Kotlin

Basic Kotlin Interview Questions (1-25)

1. **What is Kotlin?**
    - Kotlin is a statically typed programming language that runs on the Java Virtual Machine (JVM). It is fully interoperable with Java and is designed to be more concise and expressive.
2. **What are the advantages of Kotlin over Java?**
    - Kotlin offers concise syntax, null safety, extension functions, smart casts, coroutines for asynchronous programming, and better support for functional programming.
3. **What is the difference between `val` and `var` in Kotlin?**
    - `val` is used for immutable references (like `final` in Java), while `var` is used for mutable references.
4. **What is the `fun` keyword in Kotlin?**
    - The `fun` keyword is used to declare a function in Kotlin.
5. **What are data classes in Kotlin?**
    - Data classes are classes that are used to hold data. They automatically generate functions like `toString()`, `equals()`, and `hashCode()`.
6. **What is null safety in Kotlin?**
    - Null safety in Kotlin prevents null pointer exceptions by distinguishing between nullable and non-nullable types.
7. **What is a companion object in Kotlin?**

- A companion object is an object declared inside a class, which is used to hold static-like members in Kotlin.

8. **Explain the concept of extension functions in Kotlin.**

   - Extension functions allow adding functionality to existing classes without modifying their source code.

9. **What is the difference between `==` and `===` in Kotlin?**

   - `==` checks for value equality (similar to `equals()`), while `===` checks for reference equality (i.e., whether two references point to the same object).

10. **What are `lateinit` and `lazy` in Kotlin?**

- `lateinit` is used to initialize a variable later, while `lazy` is used for lazy initialization of a value.

11. **What are the different types of variables in Kotlin?**

- Variables in Kotlin can be of types `val` (immutable) or `var` (mutable).

12. **What is a primary constructor in Kotlin?**

- A primary constructor is a part of the class declaration and can be used to initialize the properties of a class.

13. **What is a secondary constructor in Kotlin?**

- A secondary constructor is a constructor that is not part of the class declaration and is used to provide additional ways to initialize the class.

14. **How do you handle exceptions in Kotlin?**

- Exceptions are handled using `try`, `catch`, and `finally` blocks in Kotlin, similar to Java.

15. **What is the difference between a class and an object in Kotlin?**

- A class is a blueprint for creating objects, while an object is an instance of a class.

16. **What is the difference between `Array` and `List` in Kotlin?**

- `Array` is a fixed-size collection that can hold elements of any type, while `List` is a read-only collection that can hold a dynamic number of elements.

17. **What are coroutines in Kotlin?**

- Coroutines are a way of handling asynchronous programming and background tasks in Kotlin, allowing you to write asynchronous code in a sequential manner.

18. **What is the difference between `is` and `as` in Kotlin?**

- `is` is used to check the type of a variable, while `as` is used for casting a variable to a specific type.

19. **Explain the `when` expression in Kotlin.**

- The `when` expression is similar to `switch` in other languages and is used for branching based on a condition or value.

20. **What is the purpose of `in` keyword in Kotlin?**

- The `in` keyword is used for checking if a value is within a range or for iterating over collections.

21. **What are sealed classes in Kotlin?**

- Sealed classes restrict class hierarchies to a limited set of subclasses, providing more control over inheritance.

22. **How do you define default parameter values in Kotlin?**

- You can define default values for function parameters in Kotlin by specifying them in the function signature.

23. **What is the purpose of the `apply` function in Kotlin?**

- The `apply` function is used for configuring objects, executing multiple operations on an object in a block, and returning the object itself.

24. **What is the `with` function in Kotlin?**

- The `with` function allows you to execute a block of code on an object without having to refer to the object explicitly every time.

25. **What is the difference between `StringBuilder` and `StringBuffer` in Kotlin?**

- `StringBuilder` is used for mutable strings in single-threaded contexts, while `StringBuffer` is used in multi-threaded contexts.

Intermediate Kotlin Interview Questions (26-50)

26. **What are higher-order functions in Kotlin?**

- Higher-order functions are functions that take other functions as parameters or return functions.

27. **What are lambda expressions in Kotlin?**

- Lambda expressions are anonymous functions that can be passed around as values.

28. **What is the purpose of `inline` functions in Kotlin?**

- The `inline` keyword is used to reduce the overhead of lambda expressions by inserting the function body directly into the calling function.

29. **What is a generic class in Kotlin?**

- A generic class in Kotlin allows you to define classes, interfaces, and functions that work with any type.

30. **What is the `out` keyword in Kotlin?**

- The `out` keyword is used for covariance, allowing you to return a subtype of a specified type.

31. **What is the `in` keyword in Kotlin?**

- The `in` keyword is used for contravariance, allowing you to pass a supertype of a specified type.

32. **What is an `interface` in Kotlin?**

- An interface in Kotlin is a contract that classes can implement. It can contain abstract methods as well as default method implementations.

33. **How does Kotlin handle null references?**

- Kotlin uses nullable types (`Type?`) and null safety features to prevent null pointer exceptions.

34. **What is the purpose of the `@JvmStatic` annotation in Kotlin?**

- The `@JvmStatic` annotation is used to mark a method as static when generating Java bytecode, allowing it to be accessed without an instance of the class.

35. **What are extension properties in Kotlin?**

- Extension properties allow you to add properties to existing classes without modifying their source code.

36. **What is destructuring declaration in Kotlin?**

- Destructuring declaration allows you to unpack a data class into separate variables.

37. **What are the different types of collections in Kotlin?**

- Kotlin provides `List`, `Set`, and `Map` collections, each of which can be mutable or immutable.

38. **What is the `lazy` initialization in Kotlin?**

- `lazy` initialization is a way to initialize a variable only when it is first accessed.

39. **What are the benefits of using `sealed classes`?**

- Sealed classes allow you to define a limited set of subclasses, making it easier to handle exhaustive `when` expressions.

40. **How does `run` function work in Kotlin?**

- The `run` function is used for executing a block of code and returning the result of the last expression in that block.

41. **What are `try-catch-finally` blocks used for in Kotlin?**

- These blocks are used to handle exceptions in a structured way, where `finally` executes regardless of whether an exception is thrown or not.

42. **What is `apply` vs `also` in Kotlin?**

- Both `apply` and `also` are used for performing operations on an object, but `apply` returns the object itself, while `also` returns the result of the lambda.

43. **What is `reified` in Kotlin?**

- The `reified` keyword is used in inline functions to access the actual type of a generic type parameter.

44. **How does Kotlin handle multiple inheritance?**

- Kotlin allows multiple inheritance through interfaces, but only single inheritance for classes.

45. **What is the purpose of the `override` keyword in Kotlin?**

- The `override` keyword is used to indicate that a function or property in a subclass is overriding a function or property in a superclass.

46. **What is `sealed interface` in Kotlin?**

- A `sealed interface` is an interface that restricts its implementation to a limited set of types, similar to sealed classes.

47. **How does `ArrayList` work in Kotlin?**

- `ArrayList` in Kotlin is a resizable array-backed collection, part of the `MutableList` interface.

48. **What is the difference between `set` and `list` in Kotlin?**

- A `set` is a collection of unique elements, while a `list` is an ordered collection that may contain duplicates.

49. **How do you handle concurrency in Kotlin?**

- Kotlin provides coroutines for handling concurrency and asynchronous tasks in a more lightweight and efficient way.

50. **What are `reified` type parameters in Kotlin?**

- `reified` type parameters are used in inline functions to preserve type information for generics.

Advanced Kotlin Interview Questions (51-75)

51. **How do coroutines work in Kotlin?**

- Coroutines are lightweight threads that allow asynchronous and non-blocking code execution, simplifying concurrency.

52. **What is a `CoroutineScope` in Kotlin?**

- A `CoroutineScope` is used to manage and control the lifecycle of coroutines, ensuring that they are cancelled when no longer needed.

53. **What is a `Channel` in Kotlin coroutines?**

- A `Channel` is a way of communicating between coroutines, providing a safe way to send and receive data.

54. **What is the `suspend` keyword in Kotlin?**

- The `suspend` keyword is used to mark a function that can be paused and resumed without blocking the thread.

55. **What is `CoroutineDispatcher` in Kotlin?**

- `CoroutineDispatcher` is used to control the thread or thread pool on which a coroutine runs.

56. **What is `Flow` in Kotlin?**

- `Flow` is an asynchronous stream of values that can be collected and processed in a non-blocking manner.

57. **What is the difference between `launch` and `async` in Kotlin coroutines?**

- `launch` is used for launching a coroutine without returning a result, while `async` is used for launching a coroutine that returns a result.

58. **What is `withContext` in Kotlin coroutines?**

- `withContext` is used to change the context (thread) of a coroutine, typically used for switching between different dispatchers.

59. **Explain the concept of `actor` in Kotlin.**

- An `actor` is a special type of coroutine used for managing state in a concurrent manner.

60. **What is the purpose of `Job` in Kotlin coroutines?**

- `Job` is used to manage the lifecycle of a coroutine and can be cancelled when necessary.

61. **What are `suspend` functions and how are they used in Kotlin?**

- `suspend` functions are special functions that can be suspended during their execution and resumed later, making them suitable for asynchronous operations.

62. **How do you handle exceptions in Kotlin coroutines?**

- Exceptions in coroutines are handled by using `try-catch` blocks or using structured concurrency principles with `supervisorScope`.

63. **What is `CoroutineExceptionHandler` in Kotlin?**

- `CoroutineExceptionHandler` is used to handle uncaught exceptions in coroutines.

64. **What is the `GlobalScope` in Kotlin?**

- `GlobalScope` is a global singleton scope for launching top-level coroutines that are not bound to any specific lifecycle.

65. **What is `runBlocking` in Kotlin coroutines?**

- `runBlocking` is used to start a coroutine in the main thread, blocking the thread until the coroutine completes.

66. **How does Kotlin handle memory management?**

- Kotlin relies on the JVM's garbage collection system for memory management, but it also has its own memory optimizations for handling null safety and object lifecycles.

67. **What is `Kotlin Native`?**

- Kotlin Native is a technology that compiles Kotlin code to native binaries, enabling Kotlin to be used for developing applications for platforms like iOS and embedded systems.

68. **What is the `@Inject` annotation in Kotlin?**

- The `@Inject` annotation is used for dependency injection in Kotlin, typically with frameworks like Dagger or Koin.

69. **What is `Kotlin Multiplatform`?**

- Kotlin Multiplatform allows you to write code that can run on multiple platforms (Android, iOS, JVM, JS) without rewriting the logic for each platform.

70. **How do you perform testing in Kotlin?**

- Kotlin supports testing using libraries like JUnit, TestNG, and others, and can be used in combination with mocking libraries like Mockito.

71. **What are the pros and cons of using Kotlin over Java?**

- Pros include null safety, concise syntax, and enhanced functionality. Cons include learning curve and sometimes larger binary size.

72. **What is `Kotlin DSL`?**

- Kotlin DSL (Domain Specific Language) allows you to create expressive and domain-specific languages using Kotlin.

73. **What is the difference between `apply` and `also` in Kotlin?**

- Both `apply` and `also` are used for operations on an object, but `apply` returns the object itself, while `also` returns the result of the lambda.

74. **How do you manage concurrency using Kotlin?**

- You can manage concurrency using Kotlin coroutines and related constructs like `Channel`, `Flow`, and `Job`.

75. **Explain the concept of `Inline` functions in Kotlin.**

- `inline` functions are used to avoid the overhead of lambda expressions by inlining the code of the lambda at the call site.

Technical Kotlin Interview Questions (76-100)

76. **How does Kotlin interact with Java?**

- Kotlin is fully interoperable with Java, meaning that Kotlin code can call Java classes and vice versa.

77. **What is `annotation class` in Kotlin?**

- `annotation class` is used to define annotations, which are metadata for code that can be accessed during runtime.

78. **How do you convert a `String` to `Int` in Kotlin?**

- Use `toInt()` to convert a `String` to an `Int`, or `toIntOrNull()` to avoid exceptions if the string is not a valid number.

79. **Explain the `Array` functions in Kotlin.**

- Kotlin provides various functions to manipulate arrays, like `map()`, `filter()`, `forEach()`, etc.

80. **What is the role of `object` in Kotlin?**

- The `object` keyword is used to define a singleton, an anonymous class, or a companion object.

81. **How do you define a custom exception in Kotlin?**

- Custom exceptions in Kotlin are defined by inheriting from `Exception` or any subclass of it.

82. **What is `super` used for in Kotlin?**

- `super` is used to access members (methods or properties) of a superclass.

83. **What is a `map` in Kotlin?**

- A `map` is a collection that holds key-value pairs, where each key maps to exactly one value.

84. **How do you access Java classes from Kotlin?**

- Kotlin allows direct access to Java classes without requiring additional syntax, thanks to its interoperability with Java.

85. **What are `when` expressions used for in Kotlin?**

- `when` expressions are used for branching based on different conditions, similar to a `switch` statement in other languages.

86. **What is `with` function in Kotlin?**

- `with` is a scoping function used to operate on an object within a block of code.

87. **What is `takeIf` in Kotlin?**

- `takeIf` is used to return the object if it satisfies a condition, or `null` otherwise.

88. **How do you use `for` loops in Kotlin?**

- Kotlin provides traditional `for` loops and enhanced `forEach` methods for iterating over collections.

89. **What is `uninitialized` in Kotlin?**

- Uninitialized properties are declared but not yet assigned a value. Using them can lead to errors unless handled properly.

90. **What is `class delegation` in Kotlin?**

- Class delegation is a mechanism in Kotlin where a class delegates the implementation of some methods to another class.

91. **What is `Kotlin Reflection`?**

- Kotlin Reflection is a feature that allows the inspection of classes, functions, and properties at runtime.

92. **Explain the `delegates` in Kotlin.**

- `delegates` in Kotlin are used to delegate the implementation of properties to another object.

93. **What is `@JvmOverloads` in Kotlin?**

- The `@JvmOverloads` annotation generates overloaded methods with default values to make Kotlin code compatible with Java.

94. **What is `CoroutineScope` in Kotlin?**

- `CoroutineScope` defines the context in which coroutines are launched and their lifecycle.

95. **How do you handle background tasks in Kotlin?**

- Background tasks can be handled using coroutines for non-blocking execution.

96. **What is `lambda` with `receiver` in Kotlin?**

- A lambda with receiver is a lambda that can access members of the receiver object directly.

97. **What is `runBlocking` used for in Kotlin?**

- `runBlocking` is used to start a coroutine in the main thread and block until it completes.

98. **Explain the concept of `generics` in Kotlin.**

- Generics allow classes and functions to work with any type, increasing code reusability and safety.

99. **What is `@JvmName` annotation used for in Kotlin?**

- `@JvmName` allows renaming a function when it is compiled to JVM bytecode, ensuring compatibility with Java.

100. **How do you declare a singleton in Kotlin?**

- A singleton in Kotlin is declared using the `object` keyword, which creates a single instance of a class.

IDM TECHPARK

GUIDE`S FOR PERFECT CAREER PATHWAY