INE5416 - Paradigmas de Programação (2015/2)
Gustavo Zambonin
Relatório 6 - Classes e Tipos

- O conceito de **polimorfismo** pode ser definido como a habilidade de objetos diferentes responderem, cada um à sua maneira, para uma mensagem idêntica provinda de uma certa *interface*. Por exemplo, seja uma classe `Veiculo` e três classes filhas `Bicicleta`, `Carro` e `Caminhao`. A definição e chamada de um método `NumeroDeRodas()` retornaria valores diferentes para cada tipo de objeto: respectivamente 2, 4 e 18 rodas.
  - O polimorfismo ***ad-hoc*** refere-se a funções polimórficas que podem ser aplicadas a argumentos de diferentes tipos, e retornam valores referentes a esses tipos de objetos. Por exemplo, implementações de sobrecarga em operadores em diversas linguagens funcionam desta maneira: o operador `+` funciona como soma, quando aplicado a números, e como concatenação, quando aplicado a *strings*.
  - O polimorfismo **paramétrico** habilita uma função, ou objeto, a ser descrita genericamente para que possa gerenciar valores uniformemente independente de seus tipos. Implementações de tipos genéricos e *templates*, em diversas linguagens de programação, são exemplos de polimorfismo paramétrico.
- Diversos recursos polimórficos diferem de linguagem para linguagem. Por exemplo, a utilização de *templates* em C++ habilita a linguagem a trabalhar com tipos genéricos. Esta estratégia também existe em Python de modo menos verboso, mas funciona de modo diferente a nível de compilação/interpretação. assemelhando-se mais às funções virtuais em C++. Em Java, sobrecarga de operadores não existe, funções abstratas provindas de interfaces forçam implementações derivadas em subclasses, e tipos genéricos precisam ser subclasses de outros tipos. Haskell, por outro lado, não suporta sobrecarga de funções completamente, apenas através de *type classes*, construtos que habilitam subclasses a utilizar, implementar e sobrecarregar métodos disponíveis.
- A orientação a objetos não é tão natural em Haskell como em outras linguagens. Embora esta exista, é comum abordar problemas de outras maneiras quando utiliza-se uma linguagem funcional. Não existe, por exemplo, uma superclasse universal como `Object`, e o encapsulamento deve ser feito através do gerenciamento de módulos.
- Listas e tuplas são estruturas fundamentais para manipulação de grupos de valores. A restrição mais importante é que todos os elementos da lista devem ser do mesmo tipo. Existe um operador de adição de elementos à lista que pode ser utilizado da seguinte maneira: `<elemento>:[]`, onde a lista é delimitada por colchetes. De modo contrário, tuplas aceitam elementos de tipos diferentes, mas são imutáveis no seu tamanho, ou seja, o operador `:` não deve funcionar. n-uplas são tuplas de tamanho n, ou seja, que contêm n elementos.