.NET class

FINAL TEST - ANSWERS

1. W jaki sposób WCF "podpisuje się kontrakt"?
    a. Poprzez implementację interfejsu.
    b. Poprzez implementację interfejsu, który opakowano atrybutem [OperationContract].
    c. Poprzez implementację interfejsu, który opakowano atrybutem [ServiceContract].
    d. Nie trzeba tego w specjalny sposób "podpisywać" - WCF sam wyciągnie metody publiczne z klasy.

    Kontrakt definiuje się poprzez nadanie atrybutu ServiceContract (na interfejsie lub klasie) oraz OperationContract na konkretnej metodzie. Z tym pytaniem nie było żadnych problemów!

2. Co to znaczy "inversion of control"?
    a. Gdy korzystamy z IoC, aplikacja działa szybciej i sprawniej ponieważ obiekty są tworzone przez specjalnie zaprojektowane kontenery.
    b. Obiekt sam nie tworzy obiektów zależnych, lecz zostają mu one dostarczone z zewnętrznego źródła.
    c. Obiekt tworzy obiekty zależne tylko raz, a późniejsze wywołanie konstruktora korzysta z tych gotowych obiektów.
    d. Metody obiektu są kontrolowane przez zewnętrzny kontener - wszystkie metody sa prywatne, to kontener decyduje o poziomie dostępu do metod.
    e. Zastosowanie plików XML jako definicji klas dla kontenera. Na podstawie tego pliku budowane są zależności i obiekty do ich rozwiązania.

    Większość odpowiedzi zawiera cząstkę prawdy - ale tylko odpowiedź B jest w pełni poprawna! Nieco ponad połowa z Was zaznaczyła poprawną odpowiedź. Zauważcie, że kontener IoC nie gwarantuje, że aplikacja będzie działać szybciej – cel stosowania kontenerów jest inny – o co zostaliście zapytani w kolejnym zadaniu.

3. Jakie są korzyści ze stosowania IoC/DI? (UWAGA, możliwe WIELE poprawnych odpowiedzi)
    a. Szybsze działanie programu.
    b. Łatwiejsze testowanie programu.
    c. Tzw. loose coupling.
    d. Łatwość refaktoryzacji programu.
    e. Mniejsza ilość błędów w aplikacji.

Tutaj poprawne były 3 odpowiedzi. Zauważcie, że zastosowanie kontenera nie ma zupełnie związku z liczbą błędów w aplikacji – to bezpośrednio od developera zależy, czy poprawnie wykorzysta dostarczone mu obiekty. Nie ma gwarancji, że obiekty te (a w szczególności ich metody) są poprawnie napisane – logika nie jest zależna od kontenera. Dokładnie to samo możemy odnieść do szybkości działania aplikacji – kontenery zastępują tylko proces tworzenia obiektów – złożoność programów najczęściej wynika z ich złożonej logiki oraz przetwarzania obiektów, a nie samego ich tworzenia!

Pozostałe odpowiedzi poruszaliśmy podczas zajęć – łatwiejsze testowanie (zamiana jednego obiektu na inny np. Poprzez zastosowanie Mock'ów), loose coupling (czyli nie interesuje mnie konkretna implementacja – mogę wykorzystać dowolną, o ile implementuje ona konkretny interfejs, czasami nazywany też kontraktem – choć nie jest to do końca poprawne, jeśli wszystkie metody nie są „pure methods"!) a także łatwość refaktoryzacji.

4. [AUTOFAC] Zakładając, że w Twoim programie są

```
public interface IOutput
{
  void Write(string content);
}

public class ConsoleOutput : IOutput
{
  public void Write(string content)
  {
    Console.WriteLine(content);
  }
}
```

W Twoim programie istnieje klasa, która korzysta z typu `ConsoleOutput` - uzupełnij fragment kodu odpowiedzialny za odpowiednie zarejestrowanie typu `ConsoleOutput`, jako typ reprezentujący `IOutput`.

```
var builder = new ContainerBuilder();
// wpisz fragment kodu poniżej:
```

Zadanie prosto z QuickStartGuide do autofac:

```
// Create your builder.
var builder = new ContainerBuilder();

// Usually you're only interested in exposing the type
// via its interface:
builder.RegisterType<SomeType>().As<IService>();

// However, if you want BOTH services (not as common)
// you can say so:
builder.RegisterType<SomeType>().AsSelf().As<IService>();
```

Poprawna odpowiedź, to ta środkowa:
builder.RegisterType<ConsoleOutput>().As<IOutput>();

Wszystko było w treści. Oczywiście typ można zarejestrować na różne sposoby – jestem pewien, że łatwiej by Wam było gdyby to robić w VS. Dlatego starałem się rozdawać punkty za każdą próbę rejestracji tego typu.

5. W jaki sposób muszą zostać skonfigurowani konsumenci, tak żeby RabbitMQ rozdzielało wiadomości równomiernie pomiędzy nich?
    a. Każdy konsument tworzy własną Queue, podłączoną do tego samego Exchange działającego w trybie fanout.
    b. Każdy konsument tworzy własny Topic, podłączony do tego samego Exchange.
    c. Każdy konsument korzysta z tego samego Queue, podłączonego do jednego Exchange.
    d. Niemożliwe jest takie skonfigurowanie konsumentów.

Round-robin bez priorytetów – każdy klient równomiernie pobierze z kolejki.

6. Exchange w trybie fan-out dostarcza wiadomości do:
    a. Jednego konsumenta z pasującym kluczem Queue (wiadomości będą rozdzielane równomiernie).
    b. Wszystkich konsumentów z pasującym kluczem Queue.
    c. Wszystkich konsumentów, klucz Queue jest ignorowany.
    d. Jednego konsumenta, klucz Queue jest ignorowany (wiadomości będą rozdzielane równomiernie).

Tryb fan-out to przykład broadcastowania – Exchange bedzie dostarczac wiadomosci do wszystkich kolejek, jakie zna.

7. Biorąc pod uwagę poniższy fragment kodu z pliku xaml:
```
<TextBlock Text={Binding UserName} Visibility={Binding Status,
Converter={StaticResource StatusToVisibility}} />
```

Uzupełnił poniższy kod konwertera (do którego odnosi się zasób StatusToVisibility) tak, żeby element TextBlock był ukryty jeżeli wartość właściwości `Status == "Offline"` i widoczny w każdym innym przypadku:

```
public class StatusToVisibilityConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object
  parameter, CultureInfo culture)
   {
       var stringValue = value as string;
       if(stringValue != null)
       {
           if(stringValue.Equals(„Offline”))
           {
               return Visibility.Visible;
           }
       }
      return Visibility.Hidden;
   }
```

1. *Trzeba rzutować object na string by CLR odpowiednią metodę do porównania rozwiązał.*
2. *Wolimy „bezpiecznie rzutować” – czyli korzystać z patern as oraz !=null a nie rzucać wyjątkami wewnątrz metody (chociaż czasami jest to również dobre rozwiązanie).*
3. *Sprawdzamy przychodzącą wartosć ze stałym tekstem. Oczywiście tutaj można to zrobić na wiele sposobów -czy to metodą Equals, czy też CompareTo.*

```
public object ConvertBack(object value, Type targetType, object
                          parameter, CultureInfo culture)
    {
        throw new NoSupportedException();
    }
}
```

8. Dany jest nastepujacy kod:

```
class MyControl : UserControl
{
     public void Initialize()
     {
         var tb = StatusBar; //assume this is defined ctrl in
     XAML
         tb.Text = "loading...";
```

```
            var thread = new Thread( () => UpdateUI(tb, "hello
        world"));
            try
            {
               thread.Start();
            }
            catch (Exception err)
            {
               tb.Text = "Error!";
            }
        }

        private void UpdateUI(TextBlock block, string value)
        {
          try
          {
             block.Text = value;
           }
           catch (Exception err)
           {
              block.Text = "blad!";
           }
        }

    }
```

Jaki będzie rezultat dzialania programu?

a. wartosc block.Text bedzie ustawiona na "hello world"
b. wartosc block.Text bedzie ustawiona na "Error!"
c. wartosc block.Text bedzie ustawiona na "blad!"
d. wartosc block.Text bedzie ustawiona na "loading"
e. **aplikacja zostanie zamknieta**

Wątek przez nas stworzony próbuje zaktualizować UI – spowoduje to zatrzymanie aplikacji z pieknym Exceptionem. ☺

9. Dany jest nastepujacy fragment programu:

```
class Calculator {
      private int _totalSum = 0;
      private int _operationsLeft;
      private Action<int> _callback;


private void CalculateValue(int a, int b)
{
```

```
        // dodaj a do b, i wynik zapisz w polu _totalSum
    }


    private void NotifyCaller()
    {
        // kiedy wszystkie operacje sie zakonczyly, wykonaj:
        _callback(_totalSum);
    }


    public void Sum(int n, Action<int> resultCallback) {
        _callback = resultCallback;
        _operationsLeft = n;

        for (int i = 0; i<n; ++i)
        {
        int current = i;
        ThreadPool.QueueUserWorkItem( _ =>
                                    {
                                        CalculateValue(i, i + 1)
                                        NotifyCaller();
                                    });
        }
      }
    }
```

Zaimplementuj zawartosc metody CalculateValue:

Metoda była na prawdę prosta do zaimplementowania – wystarczyło pamiętać o kilku rzeczach:

1. *Wartosci w metodzie są niezależne od wątku – czyli operacja nie musi być atomowa.*
2. *Dodanie sumy do _totalSum musi byc operacją atomową, ponieważ wiele wątków będzie pisać do tego miejsca w pamięci.*

Dlatego, jedno z proponowanych rozwiazań:

```
var mySum = a+b;

Interlocked.Add(ref _totalSum, mySum);
```

W pierwszej linii – niezależnie dodajemy dostarczone wartości. Najważniejsze jest zastosowanie ATOMOWEJ operacji dodawania, jaki zapewnia klasa Interlocked. Wiecej o tej metodzie zobaczyć można tutaj: http://msdn.microsoft.com/pl-pl/library/33821kfh(v=vs.110).aspx

Zaimplementuj zawartosc metody NotifyCaller:

W drugiej metodzie również należy skorzystać z metody z klasy Interlocked – tym razem by zmniejszyć liczbę operacji, jaka pozostała do wykonania:

```
if(Interlocked.Decrement(ref _operationsLeft) == 0)

        _callback(_totalSum);
```

10. Mamy metodę GetFxSpotRate(string, string) zwracającą kurs wymiany walut. Czy dane scenariusze użycia można pokryć testami jednostkowymi?

	TAK	NIE
Metoda GetFxSpotRate powinna rzucać wyjątkiem jeżeli którykowielk z parametrów jest nullem.	X	
Metoda GetFxSpotRate powinna najpierw wywołać z klasy IFxRepository metodę GetSpotRate.	X	
Metoda GetFxSpotRate dla pary „USD/EUR" zwraca te same wartości, które są w bazie danych.		X
Metoda GetFxSpotRate wywołana z poprawnymi parametrami powinna zwrócić 0 gdy serwis zwróci błąd.	X	
Po rozszerzeniu bazy o kraje afrykańskie metoda GetFxSpotRate powinna zwracać poprawne dane dla pary walut „PLN/ZAR".		X

Przypadek 3 zawiera interakcję z bazą danych i wyświetlanie konkretnych danych. Co więcej, kurs walut nie jest stały, więc raczej nie da się tego przypadku przetestować. Przypadek 5 również wymaga interakcji z bazą dannych – testem jednostkowym nie sprawdzamy czy konkrena para jest w bazie czy nie.

11. Którego protokołu serializacji nie można użyć żeby wymieniać dane między serwisami w .NET/C# i Java?
    a. JSON
    b. XML
    c. Protocol Buffers
    d. Binary Formatter

12. Które zdanie nie jest poprawne?

a. Mając Mock stworzony dla interfejsu można stworzyć testowe implementacje dla wszystkich metod tego interfejsu.
b. Mając Mock stworzony dla konkretnej klasy można stworzyć testowe implementacje tylko dla publicznych metod tej klasy.
c. Mając Mock stworzony dla konkretnej klasy można stworzyć testowe implementacje tylko dla wirtualnych metod tej klasy.
d. Mając Mock stworzony dla konkretnej klasy nie można zamieniać wartości pól (*fields*) klasy przy odwołaniach do nich.

Mock (zakładając, że używamy Moq, chociaż ograniczenie to dotyczy każdej biblioteki) wewnętrznie używa biblioteki DynamicProxy, która dziedziczy z mockowanej klasy i przeciąża metody, które chcemy nadpisać. Żeby metodę przeciążyć musi byc ona wirtualna, ale nie koniecznie publiczna.