

我要—PASS



質 量 更 高 務 服 更 好

一年免費升級服務

<http://www.51-pass.com>

Exam : 70-536Chinese(C++)

**Title : TS:MS.NET Framework
2.0-Application Develop
Foundation**

Version : DEMO

1. 您正在使用应用程序的调试版本。

您需要找出导致异常抛出的代码行。

您应使用 `Exception` 类的哪个属性来达到此目的？

- A. Data
- B. Message
- C. StackTrace
- D. Source

Answer: C

2. 您正在编写一个方法，该方法返回名为 `al` 的 `ArrayList`。

您需要确保以线程安全的方式对 `ArrayList` 执行更改。

您应该使用哪个代码段？

A. `ArrayList^ al = gcnew ArrayList();`

`lock (al->SyncRoot)`

```
{  
    return al;  
}
```

B. `ArrayList^ al = gcnew ArrayList();`

`lock (al->SyncRoot.GetType())`

```
{  
    return al;  
}
```

C. `ArrayList^ al = gcnew ArrayList();`

`Monitor::Enter(al);`

`Monitor::Exit(al);`

`return al;`

D. `ArrayList^ al = gcnew ArrayList();`

`ArrayList^ sync_al = ArrayList::Synchronized(al);`

`return sync_al;`

Answer: D

3. 您正在创建一个类，用于比较经过特殊格式设置的字符串。默认的排序规则比较不适用。

您需要实现 `Comparable<string>` 接口。

您应该使用哪个代码段？

A.

```
public ref class Person :public Comparable<String^>{  
    public :virtual Int32 CompareTo(String^ other){  
        ...  
    }  
}
```

B.

```
public ref class Person :public Comparable<String^>{  
    public :virtual Int32 CompareTo(Object^ other){  
        ...  
    }  
}
```

C.

```
public ref class Person :public Comparable<String^>{  
    public :virtual Boolean CompareTo(String^ other){  
        ...  
    }  
}
```

D.

```
public ref class Person :public Comparable<String^>{  
    public :virtual Boolean CompareTo(Object^ other){  
        ...  
    }  
}
```

Answer: A

4. 您正在编写自定义字典。该自定义字典类名为 `MyDictionary`。

您需要确保该字典是类型安全的字典。

您应该使用哪个代码段？

A.

```
public ref class MyDictionary :public Dictionary<String^, String^>{};
```

B. `public ref class MyDictionary :public Hashtable{};`
C. `public ref class MyDictionary :public IDictionary{};`
D. `public ref class MyDictionary{};`
`Dictionary<String^, String^> t = gcnew Dictionary<String^, String^>();`
`MyDictionary dictionary = (MyDictionary)t;`

Answer: A

5. 您正在开发一个协助用户进行电子调查的应用程序。调查由 25 个对错判断题组成。

您需要执行下列任务：

☒ 将每个答案预置为是。

☒ 最大程度地减少每次调查使用的内存量。

您应该选择哪个存储选项？

A. `BitVector32^ answers = gcnew BitVector32(1);`
B. `BitVector32^ answers = gcnew BitVector32(-1);`
C. `BitArray^ answers = gcnew BitArray (1);`
D. `BitArray^ answers = gcnew BitArray(-1);`

Answer: B

6. 您正在创建名为 `Age` 的类。

您需要确保编写的 `Age` 类的对象所构成的集合能够被排序。

您应该使用哪个代码段？

A. `public ref class Age {`
 `public :Int32 Value;`
 `public :virtual Object CompareTo(Object^ obj) {`
 `if (obj->GetType() == Age::GetType()) {`
 `Age^ _age = (Age^) obj;`
 `return Value.CompareTo(obj);`
 `}`
 `throw gcnew ArgumentException("object not an Age");`
 `}`

```
};
```

```
B. public ref class Age {
```

```
    public :Int32 Value;
```

```
    public :virtual Object CompareTo(Int32^ iValue) {
```

```
        try {
```

```
            return Value.CompareTo(iValue);
```

```
        } catch (Exception^ ex) {
```

```
            throw gcnew ArgumentException ("object not an Age");
```

```
        }
```

```
    }
```

```
};
```

```
C. public ref class Age :public IComparable {
```

```
    public :Int32 Value;
```

```
    public :virtual Int32 CompareTo(Object^ obj) {
```

```
        if (obj->GetType() == Age::GetType()) {
```

```
            Age^ _age = (Age^) obj;
```

```
            return Value.CompareTo(_age->Value);
```

```
        }
```

```
        throw gcnew ArgumentException("object not an Age");
```

```
    }
```

```
};
```

```
D. public ref class Age :public IComparable {
```

```
    public :Int32 Value;
```

```
    public :virtual Int32 CompareTo(Object^ obj) {
```

```
        try {
```

```
            return Value.CompareTo(((Age^) obj)->Value);
```

```
        } catch (Exception^ ex) {
```

```
            return -1;
```

```
        }
```

```
    }
```

};

Answer: C

7. 您编写以下代码。

```
public delegate void FaxDocs(Object^ sender, FaxArgs^ args);
```

您需要创建一个将调用 `FaxDocs` 的事件。

您应该使用哪个代码段？

A. `public :static event FaxDocs^ Fax;`

B. `public :static event Fax^ FaxDocs;`

C. `public ref class FaxArgs :public EventArgs {`

`public :`

`String^ CoverPageInfo;`

`FaxArgs (String^ coverInfo) {`

`this->CoverPageInfo = coverInfo;`

`}`

`};`

D. `public ref class FaxArgs :public EventArgs {`

`public :`

`String^ CoverPageInfo;`

`};`

Answer: A

8. 您需要编写一个接受 `DateTime` 参数并返回布尔值的多路广播委托。

您应该使用哪个代码段？

A. `public delegate int PowerDeviceOn(bool,`

`DateTime);`

B. `public delegate bool PowerDeviceOn(Object,EventArgs);`

C. `public delegate void PowerDeviceOn(DateTime);`

D. `public delegate bool PowerDeviceOn(DateTime);`

Answer: D

9. 您开发一个名为 **FileService** 的服务应用程序。您将该服务应用程序部署到网络上的多台服务器。

您执行以下代码段。（包括的行号仅供参考。）

```
01 public :  
02     void StartService(String^ serverName){  
03  
04         ServiceController^ ctrl = gcnew  
05             ServiceController("FileService");  
06         if (ctrl->Status == ServiceControllerStatus::Stopped){  
07     }
```

您需要开发一个例程，如果 **FileService** 停止，该例程将启动它。该例程必须在由 **serverName** 输入参数确定的服务器上启动 **FileService**。

您应该将哪两行代码添加到代码段？（每个正确答案都仅给出了部分解决方案。请选择两个答案。）

A. 在 03 行和 04 行之间插入以下代码行：

```
ctrl.ServiceName = serverName;
```

B. 在 03 行和 04 行之间插入以下代码行：

```
ctrl.MachineName = serverName;
```

C. 在 03 行和 04 行之间插入以下代码行：

```
ctrl.Site.Name = serverName;
```

D. 在 04 行和 05 行之间插入以下代码行：

```
ctrl.Continue();
```

E. 在 04 行和 05 行之间插入以下代码行：

```
ctrl.Start();
```

F. 在 04 行和 05 行之间插入以下代码行：

```
ctrl.ExecuteCommand(0);
```

Answer: BE

10. 您正在开发一个用于执行数学计算的应用程序。您开发名为 **CalculationValues** 的类。您编写一个名为 **PerformCalculation** 的过程，该过程在类的实例上进行操作。

您需要确保应用程序的用户界面在计算正在执行时能保持响应。您需要编写一个调用 **PerformCalculation**

过程的代码段来达到此目的。

您应该使用哪个代码段？

A. public ref class CalculationValues {...}

```
public ref class Calculator {  
    public :  
        void PerformCalculation() {}  
};  
  
public ref class ThreadTest{  
    private :  
        void DoWork (){  
            CalculationValues^ myValues = gcnew CalculationValues();  
            Calculator ^ calc = gcnew Calculator();  
            Thread^ newThread = gcnew Thread(  
                gcnew ThreadStart(calc, &Calculator::PerformCalculation));  
            newThread->Start(myValues);  
        }  
};
```

B. public ref class CalculationValues {...}

```
public ref class Calculator {  
    public :  
        void PerformCalculation() {}  
};  
  
public ref class ThreadTest{  
    private :  
        void DoWork (){  
            CalculationValues^ myValues = gcnew CalculationValues();  
            Calculator ^ calc = gcnew Calculator();  
            ThreadStart^ delStart = gcnew  
                ThreadStart(calc, &Calculator::PerformCalculation);  
            Thread^ newThread = gcnew Thread(delStart);
```

```
        if (newThread->IsAlive) {  
            newThread->Start(myValues);  
        }  
    }  
};
```

C. public ref class CalculationValues {...-}

```
public ref class Calculator {  
    public :  
        void PerformCalculation(CalculationValues^ values) {}  
};  
  
public ref class ThreadTest{  
    private :  
        void DoWork (){  
            CalculationValues^ myValues = gcnew CalculationValues();  
            Calculator ^ calc = gcnew Calculator();  
            Application::DoEvents();  
            calc->PerformCalculation(myValues);  
            Application::DoEvents();  
        }  
};
```

D. public ref class CalculationValues {...-}

```
public ref class Calculator {  
    public :  
        void PerformCalculation(Object^ values) {}  
};  
  
public ref class ThreadTest{  
    private :  
        void DoWork (){  
            CalculationValues^ myValues = gcnew CalculationValues();  
            Calculator ^ calc = gcnew Calculator();
```

```
Thread^ newThread = gcnew Thread(  
    gcnew ParameterizedThreadStart(calc,  
        &Calculator::PerformCalculation));  
newThread->Start(myValues);  
}  
};
```

Answer: D