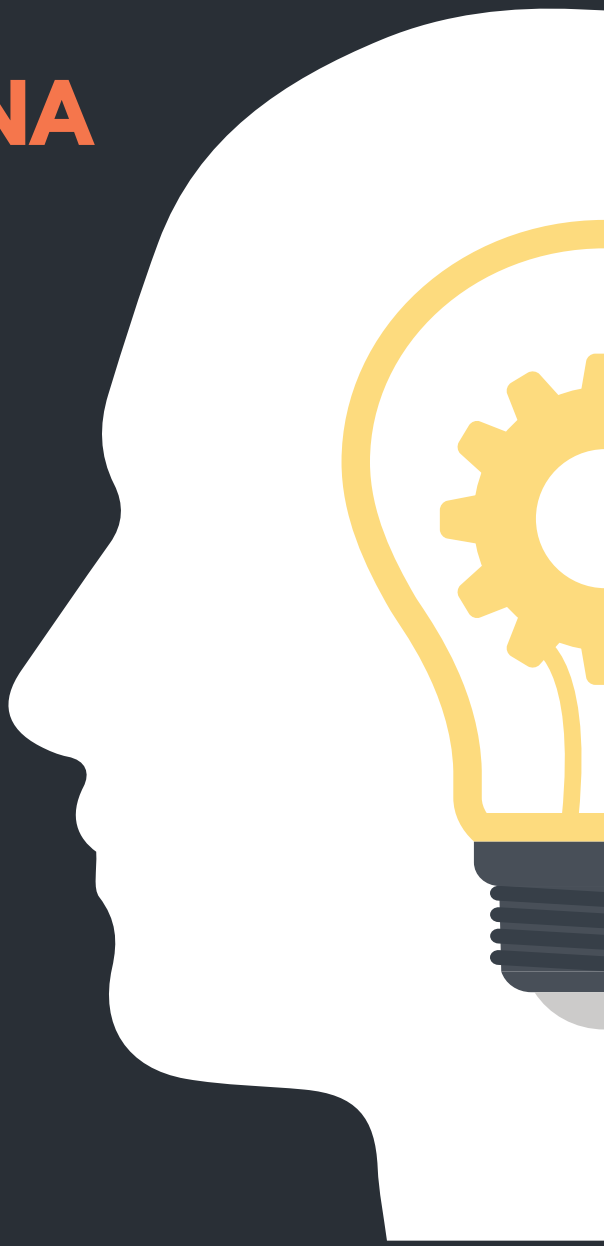
# 10 NAJCZĘSTSZYCH PYTAŃ (I ODPOWIEDZI!) Z ROZMÓW KWALIFIKACYJNYCH NA JUNIOR C# /.NET DEVELOPERA. NIE POPEŁNIJ PROSTYCH BŁĘDÓW, KTÓRE SKREŚLĄ CIĘ JUŻ NA STARCIE!



MODEST PROGRAMMER

**PRZYGOTOWAŁ**: KAZIMIERZ SZPIN

## WSTĘP:

Nieodłącznym elementem kariery programisty jest udział w rozmowach kwalifikacyjnych. Sam nie przepadam za takimi rozmowami, ale bardzo dużo już mam takich rozmów za sobą. Szczególnie często uczęszczałem na rozmowy kwalifikacyjne, szukając pracy na stanowisko młodszego programisty C#/.NET. Rozmowy były bardzo różne, zapewne podzielę się z Tobą wkrótce, szczegółami na moim blogu jak dokładnie wyglądały krok po kroku, najpierw jednak chciałem przedstawić Ci 10 pytań, które najczęściej pojawiały się na moich rozmowach kwalifikacyjnych. Większość z nich nie jest trudna, w końcu to pytanie dla początkujących programistów, warto jednak przed każdą rozmową sobie je powtórzyć. Oczywiście nie obiecuję Ci, że wszystkie te pytania pojawią się również na Twojej rozmowie kwalifikacyjnej, myślę jednak, że kilka z nich na pewno się powtórzy.

## 10 Najczęstszych Pytań (i odpowiedzi!) z Rozmów Kwalifikacyjnych Na Junior C#/.NET developera. Nie popełnij Prostych Błędów, Które Skreślą Cię Już Na Starcie!

## #1 Czym się różni typ wartościowy, od typu referencyjnego?

Typ wartościowy to elementy, które rozszerzają System.ValueType i są to typy proste takie jak na przykład int, long, double, decimal, bool. Przechowywane są one na stosie. Pokaże na przykładzie, jak działa przypisywanie wartości w przypadku typów wartościowych.

```
var minAge = 10;
var maxAge = 20;
minAge = maxAge;
maxAge = 21;
Console.WriteLine(minAge);//20
```

Przypisując wartość zmiennej maxAge do minAge, tak naprawdę skopiowaliśmy to, co jest na stosie, czyli wartość. Dlatego, jeżeli w kolejnym kroku została zmieniona wartość zmiennej maxAge, minAge pozostała bez zmian. Trochę inaczej wygląda taka operacja, gdy działamy na typach referencyjnych.

Typ referencyjny są to elementy dziedziczące po klasie System.Object oraz System.String, czyli głównie klasy, tablice, listy, stringi. Referencja do pamięci umieszczana jest na stosie, ale obszar pamięci, do jakiego prowadzi referencja, znajduje się na stercie. Są usuwane z pamięci za pomocą Garbage Collectora. Rozważmy znowu przykład z kopiowaniem wartości tym razem w przypadku typów referencyjnych.

```
public class Animal
{
   public int Age { get; set; }
}

static void Main(string[] args)
{
   var elephant = new Animal() { Age = 10 };
   var dog = new Animal() { Age = 2 };
   elephant = dog;
   dog.Age = 5;
   Console.WriteLine(elephant.Age);//5
}
```

Jak można zauważyć w przypadku typów referencyjnych, podczas przypisania została skopiowana referencja obiektu i oba obiekty wskazują teraz takie same referencje, czyli wskazują w to samo miejsce w pamięci.

## #2 Jakie są różnice pomiędzy interfejsem a klasą abstrakcyjną?

Najważniejsze różnice:
-W deklaracji interfejsu używamy słowa kluczowego interface, a w klasie abstrakcyjnej abstract class.
-Klasa może implementować dowolną ilość interfejsów, a dziedziczyć może tylko po 1 klasie abstrakcyjnej.
-Składowe interfejsu nie mogą zostać oznaczone atrybutami dostępu (domyślnie ustawiony jest public), a w klasie abstrakcyjnej mogą zostać oznaczone atrybuty dostępu.
-Interfejs umożliwia jedynie deklarację składowych, bez ich pełnych implementacji, a klasa abstrakcyjna może mieć zarówno deklarację składowych, jak również pełną implementację. Implementacja ta może zostać przesłonięta w klasie pochodnej.
-W interfejsie nie można deklarować pól, a w klasie abstrakcyjnej można to robić.
-Interfejs nie może mieć konstruktora, a w klasie abstrakcyjnej może być implementacja konstruktora domyślnego.

## #3 Różnice pomiędzy słowami kluczowymi const oraz readonly?

-Zmienna zadeklarowana jako readonly może być inicjalizowana z poziomu deklaracji oraz konstruktora, zmienna const można inicjalizować tylko w deklaracji.
-Zmienna readonly zostaje przetwarzana na poziomie runtime, a wartość zmiennej const zostaje przypisywana w czasie kompilacji.
-Zmienna readolny może być zadeklarowana jako static, a zmienna const nie.

## #4 Jaka się różnica pomiędzy słowami kluczowymi ref oraz out?

W C# typy wartościowe przekazywane są przez wartość, między innymi oznacza to też, że zmiany wykonywane na polu wewnątrz metody nie są widoczne na zewnątrz. Aby tak się stało, zmienna musi zostać przekazana przez adres i właśnie do tego używamy słów ref oraz out. Przykład jak można to zrobić (bez ref i out):

```
private static void IncreaseAge(int age)
{
    age++;
}

static void Main(string[] args)
{
    int myAge = 30;
    IncreaseAge(myAge);
    Console.WriteLine(myAge);//30
}
```

Przykład z ref:

```
private static void IncreaseAge(ref int age)
{
    age++;
}

static void Main(string[] args)
{
    int myAge = 30;
    IncreaseAge(ref myAge);
    Console.WriteLine(myAge);//31
}
```

Przykład z out (trochę w tym przypadku nie ma sensu, chcę tylko Ci pokazać różnice w implementacji):

```
private static void IncreaseAge(out int age)
{
     age = 30;
     age++;
}

static void Main(string[] args)
{
     int myAge;
     IncreaseAge(out myAge);
     Console.WriteLine(myAge);
}
```

Czym się różnią te implementacje z ref oraz out:
-Zmienna przekazana przez ref musi być już wcześniej zainicjalizowana.
-Zmienna przekazana przez out nie musi być wcześniej zainicjalizowana, ale może. Zmienna out musi zostać zainicjalizowana w ciele metody.

## #5 Czym są metody rozszerzające i jak wygląda implementacja?

Przyznam, że dosyć często w swojej pracy piszę metody rozszerzające, także wydaje mi się, że i Tobie się przydadzą, także warto wiedzieć, czym one są. Dzięki metodzie rozszerzającej możemy rozszerzyć funkcjonalność typów już istniejących. Metoda rozszerzająca musi być statyczna oraz zostać zadeklarowana w klasie statycznej i przed pierwszym parametrem musi zawierać słowo kluczowe this przed typem, który chcemy rozszerzyć. Najlepiej będzie to zademonstrować na przykładzie. Pokaże Ci jak rozszerzyć stringa, tak aby wypisywał swoją wartość w konsoli. Czyli chcemy, aby zamiast wpisywać Console.WriteLine('tekst'), wystarczyło wpisać zmienna.Display().

```
public static class StringExtensions
{
     public static void Display(this string value)
     {
         Console.WriteLine(value);
     }
}

public class Program
{
    static void Main(string[] args)
    {
        var title = "Programowanie w C#";
        title.Display();//Programowanie w C#
    }
}
```

## #6 Jaki jest cel używania słowa kluczowego using w C#?

Słowo kluczowe using używane jest w 2 przypadkach. Po pierwsze służy do dołączania przestrzeni nazw (np. using System;). Drugi sposób służy do ograniczenia czasu zużycia zasobów przez tworzony obiekt. Jeżeli użyjemy w klauzuli using jakiegoś zasobu to zawsze zostanie wywołane na tym obiekcie metoda Dispose(), która zajmuje się zwalnianiem zasobów. Trzeba pamiętać, że usinga można używać tylko na obiektach, które implementują interfejs IDisposable. Przykład:

```
using (var sqlConnection = new
SqlConnection(connectionString))
{
    //...
}
```

## #7 Co to są testy jednostkowe?

Testy jednostkowe (unit tests) to metody, których celem jest sprawdzenie działania innych metod. Testy jednostkowe mogą stanowić dokumentację dla innych programistów, zapewniać, że dany kod działa zgodnie z założeniami oraz przydają się gdy chcemy zrobić refaktoryzację kodu. Mamy wtedy pewność, że podczas refaktoryzacji nie utraciliśmy jakiejś ważnej funkcjonalności. Testy jednostkowe dzielą się na 3 etapy (AAA): Arrange, Act oraz Assert. Pierwszy z nich, arrange to przygotowanie obiektów do testów, kolejny act to testowanie funkcjonalności oraz ostatni arrange sprawdzenie, czy wyniki są zgodne z oczekiwaniami.

# #8 Co to jest polimorfizm?

Polimorfizm (wielopostaciowość) to jeden z podstawowych założeń dotyczących programowania obiektowego. Polimorfizm można podzielić na 2 grupy: polimorfizm statyczny, a także polimorfizm dynamiczny. Polimorfizm statyczny w C# to przeciążanie funkcji oraz operatorów. Chodzi o to, że w 1 klasie może istnieć dużo metod o takiej samej nazwie, lecz różniących się tylko parametrami. Polimorfizm statyczny zachodzi podczas kompilacji programu, to która metoda zostanie wybrana, zależy od ilości, a także typu przekazanych do metody argumentów. Warto wspomnieć, że nie można przeciążyć metody, która różni się tylko zwracanym typem. Polimorfizm dynamiczny w C# jest powiązany z funkcjami wirtualnymi i abstrakcyjnymi, jest to tak zwane przesłanianie funkcji. Idea polimorfizmu (dynamicznego) kręci się wokół dziedziczenia klas. Aby przesłonić metodę w klasie bazowej, należy użyć słowa kluczowego virtual, a w klasie podrzędnej override.

Przykład:

```
public class Animal
{
   public virtual void Eat()
   {
      Console.WriteLine("Animal.Eat()");
   }
}

public class Dog : Animal
{
    public override void Eat()
    {
       Console.WriteLine("Dog.Eat()");
    }
}

class Program
{
   static void Main(string[] args)
   {
      Animal animal = new Dog();
      animal.Eat(); //Dog.Eat()
   }
}
```

## #9 Zaimplementuj na kartce metodę, która z przedziału od 1 do 10 wyświetli tylko liczby parzyste.

Nie jest napisane, w jaki sposób ma wyświetlać, więc założę, że będzie to aplikacja konsolowa. Przykładowy metoda może wyglądać w taki sposób:

```
public void DisplayEvenNumbers()
{
   for (int i = 1; i < 11; i++)
   {
       if (i % 2 == 0)
          System.Console.WriteLine(i);
   }
}
```

# #10 Zaimplementuj na kartce algorytm FizzBuzz.

Czym jest FizzBuzz? Na podstawie przekazanego do metody argumentu zwracasz odpowiednią wartość. Jeżeli argument jest podzielny przez 3, zwracasz Fizz, jeżeli argument jest podzielny przez 5, zwracasz Buzz, a jeżeli jest podzielny przez 3 i 5 to wtedy zwracasz FizzBuzz. Jeżeli argument nie jest podzielny przez 3 ani przez 5 to wtedy zwracasz wartość argumentu.
Przykładowy kod może wyglądać w taki sposób:

```
public string FizzBuzz(int value)
{
   if (value % 3 == 0 && value % 5 == 0)
      return "FizzBuzz";

   if (value % 3 == 0)
      return "Fizz";

   if (value % 5 == 0)
      return "Buzz";

   return value.ToString();
}
```

To zadania wydaje się bardzo łatwe, ale dużo kandydatów robi w tym zadaniu błąd. Musisz uważać, aby warunki, które piszesz były w odpowiedniej kolejności. Najczęściej kandydaci zaczynają od warunku:

```
if (value % 3 == 0)
    return "Fizz";
```

I to jest błąd, ponieważ na przykład wartość 15 zwróci nam wtedy "Fizz", a powinno zwrócić "FizzBuzz". Najlepiej po napisaniu kodu, jeszcze raz na spokojnie sobie przemyśleć różne przypadki.

## PODSUMOWANIE:

Jak widzisz, pytania nie są trudne, więc nie masz się czego obawiać. Warto jednak przed każdą rozmową kwalifikacyjną sobie je powtórzyć. Jak widzisz pytanie 9 oraz 10 miałem do zaimplementowania na kartce i uwierz mi, że jest to dość popularna praktyka na rozmowach kwalifikacyjnych. Proponuję Ci, abyś sobie w domu też poćwiczył takie programowanie na kartce, bo za pierwszym razem może Ci być trudno. Czasem na pytania teoretyczne odpowiadałem ustnie, a czasem były to pytania do rozpisania na kartce papieru. Oprócz tych pytań miałem jeszcze dużo innych, ale w tym artykule chciałem się z Tobą podzielić tylko tymi, które się najczęściej pojawiały. Jeżeli chcesz, abym się z Tobą podzielił również tymi mniej popularnymi pytaniami, które miałem na swojej drodze, to proszę, poinformuj mnie o tym, a na pewno w najbliższym czasie napiszę kolejny artykuł na ten temat. Powodzenia!

Kontakt: kazimierz.szpin@modestprogrammer.pl