Herança

Site:	HackaTruck MakerSpace	Impresso por:	Matheus Ferreira Santos
Curso:	Conceitos e Fundamentos: Algoritmos e Programação Orientada a Objetos com Swift	Data:	terça-feira, 15 ago. 2023, 18:16
Livro:	Herança		

Índice

1. Herança

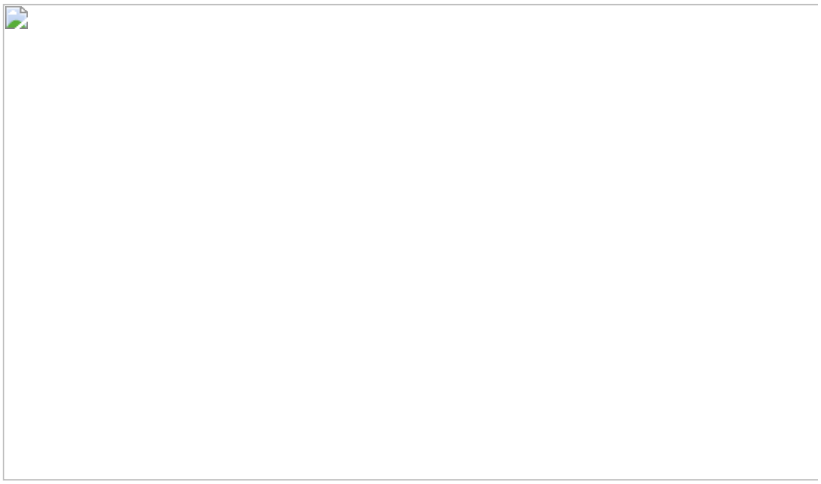
ORIENTAÇÃO A OBJETOS - HERANÇA

HERANÇA

Herança é um conceito muito importante em Orientação a Objetos (OO), pois permite uma melhor organização e reaproveitamento de código. Por meio desse conceito, as classes *filhas* compartilham os atributos e métodos da *classe mãe*.

Os apelidos “classe pai”, “classe mãe”, “supertipo”, “superclasse” e “classe base” são a mesma coisa, então não se assuste ao ver algum desses termos, pois todos se referem à classe original a ser herdada.

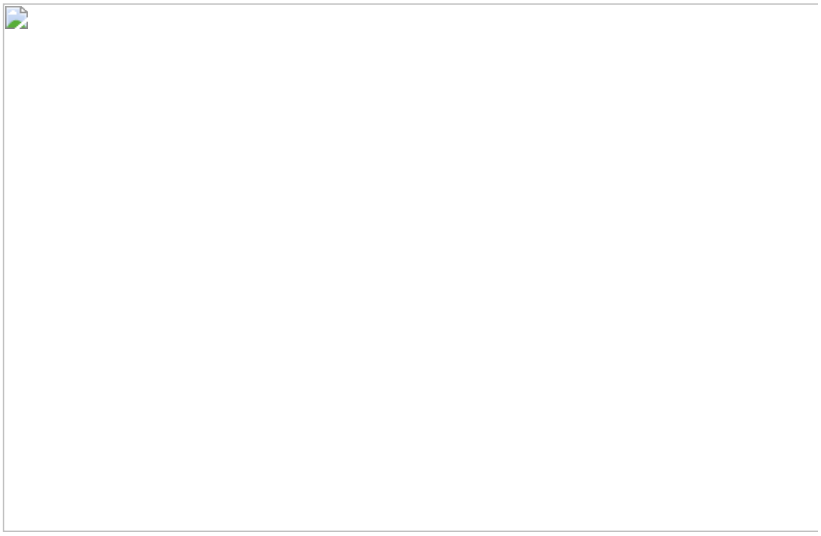
Já os termos “classe filha”, “subtipo”, “subclasse” e “classe derivada” também são sinônimos, e se referem às classes que herdam os atributos e métodos de uma superclasse.



Para dizer que uma classe herda o comportamento de outra, usamos a palavra **“estende”**. No exemplo abaixo veremos “Carro estende Veiculo”, onde a classe Carro é a subclasse e a classe Veiculo é a superclasse. Desse modo, Carro terá todos os atributos públicos da classe Veiculo, e também poderá utilizar seus métodos públicos (ou até mesmo modificá-los, se necessário, conforme veremos no capítulo de Polimorfismo). É justamente por isso que dizemos que as classes filhas herdam o comportamento das classes mães, e também podem acrescentar outras características (atributos) ou novas funcionalidades (novos métodos). É importante notar que herança também segue os princípios de visibilidade e encapsulamento, portanto, o que é privado em uma classe não será observado nas suas subclasses.

O conceito de herança está claramente relacionado com o conceito de **“é um(a)”**. No nosso exemplo anterior, Carro é um Veículo. A herança também se relaciona com os conceitos:

Generalização - Quando partimos de uma classe e chegamos à sua superclasse.

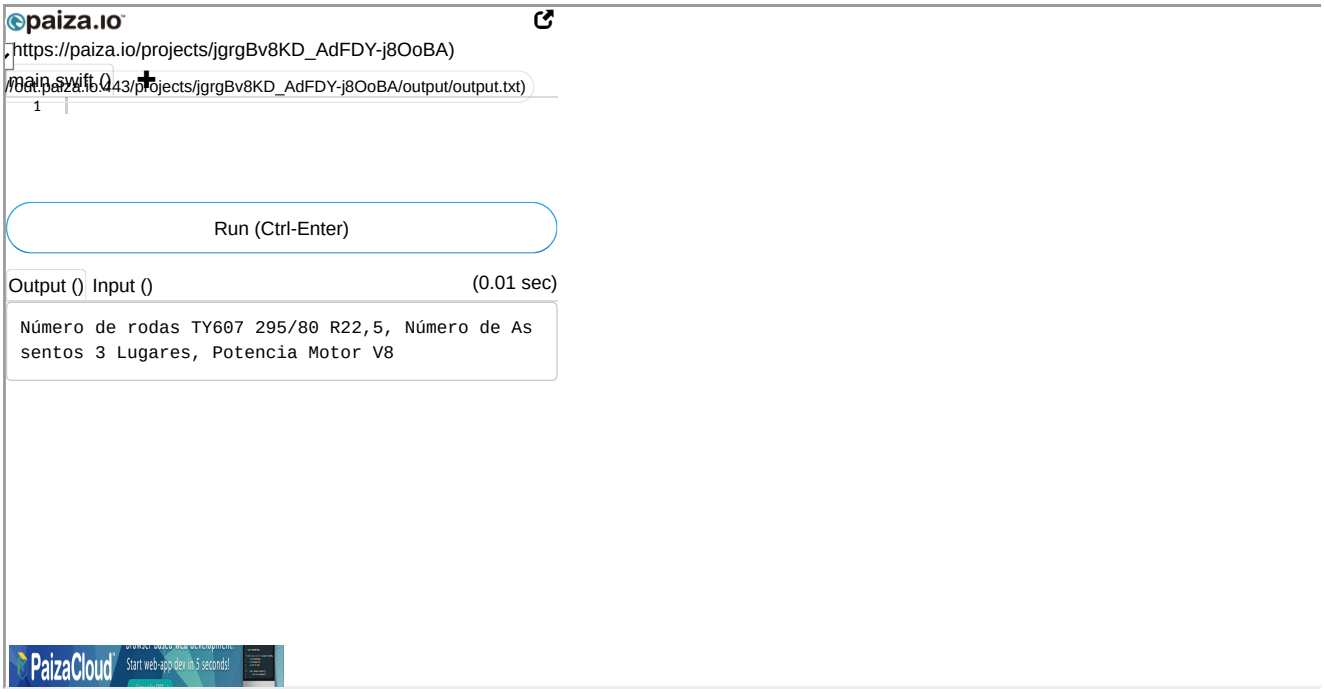


Especialização - Quando partimos de uma superclasse e chegamos na sua subclasse.

Assim, Carro é uma especialização de Veiculo e Veiculo é uma generalização de Carro. Do mesmo modo será com qualquer outro meio de transporte se definirmos que “MeioDeTransporte estende Veiculo”. Isso exemplifica outro ponto relevante: uma superclasse pode ser estendida por infinitos subtipos, no entanto, um subtipo ***NORMALMENTE*** herda as características de uma única superclasse por vez.

💡 É possível herdar mais de uma superclasse com o uso de Protocolos, mas não se preocupem com isso neste momento, apenas saibam que é possível!

Vejamos nosso exemplo acima em código:



Vamos treinar?

Corrija os erros, instancie um objeto e adicione valores para Motocicleta e Avião. Usem como exemplo o código disponibilizado no exemplo anterior:

💡Dica

Em nosso EAD utilizamos compiladores online, caso algum deles não carregue, basta clicar em RUN .

Caso persista, recarregue a página!

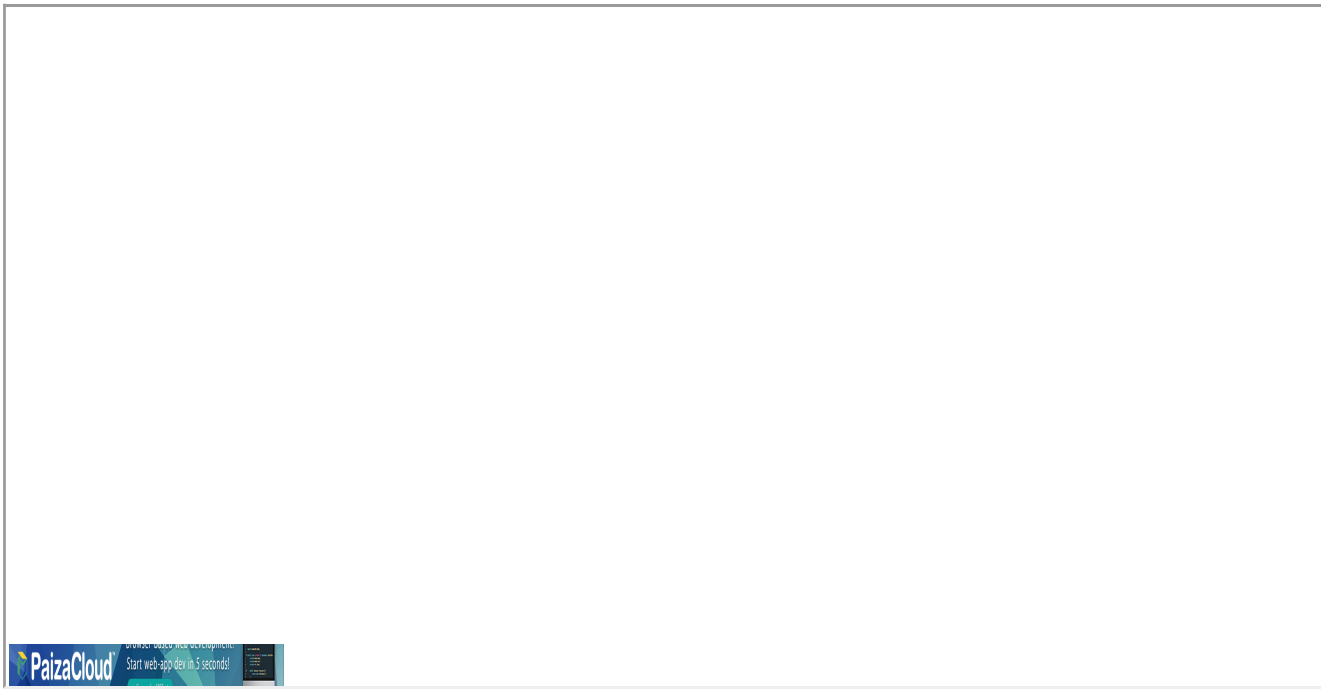
Outras dúvidas ou sugestões entre em contato com contato@hackatruck.com.br.

2. Herança - Continuação

ORIENTAÇÃO A OBJETOS - HERANÇA - CONTINUAÇÃO

Como vimos nos exemplos anteriores, podemos também adicionar atributos na subclasse, e isso se estende também a adicionar novos métodos, a fazer *overloading,* e até mesmo reescrevê-los, *overriding* (detalharemos no próximo capitulo essa definição).

Vejamos os exemplos a seguir:



Adicionamos novos atributos, cilindradas e numMotores respectivamente, e também novos métodos, descansar() para Motocicleta e voar() para Avião. Dessa forma, adicionamos comportamentos que são cabíveis somente para a realidade da própria classe.

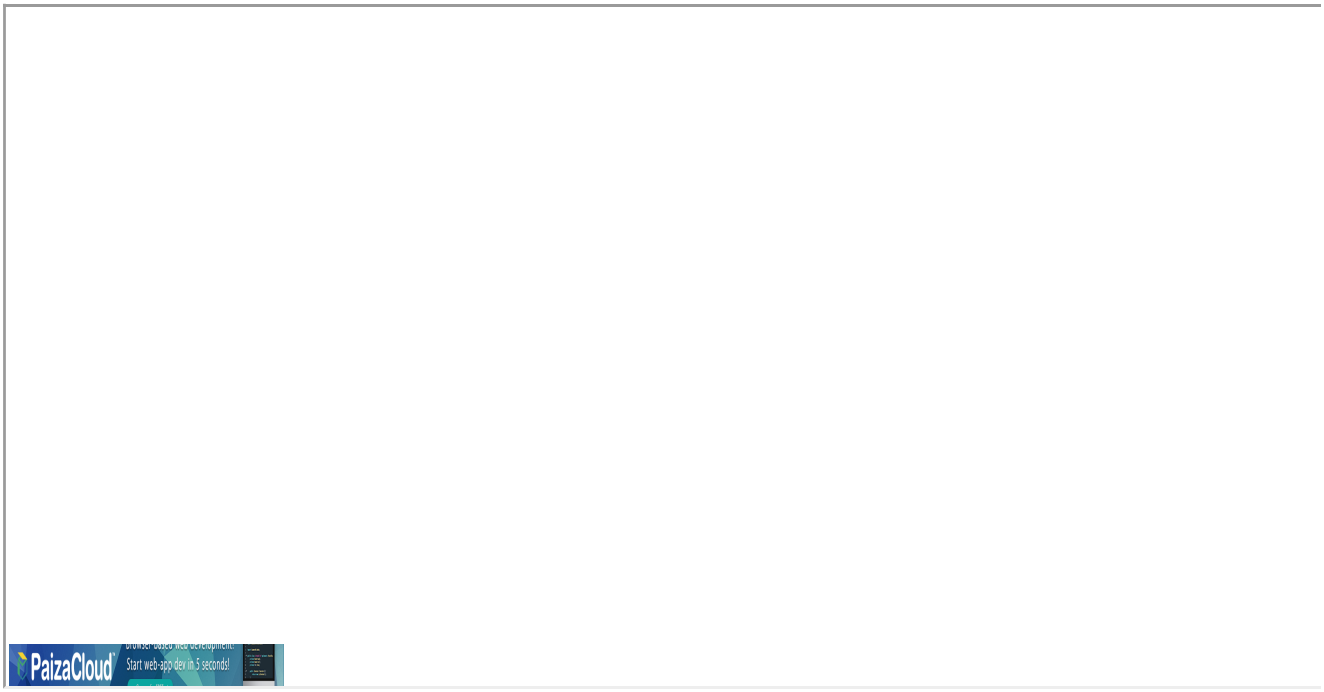
Com o *overloading* (sobrecarga), conseguimos especificar mais o comportamento da classe por meio de novos métodos com o mesmo nome e ações distintas a serem executadas, e isso é fazer sobrecarga. Para que isso ocorra e se torne possível, existem as assinaturas dos métodos que consistem em validar a soma de **Nome do Método + (Tipo dos Parâmetros) + Tipo do Retorno**. Para que nosso conceito seja válido, essa soma não pode ser repetida, ou seja, precisamos de assinaturas únicas. Exemplos:

func calcula(a: Int, b: Int) -> retorno INT

func calcula(a: Double, b: Double) -> retorno DOUBLE

func calcula(a: String, b: String) -> retorno STRING

Então, podemos definir e criar um método mais genérico, e fazer nossas especificações somente por meio de diferentes assinaturas, ou seja, métodos com o mesmo nome, mas com diferenças nos parâmetros ou até mesmo no tipo de retorno. Estes seriam exemplos de diferentes assinaturas para o método calcula. Vamos olhar no código como eles se comportariam?



Faça já os exercícios desde capítulo.

Exercícios – Herança ▶️