PASS QUESTION

QUESTION & ANSWER

HIGHER QUALITY, BETTER SERVICE

Provide One Year Free Update!

https://www.passquestion.com

Exam : **70-483**

Title : Programming in C#

Version : DEMO

1. **トピック 1、ボリューム A**
あなたは Order という名前のクラスを含むアプリケーションを開発しています。 アプリケーションは Order オブジェクトのコレクションを保存します。
コレクションは以下の要件を満たす必要があります。
•強く型付けされたメンバーを使う。
•先入れ先出し法で Order オブジェクトを処理します。
•各 Order オブジェクトの値を保存します。
•ゼロベースのインデックスを使用する。
要件を満たすコレクションタイプを使用する必要があります。
どのコレクションタイプを使うべきですか?
A.キュー<T>
B. SortedList
C. LinkedList <T>
D.ハッシュテーブル
E.アレイ<T>
Answer: A
Explanation:
キューは、順次処理のために受信した順序でメッセージを格納するのに役立ちます。 Queue <T>に格納されているオブジェクトは一方の端に挿入され、もう一方の端から削除されます。
http://msdn.microsoft.com/en-us/library/7977ey2c.aspx

2. あなたはアプリケーションを開発しています。 アプリケーションは、employeeIds という名前の整数の配列を返すメソッドを呼び出します。 employeeIdToRemove という名前の整数変数を定義し、それに値を割り当てます。 あなたは filteredEmployeeIds という名前の配列を宣言します。
以下の要件があります。
•employeeIds 配列から重複する整数を削除します。
•最大値から最小値の順に配列を並べ替えます。
•employeeId 配列から employeeIdToRemove 変数に格納されている整数値を削除します。
要件を満たすには LINQ クエリを作成する必要があります。
どのコードセグメントを使うべきですか?

```
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.オプション A
B.オプション B
C.オプション C
D.オプション D
Answer: C

3. 次のコードセグメントを含むアプリケーションを開発しています。（行番号は参照用にのみ含まれています。）

```
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)sqlDataEeader["Name"];
20         animal.Color = (string)sqlDataEeader["ColorName"];
21         animals.Add(animal);
22       }
23     }
24   }
25   return customers;
26 }
```

GetAnimals（）メソッドは以下の要件を満たしている必要があります。
•Microsoft SQL Server データベースに接続します。
•Animal オブジェクトを作成し、それらにデータベースからのデータを入力します。
•移入された動物オブジェクトのシーケンスを返します。
あなたは要件を満たす必要があります。
どの 2 つのアクションを実行する必要がありますか？ （正解はそれぞれ解の一部を表しています。2 つ選んでください。）
A. 16 行目に次のコードセグメントを挿入します。
while（sqlDataReader.NextResult（））
B. 13 行目に次のコードを挿入します。
sqlConnection.Open（）;
C. 13 行目に次のコードを挿入します。
sqlConnection.BeginTransaction（）;
D. 16 行目に次のコードを挿入します。
while（sqlDataReader.Read（））
E. 16 行目に次のコードを挿入します。
while（sqlDataReader.GetValues（））
Answer: B, D
Explanation:
SqlConnection.Open - で指定されたプロパティ設定でデータベース接続を開きます。
ConnectionString。
http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.open.aspx
SqlDataReader.Read - SqlDataReader を次のレコードに進めます。

http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.read.aspx

4. ドラッグドロップ
Loan class というクラスの LoanCollection というカスタムコレクションを開発しています。 foreach ループを使用して、LoanCollection コレクション内の各 Loan オブジェクトを確実に処理できるようにする必要があります。
関連するコードをどのように完成させるべきですか? （回答するには、適切なコードセグメントを回答領域の正しい場所にドラッグします。各コードセグメントは、1 回、複数回、またはまったく使用しないことができます。コンテンツを表示するには、分割バーをドラッグするかスクロールする必要があります。）

```
: 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. Microsoft ADO.NET Entity Framework を使用して Microsoft SQL Server データベースから注文情報を取得するアプリケーションを開発しています。
アプリケーションには次のコードが含まれています。 （行番号は参照用にのみ含まれています。）

```
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 }
```

アプリケーションは以下の要件を満たす必要があります。

•null 以外の OrderDate 値を持つ注文のみを返す。

•OrderDate プロパティで指定された年またはそれ以降の年に行われた注文のみを返します。

アプリケーションが要件を満たしていることを確認する必要があります。

どのコードセグメントを 08 行目に挿入する必要がありますか。

A.ここで、order.OrderDate.Value！= null && order.OrderDate.Value.Year> = year

B. order.OrderDate.Value = = null && order.OrderDate.Value.Year = = year の場合

C.ここで、order.OrderDate.HasValue && order.OrderDate.Value.Year = =年

D.ここで order.OrderDate.Value.Year = =年

Answer: A

Explanation:

* null 以外の OrderDate 値を使用するという要件については、

OrderDate.Value！= null

*今年またはそれ以降の年に OrderDate 値を使用するという要件については、次のように使用します。

OrderDate.Value> =年