Esta documento tem somente o intuito de auxiliar no aprendizado da linguagem Kotlin.

Pré-requisito

Neste curso, assumimos que você conheça o conteúdo apresentado no curso de primeiros passos com o Java. Os principais pontos são:

Máquina virtual e a JVM;

Variáveis e seus tipos;

Tenha o JDK instalado.

Esse é um curso classificado como básico, logo, a maioria dos exercícios terão o passo a passo de implementação do que será visto em aula.

Preparando o Ambiente

Antes de instalar o IntelliJ IDEA, tenha certeza que o JDK está instalado na sua máquina. Caso não tenha e não saiba como instalar, confira esta aula do curso de primeiros passos de Java que apresenta as instruções necessárias. Neste curso, utilizamos a versão 12 do JDK.

Porém, recomendamos que utilize o Toolbox, uma ferramenta da Jetbrains que baixa e instala a IDE automaticamente.

Lembre-se que existem duas versões:

Community: versão gratuita que usaremos durante o curso;

Ultimate: versão paga que tem os mesmos recursos da Community com plugins e suporte a mais.

Se preferir, você pode baixar o IntelliJ IDEA diretamente pelo site da Jetbrains.

Após finalizar a instalação, uma página de wizard, janela com etapas, será apresentada, finalize as etapas até apresentar a seguinte janela ou similar:

Criando o projeto Kotlin

Com o IntelliJ pronto, crie o projeto um projeto clicando em Create New Project.

Em seguida, no menu à esquerda, clique em Kotlin, então escolha a opção à direita JVM | IDEA e clique em Next.

Vai abrir o formulário para preencher os campos para nome, local do projeto e o SDK:nome: insira o nome como bytebank;

local do projeto: preencha com um local onde costuma deixar seus projetos (não há problema se quiser manter o padrão apresentado pelo IntelliJ);

SDK: selecione o JDK instalado no seu computador.

Nessa etapa é comum apresentar problemas na seleção do JDK, certifique-se que a instalação está correta.

Se estiver mas não aparecer, é necessário buscar o JDK no local onde foi instalado.

Após preencher todos os campos clique em Finish. Aguarde o IntelliJ configurar e indexar os arquivos.

Ao finalizar, abra a aba project e dentro do diretório src, clique com o botão direito do mouse e vai escolha a opção New > Kotlin File/Class.

Mantenha a opção File selecione e dê o nome de main.

Em seguida crie a função main() e, dentro de seu escopo, chame a função print() enviando a mensagem de boas vindas.

Em vídeo utilizamos a "Bem vindo ao Bytebank", mas fique à vontade para enviar a mensagem que preferir.

Execute a função main() e veja se apresenta o console com a mensagem esperada.

Criando o titular para a conta

De acordo com a proposta de implementação enviada pelo cliente. Crie uma variável que receba o nome do titular da conta e faça a impressão. Nesta impressão, utilize o println() para pular as linhas.

Você pode definir essa variável tanto como mutável (var) como imutável (val).

Para a impressão, considere o uso do String template, que recebe variáveis dentro de uma String.

Adicionando as demais variáveis

Implemente as variáveis para representar o número e saldo da conta. Para o número da conta, considerando que não muda o valor, utilize uma variável imutável, para o saldo, considerando que pode aumentar ou diminuir, utilize uma variável mutável.

Em seguida, faça alguns ajustes na variável do saldo para que receba um novo valor e depois uma soma. Por fim, imprima as novas variáveis para que todas as informações da conta sejam apresentadas.

Testando if e when

Caso você precise do projeto com todas as alterações realizadas na aula passada, você pode baixá-lo por meio deste link.

Agora que conhecemos as possibilidades para controle de fluxo do Kotlin, teste as estruturas condicionais do Kotlin, verificando se uma conta é positiva, neutra ou negativa em relação ao valor do saldo.

Para esse exercício, use a implementação com if, else if e else, e então faça a mesma verificação com o when.

Lembre-se que o plugin do Kotlin para o IntelliJ verifica as possíveis conversões. Para isso, utilize o atalho Alt + Enter quando aparecer o código sublinhado.

Após implementar o código, ajuste o valor do saldo para testar todas as condições e confira se funciona como esperado.

Testando o for loop e while

Continuando com as possibilidades de controle de fluxo, teste as estruturas de repetição for loop e while.

Para o teste, crie contas e faça a impressão das mesmas com ambas abordagens.

Para o for, testes as situações que tem o range de

1 até 5,

5 até 1,

5 até 1 de 2 em 2.

Por fim, utilize o while com o índice menor do que 5.

Lembre-se que o while precisa ajustar o contador manualmente para não resultar em um loop infinito.

Para facilitar os testes, extraia o código do if e when em uma função com o nome testaCondicoes(). Então faça os devidos teste e confira se tudo funciona como o esperado.

Vimos que o Kotlin oferece suporte para o break e continue dentro de laços, dessa forma, somos capazes de controlar até mesmo as iterações do for loop.

Além do controle básico, podemos também utilizar break ou continue dentro de loops aninhados:

```
loop@ for (i in 1..100) {
   println("i $i")
   for (j in 1..100) {
      println("j $j")
      if (j == 5) break@loop
   }
}
```

Neste trecho de código, temos o seguinte resultado:

```
i 1
j 1
j 2
j 3
j 4
j 5
```

Em muitas linguagens de programação escrevemos ; no final da linha de código para indicar que finalizou o trecho, como por exemplo:

```
var i = 0;
while(i < 5){
   val titular: String = "Alex $i";
   val numeroConta: Int = 1000 + i;
   var saldo = i + 10.0;

   println("titular $titular");
   println("número da conta $numeroConta");
   println("saldo da conta $saldo");
   println();
   i++;
}
```

O código compila e roda da mesma maneira, mas é totalmente opcional, isso significa que você pode escrever com ; mas entenda que é um código inesperado e evitado pela comunidade.

Criando a classe Conta

Caso você precise do projeto com todas as alterações realizadas na aula passada, você pode baixá-lo por meio deste link.

Antes de começar a atividade, extraia o código de teste para uma função, conforme demonstrado em vídeo.

Após extrair o código, crie a classe para representar a conta, utilizando a palavra reservada class, seguida do nome Conta.

Lembre-se que o padrão para escrever nomes de classes segue o camelCase.

Em seguida, crie as variáveis para a classe que vai receber os valores para:

titular,número de conta,saldo.

Na sequência, o Kotlin vai solicitar a inicialização. Faça a mesma inicialização, conforme as variáveis que foram criadas na função main().

Então, crie 2 objetos do tipo Conta() e os atribua para uma variável que vai representar contas distintas. Em vídeo, foram as contas do Alex e da Fran, mas você pode testar com o seu nome e de outra pessoa que queira.

Modifique as variáveis de cada objeto para que tenha um número de conta e saldo diferentes, por fim, faça a impressão das variáveis de cada objeto.

Durante a aula, foi mencionado que uma das regras para criar a classe é escrevê-la em nível de arquivo. Porém, isso não significa que não é possível criá-la no escopo de funções, ou até mesmo em outras classes.

Em outras palavras, é possível criar a nossa classe dentro de funções, ou criar uma outra classe dentro da nossa, conforme os exemplos abaixo:

```
fun main() {

   println("Bem vindo ao Bytebank")

   class Conta {

      var titular = ""

      var numero = 0

      var saldo = 0.0

   }

   val contaAlex = Conta()

   // restante do código

}
```

Por mais que exista a possibilidade de criar dentro de funções, dessa forma não é possível criar objetos do tipo Conta em outras partes do programa, ou seja, perdemos a reutilização de código.

```
class Conta{

   val titular = ""

   class OutraClasse {

      var variavelQualquer = 10

   }

}
```

Mesmo que exista a possibilidade, criar uma classe dentro de outra é uma solução bem vinda para casos específicos. Portanto, se pretende criar uma nova classe, muito provavelmente será em nível de arquivo.

Testando cópia e referência

Modifique o código da função main() para testar os comportamentos de atribuição de variáveis.

Crie duas variáveis de tipos de dados primitivos, como Int ou Double. Então, inicialize a segunda com o valor da primeira e teste se, ajustar a segunda variável é mantido o comportamento de cópia.

Faça o mesmo para duas variáveis do tipo Conta. A diferença é que o ajuste é feito a partir da variável titular do objeto. Confira se o comportamento de referência é mantido.

Implementando a função de depósito

Implemente a funcionalidade para depositar na conta.

Antes de implementar o código, extraia todo o código que testa a cópia e referência para uma função. Também remova os comentários das impressões que apresentavam as informações das contas criadas.

Em seguida, crie a função deposita() — que recebe um parâmetro do tipo Conta — e um valor do tipo Double. Dentro da função, modifique o saldo da conta recebida para somar o valor recebido.

Utilize a função para as contas que foram criadas e confira se funciona, imprimindo o saldo após chamá-la.

Criando métodos da classe

Caso você precise do projeto com todas as alterações realizadas na aula passada, você pode baixá-lo por meio deste link.

Ajuste o código para que a própria classe mantenha os comportamentos dentro do seu escopo.

Para isso, migre a função deposita() para que seja uma função membro de Conta, tornando-a um método ou comportamento da classe.

Em seguida, modifique o código para que o método deposita() deixe de receber uma conta e receba apenas o valor como parâmetro. Então, modifique o atributo saldo diretamente.

Você pode usar this para deixar claro que o saldo modificado é referente ao saldo do objeto.

Na sequência, ajuste a chamada para que cada objeto chame o deposita() e envie apenas o valor. Teste o comportamento de depósito e confira se funciona como antes.

Em seguida, implemente o método de saque — saca() —, aplique as mesmas técnicas no deposita(). A diferença é que a regra de negócio de saque permite apenas que a subtração do valor seja feita se o valor for menor ou igual ao saldo. Adicione essa lógica e teste o saque.

Transferindo de uma conta para outra

Implemente o método de transferência. Utilize as mesmas técnicas nos métodos de depósito ou saque.

Nesta implementação, é necessário: o uso da conta origem, valor a ser transferido, conta destino.

Considerando que é um método, a conta de origem é quem chama o método, portanto, o método deve receber apenas o valor e a conta destino.

A lógica para transferir é a mesma de saque. A diferença é que na transferência, após subtrair o saldo, deve somar o valor recebido ao saldo da conta destino.

Após aplicar a lógica, ajuste o método para que devolva um valor do tipo Boolean, que indica sucesso ou falha. Retorne true no final da instrução if e false no final do método transferir.

Por fim, teste o novo comportamento da conta. Neste teste, faça a impressão de uma mensagem de sucesso ou falha ao executar a transferência. Então, confira o saldo de ambas as contas e verifique se a transferência ocorreu como esperado: tanto quando tem saldo suficiente ou quando não tem.

Criando getter e setter do saldo

Modifique o código da classe Conta para que o saldo seja privado e acessado por métodos de acesso setter e getter.

No setter, adicione o filtro para que o saldo seja inserido apenas com valor positivo. No getter, devolva apenas o valor do saldo.

Ajuste o código inicial para que utilizem os métodos de acesso e rode o programa verificando se funciona como esperado.

Mais sobre modificadores de acesso: Durante o encapsulamento do saldo, utilizamos o modificador de acesso privado que é o mais restrito de todos.

Porém, existem outros modificadores de acesso disponíveis no Kotlin. Caso tenha interesse em conhecê-los, confira a documentação[1].

É importante ressaltar que durante esse curso não exploraremos os demais modificadores, pois existem outros assuntos de base que precisam ser abordados antes para esclarecer a utilidade dos demais.

Na documentação[1], eles são identificados como Visibility Modifiers, que na tradução seria "Modificadores de Visibilidade". Não se assuste, pois o objetivo é o mesmo.

Utilizando getter e setter da property

Escreva um código mais idiomático com o Kotlin utilizando property ao invés de métodos de acesso para o saldo.

Para isso, remova o getter e setter que foi implementado para o saldo, em seguida, ajuste o código para que utilize a property saldo.

Após ajuste, adicione a lógica que impede valores negativos via método deposita(), então deixe o setter do saldo privado para que apenas a própria classe altere o seu valor diretamente.

Por fim, modifique o código para que adicione saldo via deposita() e leia o saldo via property. Então remova os comentários dos códigos anteriores, rode o programa e confira se funciona como esperado.

Mais detalhes de properties: Aprendemos a usar properties e exploramos o uso básico, porém esse recurso oferece bastante recurso e alternativas de implementação.

Para mais detalhes de outros recursos, confira a documentação[2].

Criando contas com construtor personalizado

Modifique o código da classe Conta para que exija o envio de titular e número de conta.

Para isso, adicione um construtor primário que recebe titular e número como properties. Em seguida, modifique todas as instâncias de Conta para que envie titular e número de conta via construtor.

Na sequência, remova os códigos de atribuição de titular e número da conta, dentro da função main().

Rode o programa e confira se funciona como esperado.

Aprendemos que existem 2 construtores no Kotlin: o primário e o secundário. Em situações que desejamos executar trechos de código a mais, o construtor secundário é mais interessante. Caso seja só inicialização, o construtor primário é o esperado.

Entretanto, também temos a possibilidade de usar o construtor primário e executar trechos de código durante a inicialização. Para isso, utilizamos o bloco init:

```
class Conta(

   var titular: String,

   var numero: Int

) {

   var saldo = 0.0

      private set

   init {

    //Executa alguma coisa durante a construção.

   }

}
```

Existem outros sobre construtores que serão discutidos em cursos futuros, porém, caso tenha interesse em se aprofundar, confira a documentação[3].

Enviando argumentos nomeados

Modifique o código para que a property numero seja val. Então, ao construir os objetos, utilize as labels para identificar o que argumento representa, por exemplo, titular = "Alex" para enviar o titular.

Teste ordem diferentes no envio dos argumentos com as labels, rode o programa e confira se tudo funciona como esperado. Ao enviar as labels como argumento, tecnicamente utilizamos os argumentos nomeados, que na documentação[3] encontramos como Named Arguments. Label é uma maneira informal de chamar os nomes dos argumentos.

https://kotlinlang.org/docs/visibility-modifiers.html ([1])

https://kotlinlang.org/docs/properties.html (2)

https://kotlinlang.org/docs/classes.html#constructors (3)