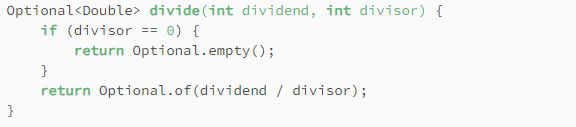
# Functor

* É um termo que vem da matemática, especificamente teoria de categoria, onde um functor é um conceito que representa o mapeamento entre duas categorias (tipos)
* Em software um functor pode ser visto como um tipo que armazena outro tipo e que permite fazer operações de mapeamento desse tipo interno. Para que um tipo seja considerada um functor ela deve seguir leis+
* Eles seguem duas leis:
  + **Identity**: Quando um functor é mapeado por uma identity function (função que retorna o mesmo valor que o parâmetro passado para ela), o functor não deve retornar o valor modificado
    - Ou seja, quando eu aplicar uma identity a um functor ele retorna um functor com o mesmo valor interno
  + **Composition/Associativity:** Quando operações de functor são chained a ordem das operações não afetam o resultado. É similar a multiplicação onde a ordem dos fatores não altera o resultado
* Na prática um functor é um design pattern funcional que permite usar um tipo container guardar um valor aplicar funções a esse valor sem afetar o dado que originou o functor



# Monads

* Em FP **efeitos** são operações que causam mudanças além do escopo da função ou componente que elas são executadas
* Em FP uma **função pura** é uma função que só depende dos seus inputs e seu algorítimo para produzir um output, de modo que ela não lê nenhum valor externo
* Monads são constructos usados para lidar com efeitos com mais facilidade, um monad pode ser pensado como um container que nos permite lidar com os efeitos fora do escopo da função preservando a pureza da função
* Optional, List, Stream são exemplos de monads da lib padrão Java, Optional é usado como container para valores null, List e Stream são usados para containers de coleções de dados



* Esse é um exemplo de monad sendo usado que torna a função pura ao garantir que ela não gere nenhum efeito como ArithmeticException ao dividir por zero e encapsula a operação de divisão no container.
* Quando um monad precisa ser transformado são usados functors, um exemplo disso são os métodos map de Optional e Stream que recebem uma função como argumento para mapear um tipo para outro
  + Todo monad é um functor, mas nem todo functor é um monad



* As operações suportadas por monads são definidas por:
  + **Construtor ou return definido por “a -> M a”:** onde a é um tipo genérico qualquer (Ex: Integer), M é o tipo do Monad (Optional, List, Stream) e -> indica que um tipo é mapeado para outro então:
    - Um tipo genérico a é mapeado para o tipo do monad um exemplo seria: **Optional<Integer> opt = Optional.of(1);**
  + **Bind definido por “M a -> (a -> M b) -> M b”:** M a é um monad M contendo um valor do tipo a, (a -> M b) é uma função que pega um valor do tipo M a e transforma em M b e M b é o resultado da transformação
    - Bind monad tem a capacidade de receber uma função transformadora e transforma o monad em outro monad com outro valor, no caso de Optional seria algo como Optional.of(“1”).map(Integer::parse), onde map seria o bind
    - Bind, map e flatmap são a mesma coisa, eles transformam o valor monádico em outro
* Bind pode ser encadeado assim como em functors

