81 Xamarin Interview Questions

By Kristijan Kralj

Table of Contents

Introduction

Welcome, dear colleague developer. I'll keep this introduction short since no one reads this anyway.

I made this book while working on the **MVVM in Xamarin.Forms Udemy course**.

The purpose of this eBook is to collect at one place all Xamarin interview questions that might pop up at your next interview.

I have a favor to ask: if you see I made a mistake anywhere in this ebook or would like to suggest a question, please let me know. I'll be happy to correct that in the next version of this book.

You can reach me at kristijan@blueinvader.com.

Ok, enough with this introduction! Let's start with the questions.

About the author

Hi,

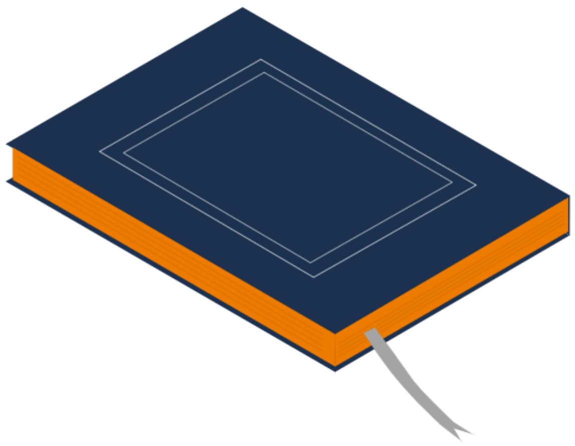
My name is Kristijan. After more than 10 years of development experience, I've decided to start giving back to the community.

Before this eBook, I wrote a free eBook 50 iOS Interview Questions.

You can also improve your skills with my courses:

- Refactoring in C#
- MVVM in Xamarin.Forms

The C# Basics



1. What is a reference type?

Reference type means that variable which is a reference type has a reference to the address where the value is stored. With reference type, two variables can reference the same object. If you change one variable, you will also change another variable.

The following keywords are used to define reference types: `class`, `interface` and `delegate`. C# also has built-in reference types: `dynamic`, `object` and `string`.

2. What is the value type?

Value type means that variable which is a value type directly contains the value. So, if a variable is `int number = 3;`

then 3 in memory is stored directly in the *number* variable. When you pass the value type variable to a method or assign it to another variable, then its value is copied. This means that change to one variable does not affect the value of another variable.

Some examples of value types are `bool`, `int`, `decimal`, `double`, `enum` and `struct`.

3. What is the difference between class and struct?

Classes are reference types, while structs are value types. Classes support inheritance and can be null, and should be used for complex objects. Structs don't support inheritance, they are value types and are typically used to represent small objects.

4. What is an interface?

An interface defines a contract. Any class or struct that implements that contract must provide an implementation of the members defined in the interface. Beginning with C# 8.0, an interface may define a default implementation for members. It may also define static members in order to provide a single implementation for common functionality.

5. What is the difference between an interface and an abstract class?

The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share. So, they are pretty similar, and before C# 8.0, the main difference was that interfaces couldn't have a default implementation for methods and properties. Starting with C# 8.0, an interface may define a default implementation for methods and properties.

The other difference is that a class can inherit only one abstract class, while it can implement multiple interfaces.

6. Where do we use async and await in C#?

`async` and `await` are keywords used to create and execute asynchronous code. Asynchronous execution means that the code execution is performed on a thread which is not the main thread. This enables us to have a responsive UI.

`async` keyword is used while declaring an asynchronous method, while `await` is used to call that kind of method.

7. What are generics?

Generic class, struct, method or interface is used to define a code that doesn't have a concrete type.

Example:

```
// Declare the generic class.
public class GenericClass<T>
{
    public void SomeMethod(T input) { }
}
```

This enables us to write one functionality, but reuse it for multiple types. Instead of using a specific data type, we use a generic type parameter, usually T.

8. Can we use this keyword within a static method?

We can't use `this` in a static method because that keyword returns a reference to the current instance of the class containing it. Static methods do not belong to a particular instance. They exist without creating an instance of the class and are called with the name of a class, not by instance.

9. What is method overloading?

Method overloading is having several methods with the same name, but they have different signatures (different parameters). Multiple methods can have the same name as long as the number and/or type of parameters are different.

10. What is LINQ?

LINQ stands for Language-Integrated Query. This is a query syntax to retrieve data from different sources and formats. It is already integrated in C# language, you just need to import `System.Linq` namespace.

Example of querying list of development languages:

```
List<string> developmentLanguages = new List<string> { "C#", "Java",
"Python", "Go" };

// LINQ Query
var result = from name in developmentLanguages
                    where name.Contains('a')
                    select name;

foreach (var name in result)
    Console.Write(name + " "); // "Java" is the output
```

The alternative way to perform LINQ queries is through method chaining:

```
// the same query as above
var result = developmentLangugages.Where(name => name.Contains('a'));
```

As you can see, this is a very concise way of filtering data.

11. What is the difference between "as" and "is" operators?

The `as` operator is used to cast an instance of an object to a particular type. If it fails, it will return null.

The `is` operator is used to check whether an object can be casted to a particular type. It returns `true` if the object can be casted to a particular type, `false` if it can't.

12. What is the difference between "throw" and "throw ex" in C#?

When called in the `catch` block, the `throw` statement preserves the original stack trace. `throw` ex will have stack trace only from that catch block. You should try to use the `throw` keyword whenever you can because it provides more error information.

13. Why do we use reflection?

You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them.

14. What are extension methods?

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are static methods, but they're called as if they were instance methods on the extended type.

The next example shows how to extend `DateTime` type to provide a new method to it, to check whether or not the day of week for a date is weekend:

```
public static class DateTimeExtensions
{
    public static bool IsWeekend(this DateTime dateTime)
    {
        return dateTime.DayOfWeek == DayOfWeek.Saturday ||
                dateTime.DayOfWeek == DayOfWeek.Sunday;
    }
}
```

And the usage:

```
public bool CanScheduleEvent()
{
    bool canSchedule = DateTime.Now.IsWeekend();
    return canSchedule;
}
```

15. What is a virtual method?

A virtual method is a method that can be overridden in the subclasses. A virtual method has an implementation in the base class but subclass can choose to replace that implementation. We use the `virtual` keyword to indicate that the method can be overridden.

16. What are anonymous types?

Anonymous types allow us to create new types without defining them. This is a way of defining read-only properties in a single object without having to define each type explicitly. The type is generated by the compiler and is accessible only in the current block of code. The type of properties is also inferred by the compiler. We can create anonymous types by using the `new` keyword together with the object initializer.

17. What are nullable value types?

A nullable value type is a type that can have value type or null. We define nullable value types by adding the "?" to the type: `var int? = null`.

18. What are nullable reference types?

Nullable reference types are available beginning with C# 8.0, in code that has opted in to a *nullable aware context*. Nullable reference types, the null static analysis warnings, and the null-forgiving operator are optional language features. All are turned off by default. A nullable context is controlled at the project level using build settings, or in code using pragmas.

In a nullable aware context:

- A variable of a reference type `T` must be initialized with non-null, and may never be assigned a value that may be null.
- A variable of a reference type `T?` may be initialized with null or assigned null, but is required to be checked against null before dereferencing.
- A variable `m` of type `T?` is considered to be non-null when you apply the null-forgiving operator, as in `m!`.

Nullable reference types aren't new class types, but rather annotations on existing reference types. The compiler uses those annotations to help you find potential null reference errors in your code.

19. What are boxing and unboxing?

Boxing is a process of converting value type to a reference type of type `object`. The unboxing is a process of explicit conversion of that same reference type back to the value type.

20. What is the difference between Continue and Break keyword?

In loops, the `break` keyword is used to finish the loop execution. The `continue` keyword is used to only finish the current iteration and continue program execution with the next iteration.

Xamarin.Forms

21. What is Xamarin?

Xamarin is an open-source platform for cross-platform mobile development. It uses C# and .NET to build iOS and Android apps.

Xamarin extends the .NET platform with tools and libraries specifically for building apps for iOS, Android and UWP (Universal Windows Platform). With Xamarin, you can write one code that will be compiled and executed on all 3 platforms.

22. What is XAML?

XAML stands for Extensible Application Markup Language. It's a declarative XML-based language developed by Microsoft. In Xamarin, it's used to create an application's UI.

23. What is Page in Xamarin.Forms?

In Xamarin.Forms, the Page class is used to define a visual element that occupies the entire screen. In other words, one page represents one screen of the mobile app.

Xamarin.Forms supports the following page types:

- ContentPage
- MasterDetailPage
- NavigationPage
- TabbedPage
- CarouselPage
- TemplatedPage

The page types mentioned above derive from the Xamarin.Forms Page class.

24. What is ContentPage?

ContentPage is a page that displays a single view. Since you usually need to show more than one view on a page, you group them in a container view such as `StackLayout` or `ScrollView`.

25. What is MasterDetailPage?

MasterDetailPage is a page that manages two panes of information: A master page that presents data at a high level, and a detail page that displays low-level details about information in the master.

26. What is NavigationPage?

NavigationPage is a page that manages navigation throughout the app. It contains a stack of other pages.

27. What is TabbedPage?

TabbedPage is a page where the user interface consists of a list of tabs and a larger detail area. On iOS, the list of tabs appears at the bottom of the screen, and the detail area is above. On Android and Windows phones, the tabs appear across the top of the screen. The user can scroll the collection of tabs that are across the top of the screen if that collection is too large to fit on one screen.

28. What is CarouselPage?

CarouselPage is a page that users can swipe from side to side to display pages of content, like a gallery.

29. What is TemplatedPage?

TemplatedPage is a page that displays full-screen content with a control template, and the base class for the `ContentPage` .

30. What is the purpose of the InitializeComponent() method in Page?

This method is auto-generated when we add any new XAML Page to the project. It initializes all the objects which were defined in the XAML file. It connects them with each other based on their parent child relations. Then, it generates the whole tree of objects like the content of a page.

31. What is the difference between CollectionView and ListView?

ListView is used to display a collection of data as a vertical list. CollectionView is used for presenting lists of data using different layout specifications. It aims to provide a more flexible, and performant alternative to ListView. Some major differences are:

- CollectionView has a flexible layout model, which allows data to be presented vertically or horizontally, in a list or a grid.
- CollectionView supports single and multiple selections.
- CollectionView reduces the API surface of ListView. Many properties and events from ListView are not present in CollectionView.
- CollectionView does not include built-in separators.

CollectionVIew is available from Xamarin.Forms 4.3.

32. What is the difference between native and cross-platform application mobile development?

Native mobile development is a type of development where a specific native language is used to develop mobile apps for iOS (Swift or Objective-C) and Android (Kotlin or Java).

Cross-platform development uses a single codebase to develop an app for iOS and Android at the same time. 3 most popular cross-platform development technologies are:

- Xamarin.Forms
- React Native
- Flutter

They act as a wrapper around native controls.

33. What is the default solution structure of a Xamarin.Forms project?

When you create a new Xamarin.Forms solution, you get the following project structure (*App* is the name of the app):

- App (the main project)
- App.Android
- App.iOS
- App.UWP

The main project contains UI and Business logic inside it. iOS, Android, and UWP contain platform-specific code.

34. What is a Custom Renderer?

Custom renderers provide an approach for customizing the appearance and behavior of Xamarin.Forms controls. They can be used for small styling changes or sophisticated platform-specific layout and behavior customization. With Custom renderers, you get the native look and feel for UI controls.

35. What are Effects?

Effects allow the native controls on each platform to be customized, and are typically used for small styling changes.

The process for creating an effect in each platform-specific project is as follows:

- Create a subclass of the `PlatformEffect` class.
- Override the `OnAttached` method and write logic to customize the control.
- Override the `OnDetached` method and write logic to clean up the control customization, if required.
- Add a `ResolutionGroupName` attribute to the effect class. This attribute sets a company wide namespace for effects, preventing collisions with other effects with the same name.
- Add an `ExportEffect` attribute to the effect class. This attribute registers the effect with a unique ID that's used by Xamarin.Forms, along with the group name, to locate the effect prior to applying it to a control.

The effect can then be consumed by attaching it to the appropriate control.

36. What are Triggers?

Triggers allow you to express actions declaratively in XAML that change the appearance of controls based on events or property changes.

You can assign a trigger directly to a control, or add it to a page-level or app-level resource dictionary to be applied to multiple controls.

37. What are property triggers?

Property triggers are triggers that change how the control looks like whenever some property changes. The next example shows a trigger that changes an `Entry` background color when it receives focus:

```
<Entry Placeholder="enter name">
    <Entry.Triggers>
        <Trigger TargetType="Entry"
                 Property="IsFocused" Value="True">
            <Setter Property="BackgroundColor" Value="Yellow" />
        </Trigger>
    </Entry.Triggers>
</Entry>
```

38. What are data triggers?

Data triggers use data binding to monitor another control to cause the Setters to get called. Instead of the Property attribute in a property trigger, set the Binding attribute to monitor for the specified value.

The next example shows how to use the data binding syntax `{Binding Source={x:Reference entry}, Path=Text.Length}` which is how we refer to another control's properties. When the length of the entry is zero, the trigger is activated. In this sample the trigger disables the button when the input is empty.

```
<Entry x:Name="entry"
       Text=""
       Placeholder="required field" />

<Button x:Name="button" Text="Save"
        FontSize="Large">
    <Button.Triggers>
        <DataTrigger TargetType="Button"
                     Binding="{Binding Source={x:Reference entry},
                                       Path=Text.Length}"
                     Value="0">
            <Setter Property="IsEnabled" Value="False" />
        </DataTrigger>
    </Button.Triggers>
</Button>
```

39. What are event triggers?

Event triggers are triggers which respond to events, such as the `Clicked` event.

40. What are multi triggers?

A `MultiTrigger` looks similar to a `Trigger` or `DataTrigger` except there can be more than one condition. All the conditions must be `true` before the Setters are triggered.

41. What are state triggers?

State triggers have been introduced in Xamarin.Forms 4.5, and are a specialized group of triggers that define the conditions under which a `VisualState` should be applied.

42. What is Xamarin.Essentials?

Xamarin.Essentials provides developers with cross-platform APIs for their mobile applications.

Android, iOS, and UWP offer unique operating system and platform APIs that developers have access to call in C# using Xamarin. Xamarin.Essentials provides a single cross-platform API that works with any Xamarin.Forms, Android, iOS, or UWP application that can be accessed from shared code no matter how the user interface is created.

Some Xamarin.Essentials features are:

- Access to clipboard
- App information
- Checking network access
- Email
- Maps
- Open browser
- SMS
- Text-to-speech

The full list of features can be found here.

43. What is Xamarin.Forms Shell?

Xamarin.Forms Shell is an application template that aims to reduce the complexity of mobile development by providing common navigation user experience, a URI-based navigation scheme, and an integrated search handler.

The process for creating a Xamarin.Forms Shell application is to create a XAML file that subclasses the `Shell` class, set the `MainPage` property of the application's App class to that `Shell` class, and then define the visual hierarchy of the application in the subclassed `Shell` class.

44. What is MessagingCenter?

MessagingCenter is a class that implements a publish-subscribe pattern. It’s used for communication between classes. This mechanism allows publishers and subscribers to communicate without having a reference to each other, helping to reduce dependencies between them.

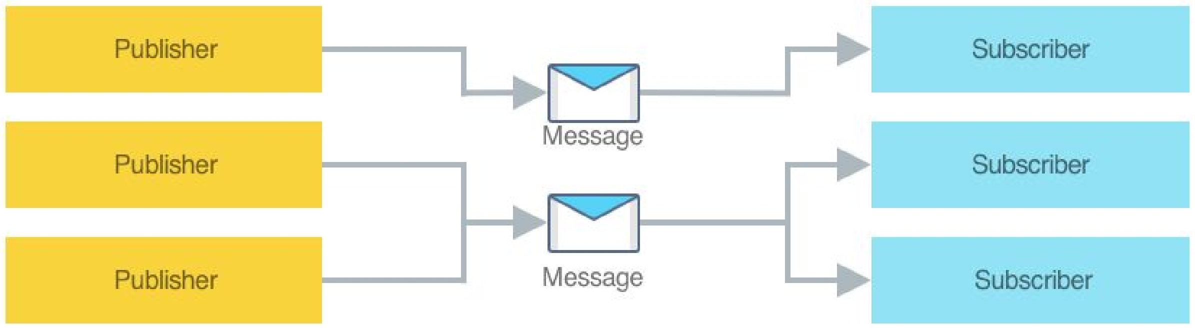


Image credit: [Xamarin.Forms MessagingCenter - Xamarin]

MessagingCenter needs to be used carefully, or it can cause memory management issues.

45. What is ResourceDictionary?

ResourceDictionary is a repository for resources that are used by a Xamarin.Forms application. Typical resources that are stored in a ResourceDictionary include styles, control templates, data templates, colors, and converters.

MVVM in Xamarin.Forms

46. What is MVVM?

Model-View-ViewModel (MVVM) is a design pattern that facilitates a separation of development of the UI from the development of the business logic or back-end logic (the data model).

The view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented in the view.

- **Model** - Model objects encapsulate the data specific to an application and define the business logic for manipulating and processing that data. For example, a model object might represent a character in a game or a contact in an address book.
- **View** - The view is the structure, layout, and appearance of what a user sees on the screen. A view object knows how to draw itself and can respond to user actions. The major purpose of view objects is to display data from the application's model objects and to enable the editing of that data.
- **View model** - The view model is an abstraction of the view exposing public properties and commands.

MVVM has some kind of binding mechanism, which automates communication between the view and its bound properties in the view model. This concept is known as data binding.

47. What is data binding?

Data bindings allow properties of two objects to be linked so that a change in one causes a change in the other. This is a very valuable tool, and while data bindings can be defined entirely in code, XAML provides shortcuts and convenience. Consequently, one of the most important markup extensions in Xamarin.Forms is *Binding*.

48. What types of data binding exist?

Data binding mode is a way to describe in what directions data flows between two connected properties. The binding mode is specified with a `BindingMode` enumeration.

- `Default` - this means, if the mode is not specified, the default is used. And the default is `OneWay` binding.
- `OneWay` – data goes from source to target.
- `TwoWay` – data goes both ways between source and target.
- `OneWayToSource` – data goes from target to source.
- `OneTime` – data goes from source to target, but only when the `BindingContext` property of the view changes (new with Xamarin.Forms 3.0)

49. What is view-to-view binding?

View-to-view binding is a type of binding between two UI controls. For example, the source of data for a `Label` can be a `Slider`.

50. What is BindingContext?

BindingContext is a property on a page or a UI control that defines the source of data for that page or UI control. In MVVM, BindingContext for the view is the view model.

51. What is a Value Converter?

If you have a binding between properties that are not of the same type, you need to use a class that implements the `IValueConverter` interface. For example, you would use a value converter if you have to convert a source property of type `int` to the type of the target property, which is `bool`.

52. What is Command?

Command is a property in the view model that is executed in reaction to a specific activity in the view such as a button click. To allow a data binding between a Button and a view model, the Button defines two properties:

- `Command` of type `System.Windows.Input.ICommand` - the action we want to execute once the button is clicked.
- `CommandParameter` of type `Object` - parameter we want to pass to the action which is going to be executed on a button click.

53. How to pass a parameter with Command?

To pass a parameter with the command, we use `CommandParameter`.

```
<Button Text="9" Command="{Binding DigitCommand}"
CommandParameter="9" />
```

The example above shows how to pass “9” as the command parameter.

54. What is the purpose of INotifyPropertyChanged?

View models generally implement the `INotifyPropertyChanged` interface, which means that the class fires a `PropertyChanged` event whenever one of its properties changes.

The data binding mechanism in Xamarin.Forms attaches a handler to this `PropertyChanged` event so it can be notified when property changes and keep the target updated with the new value.

55. What is BindableObject?

BindableObject is a class that implements `INotifyPropertyChanged` interface. View models can inherit from this class to raise property changes with the `OnPropertyChanged` method.

56. Show me the code example of raising the PropertyChanged event?

If the view model inherits from the BindableObject class, we can use the call to the `OnPropertyChanged("SomeProperty")` method to raise the event that the property value was changed, and that will update the UI.

To avoid having calls to the `OnPropertyChanged("SomeProperty")` in multiple places, what's usually recommended is to have a backing field for the property. That field holds the value of the property. After that field is updated in the setter of the property, call `OnPropertyChanged()`.

Then the property name in the `OnPropertyChanged()` is optional since the method will use the property name from which it's being called.

Example:

```
private string _someProperty;

public string SomeProperty
{
    get => _someProperty;
    set
    {
        _someProperty = value;
        OnPropertyChanged();
    }
}
```

And we update the property:

```
SomeProperty = "John";
```

57. What is ObservableCollection?

`ObservableCollection` is a dynamic data collection that provides notifications when items get added, removed, or when the whole list is refreshed. Every time this collection changes, the view is going to be notified.

58. What are .NET MVVM frameworks?

The purpose of the MVVM frameworks is to accelerate the development of Xamarin.Forms applications using the MVVM design pattern. A good MVVM frameworks usually provides you with the:

- Navigation service
- Built-in inversion of control container
- View model-to-view model navigation with parameters
- Other helper classes

There are many MVVM frameworks that offer these features.

59. What are the most popular .NET MVVM frameworks?

The most popular MVVM frameworks are:

- MVVMCross
- FreshMVVM
- Prism.Forms
- MVVMLight
- Caliburn.Micro

Architecture

60. What are the Object-Oriented Programming principles?

The object-oriented programming principles are:

- **Abstraction** - Objects only provide access to methods and properties that are relevant to the use of other objects, and hide implementation details.
- **Encapsulation** - The implementation and state of each object should not be exposed to the public.
- **Inheritance** - Code can be reused through the hierarchy.
- **Polymorphism** - At run time, objects of a derived class may be treated as objects of a base class in places such as method parameters and collections or arrays.

61. What are the SOLID principles?

S.O.L.I.D is an acronym for the 5 object-oriented design principles by Robert C. Martin (also known as Uncle Bob).

The principles are:

- **Single-responsibility principle** - An object should only have a single responsibility, that is, only changes if one part of the application changes.

- **Open-closed principle** - Objects should be open for extension, but closed for modification.
- **Liskov substitution** - you should be able to replace objects in an application with instances of their subtypes without changing the other code that uses that supertype.
- **Interface segregation principle** - Many smaller interfaces are better than one big interface.
- **Dependency inversion principle** - An object should depend upon interfaces, not concrete types.

If a developer follows these principles, then it should be easy for him to create applications that are easy to maintain and extend.

62. What is a Dependency Injection?

Dependency injection is a technique where you give to an object the dependencies it needs. Dependencies are other objects that our object needs to fulfill its functionality.

The advantage of dependency injection is that the object who wants to call some services doesn't have to know how to construct those services. Instead, the object delegates the responsibility of providing its services to an external code.

63. What are IoC Containers?

Inversion of Control Containers are classes or packages which help us to register dependencies and then automatically resolve them when we need them.

64. What is the difference between inheritance and composition?

The composition is a type of relationship between objects where one object contains another object. For example, a `Worker` class might contain a property of type `Address`. Object composition is used to represent "*has-a*" relationships: every worker has an address.

Inheritance is a type of relationship between objects where they are arranged in a hierarchy. This is an "*is-a-type-of*" relationship. For example, class `Worker` might inherit from the class `Person`. This means that the `Worker` class is a type of `Person`.

In general, preferring composition over inheritance is a design principle that gives your code higher flexibility.

65. What are Design Patterns?

Design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.

It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

66. Explain Factory Method design pattern.

Factory method is a design pattern that provides an interface for creating objects in a superclass, but allows superclass to change the type of the object that will be created.

67. Explain Command design pattern.

Command is a design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you parameterize methods with different requests, delay or queue the request's execution, and support undoable operations. Command design pattern is used heavily in the MVVM architecture in Xamarin.Forms. It’s used to trigger action that should be executed in the view model after the user has performed an action (button tap, cell swipe...).

68. Explain Decorator design pattern.

Decorator is a design pattern that lets you attach new behavior to objects by placing these objects inside a special wrapper class that usually implements the same interface as the wrapped class.

69. Explain Singleton design pattern.

Singleton is a design pattern that ensures that a class has only one instance, while providing a global access point to this instance.

70. Explain Observer design pattern.

Observer is a design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they are observing. The example of using this pattern in Xamarin.Forms is the `MessagingCenter` class.

71. Explain Adapter design pattern.

Adapter is a design pattern that allows objects with incompatible interfaces to collaborate. The adapter implements the compatible interface and wraps an object with an incompatible interface. The other object then communicates with the original object through the adapter and compatible interface.

Testing

72. What is unit testing?

Unit testing is a type of testing where individual components of the software are tested.

The purpose of unit testing is to confirm that each unit of the software performs as designed. A unit is a small piece in a codebase, usually a single method in a class, but it can be the class itself.

73. What is the structure of a unit test?

A unit test usually consists of three main parts:

- **Arrange** objects, create and set them up as necessary.
- **Act** on an object. Call the method you want to test.
- **Assert** (check) that something works as expected.

This structure is also known as 3A.

74. What is code coverage?

Code coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. An application with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to an application with low test coverage.

75. What is integration testing?

An integration test is a type of test where you test that several classes work together as expected. Integration tests usually involve interaction with a database, file system or mail. Therefore their execution time is long since you need to set up the database first and clean it up after the test is completed.

76. What is UI testing?

UI tests are tests that make sure your app's user interface behaves correctly when expected actions are performed.

With UI testing we can find and interact with UI elements, and validate UI properties and state.

77. What are stubs?

A stub is a dummy implementation of a dependency your class under test needs to execute a testing scenario. Its main purpose is to return fake data. With stubs, you don't have to deal with network or database in your tests, which enables you to have fast unit tests.

78. What are mocks?

A mock is a fake implementation of a dependency your class under test needs to execute a testing scenario. In this, it's pretty similar to the stub.

However, the main difference between the mock and the stub is that the mock usually verifies that the class under test behaves as expected. The mock checks that some method was called and/or that it was called with correct parameters. In the assert part of the test, you check your assumptions against the mock.

79. What are mocking libraries?

Mocking library is a library that provides an automated way to define and create mocks and stubs.

The popular mocking libraries in .NET are moq, NSubstitute and FakeItEasy.

80. What is Test-Driven Development?

Test-driven development is a software development process where tests are written before the code (the actual implementation). The tests guide the developer to the path of the actual implementation. They drive the implementation, hence the name test-driven development (TDD).

The process of test-driven development follows the following phases:

- Quickly add a test.
- Run all tests and see the new one fail.
- Make a little change.
- Run all tests and see them all succeed.
- Refactor to remove duplication.

The TDD process is also known as the Red-Green-Refactor cycle.

81. What is refactoring?

Refactoring is a change made to the code to make it easier to understand and cheaper to modify without changing its observable behavior. The advantages of refactoring your code are:

- You'll **minimize the risk** of introducing bugs while changing code.
- Refactoring helps you to structure the code so it's **easy to add new features** in the future.
- Refactoring organizes your code so you can easily understand and change it. Even 6 months from now? Yes, even 6 months from now **you'll understand what the code does**.

You can also learn more about refactoring in my Refactoring in C# course on Udemy.

Conclusion

Congratulations! You've made it to the end! You are now one step closer to getting that Xamarin job you want.

Remember to take plenty of sleep before your interview.

Best of luck!

Kristijan Kralj

References

1. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/
2. https://refactoring.guru/design-patterns/catalog
3. https://en.wikipedia.org/wiki/Software_design_pattern
4. https://en.wikipedia.org/wiki/Object-oriented_programming