Higher Quality

Better Service!

EXAM SELL

Certified IT practice exam authority

Accurate study guides, High passing rate!
Exam Sell provides update free of charge in one year!

Exam : **70-483**

Title : Programming in C#

Version : DEMO

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>
Answer: A
Explanation:
Queues are useful for storing messages in the order they were received for sequential processing.
Objects stored in a Queue<T> are inserted at one end and removed from the other.
Reference: http://msdn.microsoft.com/en-us/library/7977ey2c.aspx

2.You are developing an application. The application calls a method that returns an array of integers named employeeIds. You define an integer variable named employeeIdToRemove and assign a value to it. You declare an array named filteredEmployeeIds.
You have the following requirements:
• Remove duplicate integers from the employeeIds array.
• Sort the array in order from the highest value to the lowest value.
• Remove the integer value stored in the employeeIdToRemove variable from the employeeIds array.
You need to create a LINQ query to meet the requirements.
Which code segment should you use?

```
A. int[] filteredEmployeeIds = employeeIds.Where(value => value !=
     employeeIdToRemove).OrderBy(x => x).ToArray();

B. int[] filteredEmployeeIds = employeeIds.Where(value => value !=
     employeeIdToRemove).OrderByDescending(x => x).ToArray();

C. int[] filteredEmployeeIds = employeeIds.Distinct().Where(value => value !=
     employeeIdToRemove).OrderByDescending(x => x).ToArray();

D. int[] filteredEmployeeIds = employeeIds.Distinct().
     OrderByDescending(x => x).ToArray();
```

A. Option A
B. Option B
C. Option C
D. Option D
Answer: C
Explanation:

The Distinct keyword avoids duplicates, and OrderByDescending provides the proper ordering from highest to lowest.

3.You are developing an application that includes the following code segment. (Line numbers are included for reference only.)

```
01 class Animal
02 {
03   public string Color { get; set; }
04   public string Name { get; set; }
05 }
06 private static IEnumerable<Animal> GetAnimals(string sqlConnectionString)
07 {
08   var animals = new List<Animal>();
09   SqlConnection sqlConnection = new SqlConnection(sqlConnectionString);
10   using (sqlConnection)
11   {
12     SqlCommand sqlCommand = new SqlCommand("SELECT Name, ColorName
FROM Animals", sqlConnection);
13
14     using (SqlDataReader sqlDataReader = sqlCommand.ExecuteReader())
15     {
16
17       {
18         var animal = new Animal();
19         animal.Name = (string)sqlDataReader["Name"];
20         animal.Color = (string)sqlDataReader["ColorName"];
21         animal..Add(animal);
22       }
23     }
24   }
25   return customers;
26 }
```

The GetAnimals() method must meet the following requirements:

• Connect to a Microsoft SQL Server database.

• Create Animal objects and populate them with data from the database.

• Return a sequence of populated Animal objects.

You need to meet the requirements.

Which two actions should you perform? (Each correct answer presents part of the solution. Choose two.)

A. Insert the following code segment at line 16:

while(sqlDataReader.NextResult())

B. Insert the following code segment at line 13:

sqlConnection.Open();

C. Insert the following code segment at line 13:

sqlConnection.BeginTransaction();

D. Insert the following code segment at line 16:

while(sqlDataReader.Read())

E. Insert the following code segment at line 16:

while(sqlDataReader.GetValues())

Answer: B, D

Explanation:

B: SqlConnection.Open - Opens a database connection with the property settings specified by the ConnectionString.
Reference: http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.open.aspx
D: SqlDataReader.Read - Advances the SqlDataReader to the next record. Reference: http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.read.aspx

4.DRAG DROP
You are developing a custom collection named LoanCollection for a class named Loan class.
You need to ensure that you can process each Loan object in the LoanCollection collection by using a foreach loop.
How should you complete the relevant code? (To answer, drag the appropriate code segments to the correct locations in the answer area. Each code segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.)

```
: IComparable
: IEnumerable
: IDisposable
public IEnumerator GetEnumerator()
public int CompareTo(object obj)
public void Dispose()
_loanCollection[0].Amount++;
return obj == null ? 1 : _loanCollection.Length;
return _loanCollection.GetEnumerator();
```

```
public class LoanCollection
{
    private readonly Loan[] _loanCollection;
    public LoanCollection(Loan[] loanArray)
    {
        _loanCollection = new Loan[loanArray.Length];

        for (int i = 0; i < loanArray.Length; i++)
        {
            _loanCollection[i] = loanArray[i];
        }
    }

    {

    }
}
```

Answer:

```
: IComparable

: IDisposable

public int CompareTo(object obj)
public void Dispose()
_loanCollection[0].Amount++;
return obj == null ? 1 : _loanCollection.Length;
```

```
public class LoanCollection : IEnumerable
{
    private readonly Loan[] _loanCollection;
    public LoanCollection(Loan[] loanArray)
    {
        _loanCollection = new Loan[loanArray.Length];

        for (int i = 0; i < loanArray.Length; i++)
        {
            _loanCollection[i] = loanArray[i];
        }
    }
    public IEnumerator GetEnumerator()
    {
        return _loanCollection.GetEnumerator();
    }
}
```

5.You are developing an application that uses the Microsoft ADO.NET Entity Framework to retrieve order information from a Microsoft SQL Server database.
The application includes the following code. (Line numbers are included for reference only.)

```
01 public DateTime? OrderDate;
02 IQueryable<Order> LookupOrdersForYear(int year)
03 {
04   using (var context = new NorthwindEntities())
05   {
06     var orders =
07      from order in context.Orders
08
09       select order;
10     return orders.ToList().AsQueryable();
11   }
12 }
```

The application must meet the following requirements:

• Return only orders that have an OrderDate value other than null.

• Return only orders that were placed in the year specified in the OrderDate property or in a later year.

You need to ensure that the application meets the requirements.

Which code segment should you insert at line 08?

A. Where order.OrderDate.Value != null && order.OrderDate.Value.Year > = year

B. Where order.OrderDate.Value = = null && order.OrderDate.Value.Year = = year

C. Where order.OrderDate.HasValue && order.OrderDate.Value.Year = = year

D. Where order.OrderDate.Value.Year = = year

Answer: A

Explanation:

• For the requirement to use an OrderDate value other than null use:

OrderDate.Value != null

• For the requirement to use an OrderDate value for this year or a later year use: OrderDate.Value>= year