RESUMO – PROGRAMAÇÃO ORIENTADA A OBJETOS

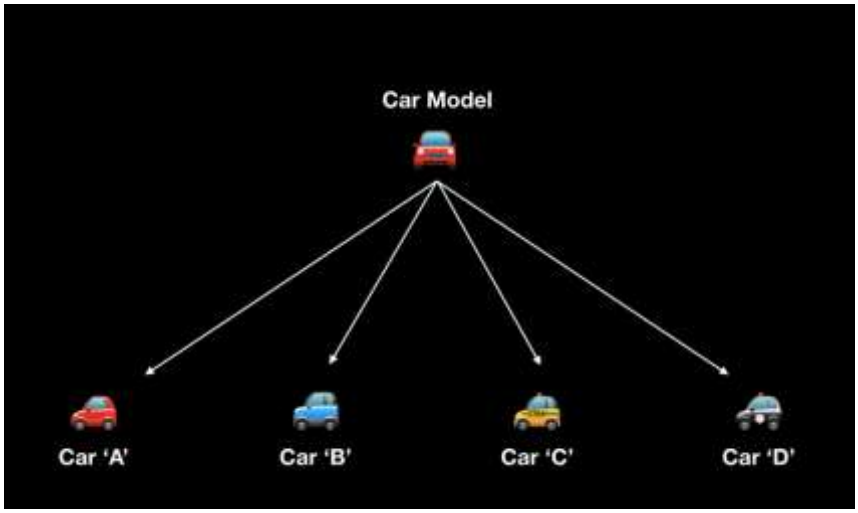
O que é um objeto?

Todo objeto pode conter características e ações que o definem, e isso também é passado para a programação. Entre eles estão:

Atributos: Características do objeto, como nome, idade, endereço.

Métodos: Ações de um objeto, como caminhar, escrever, nadar.

Devemos criar um objeto para cada coisa que abstraímos do mundo real, criando os mesmos atributos e métodos para cada um, reescrevendo? Não! Porque nós temos a classe!



O que é uma classe?

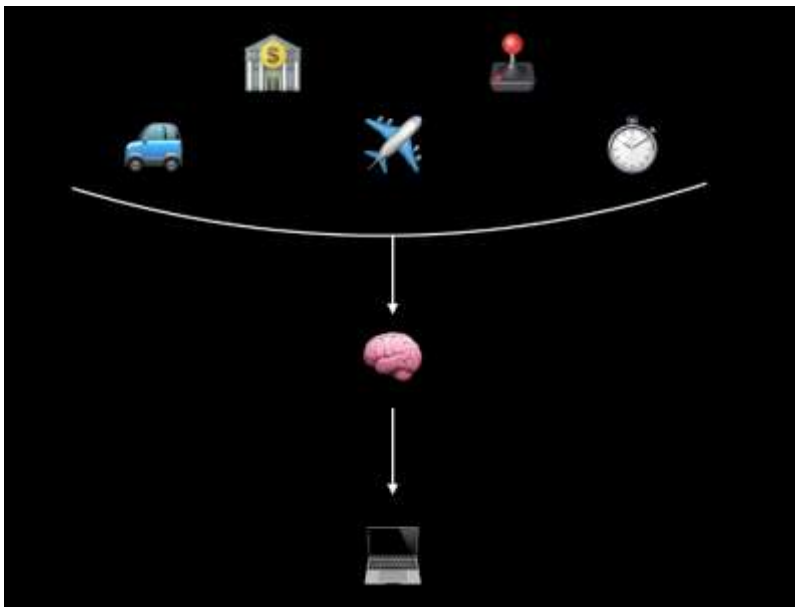
Classes são objetos do mundo real, de maneira “abstrata”, permitindo que um objeto o use com seus atributos e métodos. Classes servem como moldes para objetos, dizemos que são “instâncias” de classes, e a Classe é o modelo para a criação de objetos.

https://kotlinlang.org/docs/classes.html

https://kotlinlang.org/docs/properties.html

Quatro pilares da Orientação a Objetos

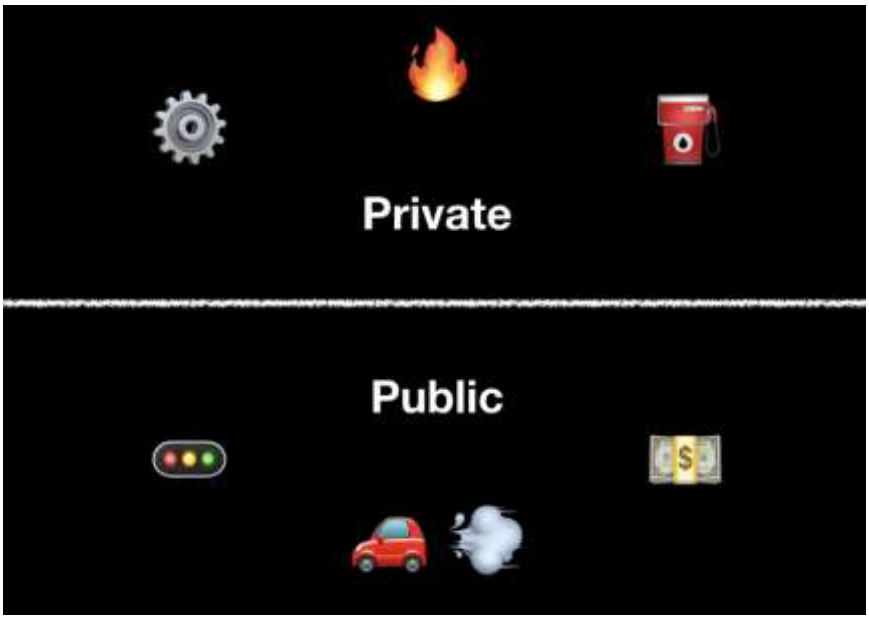
Abstração, encapsulamento, herança e polimorfismo.



Abstração

Habilidade de trazer informações relevantes do mundo real para o código e, se você fizer isso errado, poderá adicionar muita informação ou ficar confuso: se não fizer sentido no mundo real, não fará sentido no código

---



Encapsulamento

Encapsulamento é o pilar que protege o código, dizendo o que pode ser acessado por outros objetos, isto é, a visibilidade de atributos e métodos para todos. Os tipos de encapsulamento são:

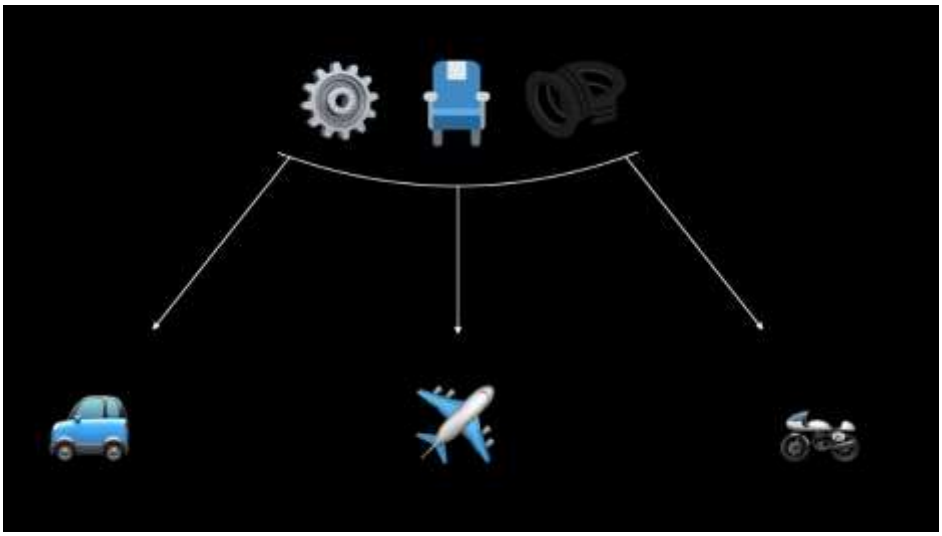
Público: Todo mundo tem acesso a esse atributo, função ou classe.

Privado: Somente a própria classe tem acesso ao atributo, função ou subclasse.

Protegido: Somente a própria classe ou classes herdadas (subclasses) dela que podem ser acessadas.

https://kotlinlang.org/docs/visibility-modifiers.html#local-declarations

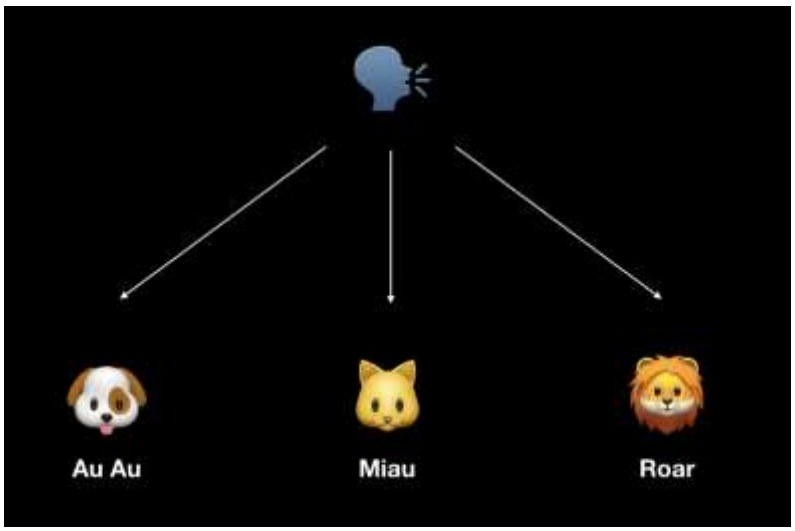
---



Herança

Quando todas as classes têm uma característica e comportamentos em comum, podemos herdá-la de uma superclasse.

https://kotlinlang.org/docs/inheritance.html



Polimorfismo

Quando as subclasses herdam de uma classe, mas a implementam de maneiras diferentes para um mesmo método.

https://kotlinlang.org/docs/inheritance.html#calling-the-superclass-implementation

---

LISTA DE EXERCÍCIOS – ENTREGA PARA 22/02

Orientações: Para cada novo exercício, crie novos pacotes (*packages*) por exemplo: “Exercicio1”, e teste todos os programas na classe principal (*main)*.

Os exercícios deverão ser entregues até a primeira aula após o recesso de Carnaval - dia 22/02 (segunda-feira) - para o e-mail de **sandyara.peres@gmail.com**, deverão subir os exercícios no GitHub de vocês e me enviar o *link* no corpo do e-mail e claro, se identifiquem.

Atentem-se a não resolver, o intuito é eu entender como é a abstração e a lógica de vocês e não ver o programa funcionando de fato, ou seja, evitem pegar o código-fonte do colega apenas para “conseguir nota”. Não se sabotem!

Aproveitem esse tempo e façam grupos de estudos, reúnam-se e tentem solucionar os exercícios em conjunto. Após a entrega, irei corrigir e darei o *feedback* individual através do e-mail enviado.

Divirtam-se e bom feriado!

Sandyara Peres

1. Crie um programa em Kotlin que represente uma pessoa: essa pessoa terá atributos privados de nome, data de nascimento e altura. Com isso, crie métodos públicos para acessar essas propriedades bem como outros dois métodos para:
    a. Exibir todos os dados da pessoa em uma *String* só, por exemplo: “Ana nasceu em 04 de Janeiro, tem 38 anos e possui 1,72m de altura”.
    b. Um método para calcular a idade da pessoa.

   Para esse exercício, não é necessária interação do usuário, é permitido instanciar os objetos diretamente na classe para teste.

2. Crie um programa em Kotlin que tenha a função de ser uma agenda de contatos. Essa agenda pode armazenar somente 10 contatos e precisa realizar as seguintes operações:
    a. Salvar contato: ao inserir o nome e o telefone de uma pessoa e ela não constar na agenda, salva então o seu contato na agenda;
    b. Remover contato: ao inserir o nome e o telefone de uma pessoa e ela constar na agenda, seu contato é removido. Caso ela não exista, informar ao usuário;
    c. Buscar contato: ao inserir um nome, retornar seus dados. Se tiver mais de um resultado, exemplo, dois contatos de nome Juliana salvos com telefones diferentes, exibir ambos;
    d. Mostrar agenda completa com todos os contatos salvos.

3. Crie um programa em Kotlin que faça o papel de gestão de RH em uma empresa: nessa empresa, os principais funcionários são divididos nesse momento entre gestores e os programadores. Por essa empresa estar passando por uma época de reajuste salarial, para lidar com isso de forma automatizada você adequará seu *software* para realizar essa função, com isso seu *software* deve:
    a. Salvar os dados (como ID, nome, CPF e salário atual) de um funcionário e ao final perguntar se ele é gestor ou programador, e só assim, instanciar o objeto e retornar a mensagem “Funcionário salvo com sucesso”.
    b. Ao pesquisar um funcionário pelo ID, é dada a opção de exibir todos os seus dados ou aumentar o seu salário.
    c. Para gestores, o reajuste é de 10%, já para programadores, de 20%.