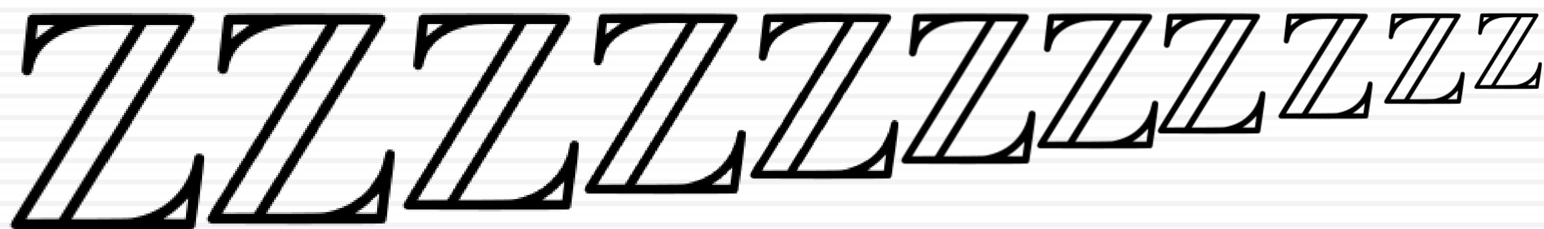
PROGRAMAÇÃO ORIENTADA PELOS OBJECTOS

Aula 6

Hierarquias de tipos

Conjuntos de inteiros



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Conjuntos de inteiros

- Os conjuntos de inteiros têm diversas aplicações em muitos domínios
  - Números do totoloto
  - Números de telefone
  - ...
- Pretendemos construir diversos tipos de conjuntos de inteiros, para depois escolher o mais adequado consoante as situações
  - Em particular, queremos
    - Um conjunto simples de inteiros
    - Um conjunto de inteiros em que seja fácil saber qual o maior
    - Um conjunto de inteiros ordenado

Estes conjuntos vão ter algumas coisas em comum

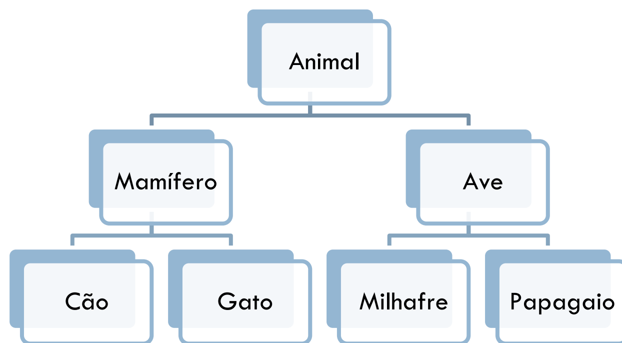
- Será que conseguimos reaproveitar esforços na definição dos vários tipos de conjuntos?
- Afinal, parece que pretendemos criar algo como uma **família de tipos relacionados**! Como fazer?

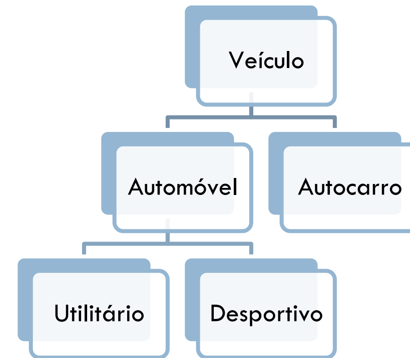
Famílias?

- Como tirar partido da abstracção de dados definindo famílias de tipos relacionados?
  - Os membros de uma família têm algum comportamento comum
    - Têm um conjunto semelhante de métodos
    - As chamadas a esses métodos resultam em comportamentos semelhantes
    - Os elementos de uma família podem divergir
      - Estendendo alguns dos comportamentos comuns da família
      - Acrescentando novos comportamentos

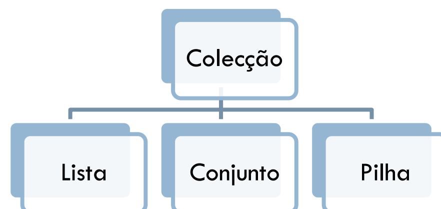
Famílias!

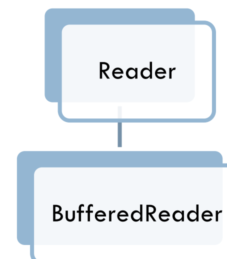
- A família pode corresponder a uma hierarquia do mundo real





- Ou no mundo da programação





Família de tipos

- Uma família de tipos é definida por uma hierarquia de tipos
  - No topo da hierarquia, está um tipo cuja especificação define o comportamento comum a todos os membros da família
    - Isto pode incluir quer as assinaturas, quer o comportamento das operações que funcionam de modo comum para os membros da família
  - Os outros elementos da família são **sub-tipos** deste tipo que está no topo da hierarquia: o **super-tipo**
    - Por sua vez, estes sub-tipos podem ter, também eles, os seus respectivos sub-tipos
      - A hieraquia pode ter mais que dois níveis!

Famílias de tipos usadas de duas formas

- Famílias usadas para fornecer diferentes implementações de um tipo
  - Neste caso, os sub-tipos não adicionam novo comportamento, com excepção de que cada um deles terá o seu respectivo construtor
  - A classe implementando o sub-tipo implementa exactamente o comportamento definido pelo super-tipo
- Famílias cujos sub-tipos estendem o comportamento do seu super-tipo
  - Por exemplo, através do fornecimento de novos métodos
  - A hierarquia nestas famílias pode ser multi-nível
  - Na parte de baixo da hierarquia, podem existir várias implementações de um determinado sub-tipo

Hierarquia de tipos

- Define uma família de tipos consistindo num super-tipo e nos seus sub-tipos
  - A hierarquia pode ter vários níveis
- Algumas famílias de tipos são usadas para fornecer diferentes implementações de um tipo: os sub-tipos fornecem implementações diferentes do seu super-tipo
- Mais genericamente, os sub-tipos estendem o comportamento dos super-tipos, por exemplo acrescentando-lhes novos métodos
- O **princípio da substituição** fornece abstracção por especificação a famílias de tipos, requerendo que os sub-tipos se comportem de acordo com a especificação dos seus super-tipos

Para que serve uma hierarquia de tipos?

- Permite "relaxar" as regras na atribuição e passagem de argumentos, e na forma como as chamadas são tratadas (em particular, na escolha sobre exactamente que código é executado em resposta a essas chamadas)

Atribuição

- Uma variável pode ser declarada como pertencendo a um tipo, mas na realidade referir-se a um objecto que é de um sub-tipo desse tipo

```
Animal a1 = new DogClass("Boby");
Animal a2 = new CatClass("Tareco");
```

- Ou seja, variáveis do tipo **`Animal`** podem, na realidade referir-se a objectos do tipo **`DogClass`**, **`CatClass`**, ou qualquer outro sub-tipo de **`Animal`**
  - Para distinguir entre ambos, chamemos ao tipo declarado o tipo **aparente**, e ao usado na instanciação o tipo **real** (em Inglês, *apparent* e *actual type*, respectivamente)

Verificação de tipos pelo compilador

- O compilador verifica os tipos com base na informação para ele disponível
  - Usa sempre os tipos *aparentes*, não os *reais*, para determinar que chamadas a métodos são legais
  - O objectivo da verificação é garantir que o objecto tem **mesmo** um método com a assinatura apropriada
    - Pode é não saber qual é o tipo *real*...
  - Recorde o conceito de *early binding*, na aula anterior

Dispatching

- Por vezes o compilador pode não saber qual é o tipo real de um objecto
- O código a correr depende do tipo real do objecto
- A chamada ao método correcto é conseguida através de um mecanismo denominado **dispatching**
  - Em vez de gerar código para chamar directamente o método, o compilador gera código para descobrir, em tempo de execução, qual o método que deve ser executado, indo depois para esse método
- Recorde o conceito de late binding, na aula anterior

Como definir uma hierarquia

- Especificação do super-tipo é, frequentemente, incompleta
  - Por exemplo, pode não ter construtores
- Especificação de sub-tipos é feita relativamente à especificação dos super-tipos
  - Foco no que o sub-tipo tem de novo, não no que se mantém sem alterações do super-tipo
    - Tipicamente, acrescenta construtores do sub-tipo
    - Métodos adicionais
    - Se o sub-tipo alterar a especificação de métodos definidos no super-tipo, então tem de fornecer a nova especificação desses métodos
      - Há limites para o tipo de alterações permitidas (já voltaremos aqui)

Implementação da hierarquia

- Por vezes, os super-tipos não são implementados de todo, ou apenas são parcialmente implementados
- A implementação do super-tipo pode disponibilizar informação extra a potenciais sub-tipos, com métodos e campos destinados exclusivamente aos sub-tipos
- Se o super-tipo é implementado, ainda que parcialmente, o sub-tipo é uma extensão da implementação do super-tipo
  - A implementação do sub-tipo pode herdar variáveis de instância e métodos do super-tipo
  - A implementação do sub-tipo pode também redefinir os métodos herdados

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Definição de hierarquias, em Java

- Utilização do mecanismo de **herança**
  - Este mecanismo permite que uma classe seja uma sub-classe de outra classe (a super-classe) e que implemente 0 ou mais interfaces
- Super-tipos definidos como classes ou interfaces
  - Em qualquer caso, a classe ou interface fornece uma especificação do tipo
    - No caso da interface, apenas fornece a especificação
    - No caso da classe, também pode fornecer uma implementação parcial ou total do super-tipo

Classes concretas vs. Classes abstractas

- Classes concretas implementam completamente o tipo
- Classes abstractas implementam apenas parcialmente o tipo
  - Não é possível criar instâncias de uma classe abstracta, tal como já acontecia com as interfaces, porque pelo menos alguns dos seus métodos não estão implementados!
  - Os métodos não implementados dizem-se abstractos
    - Cabe às sub-classes implementar esses métodos
  - O utilizador da classe abstracta não se apercebe da diferença entre métodos abstractos ou concretos
    - Esta diferença interessa apenas a quem implementa a sub-classe