

Advanced C# Exam

1. You are developing an application that includes a class named Order. The application will store a collection of Order objects.

The collection must meet the following requirements:

- ☞ Use strongly typed members.
- ☞ Process Order objects in first-in-first-out order.
- ☞ Store values for each Order object.
- ☞ Use zero-based indices.

You need to use a collection type that meets the requirements.

Which collection type should you use?

- A. Queue<T>
- B. SortedList
- C. LinkedList<T>
- D. HashTable
- E. Array<T>

The answer to this question is letter A. Queue. A queue is created when a programmer wants to add items to a collection and keep the order that they are added. Because the question asks for the collection to be first-in-first-out order that is what makes it clear that Queue<T>.

Below is a screenshot of a program creating and adding to a queue. As you can see the items of the Queue are in the order they were placed. When the Dequeue method is called the first item which was the number one is removed.

```

static void Main(string[] args)
{
    // Create order Queue
    Queue<int> order = new Queue<int>();
    // Add Items to order Queue
    order.Enqueue(1);
    order.Enqueue(2);
    order.Enqueue(3);
    order.Enqueue(4);
    order.Enqueue(5);
    // List initial items of Queue
    Console.Write("The Items in the Queue are: ");
    foreach(int i in order)
    {
        Console.Write($" {i}");
    }
    // Removes Oldest Item
    order.Dequeue();
    // List updated items of Queue
    Console.Write("\n\nThe Items in the Queue are: ");
    foreach (int i in order)
    {
        Console.Write($" {i}");
    }

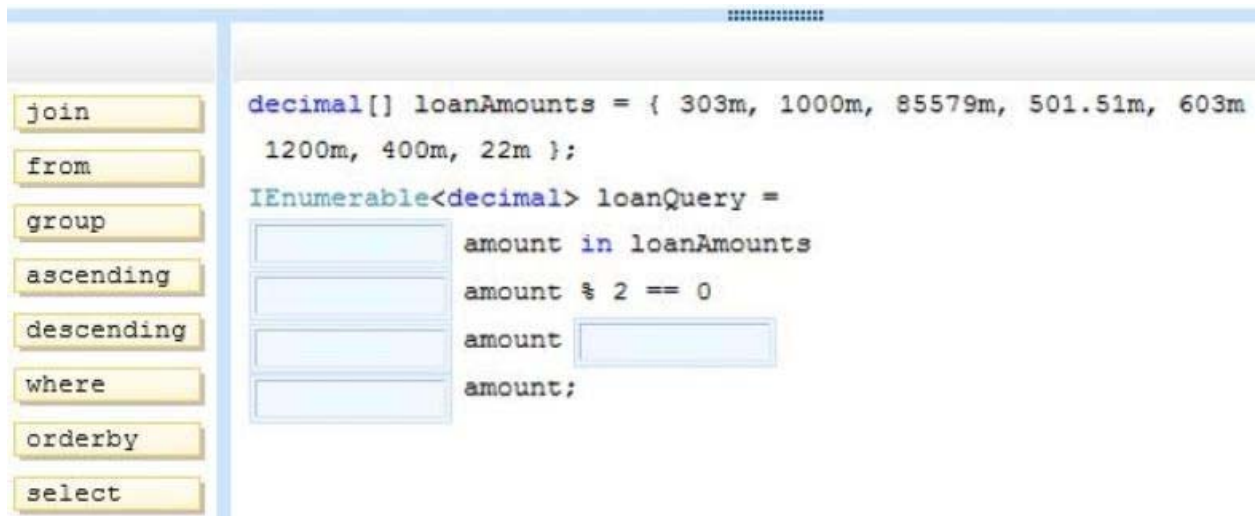
    Console.WriteLine();
}

```

The Items in the Queue are: 1 2 3 4 5

The Items in the Queue are: 2 3 4 5

6. You are developing an application by using C#. The application includes an array of decimal values named loanAmounts. You are developing a LINQ query to return the values from the array. The query must return decimal values that are evenly divisible by two. The values must be sorted from the lowest value to the highest value. You need to ensure that the query correctly returns the decimal values. How should you complete the relevant code?



```
decimal[] loanAmounts = { 303m, 1000m, 85579m, 501.51m, 603m
    1200m, 400m, 22m };
IEnumerable<decimal> loanQuery =
    [ ] amount in loanAmounts
    [ ] amount % 2 == 0
    [ ] amount [ ]
    [ ] amount;
```

The screenshot shows a LINQ query editor with a list of clauses on the left: join, from, group, ascending, descending, where, orderby, and select. The main area contains the code for the query, with four input fields corresponding to the clauses: from, where, orderby, and select.

The four blanks that I filled in were from, where, orderby, ascending, and select.

Each of these clauses mean something to the LINQ statement and is required for this program to run.

From – Specifies where the data for the query is coming from, in this case the loanAmounts array.

Where – Tells the query which items to select from the data source. In this case the query selects any decimals that have a remainder of Zero when divided by Two.

Orderby – Sorts the query in the way specified.

Ascending – The keyword used in the Orderby clause to state how the query should be sorted. Ascending sorts the values from smallest to largest.

Select – Selects the values for the query based on the previous clauses stated.

```
static void Main(string[] args)
{
    // Create array loanAmounts
    decimal[] loanAmounts = { 303m, 1000m, 85579m, 501.51m, 603m,
        1200m, 400m, 22m};
    // Select Decimals divisible by 2
    IEnumerable<decimal> loanQuery =
    from amount in loanAmounts
    where amount % 2 == 0
    orderby amount ascending
    select amount;
    // Output Decimals from Query
    foreach(decimal i in loanQuery)
    {
        Console.WriteLine($"{i} ");
    }
}
```

The output of the code is as follows to demonstrate the results.

```
22
400
1000
1200
```

17. You are implementing a method named Calculate that performs conversions between value types and reference types. The following code segment implements the method. (Line numbers are included for reference only.)

```
01 public static void Calculate(float amount)
02 {
03     object amountRef = amount;
04
05     Console.WriteLine(balance);
06 }
```

You need to ensure that the application does not throw exceptions on invalid conversions. Which code segment should you insert at line 04?

- A. `int balance = (int) (float)amountRef;`
- B. `int balance = (int)amountRef;`
- C. `int balance = amountRef;`
- D. `int balance = (int) (double) amountRef;`

The answer to this question is letter A. In this scenario the method is wanting to convert a float to an integer value. However, a float cannot be implicitly converted to the integer. It must be told to convert, if not an error occurs as shown here.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Calculate(1.00F);
    }

    1 reference
    public static void Calculate(float amount)
    {
        object amountRef = amount;
        int balance = amount;
        Console.WriteLine(balance);
    }
}
```

To solve this issue, you simply cast from float to integer. First you in parenthesis type the target value which is (int) and second you in parenthesis type the current value which is (float). When combined with the rest of the line it will look like “int balance = (int) (float)amountRef;” this will convert the float into an integer.

This code will run successfully and output the integer number that was converted.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Calculate(1.00F);
    }

    1 reference
    public static void Calculate(float amount)
    {
        object amountRef = amount;
        int balance = (int)(float)amount;
        Console.WriteLine(balance);
    }
}
```

1

44. An application includes a class named Person. The Person class includes a method named GetData.

You need to ensure that the GetData() method can be used only by the Person class and not by any class derived from the Person class.

Which access modifier should you use for the GetData() method?

- A. Public
- B. Protected internal
- C. Internal
- D. Private
- E. Protected

The answer to this question is letter D. Private. The private access modifier makes it so that only the class that the method is in can access it. This helps with security as outside classes do not have access. In the code below I have created a private and public method to demonstrate that the private method cannot be accessed from the main method, but the public method can be.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var person = new Person();

        person.GetData();

        person.GetDataPublic();
    }
}
1 reference
class Person
{
    1 reference
    private int GetData()
    {
        return 25;
    }

    1 reference
    public int GetDataPublic()
    {
        return 50;
    }
}
```

61. You are implementing a method named FloorTemperature that performs conversions between value types and reference types. The following code segment implements the method. (Line numbers are included for reference only.)

```
01 public static void FloorTemperature(float degrees)
02 {
03     object degreesRef = degrees;
04
05     Console.WriteLine(result);
06 }
```

You need to ensure that the application does not throw exceptions on invalid conversions. Which code segment should you insert at line 04?

- A. `int result = (int)degreesRef;`
- B. `int result = (int)(double)degreesRef;`
- C. `int result = degreesRef;`
- D. `int result = (int)(float)degreesRef;`

The answer to this question is letter D. To ensure that there are no exceptions or invalid conversions the float needs to be cast to an integer. This can not be done implicitly so a statement must be added to tell the program to convert the variable. This is done by adding the

statement “int result = (int)(float)degreesRef;” to line four. Without the conversion the program receives an error.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        FloorTemperature(5.00F);
    }

    1 reference
    public static void FloorTemperature(float degrees)
    {
        object degreesRef = degrees;
        int result = degreesRef;
        Console.WriteLine(result);
    }
}
```

When the conversion cast is added the program can run successfully and output the conversion.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        FloorTemperature(5.00F);
    }

    1 reference
    public static void FloorTemperature(float degrees)
    {
        object degreesRef = degrees;
        int result = (int)(float)degreesRef;
        Console.WriteLine(result);
    }
}
```