ORIENTAÇÃO À OBJETOS EM KOTLIN

Instruções para a prova: *Anote em uma folha de papel (tamanho máximo: A4) tudo aquilo que você achar necessário para a realização da prova. Essa folha é pessoal e intransferível. A folha NÃO pode ser impressa.*

KOTLIN, como outras linguagens de programação orientadas a objetos, usa classes e objetos como base para criar e gerenciar estruturas de dados complexas.

Classe: Uma classe é um molde (ou forma) para a criação de objetos. Ela define propriedades e métodos que os objetos criados irão possuir a partir da classe instanciada.

Objeto: Um objeto é uma instância (em outras palavras, criação, produção ou forma) de uma classe. Ele é criado a partir de uma classe e pode usar as propriedades e métodos definidos na classe.

Encapsulamento: É a técnica restringir o acesso e exigir de todas as interações, do qual agora devem ser realizadas por meio dos métodos de um objeto. Isso é tipicamente alcançado tornando os campos privados e fornecendo métodos *getter* e *setter* públicos. O encapsulamento ajuda a proteger os dados e a melhorar a modularidade e a manutenibilidade do código.

O encapsulamento tem um propósito principal: *proteger a integridade do objeto*. Ao restringir o acesso direto ao estado interno, o encapsulamento garante que o objeto permaneça em um estado válido. Isso significa que quaisquer alterações nos dados do objeto podem ser controladas e validadas por meio de métodos, garantindo que o objeto não entre em um estado inconsistente ou inválido.

Por padrão, em Kotlin, as propriedades de uma classe têm métodos *getter* e *setter*, ou seja, você não precisa gerar esses métodos de acesso. No entanto, você pode personalizar esses métodos se necessário.

Herança: É através desse método que podemos gerar novos comportamento em um código existente, promovendo a reutilização. Nesse processo a classe existente é chamada de superclasse e a classe a ser criada se chama subclasse. Uma subclasse herda todos os membros presentes na superclasse por padrão. Quando um conjunto de classes, geralmente concretas, exigem uma superclasse, dizemos que as características que elas compartilham serão combinadas em uma superclasse generalizada.

Toda *classe em Kotlin* possui um supertipo chamado *Any*, que possui apenas um pequeno número de funções, as quais são *equals()*, *hashCode()* e *toString()*. Vale lembrar que o *Kotlin* não suporta herança múltipla, motivo pelo qual apenas uma classe pode ser incluída na lista de supertipos de uma outra classe.

Polimorfismo: É um conceito que permite reutilizar, sobrescrever ou sobrecarregar os comportamentos (métodos) das superclasses ou interfaces, do qual normalmente acontece através da herança.

Em outras palavras o polimorfismo (poli significa muitos, e morfo significa mudar de forma) faz com que os métodos de uma classe tenham seus comportamentos alterados em outras classes do qual tenham herdado tais comportamentos.

Exercícios práticos

1- Crie um algoritmo em Kotlin que seja capaz de somar dois números. Os números devem ser informados pelo usuário, a resposta deve ser apresenta por uma função *print*.
2- Crie um algoritmo em Kotlin que possa ser capaz de armazenar nomes de pessoas em lista, pode usar um *MutableList* ou simplesmente um *List*. A tipagem da lista deve ser *String* (obviamente), o usuário deve ser o responsável por preencher a lista. Faça a validação para que o usuário digite apenas as letras do alfabeto, a recomendação é usar expressão regular. Depois que o usuário terminou de preencher a lista imprima todos os nomes na tela por ordem alfabética, use o método *sort()*. Mais informações em: https://kotlinlang.org/docs/collection-ordering.html ou https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.collections/sort.html.
3- Crie um algoritmo em Kotlin com um menu em loop contendo as seguintes opções:
    a. Somar vários números reais. Os números e a quantidades de números são informados pelo usuário.
    b. Gerar um número aleatório entre o intervalo de *x* e *y*. O valor de x e y são informados pelo usuário. Use um **range** com a função *random()*. Mais informações: https://kotlinlang.org/docs/ranges.html#progression e https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.collections/sort.html
    c. Faça a leitura do teclado de números inteiros positivos e negativos e armazene eles em lista separadas e depois ordene.
    d. Faça uma opção para sair do menu e encerrar a aplicação.
4- Faça um algoritmo capaz de contar a repetição de palavras de um determinado texto. O texto e a palavra a ser buscada são informadas pelo usuário. Mostre o valor da repetição na tela. Dica use a função *substring*(). Mais informações em:

https://www.baeldung.com/kotlin/check-substring ou https://kotlinlang.org/api/core/kotlin-stdlib/kotlin.text/substring.html.

5- Faça um jogo de pedra, papel e tesoura em Kotlin (no estilo PC vs Player, computador contra jogador). O jogo deve respeitar alguns critérios:
    a. Faça um menu com: jogar, placar e sair.
    b. Crie uma opção para que o jogador possa criar o *nickname*.
    c. Cada partida deve ser composta por 3 rodadas.
    d. Evite ao máximo o uso da condicional **if**. Caso necessário use Expressões Regulares. Dica use classes.
    e. Seja criativo!
6- Crie um algoritmo em Kotlin que possui uma lista de frases motivacionais, para cada hora do dia imprima uma mensagem da lista na tela. Dica use a biblioteca **Calendar** do Java. Mais informações em: https://www.baeldung.com/kotlin/current-date-time.
7- Crie um algoritmo em Kotlin que seja capaz de mostrar preços de rações para animais.
    a. Crie uma **classe** *Ração* com os seguintes **atributos**: *preço*, *peso* e *sabor*, depois crie um **método** chamado de *desconto()*, esse método deve alterar o preço da ração em 10%.
    b. Depois crie duas **subclasses** *RaçãoDeGato* e *RaçãoDeCachorro*. A ração para gato deve ter os seguintes **atributos**: *paraCastrado*, *ricaEmFerro*. Já a ração de cachorro deve ter os seguintes **atributos**: *fortalecimentoDePelos*, pedacos*ComCarne*. Todos os atributos listados aqui devem ser do tipo **Booleano**.
    c. Faça a **sobrescrita** (polimorfismo) das duas subclasses. No caso da ração de gato muda o desconto para 15% se a ração for para castrado ou para 20% se a ração não for para gato castrado. No caso da ração de cachorro mude o desconto para 5% se a ração conter pedaços de carne, ou para 25% se a ração não conter pedaços de carne.
8- Crie um algoritmo em Kotlin que armazene informações de restaurantes e a nota dos usuários.
    a. Faça uma classe ***Restaurante*** deve ter: nome, endereço, especialidade e nota do tipo ***Notas*** (ver abaixo) esse atributo é responsável por conter a média de notas de todos os usuários.
    b. Crie uma classe ***Notas***, essa classe deve possuir um array de notas.
    c. Crie também um método do qual altera a nota do restaurante de acordo com as notas dadas por cada usuário (em outras palavras esse atributo faz a média das notas).
    d. Crie uma classe ***Cliente***, essa classe deve possuir um método para avaliar o restaurante, alterando o atributo nota do restaurante.

e. Faça a instância de vários clientes e avalie um restaurante. Verifique se o seu algoritmo foi capaz de armazenar a média corretamente no restaurante especificado.

9- **Desafio final.** Crie um jogo da forca usando seus conhecimentos em Kotlin.