Classes

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP
Baseado nos slides do Prof. Marco Antônio Carvalho

Introdução I

- Estamos acostumados a criar programas que:
  - **Apresentam mensagens** ao usuário;
  - Obtêm dados do usuário;
  - **Realizam cálculos** e tomam decisões.
- Todas estas ações eram **delegadas à função main** ou outras;
- A partir de **agora**, nossos programas terão uma **função main e uma ou mais classes**
- Cada classe consistindo de **dados e métodos**.

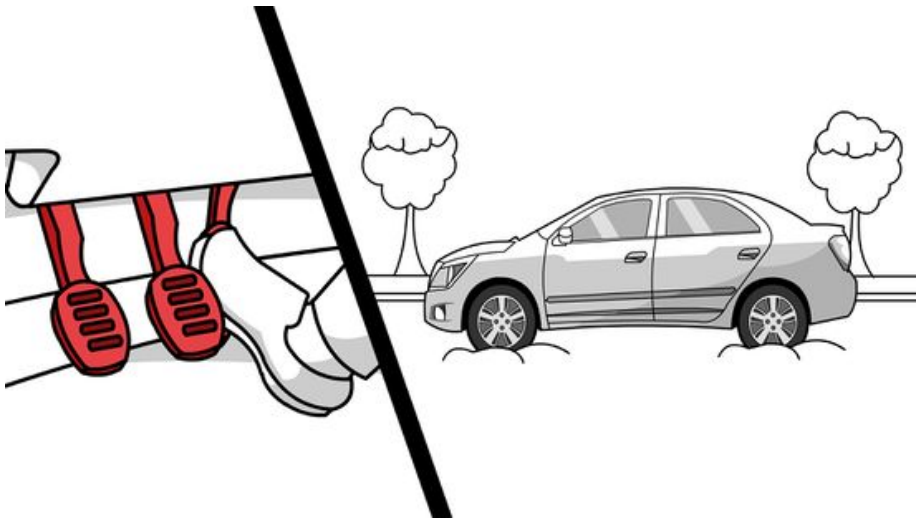
Introdução II

- Suponhamos que queremos **dirigir um carro e acelerá-lo**, de modo que ele fique mais veloz;
- Antes disto, alguém precisa **projetar e construir** o carro;
- O projeto do carro tipicamente **começa com desenhos técnicos**
  - Que incluem o pedal do acelerador que utilizaremos.

Introdução III

- De certa forma, o pedal do acelerador esconde os mecanismos complexos que fazem com que o carro acelere
  - Isto permite que pessoas sem conhecimento de mecânica acelerem um carro;
  - Acelerar é uma "interface" mais amigável com a mecânica do motor.
- Acontece que não podemos dirigir os desenhos técnicos
  - É necessário que alguém construa o carro, com o pedal.

Introdução IV



Introdução V

- Depois de construído, **o carro não andará sozinho**
- **Precisamos apertar o pedal** de acelerar.
- Vamos fazer uma analogia com programação orientada a objetos.

Introdução VI

- Realizar uma tarefa em um programa requer uma função
  - O *main*, por exemplo.
- A **função descreve os mecanismos** que realizam a tarefa
  - **Escondendo toda a complexidade** do processo, assim como o acelerador.
- **Começaremos** pela **criação de uma classe**, que abriga uma função

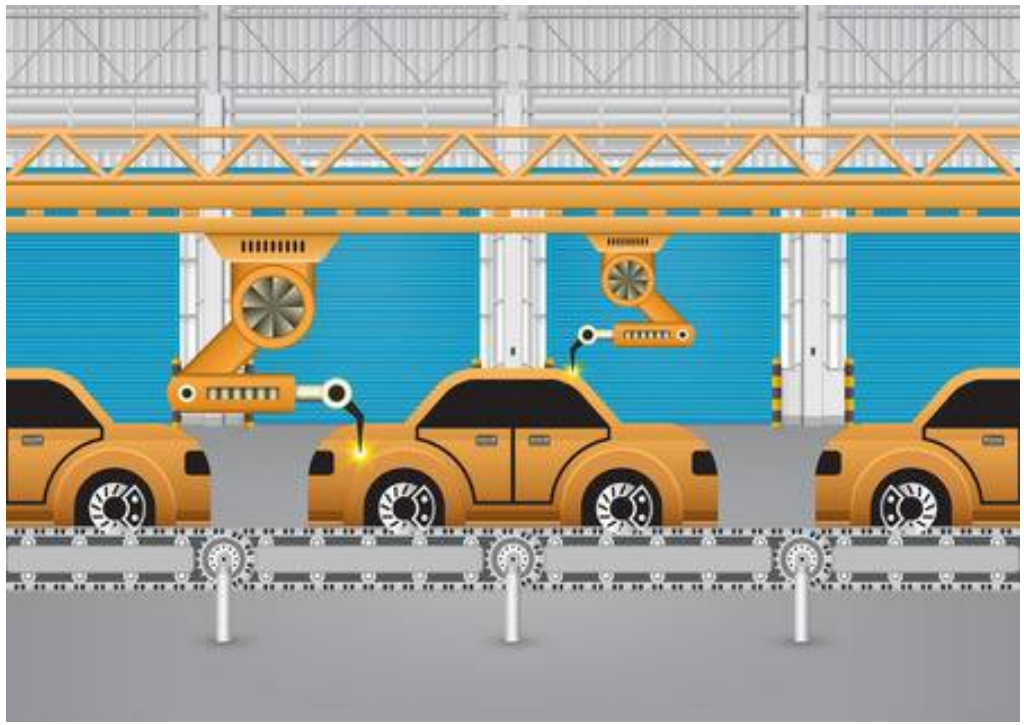
Introdução VII

- A **função** que pertencente a uma classe é **chamada método**
  - São **utilizadas** para **desempenhar as tarefas** de uma classe.
- Da mesma forma que **não é possível dirigir o projeto** de um carro, você **não pode "dirigir" uma classe**;
- É necessário **antes construir o carro** para dirigí-lo;
- É necessário **criar um objeto** de uma classe antes para poder **executar as tarefas** descritas por uma classe.

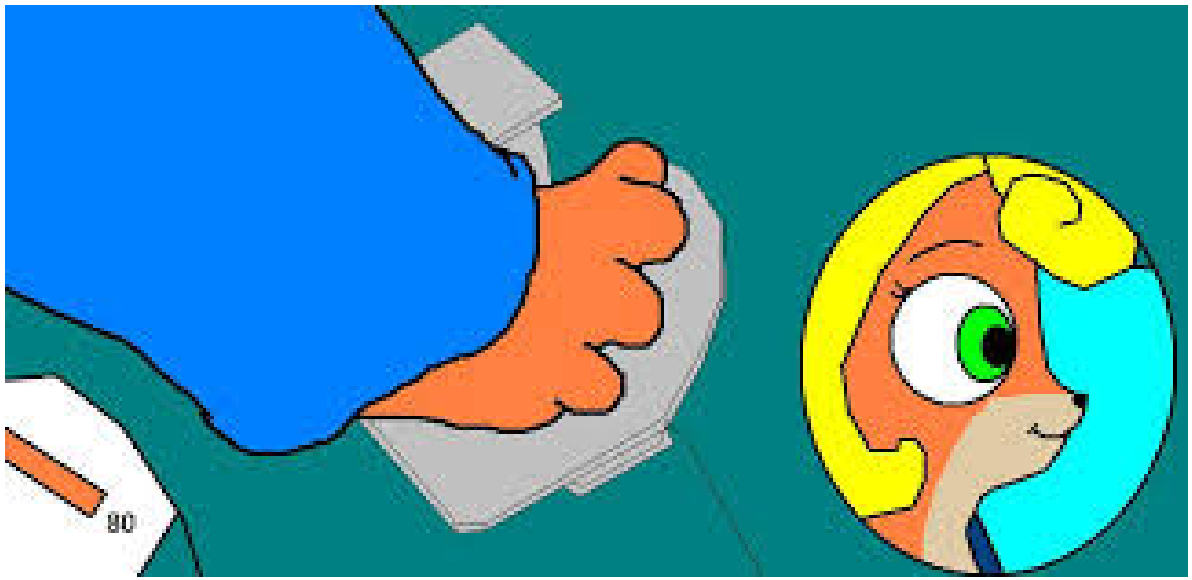
Introdução VIII

- Ainda, **vários carros podem ser criados** a partir do projeto inicial
  - Vários objetos podem ser criados a partir da mesma classe.
- Quando dirigimos um carro, pisar no acelerador **manda uma mensagem** para que o carro desempenhe uma tarefa
  - "Acelere o carro".

Introdução IX



Introdução X

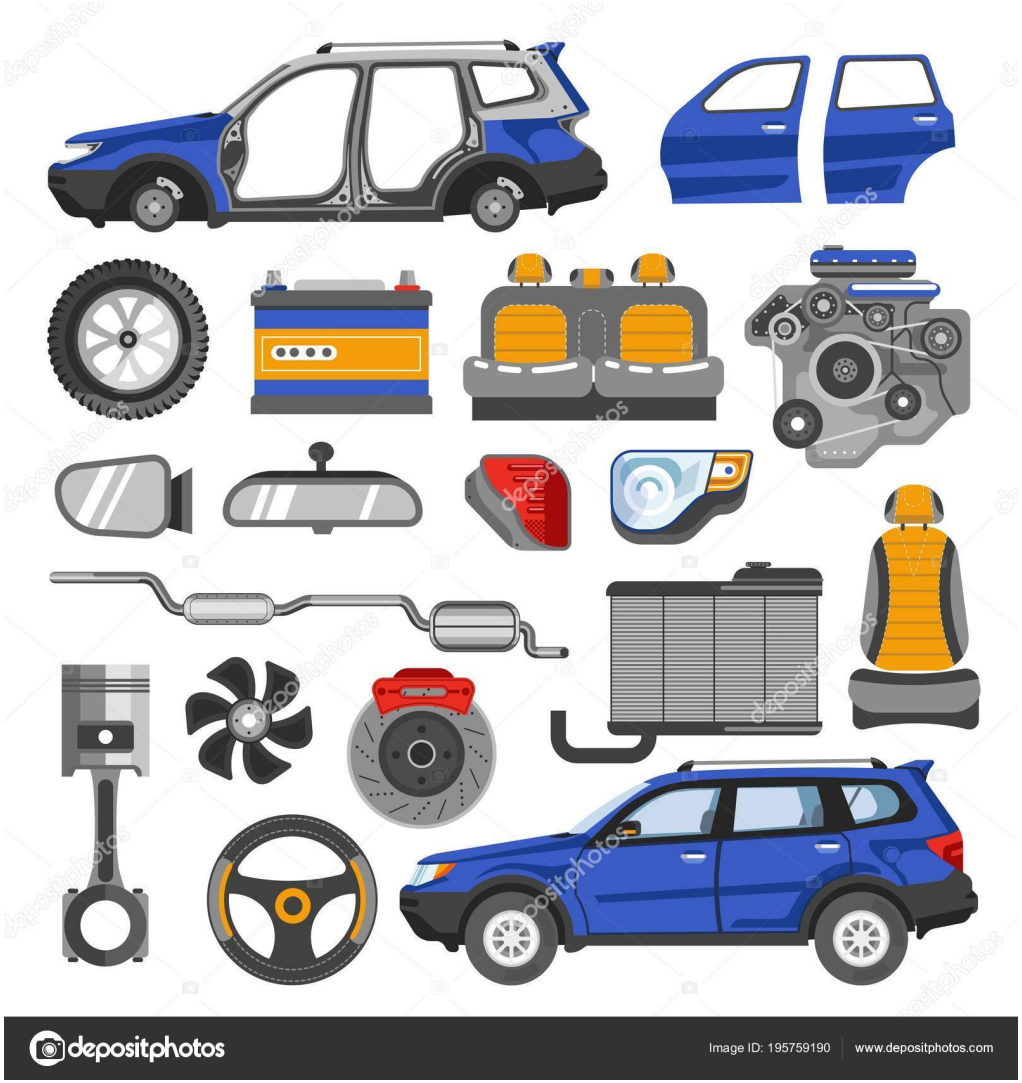


Introdução XI

Introdução XII

- Similarmente, enviamos mensagens aos objetos
  - As chamadas aos métodos;
- Além das competências de um carro, ele possui diversos atributos
  - Número de passageiros;
  - Velocidade atual;
  - Nível do tanque de combustível, etc.

Introdução XIII



Introdução XIV

- Assim como as competências, os atributos também são representados no projeto de um carro
  - Cada carro possui seus **atributos próprios**;
  - Um carro **não conhece os atributos de outro**.
- Da mesma forma ocorre com os objetos
  - Cada objeto tem os seus próprios atributos

Classes I

- Vejamos um exemplo de uma classe que descreve um "diário de classe"
  - Utilizado para manter dados sobre a avaliação de alunos.
- Vejamos também como criar objetos desta classe.

Classes II

- Definir uma classe é dizer ao compilador quais métodos e atributos pertencem à classe
  - A definição começa com a palavra class;
  - Em seguida, o nome da classe
- Por padrão, a primeira letra de cada palavra no nome é maiúscula.
- O corpo da classe é delimitado por { e };
- Não se esqueça do ; no final.

Classes III

- Declaração de uma classe

```
class Nome_Classe{

    atributos ..

public:

    métodos ..

}; // fim da classe
```

Classes IV

```
#include <iostream>
using namespace std;

// Definição da classe DiarioClasse
class DiarioClasse{
    public:
    // método que exibe uma mensagem de boas-vindas
    void mostraMensagem(){
        cout << "Seja Bem-vindo ao Diario de Classe"
             << endl;
    }
};
int main(){
    // cria um objeto da classe DiarioClasse
    DiarioClasse meuDiario;
    // chama ao método mostraMensagem
    meuDiario.mostraMensagem();
    return 0;
}
```

Classes V

- Dentro da classe definimos os métodos e os atributos
  - Separados pelo especificador de acesso (visibilidade)
  - Basicamente, público (public) , privado (private) e protegido (protected).

Classes VI

- A definição de um método se assemelha a definição de uma função
  - Possui valor de retorno;
  - Assinatura do método
  - Lista de parâmetros

Classes VII

- **Assinatura:** por padrão, começa com uma letra minúscula e todas as palavras seguintes começam com letras maiúsculas.
- **Lista de parâmetros:** va entre parênteses (tipo e identificador). Eventualmente, vazia Além disso, o corpo de um método também é delimitado por { e }.

Classes VIII

- O corpo de um método contém as instruções que realizam uma determinada tarefa
- Neste exemplo, simplesmente apresenta uma mensagem

Classes IX

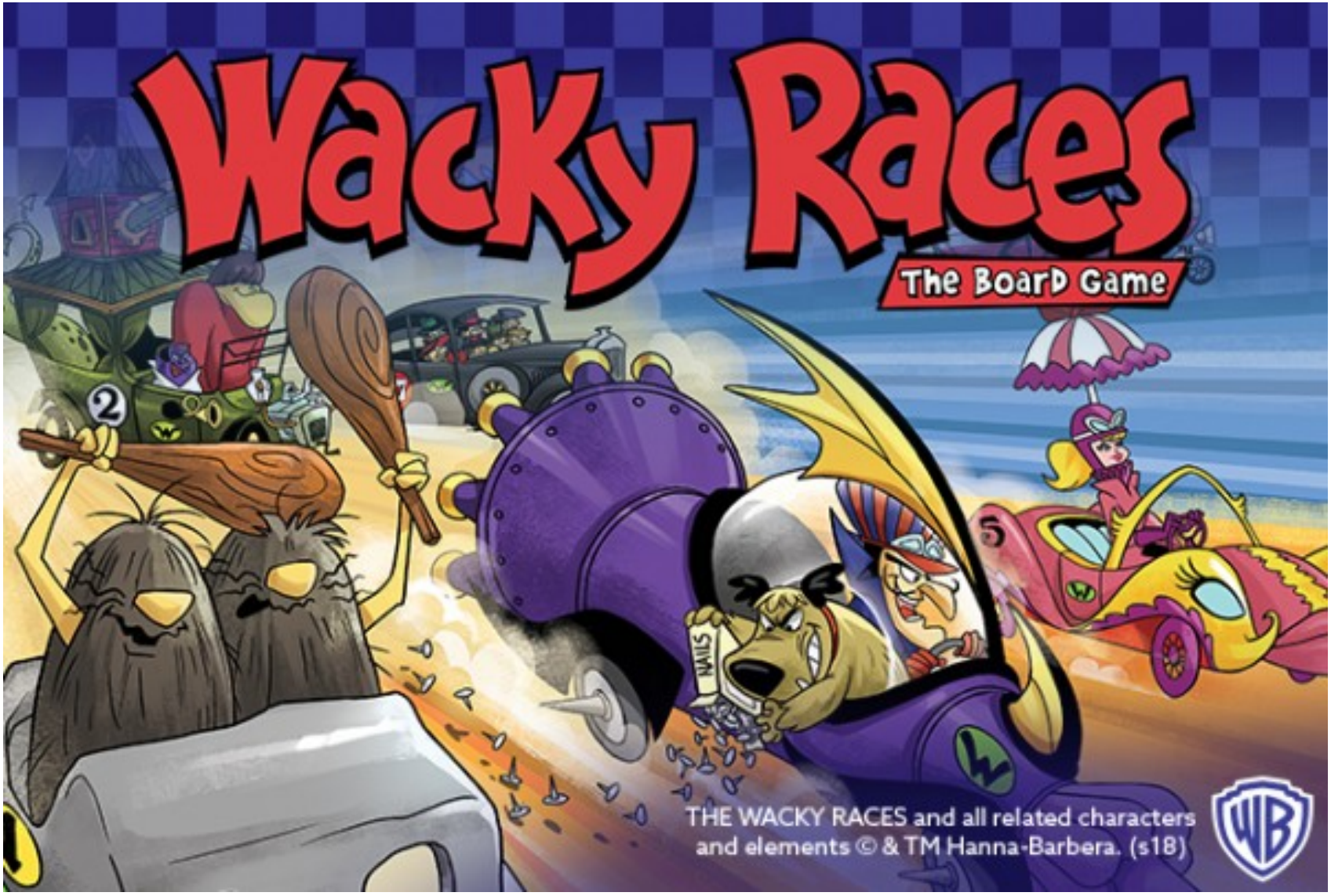
```
#include <iostream>
using namespace std;

class DiarioClasse{
    public:
    void mostraMensagem(){
        cout << "Seja Bem-vindo ao Diario de Classe"
             << endl;
    }
};
int main(){
    // cria um objeto da classe DiarioClasse
    DiarioClasse meuDiario;
    // chama ao método mostraMensagem
    meuDiario.mostraMensagem();
    return 0;
}
```

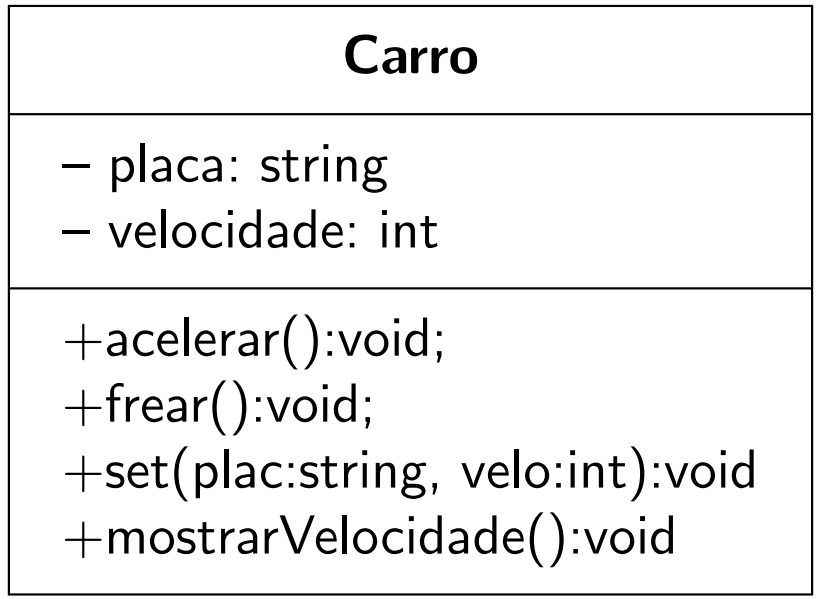
Classes X

- Como seria o diagrama de classe UML para a classe DiarioClasse?

Exemplo Carro I



Exemplo Carro II



Exemplo Carro III

```
#include <iostream>
#include <string>
using namespace std;

class Carro{
    string placa;
    int velocidade;
public:
    void acelerar(){
        velocidade += 10;
    }
    void frear(){
        velocidade -= 10;
    }
    void set(string plac, int velo){
        placa = plac;
        velocidade = velo;
    }
```

Exemplo Carro IV

```
    void mostrarVelocidade(){
            cout << velocidade << "Km/h\n";
        }
};
int main(){
    Carro herbie;
    herbie.set("THY-8888", 50);
    herbie.acelerar();
    herbie.acelerar();
    herbie.mostrarVelocidade();
    herbie.frear();
    herbie.mostrarVelocidade();
    return 0;
}
70 Km/h
60 Km/h
```

Getters e Setters I

- Atributos são definidos como privados
- Só podem ser alterados dentro da própria classe
- Tentar acessá-los fora da classe causará erro.
- Ocultação de informação;

Getters e Setters II

- Um método que altere o valor de um atributo é chamado de **setter**
  - Por padrão, a nomenclatura é set+[nome do atributo].
- Um método que retorne o valor de um atributo é chamado de getter
  - Por padrão, a nomenclatura é get+[nome do atributo].

Getters e Setters III

- Uma vez que nossa classe possui *getters* e *setters*, os atributos só devem ser acessados e alterados por eles
- **Mesmo dentro de outros métodos** que porventura necessitem acessar/alterar os atributos.
- Com *getter* e *setter*.

Getters e Setters IV

```
#include <iostream>
#include <string>
using namespace std;

class Carro{
    string placa;
    int velocidade;
public:
    void acelerar(){
        velocidade += 10;
    }
    void frear(){
        velocidade -= 10;
    }
    void setPlaca(string plac){
        placa = plac;
    }
    void setVelocidade(int velo){
        velocidade = velo;
    }
```

Getters e Setters V

```
    string getPlaca() const{
        return placa;
    }
    int getVelocidade() const{
        return velocidade;
    }
    void mostrarVelocidade(){
        cout << getVelocidade() << "Km/h\n";
    }
};
```

Getters e Setters VI

```
int main(){
    Carro herbie;
    herbie.setPlaca("THY-8888");
    herbie.setVelocidade(50);
    herbie.acelerar();
    cout << "\nVelocidade: "
         << herbie.getVelocidade() << "Km/h";
    cout << "\nPlaca: " << herbie.getPlaca();
    return 0;
}
```

```
Velocidade: 60 Km/h
Placa: THY-8888
```

Getters e Setters VII

- Para garantir a consistência dos valores dos atributos, convém realizar **validação de dados** nos getters e setters
  - O valor passado como parâmetro é adequado em relação ao tamanho ou faixa de valores?
  - Temos que definir se permitiremos valores negativos, tamanho máximo para vetores, etc.
  - Se um parâmetro for inadequado, devemos atribuir um valor padrão ao atributo
  - Zero, um, NULL, ”, etc.

Getters e Setters VIII

- Notem um detalhe interessante:
  - Quando usamos a classe string, podemos fazer atribuição direta entre os objetos, utilizando o operador de atribuição
  - De fato, podemos fazer atribuição direta entre quaisquer objetos de uma mesma classe
  - **Cuidado:** pode causar erros se entre os atributos possuirmos ponteiros para memória alocada dinamicamente.
  - Diferentemente do que ocorre com vetores de caracteres, que precisam da função strcpy.

Exercicio I

Implemente a classe Retângulo e as operações para calcular a área e perímetro.

Exercicio II

```
#include <iostream>
using namespace std;
class Retangulo{
    double altura, base;
    public:
    void setAltura(double m_altura){
        altura = m_altura;
    }
    void setBase(double m_base){
        base = m_base;
    }
    double getAltura(){return altura;}
    double getBase(){return bas;}
    double calculaArea(){
        return getAltura() * getBase();}
    double calculaPerimetro(){
        return 2 * getAltura() + 2 * getBase();}
}; // fim da classe
```

Exercicio III

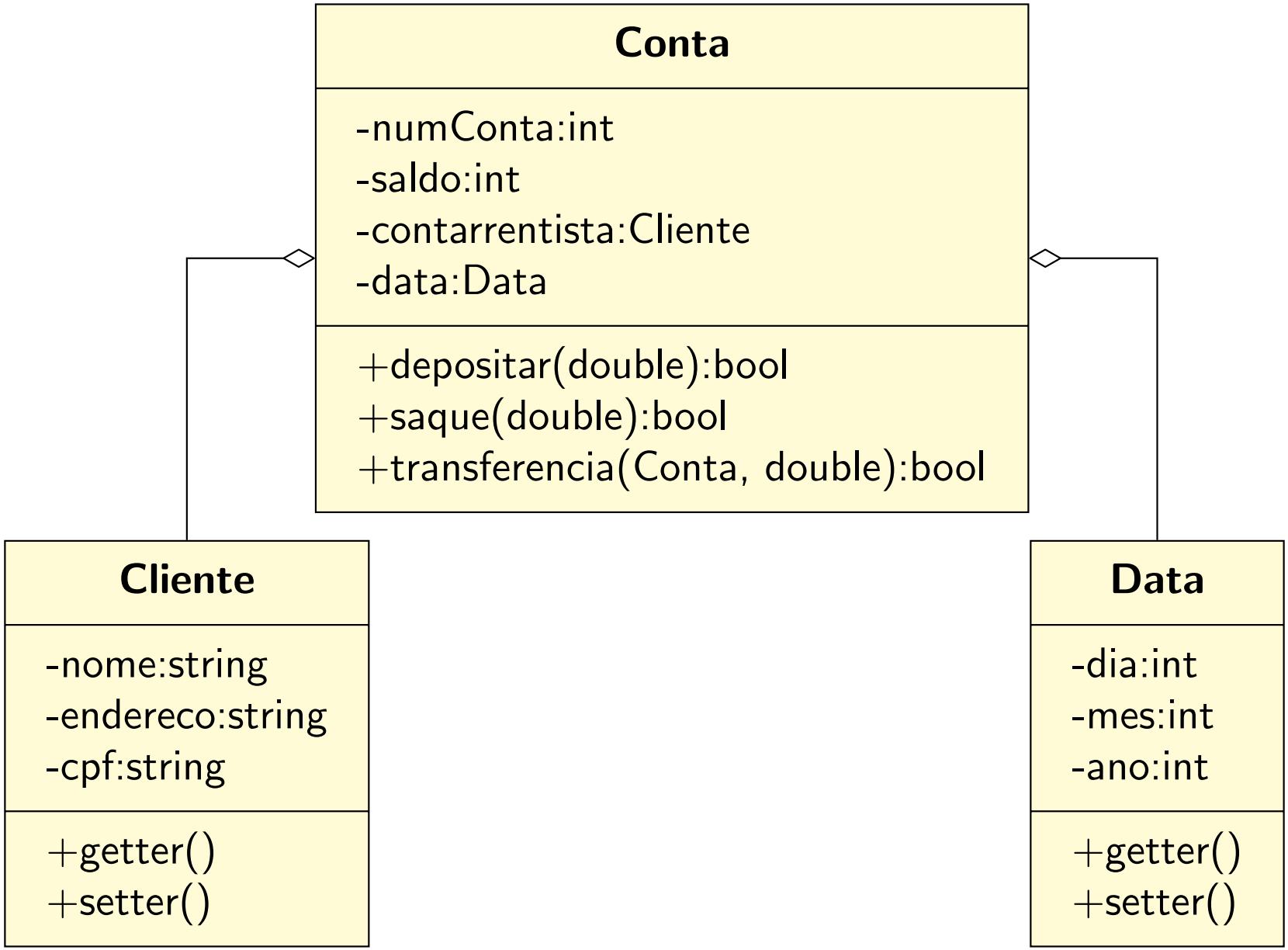
```
int main(){
    Retangulo R;
    double m_base, m_altura;
    cout << "Digite a base e altura do retangulo: ";
    cin >> m_base >> m_altura;
    R.setBase(m_base);
    R.setAltura(m_altura);
    cout << "Area: " << R.calculaArea() << endl;
    cout << "Perimetro: " << R.calculaPerimetro() << endl;
    return 0;
}
```

Exercicio I

Escreva uma ou mais classes para calcular o saldo de uma conta corrente que pertence a um correntista. Considere:

1. Toda conta se abre com pelo menos 200 reais
2. O sistema deverá permitir fazer saques, depósitos e transferências
3. O saldo nunca pode ser negativo
4. A conta corrente deve incluir a data de abertura da conta

Exercicio II



Exercicio III

```
#include <iostream>
using namespace std;
class Conta{
    int numConta;
    double saldo;
    // para simplificar, as classes Cliente e Data
    // serao tratadas como strings
    string cliente;
    string data;
public:
    // construtor verificará monto mínimo ...
    bool depositar(double valor){
        if (valor > 0){
            saldo += valor;
            return true;
        }
        return false;
    }
```

Exercicio IV

```
bool saque(double valor){
    if (saldo > valor && valor > 0){
        saldo -= valor;
        return true;
    }
    return false;
}
bool transferencia(Conta& destino, double valor){
    if ( saque(valor) ){
        destino.depositar(valor);
        return true;
    }
    return false;
}
void mostrarDados() const{
    // mostrar todos os atributos
    cout << numConta << " " << saldo
        << " " << cliente << " "
        << data << endl;
}
```

Exercicio V

```
    void set(int mconta, double msaldo, string mcliente,
             string mdata){
        numConta = mconta;
        saldo = msaldo;
        cliente = mcliente;
        data = mdata;
    }
};  // fim da classe
```

Exercicio VI

```
#include<Conta.h>
using namespace std;
int main(){
    Conta conta1, conta2;
    conta1.set(100, 500.0, "Manoel Dias", "01/03/2019");
    conta2.set(101, 1000.0, "Alex Vargas", "01/01/2019");
    conta1.mostrarDados();
    conta2.mostrarDados();
    conta1.transferencia(conta2, 200.0);
    cout << "Apos a transferencia de 200 reais";
    conta1.mostrarDados();
    conta2.mostrarDados();
    return 0;
}
```

Exercicio VII

```
100 500 Manoel Dias 01/03/2019
101 1000 Alex Vargas 01/01/2019

Apos a transferencia de 200 reais

100 300 Manoel Dias 01/03/2019
101 1200 Alex Vargas 01/01/2019
```

FIM