Student() { StudentID = 3, StudentName = "Ima Script",  Age = 18,
Major="CIT", Tuition=5500.00 } ,
      new Student() { StudentID = 3, StudentName = "Cora Coder",  Age = 65,
Major="CIT", Tuition=1500.00 } ,
      new Student() { StudentID = 4, StudentName = "Ura Goodchild" , Age = 40,
Major="Marketing", Tuition=500.00} ,
      new Student() { StudentID = 5, StudentName = "Take Mewith" , Age = 29,
Major="Aerospace Engineering", Tuition=5500.00 }
};
```

- ✔ var avgAge=
- ✔ studentList.Where(s=>s.Age>18)
- ✔ .Where(s=>s.Age<55)
- ✔ .Average(s=>s.Age)
- ✔ ;

Your answer is correct.

1.00

Given the code segment shown below, which line invokes the operator == method?

```
01    Calculator first = new Calculator();
02    Calculator second= new Calculator();
03    first.number = r.Next(10, 20);
04    second.number = r.Next(10, 20);
05    Console.WriteLine($"Number1 = {first.number} and Number2= {second.number}");
06    Console.WriteLine($"Is {first.number} = {second.number}?   {first == second}");
07    Console.WriteLine($"Is {first.number} != {second.number}?   {first != second}");
08    Console.WriteLine($"Is {first.number} > {second.number}?   {first > second}");
09    Console.WriteLine($"Is {first.number} < {second.number}?   {first < second}");
```

a. 08

b. 09

c. 05

d. 06 ✔

e. 07

Your answer is correct.

2.00

Which 2 operator methods must be overloaded together? Choose 2.

- [x] a. operator > ✔
- [ ] b. operator --
- [ ] c. operator ++
- [ ] d. operator +
- [x] e. operator < ✔
- [ ] f. operator -

Your answer is correct.

Question 18

Correct

Mark 1.00 out of 1.00

Which operator is overloaded in the code segment shown below?

```
Trip day1 = new Trip(new DateTime(2023,10,1),100,8.5f);
Trip day2 = new Trip(new DateTime(2023,10,2), 100, 7.5f);
Trip TotalTrip = new Trip();
TotalTrip += day1;
TotalTrip+= day2;
```

- a. operator +=
- b. operator + ✔
- c. operator =
- d. operator ++

Your answer is correct.

5.00

Evaluate each statement about operator overloading and select either true or false.

Overloaded operator methods must be private and static. False ✔

Overloaded operator methods can be virtual. False ✔

Overloading a unary operator will require 2 parameters. False ✔

Overloading a binary operator will require 2 parameters. True ✔

Overloading an operator within a class requires at least 1 class object as a parameter. True ✔

Your answer is correct.

Question 20

Correct

Mark 3.00 out of 3.00

Arrange the pseudocode in the correct order for a delegate that handles an event.

✔ Declare a delegate.

✔ Declare an object of the delegate with the event keyword.

✔ Create a class object

✔ Use the class object and event delete to register the event handler.

✔ Trigger the event

Your answer is correct.