Tipos de Dados

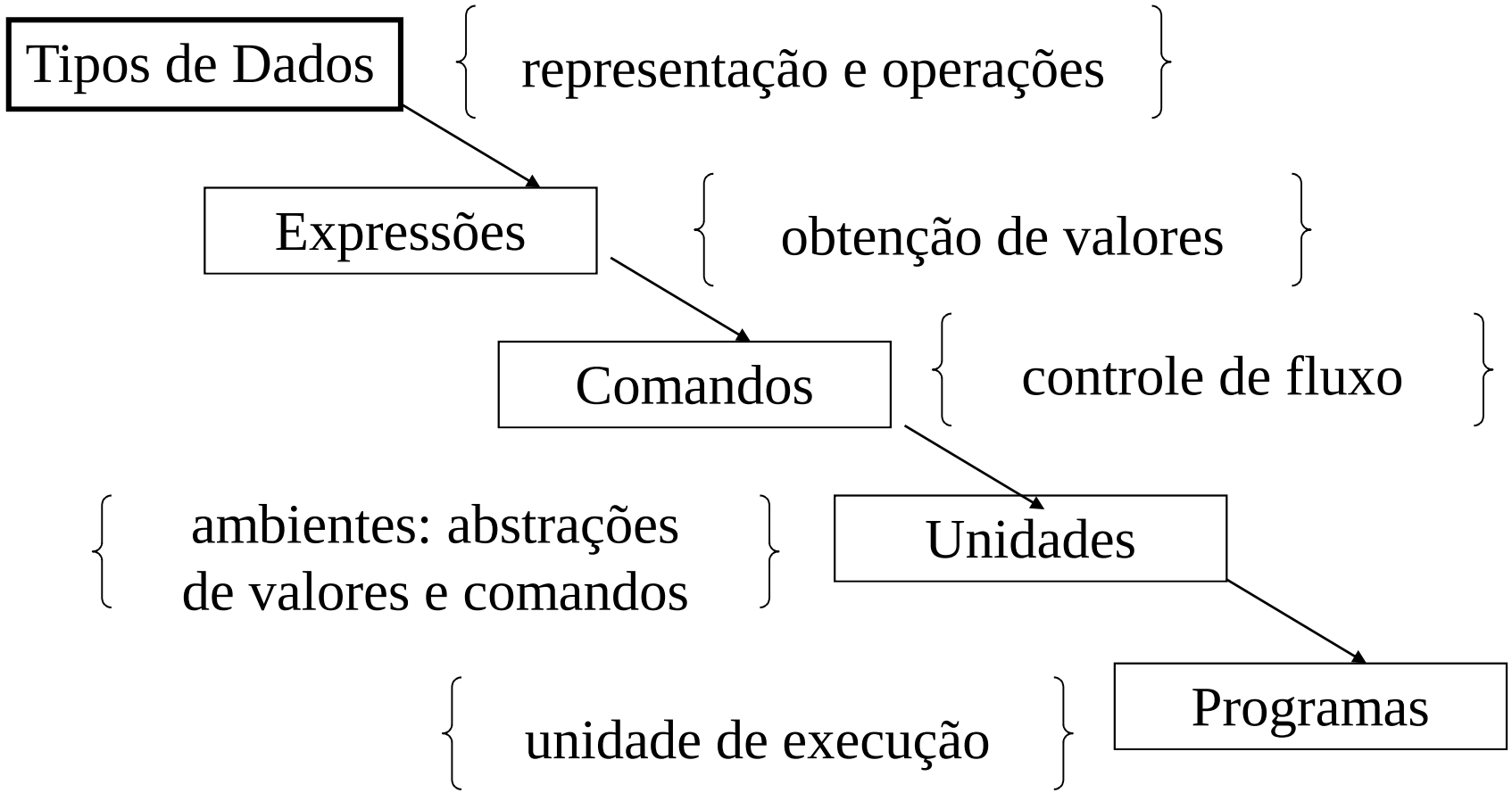
Disciplina de Modelos de Linguagens de Programação

Aula 13

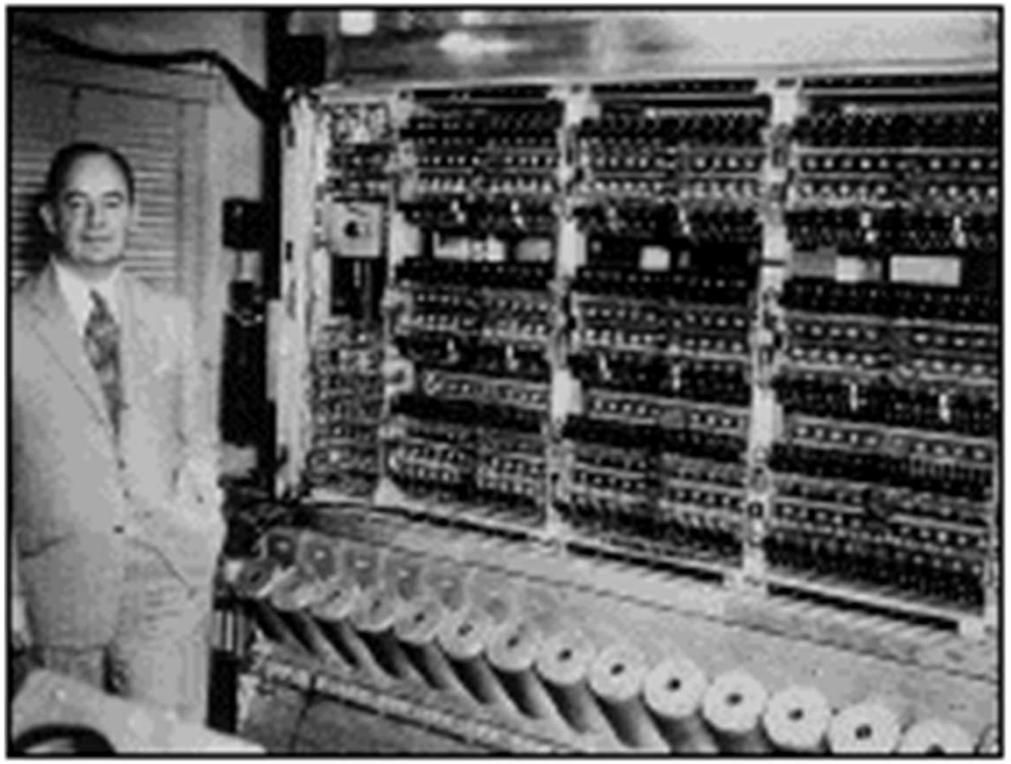
Tópicos

- Contextualização
- Sistema de tipos
- Tipos de Dados e Domínios
- Métodos de construção de domínios
- Tipos primitivos
- Tipos estruturados
- Orientação à objetos

Hierarquia de componentes de uma LP



Máquina de von Neumann



9278	dado
9279	dado
9280	dado
9281	dado
9282	dado
9283	
9284	instrução
9285	instrução
9286	instrução

Processamento de dados:
manipulações em dados (mudanças de estado).

Importância das variáveis

- Permitir reservar, acessar e manipular regiões de memória
- Manipular e processar dados de forma mais simples e menos propensa a erros
- Problemas:
  - dados são diferentes: 1, 1.0, “André”...
  - ocupam mais ou menos memória, de acordo com seu tipo ou estrutura
  - possuem operações válidas e inválidas para o seu contexto (1 + 1 = 2, mas 1 + “André” = ?)

Importância dos tipos de dados

- Determinam a classe de valores que podem ser:
  - armazenados em uma variável (na memória)
  - passados como parâmetro
  - resultantes de uma expressão
- A informação de tipo é usada para:
  - prevenir ou detectar construções incorretas em um programa
  - determinar os métodos de representação e manipulação de dados no computador

Vantagens de se definir tipos

- O conhecimento dos possíveis valores de uma variável é essencial para o entendimento de um algoritmo
- Saber quais são as operações permitidas possibilita a detecção de vários erros
- O compilador, por exemplo, pode:
  - determinar o espaço necessário para as variáveis
  - determinar como proceder para a implementação das operações e tratamento de exceções

Não é a toa que...

- Novidades na área sempre surgiram com o aparecimento das linguagens:

Fortran	☐ Tipos simples (inteiros, reais) ☐ Tipos estruturados (arranjos)
Cobol	☐ Registros
Lisp	☐ Listas
Algol68	☐ Tipos definidos pelo usuário
Atualmente	☐ TAD ☐ Classes

Alguns exemplos

- Tipos em Pascal:
  - Simples:
    - Boolean
    - Integer
    - Byte
    - Real
    - Char
    - String
  - Compostos:
    - Arrays
      - Vetores
      - Matrizes
    - Registros
  - Ponteiros

- Tipos em C:
  - Simples:
    - int
    - float
    - double
  - Compostos:
    - Arrays
      - Vetores
      - Matrizes
    - Structs
    - Unions
  - Ponteiros
  - Registradores

Sistema de Tipos

- **Mecanismo** da linguagem que serve para a definição dos tipos de dados que são manipulados pelas construções da linguagem, tais como: constantes, variáveis, parâmetros
- Conjunto de **regras** que determinam a equivalência de tipos, a compatibilidade de tipos e a inferência de tipos, para fins de verificação da validade do uso de tipos em expressões, atribuições e parâmetros

Há “linguagens não tipadas”, que justamente não usam sistema de tipos. Exemplos: LISP e Perl.

Domínios de valores

- Um domínio representa um conjunto (infinito/contínuo) de valores (e.g., números inteiros, reais, complexos)
- Problema: o computador é limitado
  - Trabalha com elementos discretos, não contínuos → mas quais?
    - Diferentes formas de representação interna foram idealizadas para definir quais são permitidos → gera incompatibilidade
      - Alternativas: padronizar a representação e/ou usar descritores de dados (metadados)

Tipos de dados

1. Tipos de dados em LP possuem representação finita
2. Dependem da forma de representação ou implementação adotada:
   - Caracteres: ASCII, UNICODE, UTF, ISO...?
   - Inteiros: 2, 4 ou 8 bytes?
   - Reais:
     - quantidade de bytes para mantissa?
     - quantidade de bytes para o expoente?
   - ...
3. São associados a um conjunto de operações válidas para manipular seus valores

Domínios versus tipos de dados

- Problema: contínuo → discreto (representação finita)
- Alternativa: adotar domínios simplificados (generalizados, abstraídos)
  - **Primitivos**: não necessitam de definição explícita, pois as características são conhecidas (e.g., domínio dos números reais)
  - **Definidos pelo usuário** (programador): seus componentes (ordinais) devem ser especificados
    - Por enumeração: criam novo domínio, com valores específicos (enumerados)
      - e.g., Estação = primavera, verão, outono, inverno
    - Por Restrição: especificam um subdomínio
      - e.g., mandato = 2000..2005

Como descrever um tipo?

- Definir seu nome (designação do tipo): Boolean
- Definir domínio de valores: lógicos
- Definir operações permitidas (operadores):
  - Aritméticos: não há para booleanos
  - Conjuntos: negação, conjunção, disjunção...
  - Relacionais: relações de igualdade, ordem...
  - ...
- Definir forma de representação: ‘True’, ‘False’
- Definir espaço ocupado: 1 bit

Tipos primitivos em LP

- São tipos atômicos, indivisíveis (não são definidos com base em outros tipos de dados)
- Geralmente refletem a estrutura de hardware, podendo ser mapeados diretamente
- Associados a um nome, a um conjunto finito de valores e a um conjunto pré-definido de operações
- Elementos de primeira ordem na maioria das LP: usados como resultado de operações, em E/S, em atribuição, como parâmetros, como valor de retorno

Tipos primitivos em Java

- boolean (1 bit)
- char (16 bits)
- byte (8 bits)
- short (16 bits), int (32 bits), long (64 bits)
- float (32 bits), double (64 bits)
- void

OBS:
- Variáveis desse tipo são colocadas diretamente na pilha
- O tamanho deles é uniforme em todas as plataformas Java

Tipos definidos pelo usuário

- Algumas vezes necessitamos manipular estruturas não suportadas pela linguagem (tipos primitivos)
- Outras queremos tornar o programa mais legível, menos complexo ou facilitar a identificação de erros
- Solução: definir novos tipos
  - Por sinonímia
  - Por restrição
  - Por enumeração
  - Por composição (tipos estruturados)

Tipos definidos pelo usuário

- Construídos por restrição:
  - Subsequência de um tipo existente
  - Exemplos em Pascal:

```
type maiusculas = 'A'..'Z';
type dias = 1..31;
type idade = 1..120;
```

Tipos definidos pelo usuário

- Construídos por enumeração:
  - Os valores desejados são descritos
  - Exemplos em Pascal:

```
type TS = ( verde, vermelho, azul );
type vetorCor = array [TS] of boolean;
```

  - Exemplos em C/C++:
    - Associação implícita:

```
enum Cores {vermelho, verde, azul};
```

    - Associação explícita:

```
enum romanos {I=1, V=5, X=10, L=50, C=100, D=500, M=1000};
enum Cartas {dois=2, tres, quatro, cinco, seis, sete, oito,
             nove, dez, valete, dama, reis, as};
```

OBS: são implementados através da associação de constantes a números inteiros.

Tipos definidos pelo usuário

- Construídos por enumeração:
  - Exemplo em Java:

```
public class Main {
    public enum Dia { SEG, TER, QUA, QUI, SEX, SAB, DOM };
    public static void main(String[] args) {
        Dia dia = Dia.SEG;
        switch (dia) {
           case SEG: System.out.println("Segunda é chato!");
                     break;
           case SEX: System.out.println("Sexta é bom");
                     break;
           case SAB:
           case DOM: System.out.println("Fim de semana!");
                     break;
           default:  System.out.println("Os dias da semana são mais ou menos");
        }
    }
 }
```

Tipos definidos pelo usuário

- Construídos por enumeração:
  - Java não permitia enumeração antes da versão 5!
  - Como resolver? *uma interface pode ser utilizada para implementar constantes, simulando-as!*

```
public interface Cores{
    public final int verde = 0;
    public final int azul = 1;
    public final int vermelho = 2;
}
public class MCores implements Cores{
   public static boolean[] vetorCor = {false, false, false};
   /* ... */
}
```

Tipos definidos pelo usuário

- Construídos por composição:
  - Os anteriores nem sempre são suficientes para representar, modelar os dados que o usuário necessita manipular...
  - Alternativa: compor domínios
    - Resultado: novo tipo de dado estruturado
    - Forma: homogêneos ou heterogêneos

Tipos estruturados

- Tipos compostos a partir de outros tipos
- Homogêneos:
  todos os componentes pertencem ao mesmo tipo
  (e.g., array (vetor ou matriz))
- Heterogêneos:
  os componentes podem ser de tipos diferentes
  (e.g., struct, union)

Métodos de composição

- Produto cartesiano:
  Faz o produto entre domínios, fornecendo tuplas ordenadas
- Mapeamento finito:
  Aplica uma função no domínio A para obter um valor de um domínio B
- Seqüência:
  Permite construir seqüências finitas $<a_1, a_2, ...a_n>$, formadas por elementos de um domínio (A)
- União:
  Faz uma união entre domínios, criando alternativas
- Conjunto potência:
  Permite a geração de subconjuntos de valores a partir de um conjunto que é definido como o domínio do tipo

Método de composição versus tipos resultantes

Método	**Formato**	**Tipo resultante**
Produto cartesiano	Heterogêneo	record, struct, class (Registro, estrutura, classe)
Mapeamento finito	Homogêneo	array, map, vector (Arranjo, mapa, vetor)
Seqüência		string, arquivo
União discriminada	Heterogêneo	variant, record, union
Conjunto potência		set

Tipos definidos pelo usuário

- Mesmo os tipos compostos (estruturados) podem não ser suficientes ou oferecer problemas e complexidades:
  - A linguagem permite operações sobre eles?
  - A linguagem faz validações e testes sobre eles?
  - Eles conseguem representar qualquer elemento do mundo real?
  - Quais são as restrições?
- Alternativa: paradigma orientado a objetos
  - Encapsula dados + métodos (operações)
  - Permite a definição de Tipos Abstratos de Dados de forma simples

Tipos abstratos de dados

- Tipos Abstratos de Dados (TAD) compreendem um tipo + conjunto de operações sobre este tipo
- Escondem os mecanismos internos de seu funcionamento, permitindo interação somente através das operações "publicadas" por eles mesmos
- Podem ser definidos pelo usuário (ou estar disponíveis em bibliotecas), oferecendo um significante benefício ao estender o sistema de tipos primitivos da linguagem
- São igualmente de primeira classe, e possuem os mesmos "privilégios" de um tipo primitivo
- Classes são a manifestação de TAD em LP OO

Classe

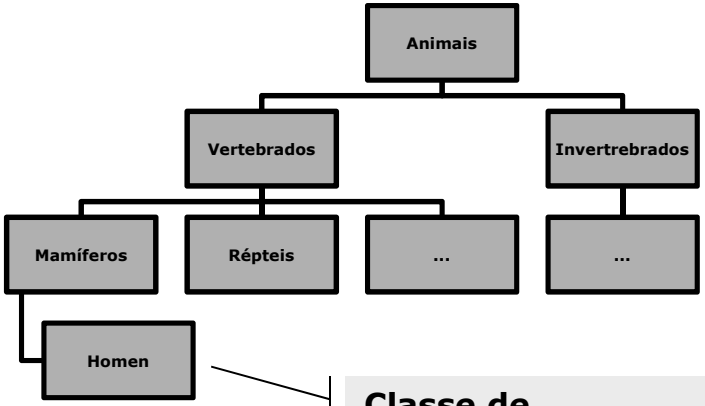
- *Blueprint*
- Define como um objeto vai se parecer
- Define a estrutura de um objeto
- Descreve o estado e as operações (comportamento) de um conjunto de objetos
  - estado: atributos (variáveis)
  - operações: métodos (funções)
- Similar a um struct (em C) ou Record (Pascal), mas com os benefícios da OO
- A "forma", segundo Aristóteles

Objeto

- Instância de uma classe
- Uma cópia construída a partir da classe
- Uma entidade (um indivíduo) independente, assíncrono e concorrente que sabe coisas (armazena dados), realiza trabalho (oferece serviços) e colabora com outros objetos (trocando mensagens) para executar as funções finais de um sistema (programa)
- A “substância”, segundo Aristóteles

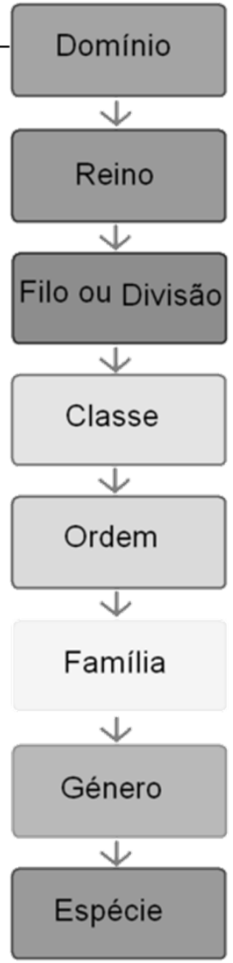
Classes vs objetos

- O cérebro armazena padrões e compara aquilo que recebe (pelos sentidos) com o que possui armazenado
- O paradigma orientado a objetos é baseado no fato de que os seres humanos tendem a abstrair os objetos (seres, componentes, entes) reais, representando-os em classes e taxonomias



Classe de entidades com características comuns: homens

Leonardo e Maria são instâncias (entidades, elementos) da classe Homen!



A hierarquia da classificação científica dos seres vivos
Fonte da figura: Wikipedia

Classes vs objetos

- Os objetos são as entidades do mundo real
- As classes representam um conjunto de entidades semelhantes, abstraindo as suas características principais e funcionamento

Exemplo: um carro

- Carros podem mover-se, parar e virar (além de outras coisas)
- Carros também têm atributos como cor, velocidade, combustível e direção

Atributos
(que todos os carros têm):
cor, velocidade, direção

Operações
(válidas para todos os carros):
mover, virar, parar

- A definição da classe vai encapsular todos as variáveis e operações de um carro (genérico)

Exemplo: um carro

- O encapsulamento tem duas vantagens:
  - A interface visível para o usuário (funções) permanece consistente (as operações para um objeto da classe Carro é a mesma para todas as suas instâncias)
  - Os detalhes de implementação podem ficar escondidos (o usuário nunca precisa saber como o carro faz para parar, mas sim que, se ele apertar o freio, o carro pára)

Especificadores de acesso

- Atributos devem ser acessados somente pelas operações definidas no ambiente encapsulado
- O ambiente encapsulado pode 'exportar' dados para outros ambientes
- Níveis de proteção (visibilidade):
  - Public: sem proteção (todos manipulam)
  - Protected: visível na classe e subclasses (e no pacote, em Java)
  - Private: visível somente na classe (default C++)

  OBS:
  - CTS/CLR .NET: Family e Assembly
  - Java: default é dito *package-private* (i.e., público no pacote)

Um exemplo em Java

```
class Carro{
 private int   velocidade;
 private float combustível,
               velMedPorLitro;
 private int   direção;
 string        cor;
```

Variáveis privadas

```
 public Carro(){  // construtor
   cor            = “Vermelho”;
   velocidade     = 0;
   combustível    = 100;
   velMedPorLitro = 10;
 }
 public void mover(int nova_velocidade) {
   combustível-=(float)velocidade / velMedPorLitro;
   if(combustivel <= 0.0) parar();
   else velocidade = nova_velocidade;
 }
 public void parar() {
    PisarNosFreios() ;
    Velocidade = 0.0 ;
 }
 /* ... */
}
```

Métodos públicos (serviços)

Um exemplo em Java

```
class Carro{
 private int   velocidade;
 private float combustível,
               velMedPorLitro;
 private int   direção;
 string        cor;

 public Carro(){  // construtor
   cor            = “Vermelho”;
   velocidade     = 0;
   combustível    = 100;
   velMedPorLitro = 10;
 }
 public void mover(int nova_velocidade) {
   combustível-=(float)velocidade / velMedPorLitr
   if(combustivel <= 0.0) parar();
   else velocidade = nova_velocidade;
 }
 public void parar() {
    PisarNosFreios() ;
    Velocidade = 0.0 ;
 }
 /* ... */
}
```

Uma classe pode definir membros (atributos ou métodos) privados ou públicos

Os **privados** só podem ser acessados e manipulados por objetos da classe Carro.

Os **públicos** podem ser acessados por qualquer outro objeto

Instanciando objetos

classe CARRO

um objeto Ferrari

- Os objetos são criados pela instanciação de uma classe (a classe é como se fosse o tipo da variável)
- Exemplo em Java:

```
Carro Ferrari;
Ferrari = new Carro(); // construtor da classe
```

- A variável "Ferrari" foi declarada como sendo do tipo 'Carro'
- Com o comando 'new' é criada uma nova instância (novo objeto)
- Esse comando aloca memória para o objeto e chama o método construtor da classe

Usando objetos

- Uma vez que o objeto tenha sido criado, o usuário pode realizar operações com ele:

```
Ferrari.mover(100);
Ferrari.parar();
Ferrari.combustível++; // Erro!
```

- Note que você pode chamar todos os métodos públicos e manipular todos os atributos públicos (combustível é um atributo privado).

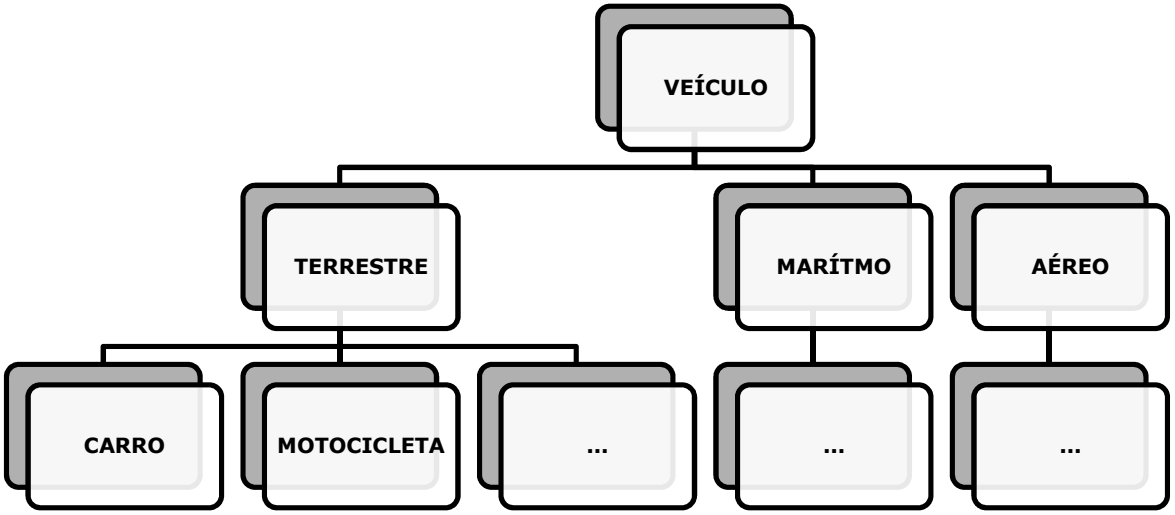
```
class Carro{
 private int   velocidade;
 private float combustível,
               velMedPorLitro;
 private int   direção;
 string        cor;
 ...
```

Métodos especiais

- Método construtor (*constructor*):
  - em Java sempre tem o mesmo nome da classe
  - utilizado para fazer a inicialização (dar valor inicial e padrão) dos atributos da instância da classe
  - sempre é chamado quando um objeto (instância) é criado (através do operador new)
- Método destruidor (*destructor*):
  - chamado toda vez que o objeto é destruído (geralmente quando o programa termina sua execução)
  - O programador pode utilizar esse método para liberar ou destruir algum recurso (dinâmico) que o objeto utilize

Herança

- Permite com que uma classe seja definida a partir de outra (chamada classe pai)
- A classe-filha herda (recebe) todos os atributos e métodos da classe pai!



Herança

```
// Define uma classe-pai:
class VEICULO{
  private int    velocidade;
  private float  combustivel;
  private int    direcao;
  private String cor;

  public VEICULO(void){
    velocidade  = 0;
    combustivel = 100;
    direcao     = 0;
    cor         = "Branco";
  }

  public mover(int vel){
    combustivel -= 1 * vel;
    if(combustivel == 0.0) parar();
    else velocidade = vel;
  }

  public parar(void){
     velocidade = 0;
  }
  public 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 trocarRodas(void){
    ...
  }
}
```

Herança

- TERRESTRE é uma especialização da classe VEICULO
- Ela herda todos os atributos e métodos de VEICULO, além de ter algumas coisas a mais, tais como: nrodas.
- Um objeto da classe TERRESTRE pode realizar todas as operações que foram definidas para um VEICULO:

```
TERRESTRE Fusca;
VEICULO   Bote;
Fusca = new TERRESTRE();
Bote  = new VEICULO();
Fusca.mover(10);
Fusca.parar();
Fusca.virar(2);
Bote.mover(20);
```

- Além dos métodos e atributos específicos da classe TERRESTRE:

```
Fusca.trocarRodas();
```

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

Resumo da aula

- Tipo:
  - conceito que criado para formalizar certos aspectos necessários à teoria dos conjuntos, ou seja, serve para caracterizar um conjunto (dizer como ele é)
  - envolve todo um formalismo que indica quais são todas as operações matemáticas envolvidas com os membros do seu conjunto (tais como soma, subtração, divisão e multiplicação)
  - permite a checagem de tipos (validação de dados), restringindo as operações (especificando o que é válido e o que não é)
- Tipos primitivos:

  Os tipos básicos oferecidos pela linguagem e que normalmente são diretamente suportados pelo hardware

Resumo da aula

- Tipos definidos pelo usuário:

  Estendem o sistema de tipos oferecidos pela linguagem, permitindo a modelagem de dados complexos, especialmente os TAD

- Classes:

  Mecanismo de definição de tipos, mas que possui características interessantes provenientes do paradigma OO, tais como herança, polimorfismo e encapsulamento

- Objetos:

  Instâncias de uma classe

Resumo da aula

- Encapsulamento (*encapsulation*):

  Permite com que um objeto encapsule, isto é, esconda seus detalhes de implementação e seus dados. Com isso, você pode criar e utilizar objetos sem precisar conhecer seu funcionamento interno

- Herança (*inheritance*):

  Novas classes de objetos podem ser criadas tendo como base classes já existentes. Isso facilita a criação de programas, já que é possível reutilizar muito do que já foi feito, além de facilitar a adição de novas características à algum componente

- Polimorfismo (*polymorphism*):

  Uma mesma função (ou método) pode ser aplicada a objetos diferentes e manter a sua funcionalidade

Diferenças entre paradigmas

- **Estruturado (PE)**:
  - Dados e procedimentos (modelados de forma separada)
  - Orientado a procedimentos
  - Seqüências de comandos realizam transformações sobre dados
- **Orientado a Objetos (OO):**
  - Organização dos dados domina
  - Dados e sua manipulação são encapsulados (modelados de forma conjunta)

Bibliografia

- Sebesta, Robert W. Tipos de Dados (capítulo 6). In: **Linguagens de Programação**. 5a. Ed. Porto Alegre: Bookman, 2003.
- Tutorial sobre **Fundamentos de Orientação a Objetos** em Java (disponível no Moodle)
- **Java Tutorial**: Enum Types http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html
- **The Java tutorial**: Controlling Access to Members of a Class http://java.sun.com/docs/books/tutorial/java/javaOO/accesscontrol.html
- Richter, Jeffrey. A Arquitetura da Plataforma de Desenvolvimento .Net Framework (cap. 1). In: **Programação Aplicada com Microsoft® .NET Framework**. Porto Alegre: Bookman, 2005.