Tipos de dados: parte 2

Disciplina de Modelos de Linguagens de Programação

Aula 15

Revisão: tipos de dados

- Servem para caracterizar e formalizar um conjunto de elementos que pode ser utilizado para a definição estruturas que manipulem dados no computador
- Envolvem contexto, formato, limites e operações válidas
- Divididos em:
  - **Tipos de dados primitivos:** oferecidos pela linguagem e, normalmente, diretamente suportados pelo hardware
  - **Tipos definidos pelo usuário:** estendem o sistema de tipos, permitindo a modelagem de dados complexos

Revisão: orientação a objetos

- **Classe:** mecanismo de definição de tipos que possui características interessantes provenientes do paradigma O.O., tais como herança, polimorfismo e encapsulamento
- **Objeto:** instância de uma classe
- **Atributos:** características dos objetos ou das classes (representados por variáveis ou constantes)
- **Métodos:** operações ou serviços oferecidos pelos objetos ou classes (Similares à funções ou procedimentos)

Para que serve tudo isso?

- Um dos principais objetivos da utilização de sistemas de tipos em linguagens de programação é permitir a detecção de erros (de tipos)
- Erro é toda e qualquer violação das regras definidas pela linguagem de programação
- A detecção de erros pode ser feita de forma estática ou de forma dinâmica (nas operações, nos operandos e nos parâmetros)

Detecção estática vs dinâmica

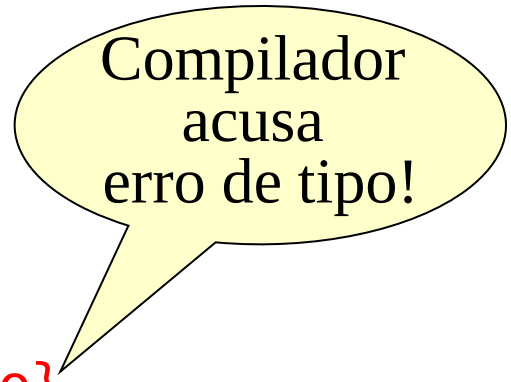
- **Estática**:
  - Violação de tipo por atribuição, detectada pelo compilador
- **Dinâmica**:
  - Violação de tipo por leitura, com valor proveniente do meio externo
  - Somente pode ser detectada pelo ambiente de execução
- Uma linguagem que não garante a ausência de erros de tipos, pelo menos de forma estática (em tempo de compilação), é dita fracamente tipada

Exemplo: o que há de errado?

```
Program Exemplo1;
type
    natural = 1..MAXINT;
var
   i: integer;
   s: shortint;
   n: natural;
begin
     i :=  32000;
     s := -10;
     n :=  i;
     n :=  s;
     writeln(n);
     n:= -10;
     i:=maxint;
     writeln(i);
     s := i;
     writeln(s);
end.
```

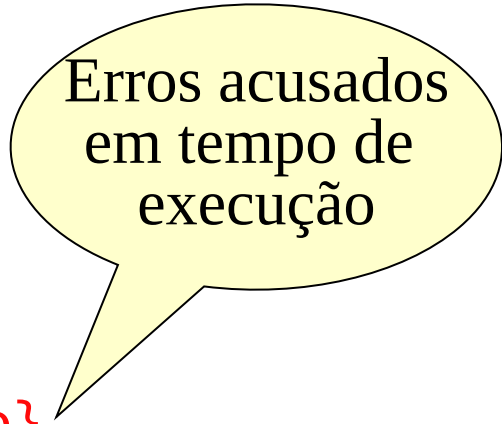
Exemplo: o que há de errado?

```
Program Exemplo1;
type
    natural = 1..MAXINT;
var
   i: integer;
   s: shortint;
   n: natural;
begin
     i :=  32000;
     s := -10;
     n :=  i;
     n :=  s;
     writeln(n);
     n:= -10;  {violação estática de tipo}
     i:=maxint;
     writeln(i);
     s := i;
     writeln(s);
end.
```



Exemplo: o que há de errado?

```
Program Exemplo1;
type
    natural = 1..MAXINT;
var
   i: integer;
   s: shortint;
   n: natural;
begin
   i := 32000;
   s := -10;
   n := i;
   n := s;  {violação dinâmica de tipo}
   writeln(n);
   n:= -10;  {violação estática de tipo}
   i:=maxint;
   writeln(i);
   s := i;   {violação dinâmica de tipo}
   writeln(s);
end.
```



Foco da aula de hoje

- Como se dá:
  - a compatibilidade e a equivalência entre tipos;
  - a conversão implícita ou explícita de tipos;
  - a inferência de tipos;
  - a checagem de tipos.

MLP - Aula 15

COMPATIBILIDADE

Compatibilidade (de plataforma)

- Nem todos os compiladores são iguais! O programador deve estar ciente disso...
- Lembre-se do discurso de Bjarne Stroustrup, criador de C++:

> *Pessoas que apregoam não se preocupar com portabilidade geralmente fazem isto porque usam um único sistema e sentem que podem se dar ao luxo de acreditar que "a linguagem é aquilo que meu compilador implementa".*
>
> *Esta é uma visão restrita e míope. Se seu programa é um sucesso, é muito provável que seja portado, de modo que alguém vai ter que procurar e corrigir problemas relacionados com características dependentes da implementação.*
>
> *Além disto, freqüentemente é necessário compilar programas com outros compiladores para o mesmo sistema e mesmo uma versão futura de seu compilador favorito pode fazer algumas coisas de maneira diferente da atual.*

Compatibilidade (de plataforma)

Como resolver, então?

- Normalmente, a própria linguagem (quando não há uma especificação clara), oferece um operador que devolve os tamanhos dos seus tipos de dados
- Em C ou C++, por exemplo, usar operador ***sizeof* :**
  - Devolve o tamanho em bytes (de uma variável ou tipo)
  - Observações:
    - Objetos em C++ são expressos em múltiplos de ***char*** (***char*** pode armazenar um caractere do conjunto de caracteres da máquina)
    - é garantido que um ***char*** tem pelo menos 8 bits, um ***short*** pelo menos 16 bits e um ***long*** pelo menos 32 bits, e que:

      $\text{sizeof(char)} \leq \text{sizeof(short)} \leq \text{sizeof(int)} \leq \text{sizeof(long)}$

Compatibilidade (entre tipos)

- Existem situações em que determinados tipos de dados são esperados:
  - Atribuições : int a = <inteiro>;
  - Operações : a % b (resto da divisão exige nºs inteiros)
  - Funções : int funcao(int, float);
- A compatibilidade define quando um tipo pode ser usado no lugar de outro
- Quando a compatibilidade ocorre, podemos usar um tipo no lugar de outro ou misturá-los, sem que ocorra erro de tipo (mas podem haver perdas!)

Exemplo em C

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
    unsigned short int a, b, c;

    a = 650000;
    b = -10;
    c = 2.4;

    printf("a=%d b=%d c=%d", a, b, c);
    system("Pause");
    return 0;
}
```

Compatibilidade (entre tipos)

- Métodos de implementação:
  - Compatibilidade por nome
  - Compatibilidade de estrutura
  - Formas mistas

Compatibilidade por nome

- Considere o exemplo:

```
type natural : 1 .. Maxint;
var n1, n2   : natural;
    i        : integer;
    n3       : natural;
```

  - Compatibilidade por nome:
    - Ocorre quando as variáveis estão na mesma declaração ou quando suas declarações tiverem o mesmo nome de tipo
    - Simples de implementar: basta comparar o nome dos tipos
    - No exemplo:
      - "n1", "n2" e "n3" são compatíveis entre si
      - "i" não é compatível com nenhuma das outras

Compatibilidade de estrutura

- Considere o exemplo:

```
type natural : 1 .. Maxint;
var n1, n2   : natural;
    i        : integer;
    n3       : natural;
```

  - Compatibilidade de estrutura:
    - Ocorre quando os tipos tiverem estrutura idêntica
    - Mais difícil de implementar, pois se deve comparar as estruturas inteiras (imagine tipos complexos, listas encadeadas…)
    - No exemplo: todas as variáveis são compatíveis!

Compatibilidade de estrutura

- Questões:
  - Se a estrutura for a mesma, mas com um nome de campo diferente, a compatibilidade deveria ser válida?

Pascal, por exemplo, não especifica claramente quando usar compatibilidade por nome ou por estrutura. Cada implementação poderia adotar regras diferentes, gerando incompatibilidades. O padrão ISO de 1982 define regras claras, mas com algumas variações das técnicas originais.

Compatibilidade de estrutura

- Exemplo:

Dado um programa em linguagem *c like*:

```
struct Data { int d, m; };
struct Data hoje;
struct OutraData { int d, m; };
struct OutraData amanha;
void mostrar(struct Data d);
// ...
```

Estruturalmente semelhantes

mostrar(amanha);

Só funciona se a linguagem utilizar método de compatibilidade por estrutura

Compatibilidade de estrutura

- Questões:
  - E se quisermos criar tipos efetivamente diferentes, tais como:

```
Type Celsius    = Real;
     Fahrenheit = Real;
```

➔ São abstrações de diferentes categorias e não poderiam ser misturados em uma mesma expressão!

Compatibilidade de estrutura

- Exemplo:

  Dado um programa em linguagem *c like*:

```
struct Ponto { int x, y; };
struct Ponto pt;
struct Data { int x, y; };
struct Data hoje;
void mostrar(struct Data d);// aceitaria qq um deles (por estrutura)

mostrar(pt); // só funciona se a linguagem utilizar compatibilidade por estrutura
mostrar(hoje); // funciona mesmo que a linguagem utilize o método por nome
```

Estruturalmente semelhantes (mas representam entidades diferentes)

Compatibilidade: formas mistas

- Equivalência de declaração (Pascal):

```
Type  tipo1 = array[1..10] of integer;
      tipo2 = array[1..10] of integer;
      tipo3 = tipo2; // declara o que é equivalente
```

- Em linguagem C, enum, struct e union definem novos tipos, mas typedef simplesmente cria sinônimos para um tipo existente.

Compatibilidade: formas mistas

- Tipos derivados versus subtipos
  - Tipos derivados (incompatíveis):

```
ADA:    type celcius is new FLOAT;
ADA:    type fahrenheit is new FLOAT;
HASKELL: type celcius = Float;
```

  - Subtipos (compatíveis):

```
ADA:  subtype idade is new INTEGER range 0..130;
Haskel: newtype idade = Integer;
```

Compatibilidade: conversão

- Quando os tipos não forem equivalentes, eles podem ser convertidos:
  - De forma implícita (automática):
    - Com base em regras pré-definidas (dependente da linguagem ou compilador)
    - Chamada de: coersão (*coersion*)
  - De forma explícita (manual):
    - Codificada pelo programador
    - Usando funções de conversão ou por *type cast*

Compatibilidade

CONVERSÃO IMPLÍCITA

Conversão implícita (*coersion*)

- Em Java, todos os tipos primitivos podem ser convertidos de um para outro (o maior é adotado), com exceção do tipo booleano:

de	para
byte	short,int,long,float,double
short	int,long,float,double
char	int,long,float,double
int	long,float,double
long	float,double
float	double

- Em C, o funcionamento é similar
- Em alguns casos a conversão pode perder informação, devido ao truncamento ou arredondamento inerente ao formato ou tamanho dos tipos de dados

Conversão implícita: exemplos

- Por atribuição:
  - Conversão para o lado esquerdo do comando
  - Exemplo:

```
float f; int i;
f = i; // int para float
```

- Por promoção:
  - Para o tipo de resultado esperado pelo operador ou pelo resultado (por inferência)
  - Exemplo:

```
float f; int i; float r;
f = r/i; // i é promovido
```

Conversão implícita: atenção

- Coerção também pode implicar num código de verificação dinâmica ou código para conversão da representação
- Isso depende do compilador ou da especificação da linguagem
- Coerções são controversas e enfraquecem a segurança do sistema de tipos da LP
- Fortran e C são consideradas por alguns como sendo fracamente tipadas, pois permitem diversas coerções no código

Compatibilidade

CONVERSÃO EXPLÍCITA

Conversão explícita

- Por *Type cast (casting):*
- Por uso de funções de conversão/*wrappers*

Conversão explícita

- *Type cast (casting):*

```
i = (int) f ;
r = (float) i/f;
v = ((*Veiculo) lst);
```

- E.g., útil em vetores de objetos, onde os elementos são construídos dinamicamente, podendo ser de subtipos diferentes

Conversão explícita: exemplo

```
// Define uma classe-pai:
class VEICULO{
  protected int velocidade, direcao;
  protected float  combustivel;
  protected String cor;
  public VEICULO(){
    velocidade  = 0;
    combustivel = 100;
    direcao     = 0;
    cor         = "Branco";
  }
  public mover(int vel){
    combustivel -= 1 * vel;
    if(combustivel == 0.0) parar();
    else velocidade = vel;
  }
  public void parar(){
    velocidade = 0;
  }
  public void virar(int direcao){
    this.direcao = direcao;
  }
}
```

```
// Classe filha (derivada de veiculo):
class TERRESTRE extends VEICULO{
  private int nrodas;
  public TERRESTRE() {
    nrodas = 4;
    cor    = "Vermelho";
  }
  public void trocarRodas(){ /* ... */ }
}
```

```
//...
Vector vetor = new Vector();
VEICULO veiculo;
for (int i=0; i<10; i++)
   vetor.add(new TERRESTRE());
for (int i=0; i<10; i++) {
  veiculo = (VEICULO)vetor.elementAt(i);
  veiculo.mover(10);
  veiculo.parar();
  veiculo.virar(2);
  //...
}
//...
```

Conversão explícita

- Por funções de conversão (Pascal, ML):

```
Var n, m: Integer; x: Real;
{ ... }
n := Round(x);
M := Trunc(x);
```

- Por uso de Wrappers (Java):

```
int a = 10; Integer b;
b = new Integer(a);
a = b.intValue();
```

- Java 5 oferece *wrapping coersion* (conversão implícita por função/*wrapper*):

```
b = a; // para a sendo int e b sendo Integer
```

Conversão sem mudança de representação

- Usada quando não é necessário mudar a representação do valor, mas sim apenas interpretar os bits de forma diferente
- *Non-converting type cast* ou *bit pattern conversion*
- Exemplo em C:

```
int n; float r;
r = *( (float *) &n); /* no run-time code */
```

  - O resultado é um float visto como um inteiro (não há conversão)
  - Funciona se e somente se:
    - n é um objeto com endereço
    - int e float ocupam o mesmo número de bytes

Conversão sem mudança de representação

- Comumente utilizada para manipular blocos de memória:
  - que foram alocados dinamicamente (e podem então ser vistos da forma que necessitarmos (char, int, structs)
  - que foram lidos de um arquivo de forma “raw” (conjunto de bytes)

- Exemplo em C:

```
void* bloco; // e.g. carregado de um arquivo
Record* header = (Record*)block; // Record é algum tipo estruturado (struct)
```

Conversão explícita

- Em C++ (e outras linguagens modernas), é possível explicitar o *tipo de casting* desejado (a fim de clarificar a intenção do programador):
  - **static_cast<type>(valor_a_ser_convertido)**
    → conversão tradicional de tipos compatíveis.
  - **dynamic_cast<type>(valor_a_ser_convertido)**
    → mais seguro, pois o compilador embute código de checagem para garantir que os *castings* são válidos

Conversão explícita

- Em C++ (e outras linguagens modernas), é possível explicitar o *tipo de casting* desejado (a fim de clarificar a intenção do programador):
  - **const_cast<type>(valor_a_ser_convertido)**
    - → permite um valor constante (*ready-only*) ser visto como um valor modificável:

```
const MyClass * cantTouchThis = CreateConstObject();
cantTouchThis->constant_value = 41; // erro de compilação
```

  - **reinterpret_cast<type>(valor_a_ser_convertido)**
    - → permite reinterpretar uma seqüência de bits como desejado (conversão sem mudança de representação):

```
MyClass* pclass = reinterpret_cast<MyClass *>(0xDEADBEEF);
```

Sobre conversões...

- As linguagens mais modernas tendem a minimizar ou eliminar a coersão, permitindo somente *casts*!

MLP - Aula 14

INFERÊNCIA

Inferência

- Verificação de tipos e compatibilidade asseguram que os componentes de uma expressão tenham os tipos apropriados
- Mas o que determina o tipo da expressão inteira?
- Na maioria dos casos, vale o bom senso:
  - O tipo dos operandos
  - A chamada de função tem o tipo declarado no cabeçalho da função
  - O resultado de uma atribuição tem o tipo do dado no lado esquerdo da atribuição
- Mas as operações em intervalos e objetos compostos, que tipo têm?

Inferência sobre subtipos

- Se o resultado de uma operação é atribuído a uma variável de um sub-tipo, verificação semântica dinâmica pode ser necessária
- Compilador pode manter a informação sobre os limites (maior e menor) de cada expressão para evitar verificações desnecessárias
- No exemplo:

```
Type tipo1 = array[1..10] of integer;
```

→ o compilador mantém a informação do tipo (1..10) para verificação

MLP - Aula 15

SOBRE A COMPATIBILIDADE DE TIPOS COMPOSTOS OU CRIADOS POR EXTENSÃO (HERANÇA)

Extensão de tipos estruturados

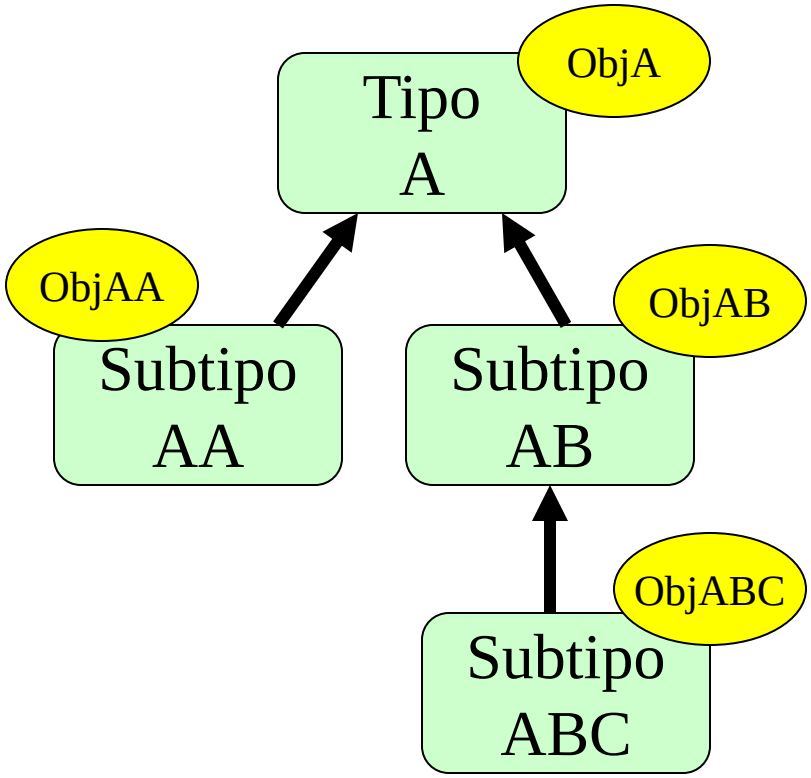
- Ocorre quando um tipo estruturado utiliza outro tipo estruturado como base
- Atualmente, normalmente utilizam o mecanismo de herança
- Úteis em estruturas de dados mais complexas, permitindo:
  - reduzir o esforço de programação;
  - aumentar a reutilização de código; e
  - aumentar a legibilidade de programas.

Extensão: definição em Java (classes)

```
class Nome {
  String primeiroNome, meioNome, ultimoNome;
  public Tnome(){ primeiroNome="Pedro”; meioNome="Alvares”;
                  ultimoNome="Cabral"; }
  // Sets e gets para atributos protegidos
}
class Pessoa{
  Nome nome;
  String email;
  char sexo;
  public Pessoa() { nome  = new Nome(); sexo = ‘M’;
                    email = “pessoa@”; }
  // Sets e gets para atributos protegidos
}
class Aluno extends Pessoa {
  int matricula;
  public Aluno() { matricula = 0; }
  // Sets e gets para atributos protegidos
}
```

Equivalência em O.O.

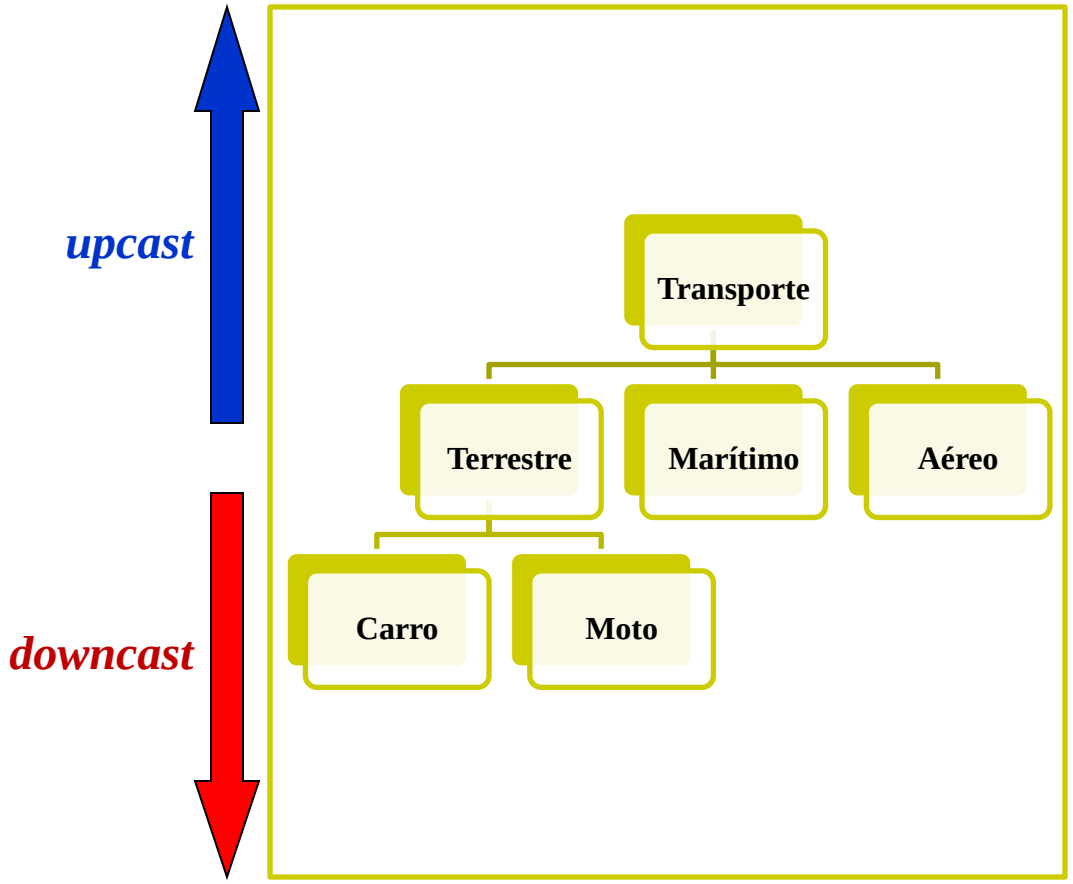
Herança estabelece uma hierarquia (ou família) de tipos:



- Um subtipo "é-um" tipo para fins de equivalência
- Atribuições são válidas somente de tipos para subtipos (*upcast*):
  - ObjA = ObjAB
  - ObjA = ObjABC
- Contra-exemplos:
  - ObjAA = ObjAB
  - ObjABC = ObjAB

Conversão (implícita) na hierarquia

- Seja a hierarquia:



- Atribuições válidas de objetos:

```
transporte = terrestre
terrestre  = carro
terrestre  = moto
```

- Atribuições inválidas de objetos:

```
terrestre = transporte
moto = aéreo
```

OBS:

- Upcast é válido e implícito!
- Informação extra é ignorada!

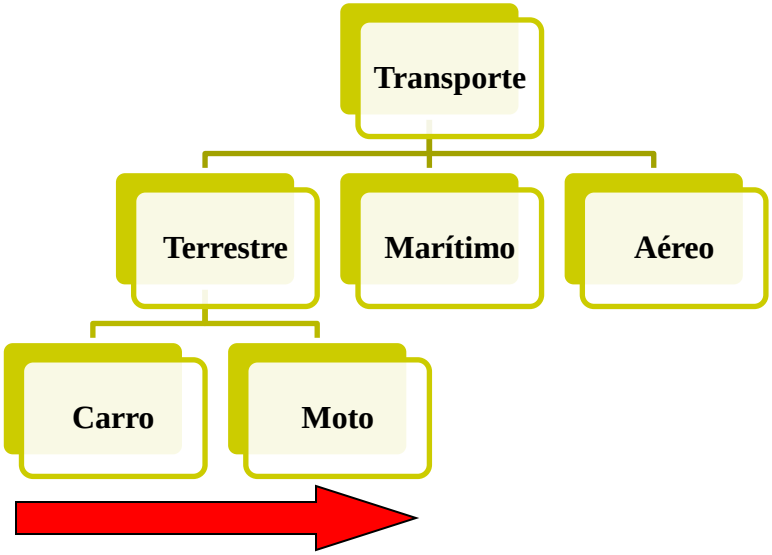
Conversões **explícitas** hierarquia

- **Questão**: como tornar válidas as atribuições de objetos como os abaixo?

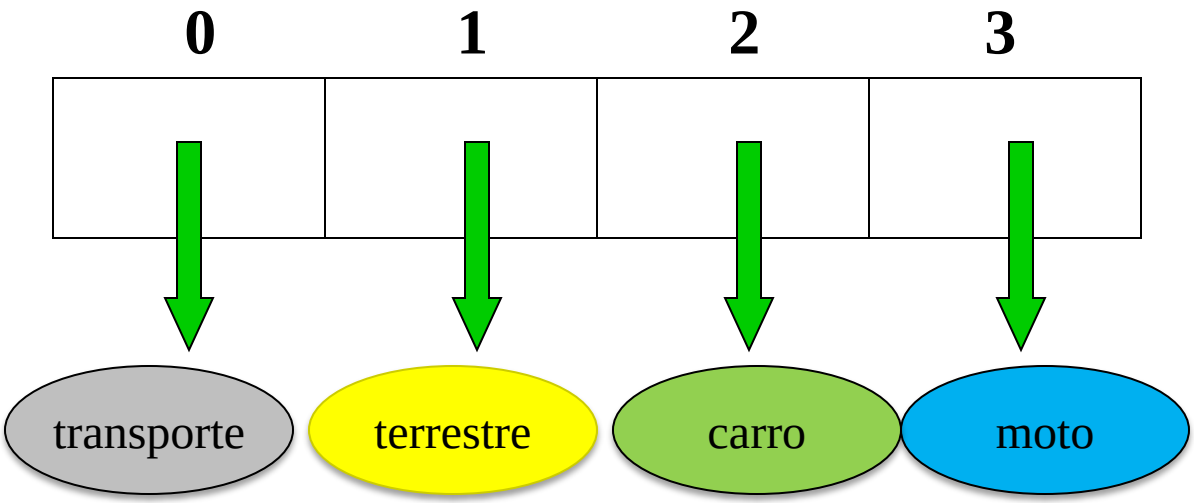
```
carro = terrestre;
carro = moto;
```

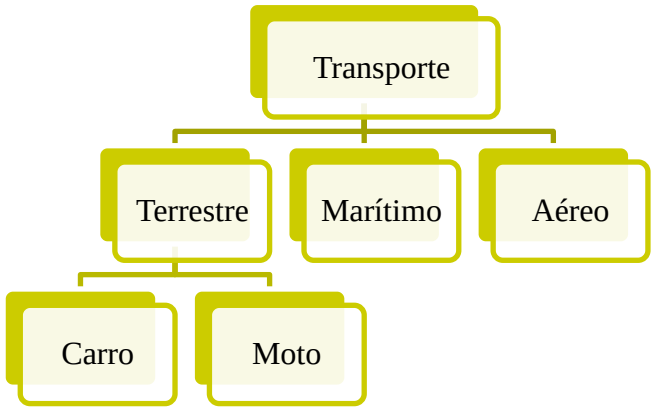
- **Solução**:
  usar "casting" de tipo!

```
Terrestre tRef;
Carro cRef;
cRef = (Terrestre) tRef;
```



Agrupando objetos equivalentes



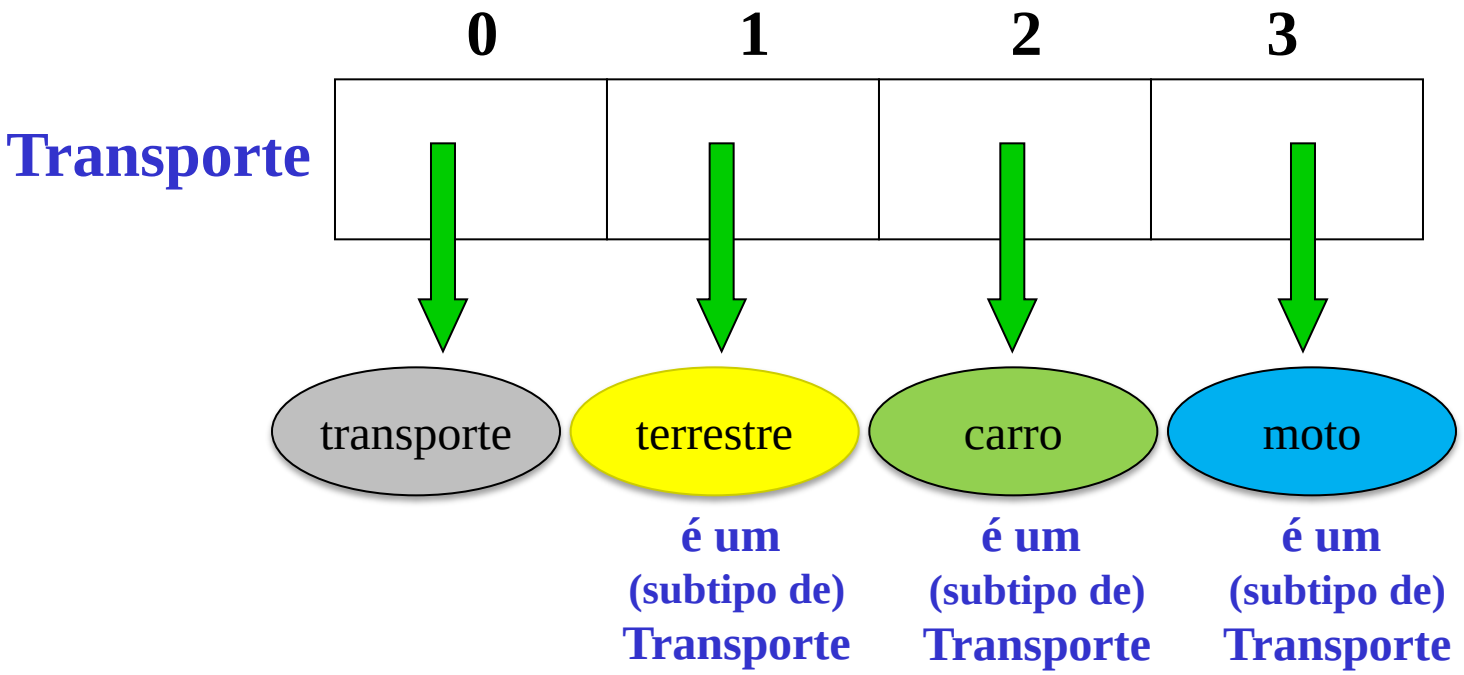


Princípio: equivalência de tipos

Agrupando objetos equivalentes

- Agrupando em ‘arrays’: mesmo tipo/subtipo
  - Exemplo:

```
Transporte grupo[] = new Transporte[4];
```



Leitura recomendada

- Sebesta, Robert W. Tipos de Dados (capítulo 6). In: **Linguagens de Programação**. 5a. Ed. Porto Alegre: Bookman, 2003.
- Sebesta, Robert W. Expressões e Instruções de Atribuição (capítulo 7). In: **Linguagens de Programação**. 5a. Ed. Porto Alegre: Bookman, 2003.