Conhecimentos Básicos - Teóricos em Kotlin

Diferenciais

- Não precisa especificar tipo de variável, onde ele é determinado pelo contexto;
- Linguagem tipada;
- Utiliza paradigmas Funcional e Orientado a Objetos;
- Funções de primeira classe, são tratadas como variáveis;
- Imutabilidade, uma vez definido o estado de um objeto, ele não pode mais ser alterado;
- Diferenças entre “val” (define um tipo imutável) e “var” (define um valor mutável que pode ser alterado);
- Benefícios do paradigma funcional (linguagem funcional, funções puras e concisa, podem ser testadas de forma fácil e isolada).

Expressão vs Declaração

- Expressão tem valor próprio
  Por exemplo em uma condição IF é uma expressão:

```
fun max(a: Int, b: Int): Int {
        return if (a>b) a else b
}
```

- A maioria das estruturas de controle são expressões;
- Exceto loops(for, do, do/while).

Corpo de bloco vs. Corpo de Expressão

- Corpo de bloco é quando uma função é escrita com seu corpo dentro de chaves{};
- Corpo de expressão é quando uma função consiste de apenas uma expressão(valor).

Estrutura de uma variável

- É declarada com a palavra chave (var/val);
- val representa um valor imutável, ou seja, não pode ser modificado;
- var representa uma variável mutável, podendo ser alterado.

Formatação de Strings

- Possibilidade de referenciar variáveis locais em strings com o $ antes do nome da variável;
- Utilizada em concatenações;
- Podem ser utilizadas com expressões.

Um pouco sobre orientação a objetos

- É um paradigma de programação baseado no conceito de objetos;
- Objetos podem conter dados(atributos) e procedimentos(métodos) e são elementos fundamentais na construção da solução;
- Objetos são instâncias de classes;
- Um objeto é uma abstração de algum fato ou ente do mundo real.

Classes

- Descrição que abstrai um conjunto de objetos com características similares;
- Descreve as propriedades/estados (atributos) possíveis e as ações/comportamentos (métodos) de um conjunto de objetos.

Propriedades em Java

- A ideia de uma classe é encapsular dados e códigos que trabalham sobre esses dados dentro de uma única entidade;
- Em java, o dado é armazenado em campos, que são usualmente privados;
- Se você necessita deixar o cliente da classe acessar esse dado, você fornece métodos assessores: getter e setter;
- Em java, a combinação de campo e seu assessor é frequentemente referenciado como uma propriedade.

Propriedade em Kotlin

- Funcionalidade de linguagem de primeira classe;
- Inteiramente substituir campos e métodos assessores.

Assessores Customizados

- Em Kotlin podemos escrever assessores de propriedades customizados;

```
class Retangulo(val altura: Int, val largura: Int) {
        val isQuadrado: Boolean
                get() {
                        return altura == largura
                }
}
```

- isQuadrado não necessita de um campo para guardar seu valor;
- Somente tem um getter customizado com a implementação fornecida;
- O valor é computado toda vez que a propriedade é acessada.

Organização: Pacotes e Diretórios

- Kotlin também tem o conceito de pacotes, similar ao Java;
- Todo arquivo Kotlin pode ter uma declaração de pacote no início;
- Todas as declarações(classes, funções e propriedades) definidas no arquivo vão ser colocadas nesse pacote;
- Declarações definidas em outros arquivos podem ser usados diretamente se eles estão no mesmo pacote, eles necessitam ser importados se eles estão em um pacote diferente;
- Kotlin não faz distinção entre importação de classes e funções(top level);
- Em Java as classes devem seguir uma hierarquia de diretórios que casam com a estrutura de pacotes;
- Em Kotlin, você pode colocar várias classes no mesmo arquivo e escolher qualquer nome para esse arquivo;
- Kotlin também não impõe qualquer restrição sobre o layout de arquivos fontes no disco;
- Na maioria dos casos, contudo, ainda é uma boa prática seguir o layout de diretórios do Java para organizar arquivos fontes dentro de diretórios de acordo com a estrutura de pacotes.

Entendendo Enums e When

```
enum class Cor {
        VERMELHO, LARANJA, AMARELO, VERDE, AZUL
}
```

- Esse é um caso raro onde uma declaração em Kotlin usa mais palavras chave do que o correspondente em Java enum class vs somente enum;
- Em Kotlin, enum é chamado de soft-keyword;
- Significado especial quando ela vem antes de class, mas podemos usá-la como um nome regular em outros lugares (val enum = “abc”);
- Assim como em Java, enums não são lista de valores: podemos declarar propriedades e métodos em classes Enum;
- Enum é uma forma poderosa de escrever grupo de constantes.

Construção “when” e enums em Kotlin

- when é um substituto do switch do Java;
- Porém mais poderoso e é usado mais frequentemente;
- when é uma expressão que retorna um valor (corpo de expressão), assim como if;
- Não precisamos escrever afirmações break em cada branch, como em Java;
- Podemos também combinar múltiplos valores no mesmo branch separando por vírgulas.

Entendendo Smart Cast

- A palavra chave is é equivalente ao instanceof no Java (Verificação de tipo de uma variável);
- Em Java, é sempre preciso fazer um cast explicito após a verificação de um tipo com instanceof;
- No Kotlin, o compilador faz esse trabalho para nós;
- Se uma verificação foi feita para um certo tipo, não é preciso fazer um cast explícito depois da verificação;
- O cast feito automaticamente pelo compilador é chamado de Smart Cast.

Blocos como ramificações de “if” e “when”

- Ambos, if e when podem ter blocos como ramificações;
- A última expressão do bloco é o resultado;
- A afirmação anterior é verdadeira em todos os casos onde um bloco pode ser usado e um resultado é esperado;
- Essa regra só não é verdadeira para funções regulares, que possuem ou um corpo de expressão que não pode ser um bloco, ou um corpo de bloco com um retorno explícito dentro (return).

Laço de Repetição while

- De todas as funcionalidades vistas até agora, iteração em Kotlin é provavelmente a mais similar ao Java;
- O laço while é idêntico ao Java;
- O laço for existe em uma única forma, que é equivalente ao laço for-each no Java;
- No Kotlin existem laços while e do-while, que não diferem dos seus correspondentes laços em Java.

Laço de Repetição for

- Em Kotlin não existe o laço for padrão do Java: (for i=0;i<10;i++){};
- Kotlin usa o conceito de ranges;
- Um range é essencialmente um intervalo entre dois valores, usualmente números: um início e um fim;
- Nós representamos isso com o uso do operador ..(1..10);
- Ranges em Kotlin são fechados ou inclusivos, significando que o segundo valor é sempre parte do intervalo;
- Em muitos casos, é mais conveniente iterar sobre intervalos semi-fechados, que não inclui o valor final especificado.

Operador IN

- Como já vimos, utilizamos o operador **in** para iterar sobre intervalos;
- Mas também podemos utilizar o operador **in** para verificar se um valor está presente em um intervalo de valores;
- Podemos verificar também se um valor não está presente em um intervalo de valores utilizando o operador com a negação (**!in**).

Lançando uma Exceção

- Exceções em Kotlin é similar em Java ou qualquer outra linguagem;
- Uma função pode completar de forma correta ou lançar uma exceção se um erro ocorrer;
- O chamado da função pode pegar e tratar a exceção ou deixar que alguém mais adiante na pilha pegue e trate da exceção;
- Para lançar uma exceção no Kotlin utilizamos a palavra reservada **throw**;
- Em Kotlin, diferente do Java, a construção **throw** é uma expressão que pode ser usada como parte de outra expressão.

Entendendo e Criando coleções em Kotlin

- Kotlin não possui seu próprio conjunto de coleções (List, Map, Set, etc);
- Kotlin usa as classes de coleção Java padrão;
- Exemplo de criação de uma coleção simples:
  Ex:
  val frutas = arrayListOf(“maça”,“banana”,”morango”)
- Apesar do Kotlin não ter seu próprio conjunto de coleções, é possível fazer muito mais com coleções na linguagem Kotlin.