Introducción a Kotlin (teoría)

Ejercicios:
https://www.jesusninoc.com/10/21/introduccion-a-kotlin/

Teoría

1. Herencia: En Kotlin, la herencia se refiere a la capacidad de una clase de heredar propiedades y comportamientos de otra clase. Esto se logra mediante la declaración de una clase hija que extiende una clase padre. La clase hija adquiere las propiedades y métodos de la clase padre y puede agregar sus propias implementaciones o modificar el comportamiento heredado.

2. Polimorfismo: El polimorfismo es la capacidad de las clases derivadas de una misma clase base para proporcionar su propia implementación de un método heredado. Esto permite tratar objetos de clases derivadas como objetos de la clase base, lo que facilita la escritura de código genérico que funciona con múltiples tipos de objetos.

3. Excepciones: Las excepciones son eventos inusuales o errores que pueden ocurrir durante la ejecución de un programa. En Kotlin, puedes usar bloques try, catch, y finally para manejar excepciones. Cuando ocurre una excepción, el control se transfiere al bloque catch, donde puedes manejar el error de manera adecuada.

4. Anotaciones: Las anotaciones son metadatos que se pueden agregar a clases, funciones, propiedades u otros elementos en el código. Las anotaciones proporcionan información adicional sobre el elemento anotado y se utilizan para varios propósitos, como la generación de código, la documentación o la validación.

5. Variables y tipos de datos: En Kotlin, puedes declarar variables utilizando las palabras clave val (inmutable) o var (mutable). Los tipos de datos incluyen enteros, flotantes, cadenas, booleanos, y otros tipos personalizados. Kotlin es un lenguaje con seguridad de tipos, lo que significa que debes declarar el tipo de una variable.

6. Operadores: Kotlin admite una variedad de operadores como aritméticos (+, -, *, /), de comparación (==, !=, <, >), lógicos (&&, ||, !), y otros para realizar operaciones en valores.

7. Conversión de entero a cadena (Int a String): Puedes convertir un entero a una cadena en Kotlin utilizando el método toString() o simplemente mediante interpolación de cadenas.

8. Conversión de cadena a entero (String a Int): Para convertir una cadena a un entero, puedes utilizar el método toInt().

9. Conversión de cadena a flotante (String a Float): La conversión de cadena a flotante se realiza con toFloat().

10. Conversión de flotante a entero (Float a Int): Para convertir un flotante a un entero, puedes usar toInt().

11. Conversión de entero a flotante (Int a Float): La conversión de entero a flotante se hace con toFloat().

12. Conversión de cadena a entero (String a Int) con parseInt: Puedes utilizar la función toInt() o Integer.parseInt() para convertir una cadena a un entero.

13. Manejo de excepción al convertir una cadena no válida: Debes usar bloques try y catch para capturar excepciones al convertir una cadena no válida a un entero o flotante.

14. Estructuras de Control (if): Las estructuras de control if se utilizan para tomar decisiones basadas en condiciones

booleanas. Puedes tener if, else, else if, y when (una versión más avanzada del switch).

15. Funciones: En Kotlin, las funciones se definen con la palabra clave fun. Pueden tener parámetros y devolver valores. También puedes definir funciones de extensión que se aplican a tipos existentes.

16. Bucle for simple: El bucle for se utiliza para iterar sobre una secuencia de elementos. En Kotlin, es común usar bucles for en lugar de bucles while debido a la versatilidad del operador in.

17. Bucle while: El bucle while se usa para repetir una serie de declaraciones mientras una condición sea verdadera.

18. Bucle do-while: El bucle do-while es similar al bucle while, pero garantiza que el bloque de código se ejecutará al menos una vez, ya que la comprobación de la condición se realiza después de la ejecución del bloque.

19. Bucle for con lista: Puedes usar un bucle for para iterar sobre elementos de una lista, colección o matriz.

20. Bucle for con índices: En Kotlin, puedes usar un bucle for para iterar sobre índices de una lista o matriz, lo que te permite acceder a elementos específicos.

21. Clases y Objetos: En Kotlin, una clase es un plano para crear objetos. Los objetos son instancias de clases. Las clases contienen propiedades y métodos que definen su estructura y comportamiento.

22. Colecciones: Las colecciones son estructuras de datos que pueden contener múltiples elementos. En Kotlin, existen diversas colecciones, como listas, conjuntos y mapas, que facilitan el manejo de datos.

23. Null Safety: Kotlin introduce conceptos de seguridad en cuanto a valores nulos. Puedes declarar explícitamente si una

variable puede ser nula (utilizando var con ?) o no nula (utilizando var sin ?). Esto ayuda a evitar errores de referencia nula.

24. Extension functions: Las extension functions permiten agregar nuevos métodos a clases existentes sin modificar su código fuente. Esto es útil para extender la funcionalidad de clases estándar de Kotlin o bibliotecas externas.

25. Tratamiento de excepciones: Ya mencionamos las excepciones, pero es importante destacar que en Kotlin puedes definir tus propias excepciones personalizadas y manejarlas de manera específica.

26. Entrada y salida: Este concepto se refiere a la lectura y escritura de datos en la consola o en archivos. Puedes usar funciones como readLine() para obtener entrada del usuario y println() para mostrar resultados en la consola.

27. Iteración y bucles: Aparte de los bucles for, while y do-while, Kotlin proporciona expresiones como forEach para simplificar la iteración sobre colecciones.

28. Entrada de datos del usuario: Se refiere a la captura de datos que el usuario introduce en la aplicación. Puedes usar funciones de entrada para recibir información del usuario.

29. Funciones recursivas: Las funciones recursivas son aquellas que se llaman a sí mismas. Esto es útil para resolver problemas que se pueden dividir en subproblemas más pequeños.

30. Lectura y escritura de archivos: Implica la capacidad de leer datos desde archivos y escribir datos en ellos. Puedes usar funciones y clases para lograr esto.

31. API de Internet (HTTP): Te permite comunicarte con servidores web y realizar solicitudes HTTP. Puedes usar bibliotecas como Retrofit para simplificar esta tarea.

32. Uso de librerías externas (GSON): GSON es una biblioteca

que facilita la conversión entre objetos Kotlin y formato JSON. Puedes usarla para trabajar con datos JSON.

33. Programación orientada a objetos (Herencia): Ya mencioné la herencia, pero en el contexto de la programación orientada a objetos, se refiere a la estructura de clases y objetos para modelar el mundo real.

34. Programación funcional (Funciones de orden superior): La programación funcional es un paradigma que se centra en el uso de funciones de orden superior, que son funciones que pueden tomar otras funciones como argumentos o devolverlas como resultados.

35. Programación concurrente (Hilos): La programación concurrente implica la ejecución simultánea de tareas. En Kotlin, puedes utilizar hilos (threads) para lograr esto.

36. Expresiones lambda y funciones anónimas: Las expresiones lambda son funciones sin nombre que se pueden pasar como argumentos. Son útiles en programación funcional.

37. Programación orientada a eventos (Escuchadores): En el desarrollo de interfaces de usuario, se utilizan escuchadores para responder a eventos como clics de botones o interacciones del usuario.

38. Uso de anotaciones: Ya mencionamos las anotaciones, que proporcionan metadatos. Pueden ser utilizadas para generar código o documentación.

39. Uso de extension functions y properties: Las extension functions permiten agregar funciones a clases existentes, y las extension properties agregan propiedades. Esto mejora la legibilidad y la reutilización del código.

40. Tratamiento de errores y excepciones personalizadas: Además de manejar excepciones, puedes definir tus propias excepciones personalizadas para situaciones específicas.

41. Manejo de listas con funciones de orden superior: Kotlin proporciona funciones de orden superior como map, filter, y reduce que facilitan la manipulación de listas y colecciones de manera elegante y funcional.

42. Uso de inicializadores de clase: Los inicializadores de clase (constructores) te permiten configurar y crear objetos de una clase. Puedes definir constructores primarios y secundarios.

43. Uso de companion objects: Companion objects son objetos que se asocian a una clase y pueden contener métodos y propiedades que pertenecen a la clase en lugar de una instancia de la clase.

44. Uso de lambdas con recorrido de mapas: Las lambdas son ampliamente utilizadas en Kotlin para procesar y transformar elementos en mapas.

45. Uso de interfaces y herencia múltiple: Kotlin admite herencia múltiple a través de interfaces. Puedes implementar múltiples interfaces en una clase.

46. Uso de enumeraciones (Enums): Las enumeraciones son un tipo de datos especial que define un conjunto fijo de constantes. Son útiles para representar opciones fijas, como los días de la semana.

47. Uso de funciones de extensión para colecciones: Las funciones de extensión aplicadas a colecciones permiten realizar operaciones específicas de manera más concisa y legible.

48. Creación de un objeto con propiedades: Puedes crear objetos en Kotlin con propiedades inmutables usando el constructor primario de la clase.

49. Creación de una clase con un constructor: Para crear una clase, necesitas definir su estructura y propiedades, junto

con un constructor.

50. Función para verificar un inicio de sesión: Puedes crear una función que verifique las credenciales de inicio de sesión y retorne un resultado.

51. Uso de hash tables: Las hash tables (tablas hash) son estructuras de datos que relacionan claves con valores. En Kotlin, puedes utilizar HashMap para este propósito.

52. Uso de listas para gestionar objetos: Las listas son ampliamente utilizadas para almacenar y gestionar conjuntos de objetos.

53. Uso de funciones de extensión para listas: Las funciones de extensión pueden ser aplicadas a listas para realizar operaciones específicas.

54. Uso de interfaz: Las interfaces definen un conjunto de métodos que las clases deben implementar. Esto permite la implementación de múltiples clases con un comportamiento común.

55. Uso de enumeraciones (juego de piedra, papel o tijera): Las enumeraciones se pueden usar para definir los posibles movimientos en juegos como «piedra, papel o tijera.»

56. Uso de JSON: Puedes trabajar con datos JSON en Kotlin, tanto para parsear como para generar estructuras de datos en este formato.

57. Uso de listas para gestionar objetos en JSON: Las listas son útiles para almacenar y manipular datos obtenidos desde fuentes JSON.

58. Uso de herencia en Kotlin (ejemplo sobre figuras geométricas): Puedes utilizar la herencia para modelar objetos del mundo real. Por ejemplo, crear una clase base «Figura Geométrica» y clases derivadas como «Círculo» y «Cuadrado.»

59. Uso de funciones de orden superior (higher-order functions): Las funciones de orden superior son esenciales en la programación funcional y permiten realizar operaciones sobre funciones.

60. Uso de claves (hash tables) para almacenar objetos JSON: Las claves pueden utilizarse para acceder a valores específicos en objetos JSON.

61. Uso de funciones closures: Las funciones closures son funciones anidadas que pueden capturar y acceder a variables del ámbito exterior. Son útiles en situaciones donde deseas mantener el estado de una variable.

62. Uso de herencia y polimorfismo: Estos conceptos están relacionados con la programación orientada a objetos. La herencia implica que las clases derivadas pueden heredar propiedades y métodos de las clases base, y el polimorfismo permite que los objetos se comporten de manera diferente según el contexto.

63. Uso de funciones de extensión en clases: Las funciones de extensión permiten agregar métodos a clases existentes sin modificar su código fuente. Esto puede ser útil para extender la funcionalidad de clases de bibliotecas.

64. Uso de colecciones y mapas: Las colecciones y los mapas son estructuras de datos fundamentales para almacenar y gestionar conjuntos de elementos. Kotlin proporciona métodos y operaciones para trabajar con ellos de manera eficiente.

65. Uso de funciones de extensión y funciones de orden superior: La combinación de funciones de extensión y funciones de orden superior puede simplificar la escritura de código más legible y conciso.

66. Uso de clases selladas (sealed classes): Las clases selladas son clases que no pueden tener subclases fuera de un archivo específico. Son útiles para modelar tipos de datos con

un número fijo de subtipos.

67. Uso de clases abstractas e interfaces: Las clases abstractas y las interfaces son importantes en la programación orientada a objetos. Las clases abstractas pueden contener métodos sin implementación, mientras que las interfaces definen un conjunto de métodos que las clases deben implementar.

68. Uso de propiedades computadas: En Kotlin, puedes definir propiedades que se calculan en tiempo real en lugar de almacenar un valor fijo. Esto es útil para propiedades cuyo valor cambia con el tiempo.

69. Uso de operadores sobrecargados: La sobrecarga de operadores te permite definir el comportamiento de operadores como +, -, ==, entre otros, para tus propias clases.

70. Uso de generics: La programación genérica te permite escribir código que puede funcionar con diferentes tipos de datos de manera segura. Puedes definir funciones y clases genéricas.

71. Uso de clases anidadas: Las clases anidadas son clases que se definen dentro de otras clases. Pueden ser útiles para organizar y encapsular la funcionalidad.

72. Uso de objetos singleton: Un objeto singleton es una instancia única de una clase que se crea una sola vez y se utiliza en toda la aplicación. Es útil cuando necesitas compartir una única instancia de una clase.

73. Uso de operadores infix: Los operadores infix permiten escribir funciones en notación infija, lo que mejora la legibilidad de ciertas operaciones.

74. Uso de expresiones lambda con receptor: Las expresiones lambda con receptor son útiles para acceder a las funciones de un objeto dentro del contexto de la expresión lambda.

75. Uso de rangos (ranges): Los rangos te permiten representar una secuencia de valores en un intervalo. Son útiles en bucles y comparaciones.

76. Uso de colecciones mutable y funciones de modificación: Las colecciones mutables permiten modificar su contenido. Puedes usar funciones de modificación como add, remove, y clear para cambiar los elementos de la colección.

77. Uso de mapas (diccionarios) y acceso a elementos: Los mapas son estructuras de datos clave-valor. Puedes utilizar claves para acceder a los valores asociados.

78. Uso de expresiones when: La expresión when es una alternativa a las declaraciones if y else if. Permite comparar un valor con múltiples opciones y ejecutar el código correspondiente a la primera coincidencia.

79. Uso de corrutinas (coroutines): Las corrutinas son útiles para realizar operaciones asincrónicas y paralelas en Kotlin de manera más eficiente que los hilos tradicionales.

80. Uso de expresiones lambda y high-order functions: Las expresiones lambda y las funciones de orden superior son fundamentales en la programación funcional de Kotlin.

81. Uso de anotaciones (annotations): Ya mencionamos las anotaciones, que se utilizan para proporcionar metadatos. También puedes crear tus propias anotaciones personalizadas.

82. Uso de expresiones lambdas con receptores de contexto: Las expresiones lambda con receptores de contexto permiten acceder a las propiedades y funciones de un objeto receptor dentro de la expresión lambda.

83. Uso de programación orientada a aspectos: La programación orientada a aspectos (AOP) te permite separar preocupaciones transversales en tu código, como el registro y la seguridad, de la lógica principal de la aplicación.

84. Uso de clases selladas (sealed classes): Las clases selladas son útiles para crear jerarquías de tipos de datos con un conjunto fijo de subtipos.

85. Uso de inicializadores de instancia: Los inicializadores de instancia permiten realizar tareas de configuración cuando se crea una instancia de una clase.

86. Uso de enumeraciones con métodos: Las enumeraciones pueden contener métodos para realizar operaciones específicas en los valores enumerados.

87. Uso de extension properties: Al igual que las funciones de extensión, las extension properties permiten agregar propiedades a clases existentes.

88. Uso de expresiones lambda y funciones de orden superior con colecciones: Estos conceptos son fundamentales para el trabajo con colecciones y transformación de datos en Kotlin.

89. Uso de expresiones regulares (Regex): Las expresiones regulares son patrones de búsqueda que se utilizan para buscar y manipular texto de manera sofisticada.

90. Uso de conversión de tipos y seguridad de tipos: Kotlin es un lenguaje con seguridad de tipos, lo que significa que debes realizar conversiones de tipos de manera segura para evitar errores en tiempo de ejecución.

91. Uso de extension functions con receptores de contexto: Las extension functions con receptores de contexto permiten interactuar con un objeto receptor dentro de la función de extensión.

92. Uso de métodos con argumentos con nombre: Puedes llamar a métodos con argumentos nombrados para mejorar la claridad y la legibilidad del código.

93. Uso de propiedades delegadas: Las propiedades delegadas permiten externalizar la lógica de acceso y modificación de

una propiedad.

94. Uso de lambdas con with y apply: Las funciones with y apply son útiles para realizar una serie de operaciones en un objeto sin tener que repetir su nombre.

95. Uso de la librería estándar: Kotlin ofrece una amplia gama de funciones y clases en su librería estándar que simplifican muchas tareas comunes en la programación.

96. Uso de la expresión when con múltiples condiciones: La expresión when te permite comparar un valor con múltiples condiciones y ejecutar código según la primera coincidencia.

97. Uso de cláusulas try y catch con recursos (try with resources): Puedes utilizar cláusulas try y catch para manejar excepciones y gestionar recursos como archivos o conexiones de manera segura.

98. Uso de expresiones de rango (range expressions): Las expresiones de rango permiten representar un rango de valores. Son comunes en bucles y comparaciones.

99. Uso de object expressions: Las expresiones de objeto permiten crear objetos anónimos en el lugar donde se necesitan, sin necesidad de definir una clase.

100. Uso de expresiones sealed y beneficios en jerarquías de clases: Las clases selladas son útiles para definir jerarquías de clases donde se requiere un conjunto fijo de subtipos.

101. Uso de anotaciones y reflección: Las anotaciones se pueden utilizar en combinación con la reflección para obtener información sobre clases y objetos en tiempo de ejecución.

102. Uso de corrutinas (coroutines) para tareas asíncronas: Las corrutinas permiten realizar operaciones asíncronas de manera más eficiente y legible que los hilos tradicionales.

103. Delegados personalizados: Los delegados personalizados

son objetos que manejan la lógica de acceso y modificación de propiedades.

104. Proyecciones de tipos genéricos (Type projections): Las proyecciones de tipos genéricos son útiles para trabajar con tipos genéricos de manera flexible.

105. Uso de expresiones regulares (Regex): Ya mencionamos las expresiones regulares, que son patrones de búsqueda utilizados para la manipulación de texto.

106. Infix functions: Las funciones infix permiten escribir llamadas de función en notación infija, lo que mejora la legibilidad.

107. Uso de destructuring declarations: Las declaraciones de desestructuración te permiten asignar componentes de objetos en múltiples variables.

108. Concurrent collections: Estas son colecciones diseñadas para ser seguras en entornos multihilo, evitando problemas de concurrencia.

109. Programación funcional: La programación funcional se centra en el uso de funciones puras y evita el estado mutable, lo que puede facilitar la escritura de código robusto.

110. Uso de data classes y copy: Las data classes son útiles para crear clases que se utilizan principalmente para almacenar datos. La función copy permite crear copias con algunos campos modificados.

111. Uso de rangos (Ranges): Ya mencionamos los rangos, que representan secuencias de valores en un intervalo.

112. Patrón delegado por propiedad (Property Delegation): Este patrón te permite externalizar la lógica de acceso y modificación de una propiedad a una clase delegada.

113. Uso de extension functions (Funciones de extensión) en

librerías externas: Puedes agregar funciones de extensión a clases de librerías externas para extender su funcionalidad.

114. Copiar un archivo en otro: Este código copia el contenido de un archivo de origen en un archivo de destino.

115. Lectura de un archivo de texto: Este código lee y muestra el contenido de un archivo de texto.

116. Escritura en un archivo de texto: Este código escribe contenido en un nuevo archivo de texto.

117. Copiar un archivo de forma simple: Este código copia el contenido de un archivo de origen en un archivo de destino utilizando una función simple.

118. Eliminar un archivo: Este código elimina un archivo si existe.

119. Listar archivos en un directorio: Este código lista los archivos en un directorio dado.

120. Creación de un directorio: Este código crea un nuevo directorio en la ruta especificada.

121. Generación de hash (SHA-256): Puedes generar un hash de un valor utilizando el algoritmo SHA-256.

122. Cifrado simétrico (AES): El cifrado simétrico AES (Advanced Encryption Standard) se utiliza para cifrar y descifrar datos de forma segura.

123. Cifrado César: El cifrado César es un cifrado de sustitución simple que desplaza cada letra en un número fijo de posiciones.

124. Firma digital (RSA): El algoritmo RSA se utiliza para firmar digitalmente datos y verificar la autenticidad de la fuente.

125. Cifrado asimétrico (RSA): El cifrado asimétrico RSA se

utiliza para cifrar y descifrar datos de manera segura utilizando una clave pública y privada.

126. Generación de contraseñas seguras: Puedes generar contraseñas seguras que sean difíciles de adivinar.

127. Cifrado simétrico con AES y modo de operación CBC: El modo de operación CBC (Cipher Block Chaining) es una técnica utilizada junto con el cifrado simétrico AES para garantizar la confidencialidad de los datos.