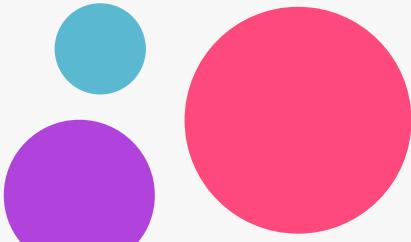


Classe e objeto

Aula 07 - Programação orientada a objetos

• • •

Professor Daniel Marques



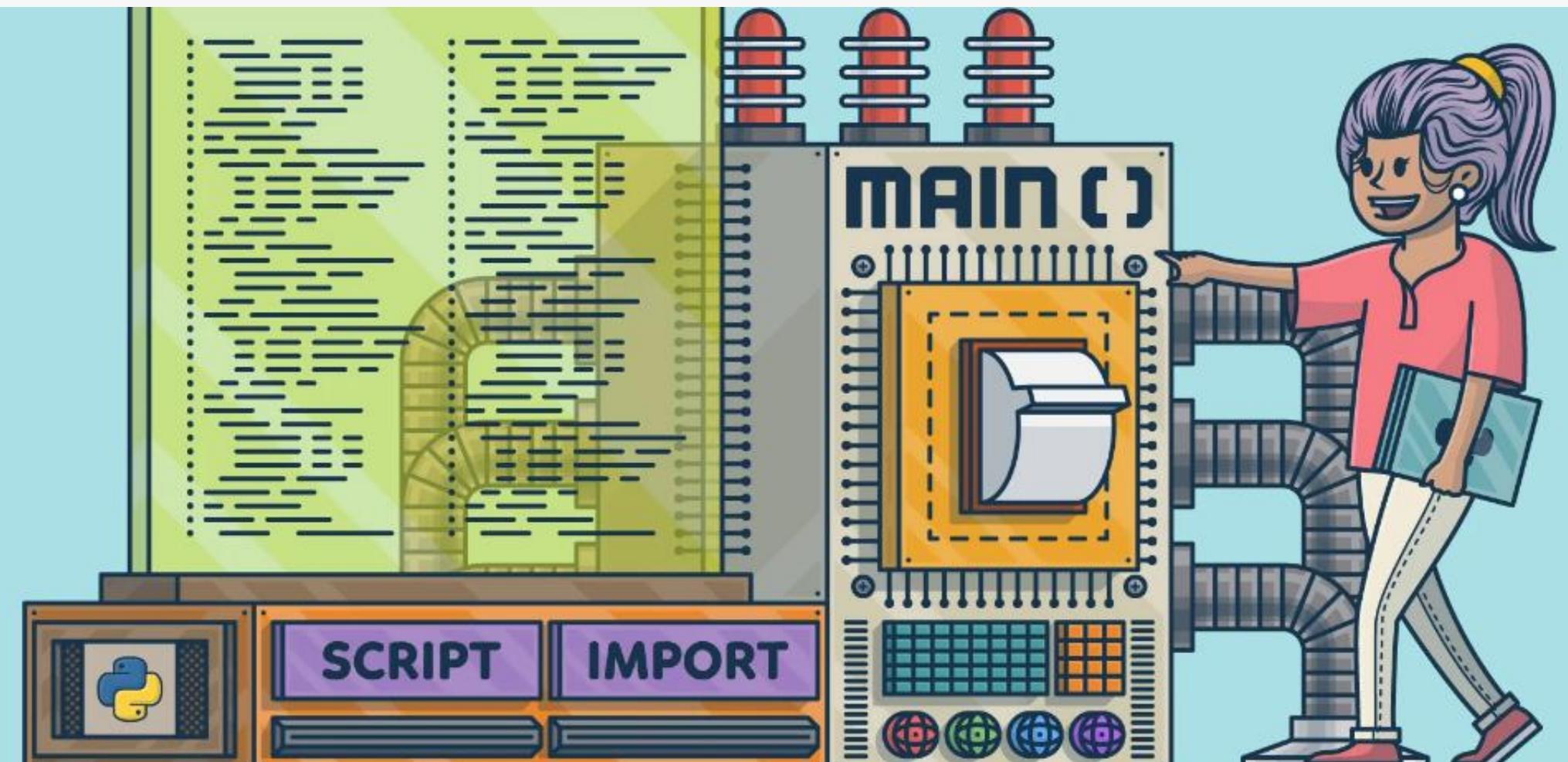
1

Introdução

Introdução

Até agora estamos habituados a criar programas que:

- Apresentam mensagens ao usuário;
- Obtêm dados do usuário;
- Realizam cálculos e tomam decisões;
- Todas essas ações delegadas a função main e outras.



-
- De agora em diante, nossos programas terão uma função main e uma ou mais classes;
 - Cada classe consistindo de dados e operações.

Introdução

De agora em diante,
nossos programas terão
uma função main e uma
ou mais classes;



Cada classe consistindo de dados e operações.

Conceito de classes e objetos

Classe: modelo para criação de objetos

Objeto: entidade física que possui características e realiza ações



O que é um objeto?

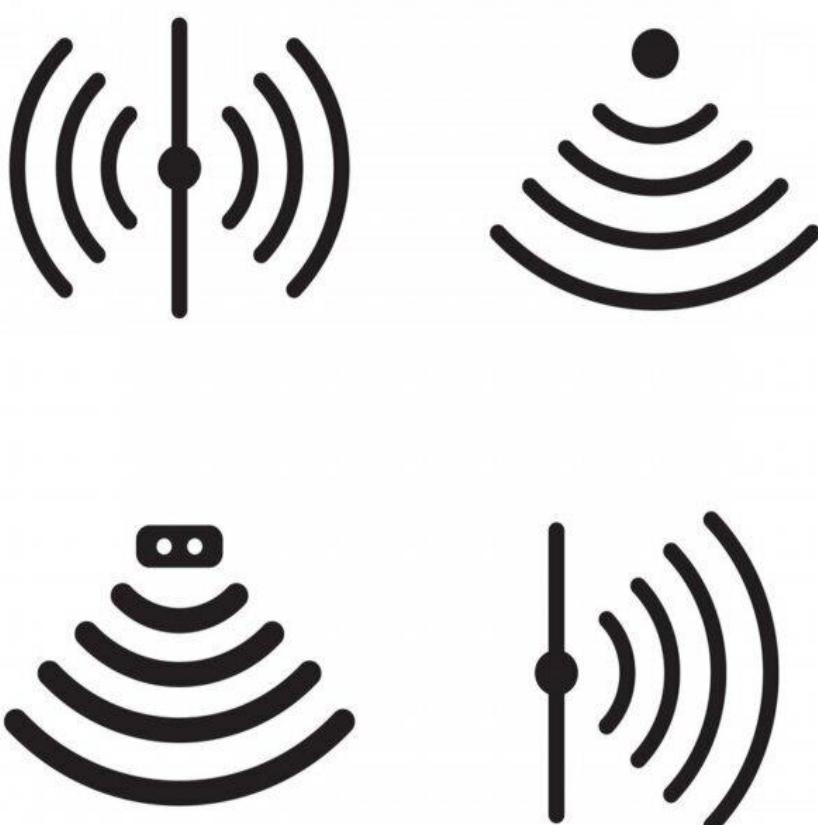
Para o ser humano, um objeto é:

- **Algo tangível e/ou visível**
- **Algo que possa ser compreendido intelectualmente;**
- **Algo para o qual um pensamento ou ação possa ser direcionado**



-
- **Um objeto modela alguma parte da realidade e é algo que existe no tempo e no espaço;**
 - **Utilizamos o conceito de objetos para entender melhor o mundo que nos cerca.**

O que é um objeto?



Objetos podem ser físicos:

- **Aluno, máquina, sensor, pessoa**

Para um programador

“Um objeto representa uma entidade, unidade ou item identificável, individual, real ou abstrato, com um papel bem definido no domínio do problema.” [Booch, 91]

Objetos podem não ser físicos

- **Reação química, número complexo**

O que é um objeto?



Nem tudo é um objeto

- Tempo, métricas, sensações e sentimentos humanos não são, em geral, modelados como objetos
- Eles são, em geral, tratados como propriedades de objetos

Domínio do problema

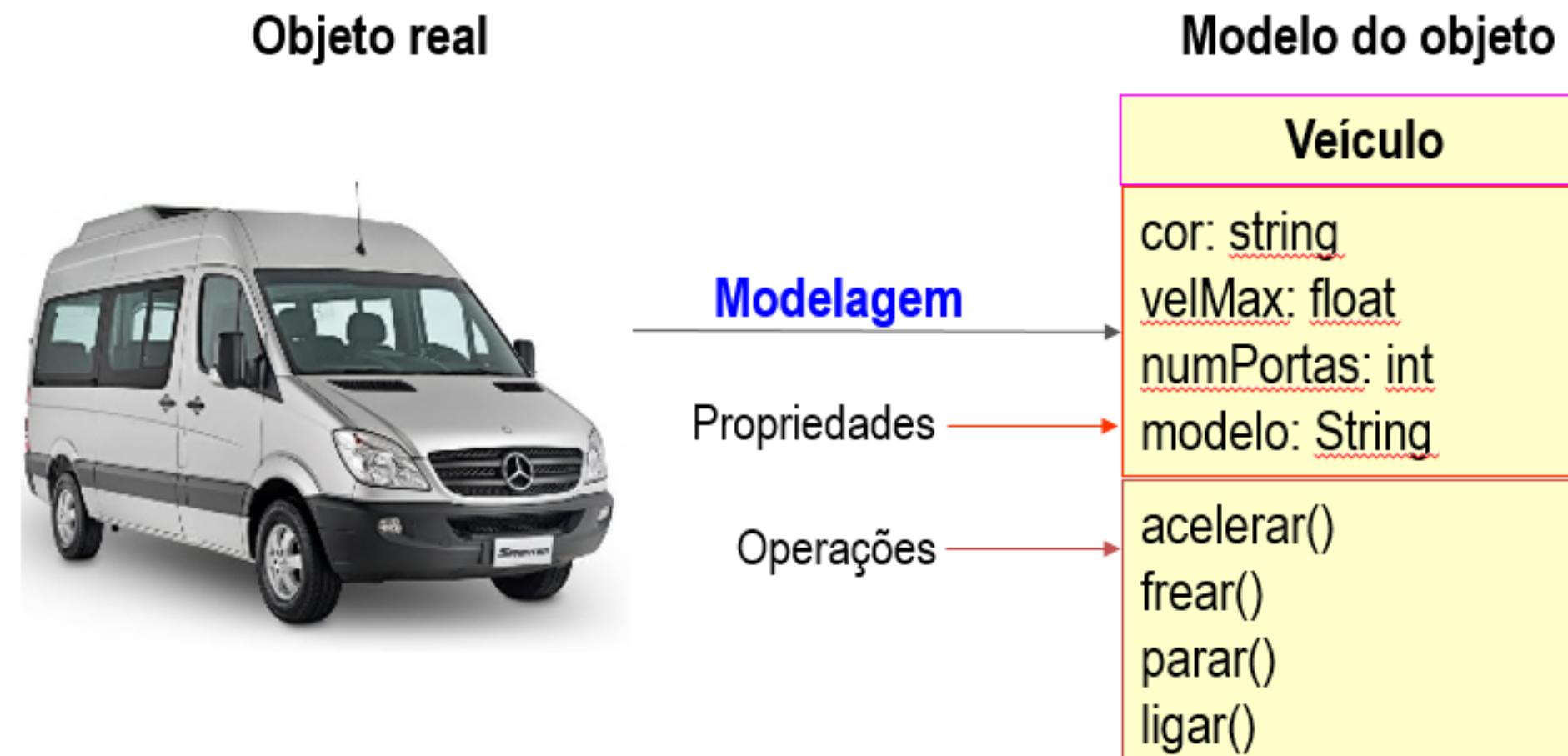


Descobrir quais entidades fazem parte do domínio do problema e identificar suas propriedades e comportamentos é tarefa do programador e dos analistas de sistemas

O que é um objeto?

Abstração

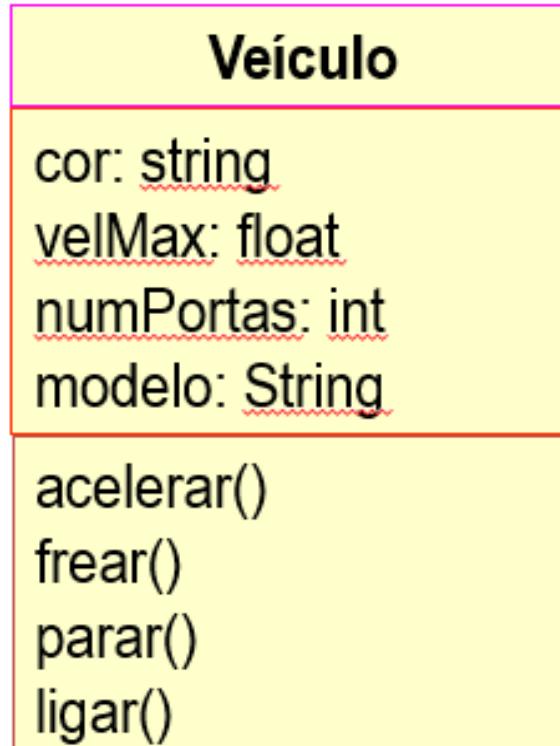
Como entidades do domínio do problema se transformam em objetos de software?



O que é um objeto?

Como entidades do domínio do problema se transformam em objetos de software?

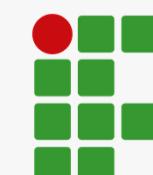
Abstração do objeto



Codificação

Implementação da abstração

```
class Veiculo {  
    private:  
        string cor;  
        float velMax;  
        int numPortas;  
        string modelo;  
  
    public:  
        void acelerar() {  
            ...  
        }  
        ...  
}
```





O que é um objeto?

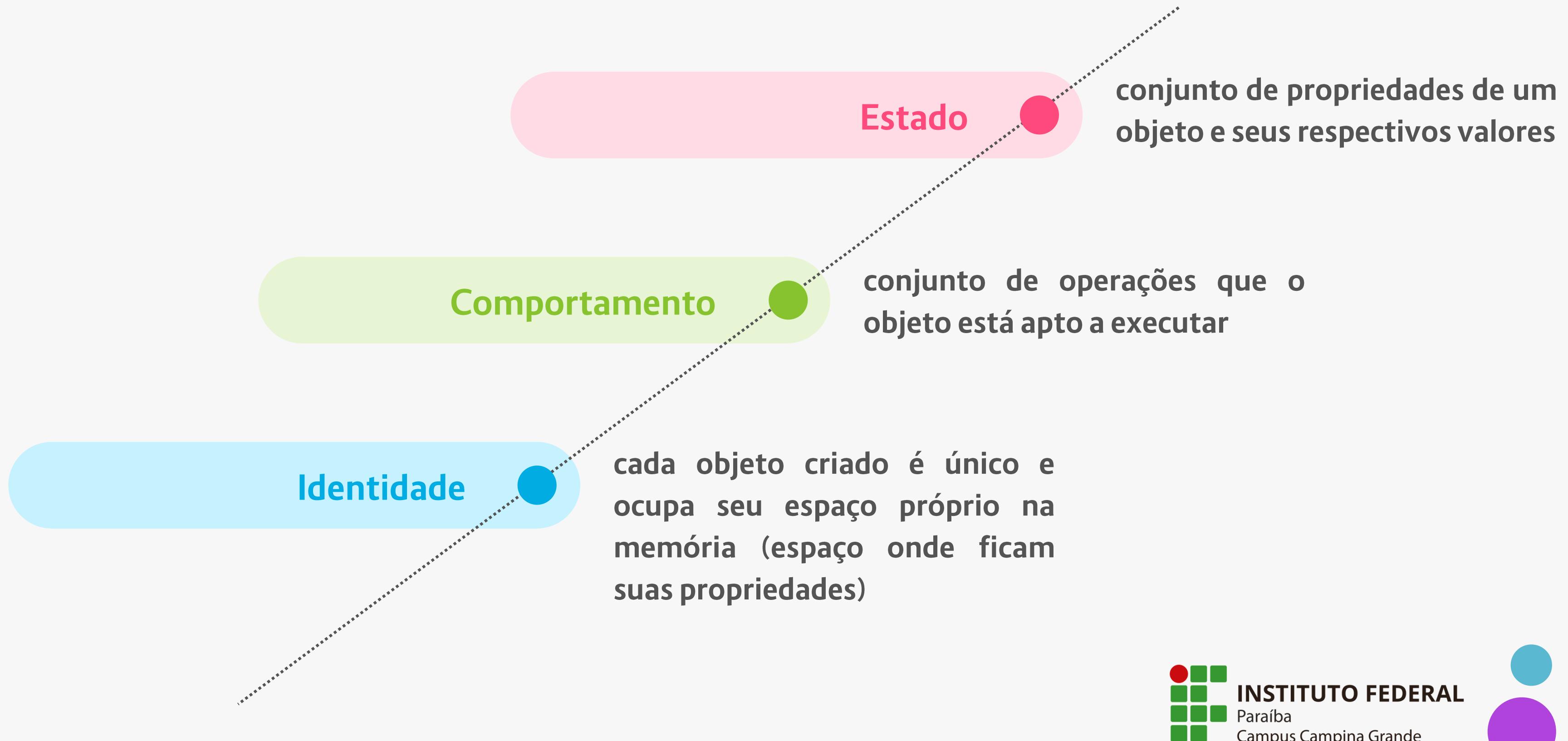
Para POO, um objeto é:

Um objeto é um bloco de software que possui um estado e uma identidade e para o qual podem ser solicitados serviços através de mensagens

“O estado de um objeto consiste de todas as propriedades do objeto mais os valores atuais destas propriedades” [Booch, 91]

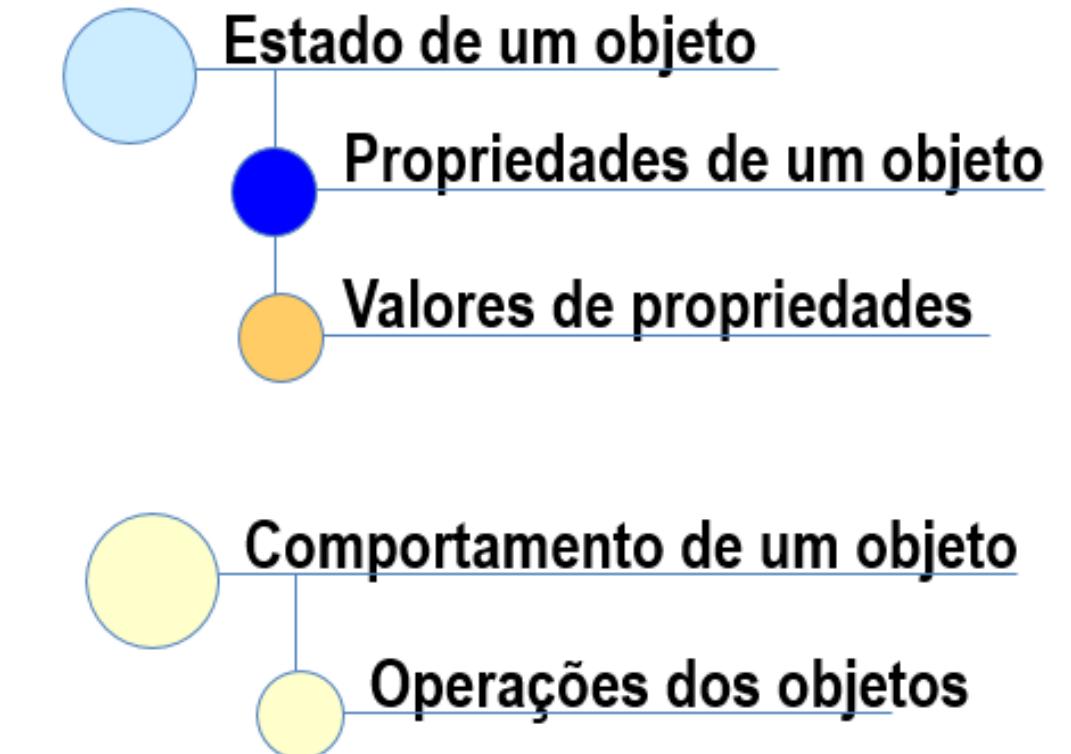
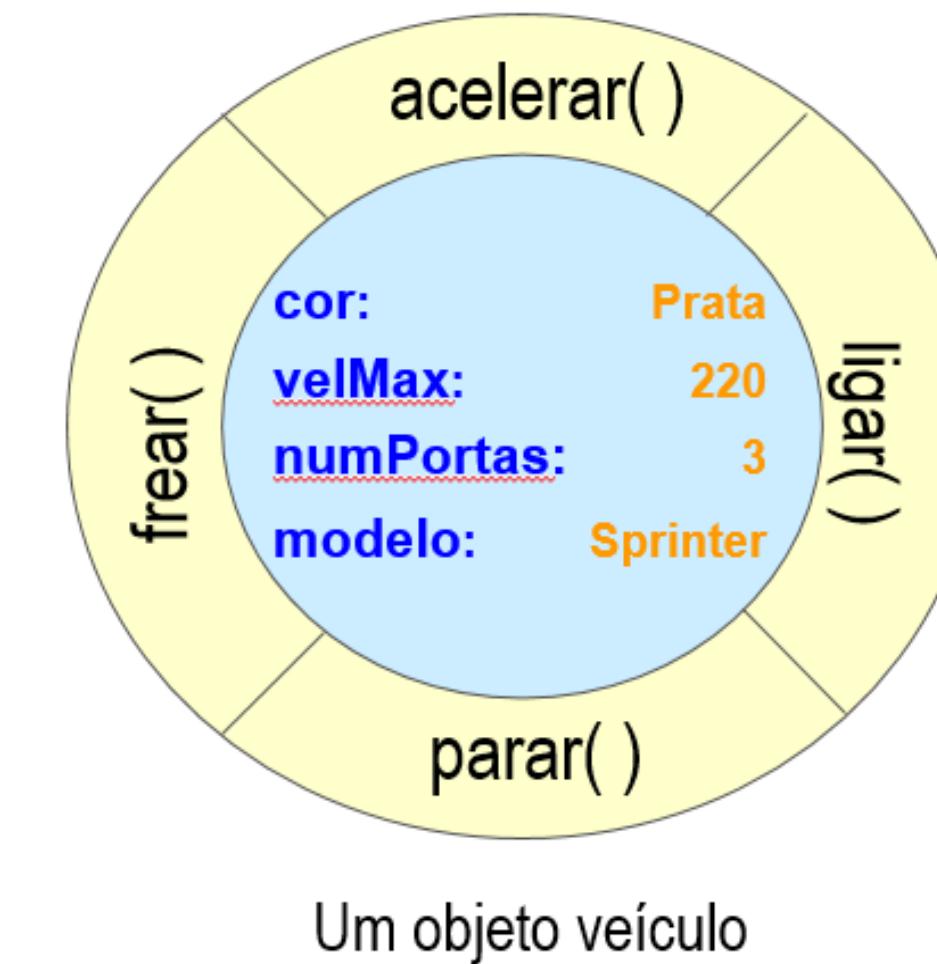
O que é um objeto?

Características chaves de um objeto:



O que é um objeto?

Estado e comportamento



O que é um objeto?



carroDeClaudio : Carro

- **cor = "verde"**
- **velMax = 140**
- **qntdPortas = 3**
- **modelo = "kombi"**

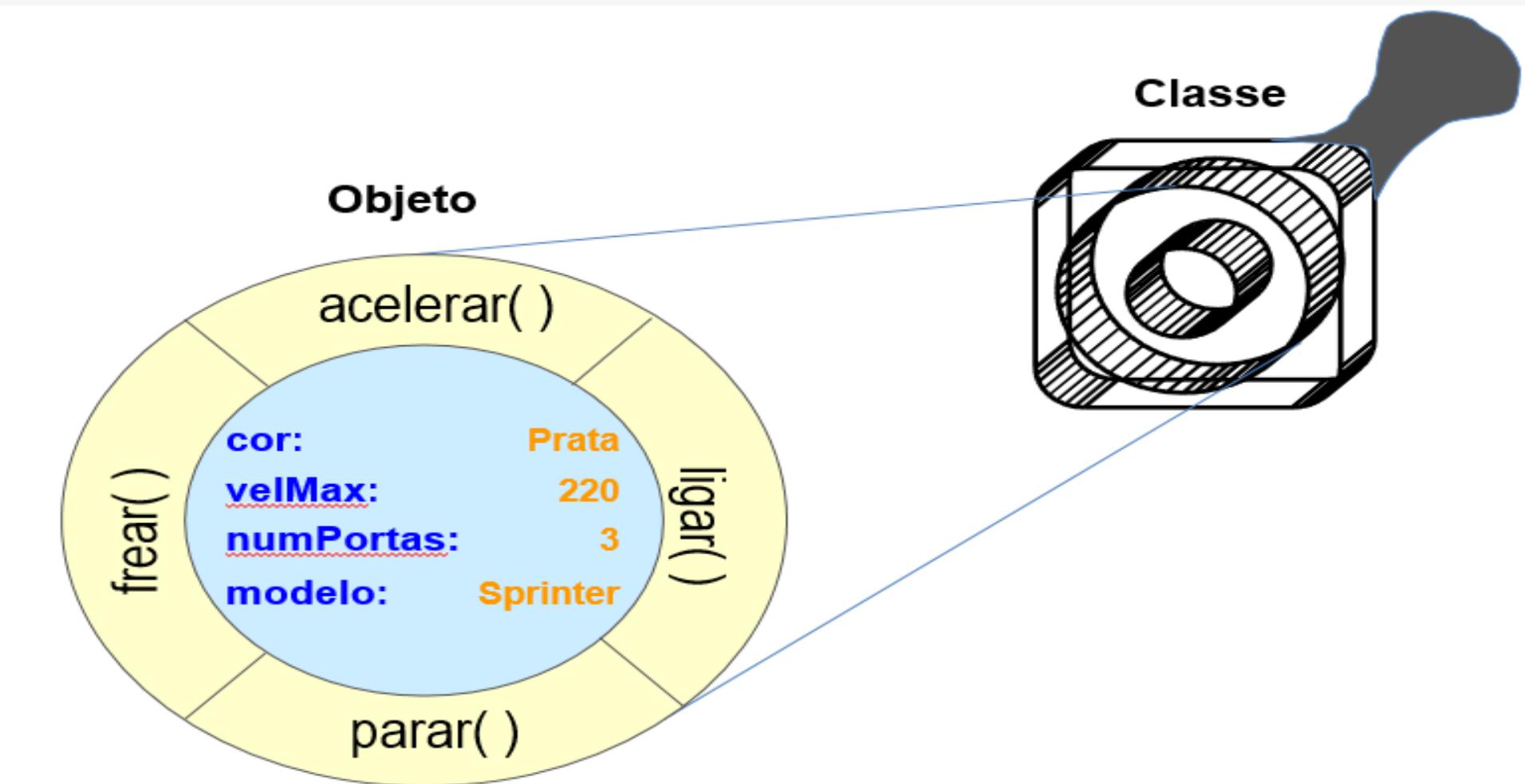
carroDeAmaury : Carro

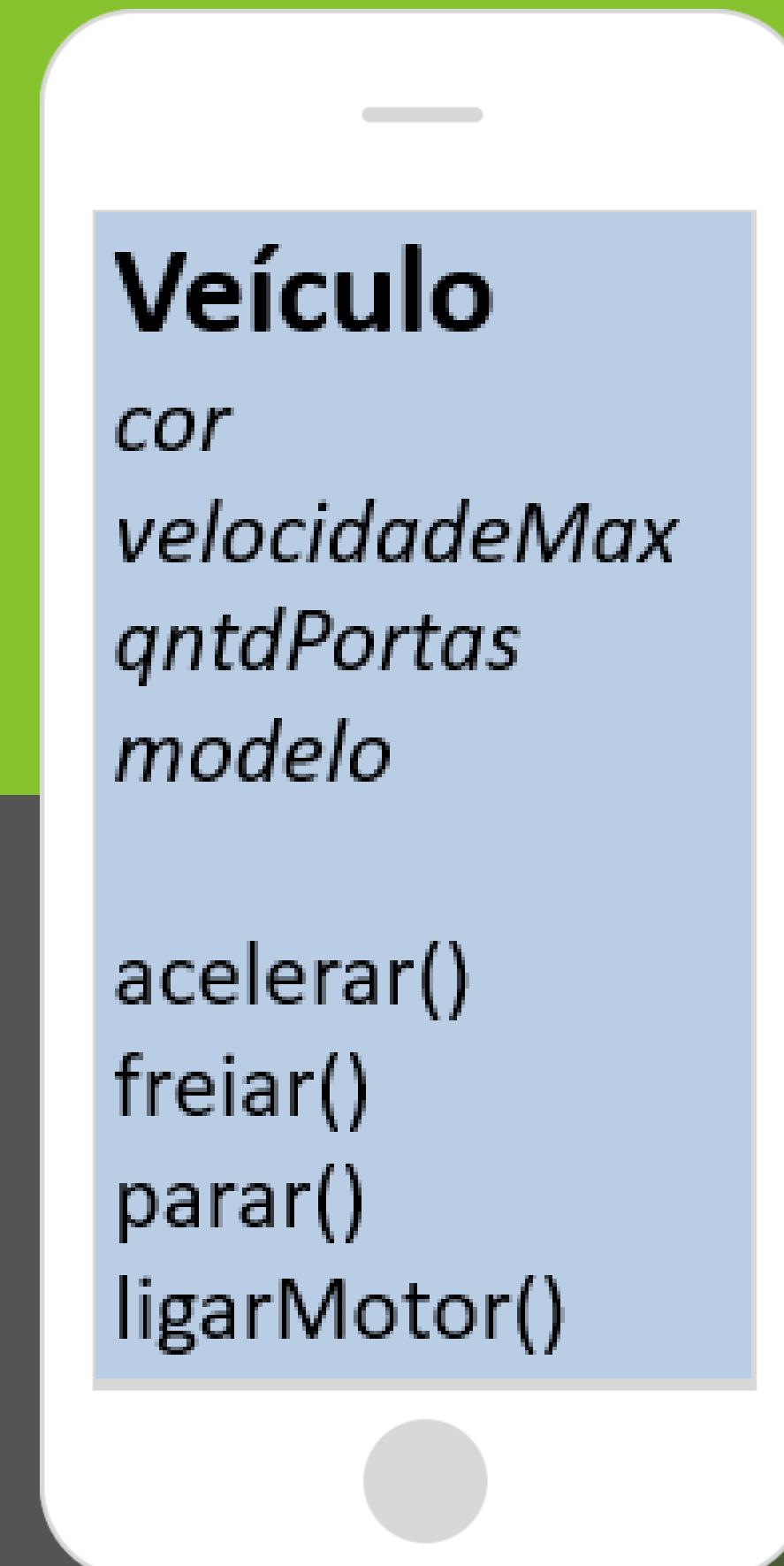


- **cor = "amarelo"**
- **velMax = 100**
- **qntdPortas = 2**
- **modelo = "fusca"**

Como é criado um objeto?

Objetos são criados
através de classes





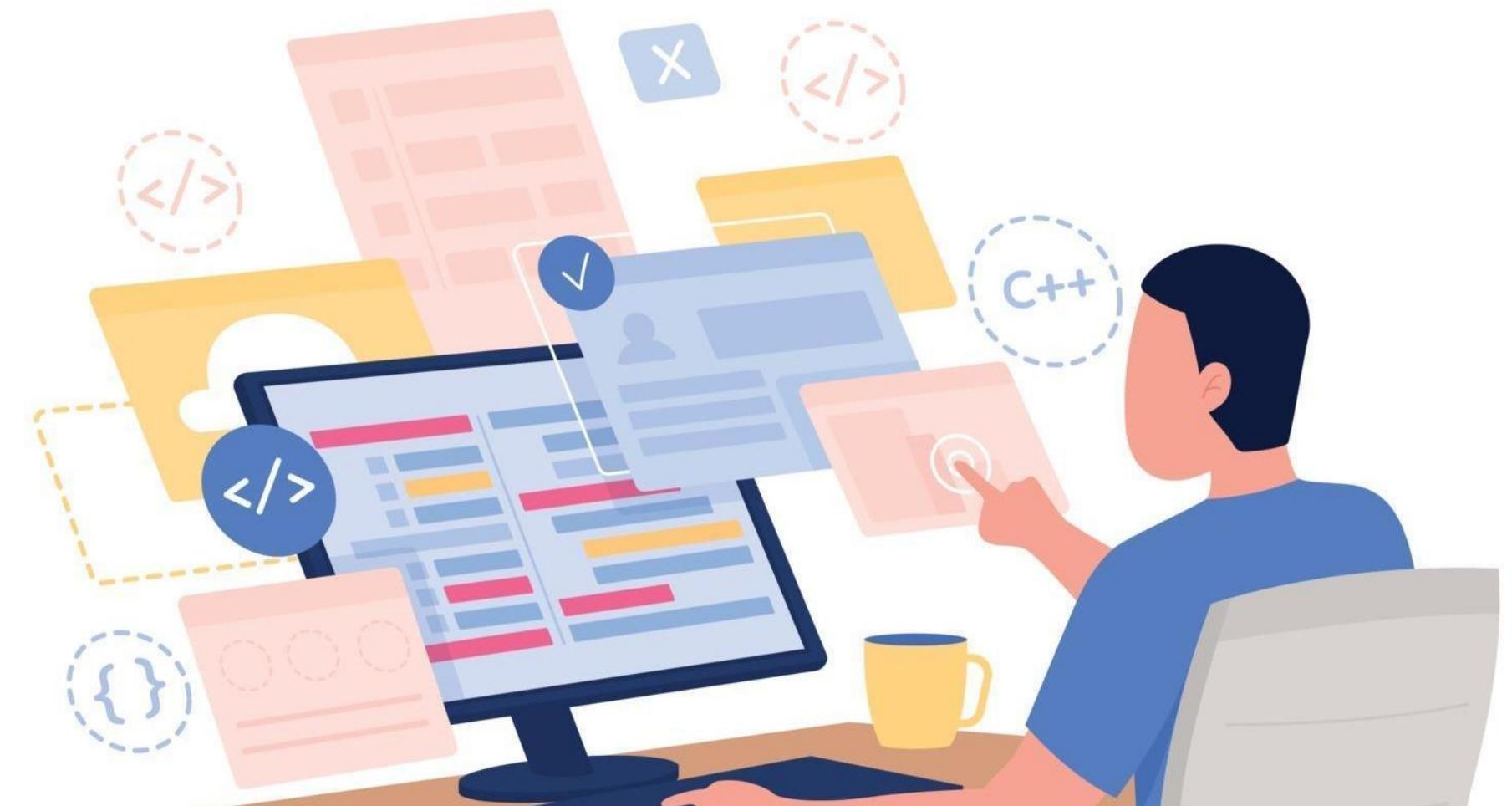
O que é uma classe?

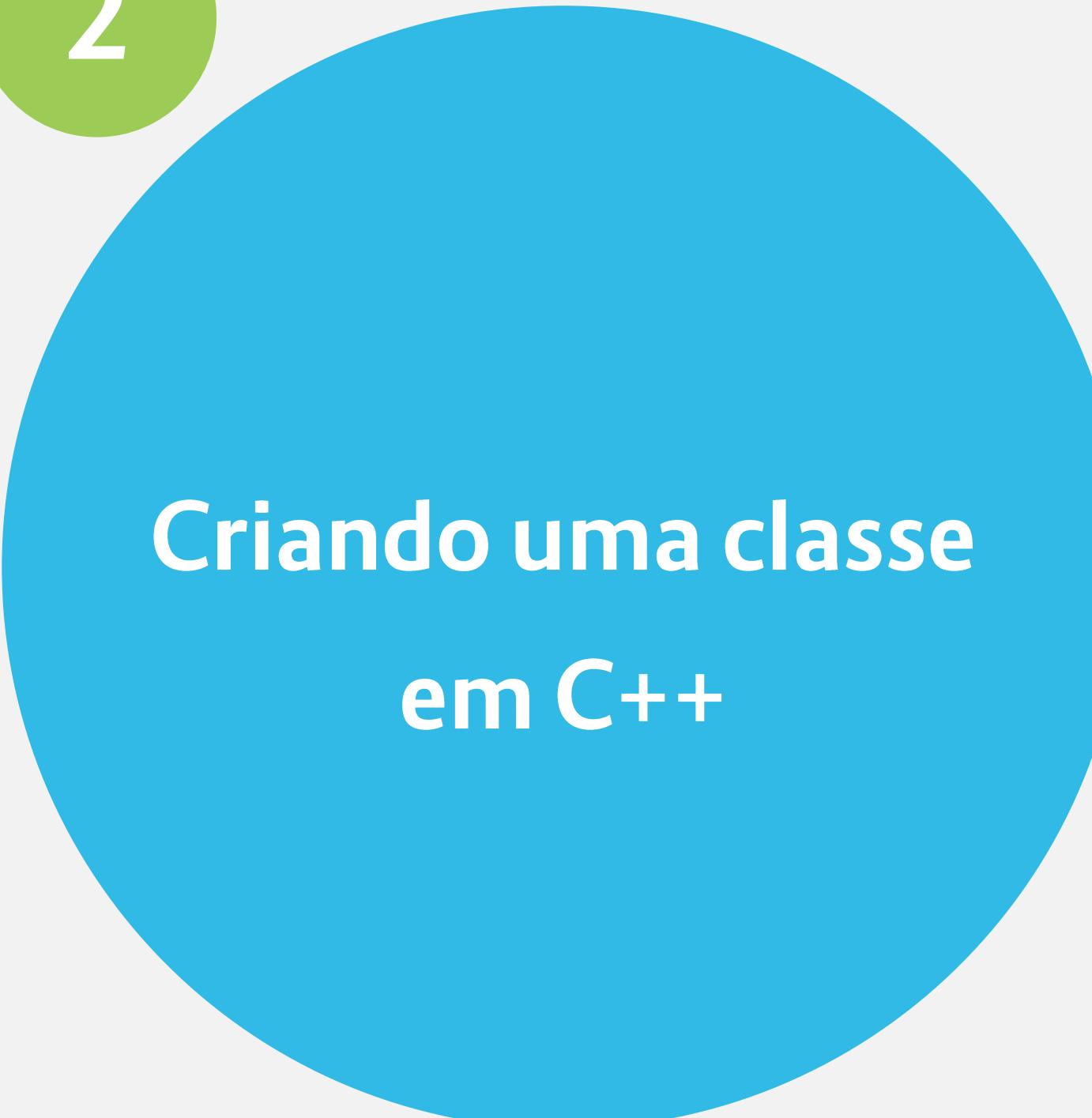
É a abstração de um determinado tipo de objeto em um determinado contexto

O que é uma classe?

Uma classe deve:

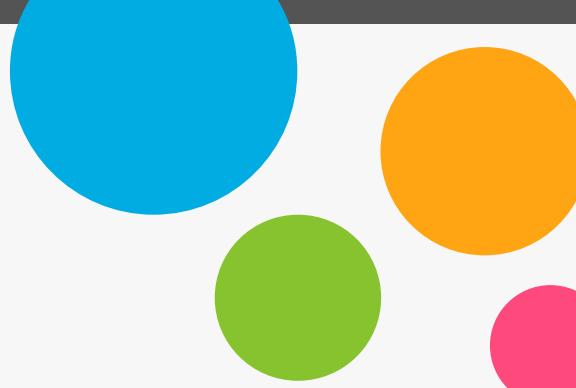
- **Representar um conceito específico;**
- **Prover uma interface bem definida para que ela seja utilizada;**
- **Ser completa e bem documentada.**





2

Criando uma classe em C++



O que é uma classe?

Uma classe tem:

```
public:  
    Veiculo();  
    void acelerar(double aumVel);  
    void frear(double dimVel);  
    double obterVelocidade();
```

```
};
```

Métodos (funções membro)

- Expressam o comportamento do objeto (interface);
- As ações que um objeto realiza para manipular seus dados .

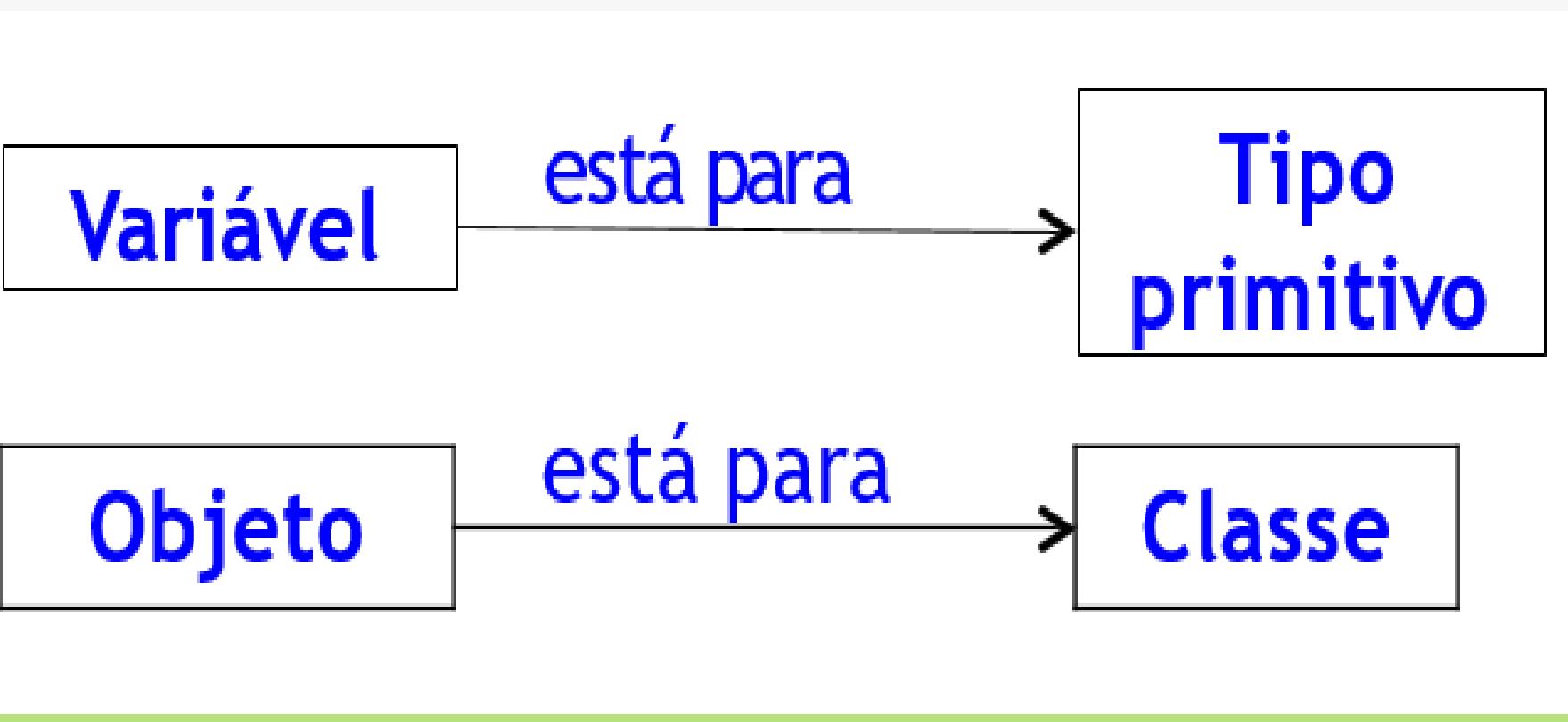
Propriedades / Atributos

Dados (variáveis)

```
class Veiculo {  
  
    private:  
        string modelo;  
        int anoFabricacao;  
        double velocidade;
```

Valores que representam características do objeto.

O que é uma classe?



Classe

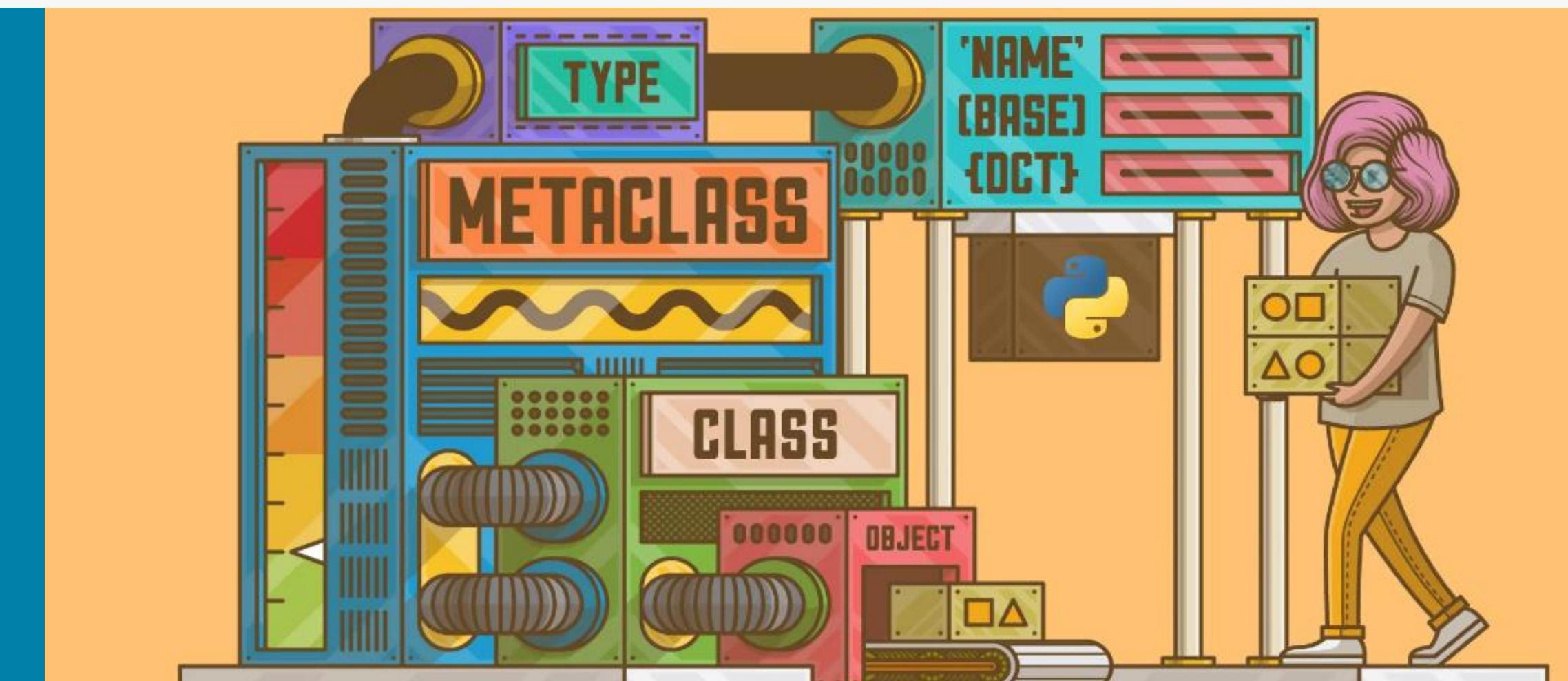
Uma classe também pode ser vista como um tipo definido pelo programador

Exemplo

```
// x é uma variável do tipo int  
int x;  
  
// y é um objeto da classe Veiculo  
Veiculo y = Veiculo();
```

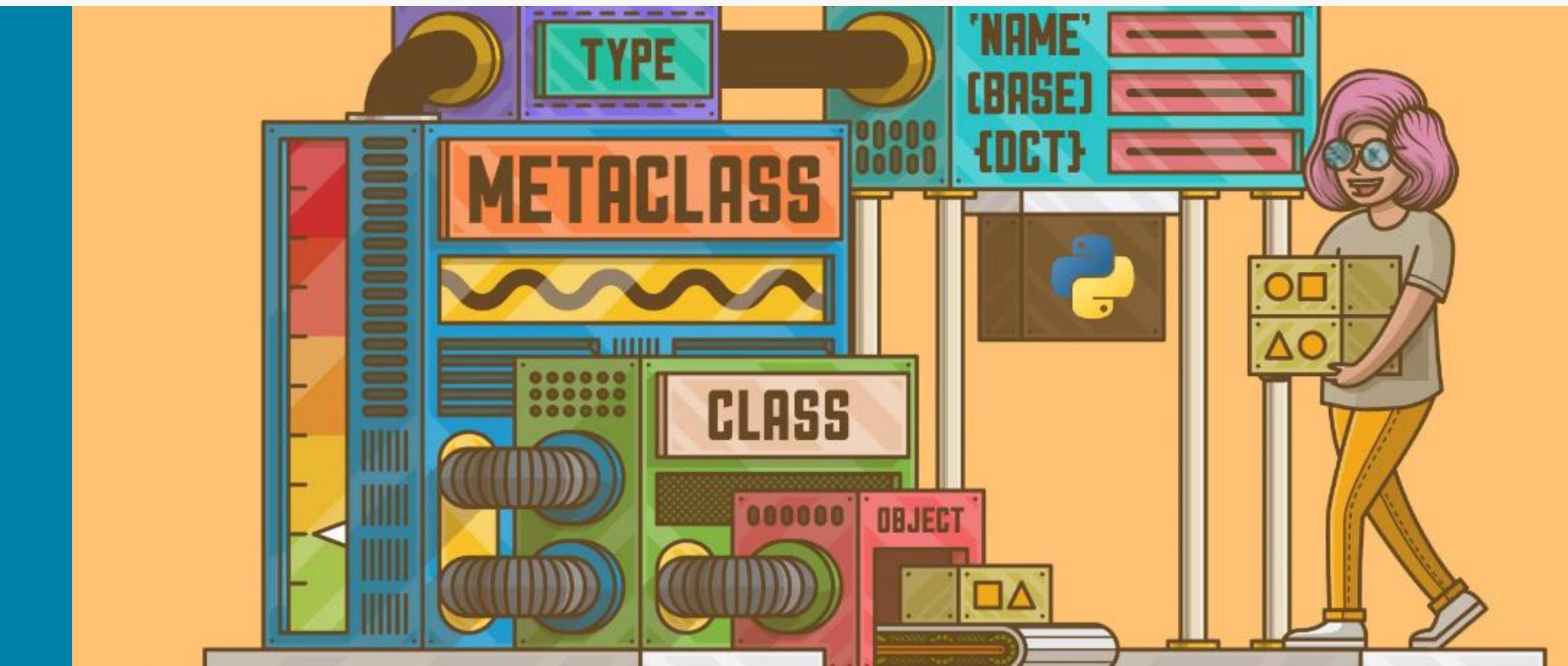
Como um objeto é criado?

É criado por uma operação de instanciação sobre uma classe



O que é uma classe?

É a unidade lógica de programação do paradigma OO onde são definidas as propriedades e métodos que uma categoria de objetos terá.



- A classe é uma espécie de "fórmula" a partir da qual objetos são criados, isto é, instanciados.
- Programar OO é, em grande parte do tempo, escrever classes!

O que é uma classe?

Como eu posso criar uma classe em C++?





Estrutura básica de uma classe

<qualificador>: indica o nível de acesso da classe.

Se não for declarado nada, o qualificador **private** será utilizado.

```
class <Nome_da_classe>{  
    <qualificador1>:  
        // atributos (estados)  
        // métodos (operações)  
    <qualificador2>:  
        // atributos (estados)  
        // métodos (operações)  
}; // tem que ter o ;
```

-
- **private**: só pode ser usada na própria classe;
 - **public**: pode ser usado livremente por qualquer classe;
 - **protected**: só pode ser usada na própria classe e sub-classes.



Criando uma classe em C++

Divisão entre arquivo de cabeçalho (.h) e arquivo de implementação (.cpp)

- Definição da classe no arquivo .h
- Implementação dos métodos da classe no .cpp
- No arquivo cpp: Nome dos métodos devem ser precedidos do nome da classe e ::

Apesar de ser possível a definição de classes em um único arquivo, é uma boa prática de programação utilizar .h e .cpp

- Proteção de código fonte (bibliotecas fechadas)
- Compartilhamento de bibliotecas sem compartilhar código fonte.

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

class Veiculo {

private:
    string modelo;
    int anoFabricacao;
    double velocidade;

public:
    Veiculo();
    void acelerar(double aumVel);
    void frear(double dimVel);
    double obterVelocidade();
};

};
```

Exemplo: Classe Veiculo (arquivo veiculo.h)

```
1 #include "Veiculo.h"
2
3 Veiculo::Veiculo() {
4     // Falaremos depois
5 }
6
7 void Veiculo::acelerar(double aumVel){
8     velocidade += aumVel;
9 }
10
11 void Veiculo::frear(double dimVel){
12     velocidade -= dimVel;
13 }
14 double Veiculo::obterVelocidade(){
15     return velocidade;
16 }
17
18
```

19 Exemplo: Classe Veiculo (arquivo veiculo.cpp)
20
21
22

```
1 #ifndef VEICULO_H
2 #define VEICULO_H
3
4 #include <iostream>
5 using namespace std;
6
7 class Veiculo {
8
9     private:
10         string modelo;
11         int anoFabricacao;
12         double velocidade;
13
14     public:
15         Veiculo();
16         void acelerar(double aumVel);
17         void frear(double dimVel);
18         double obterVelocidade();
19     };
20 #endif
21
22
```

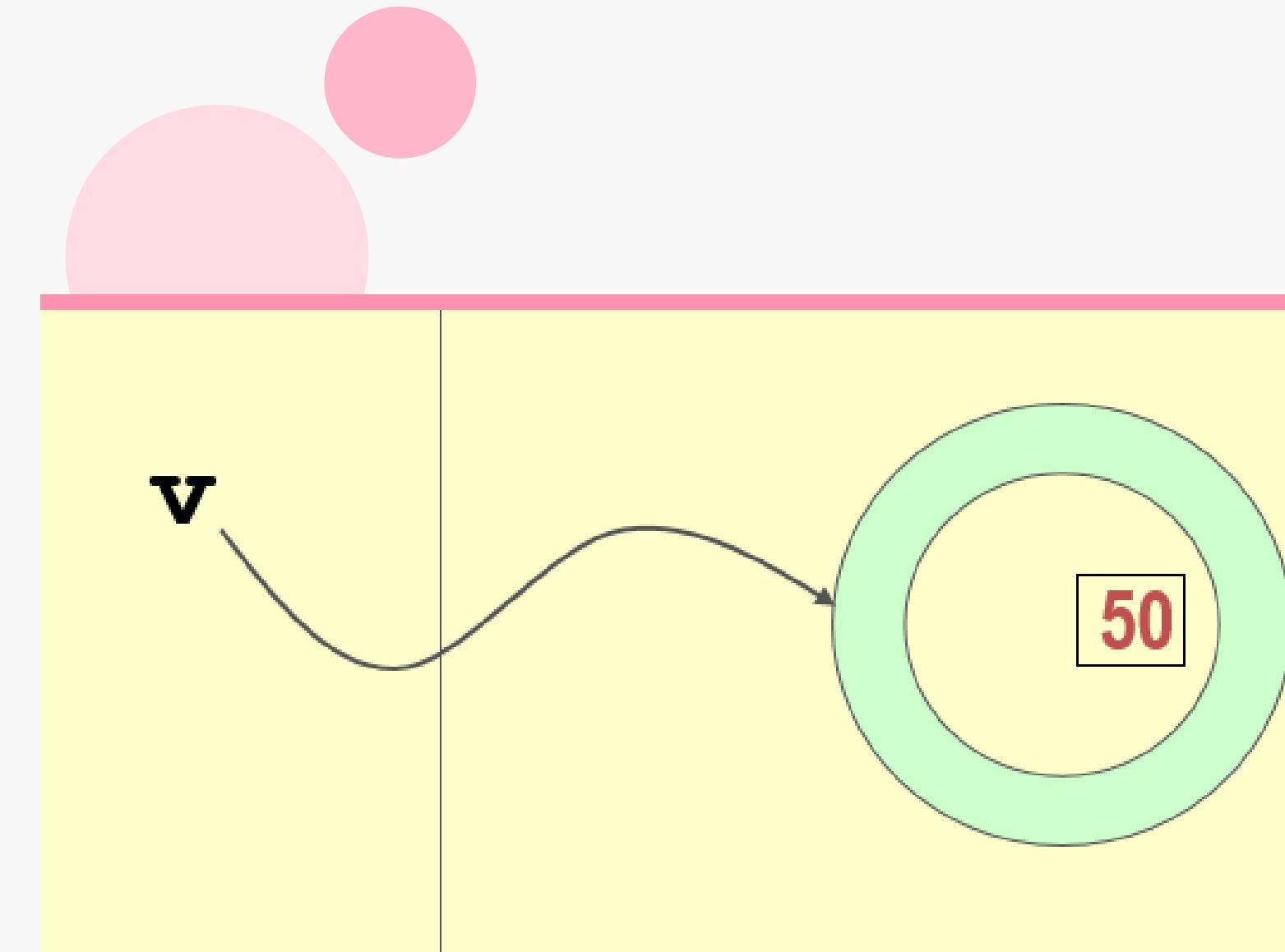
Impede que os arquivos de cabeçalho sejam incluídos mais de uma vez.

Criando e usando objetos de uma classe

Primeira forma

```
1 #include <iostream>
2 #include "veiculo.h"
3 using namespace std;
4
5 int main() {
6
7     Veiculo v;           //declaracao
8     v = Veiculo();       //instanciacao
9     v.acelerar(50);      //uso
10
11    return 0;
12 }
```

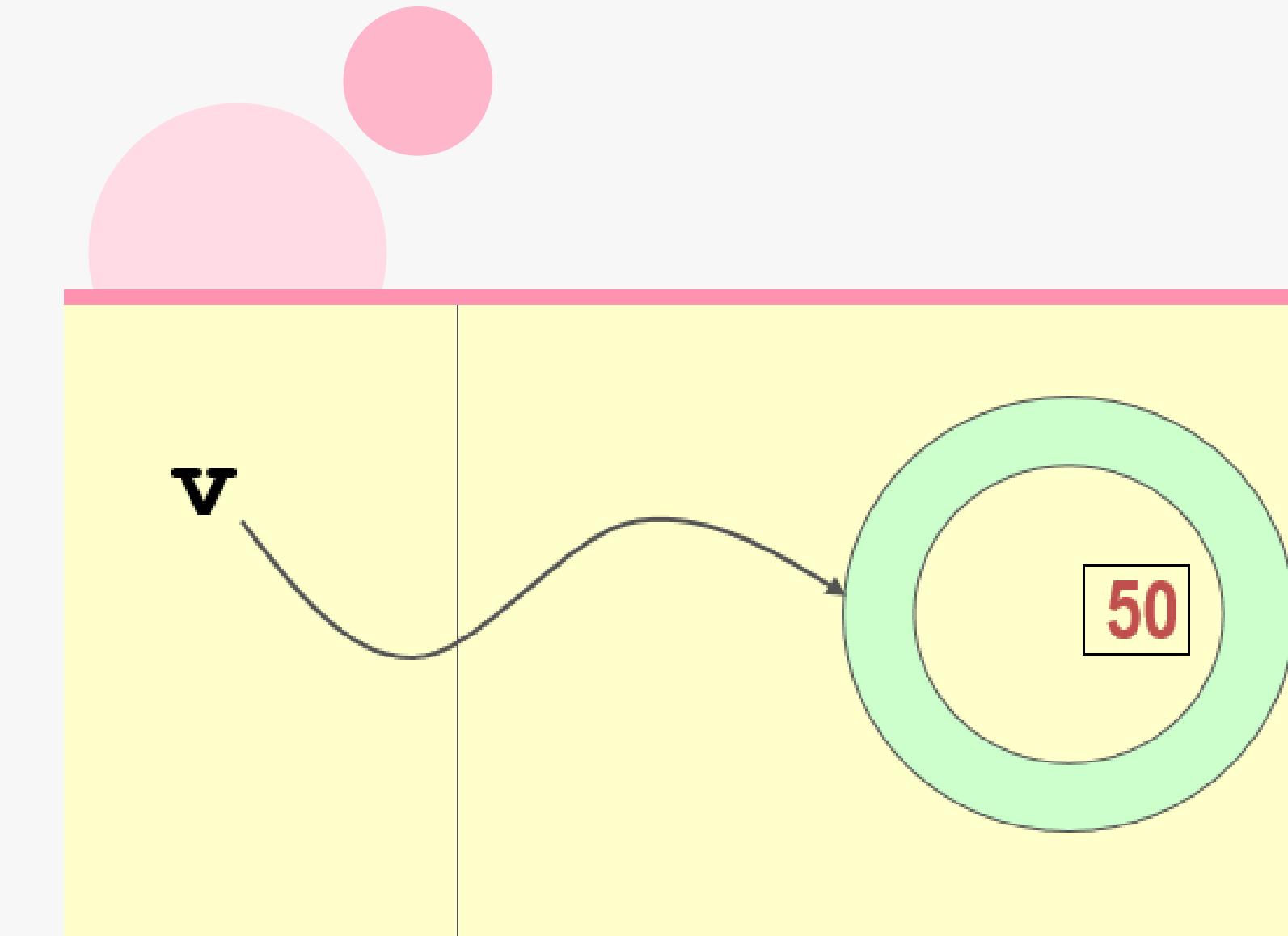
1^a Forma



Criando e usando objetos de uma classe

```
1 #include <iostream>
2 #include "veiculo.h"
3 using namespace std;
4
5 int main() {
6
7     Veiculo *v;           //declaracao
8     v = new Veiculo();    //instanciacao
9     v->acelerar(50);    //uso
10    delete v;            //destroi
11
12    return 0;
13}
14 }
```

2^a Forma



```
1 #include <iostream>
2 #include "veiculo.h"
3 using namespace std;
4
5 int main() {
6
7     Veiculo v1;                                //declaracao
8     v1 = Veiculo();                            //instanciacao
9     v1.acelerar(50);                           //uso
10
11    Veiculo v2 = Veiculo();                  //declara e cria
12    v2.acelerar(50);                           //uso
13
14    Veiculo v3;                                //declara e criar
15    v3.acelerar(50);                           //uso
16
17    return 0;
18 }
19
20
21
22
```

Observação: todas são equivalentes

```
1 #include <iostream>
2 #include "veiculo.h"
3 using namespace std;
4
5 int main() {
6
7     Veiculo *veiculo1 = new Veiculo();
8     Veiculo veiculo2 = Veiculo();
9
10    veiculo1 -> acelerar(100.0);
11    veiculo2.acelerar(50.0);
12    veiculo1 -> frear(20.0);
13
14    double v1 = veiculo1 -> obterVelocidade();
15    std::cout << v1 << std::endl;
16    delete veiculo1;
17
18    double v2 = veiculo2.obterVelocidade();
19    std::cout << v2 << std::endl;
20
21 }
```

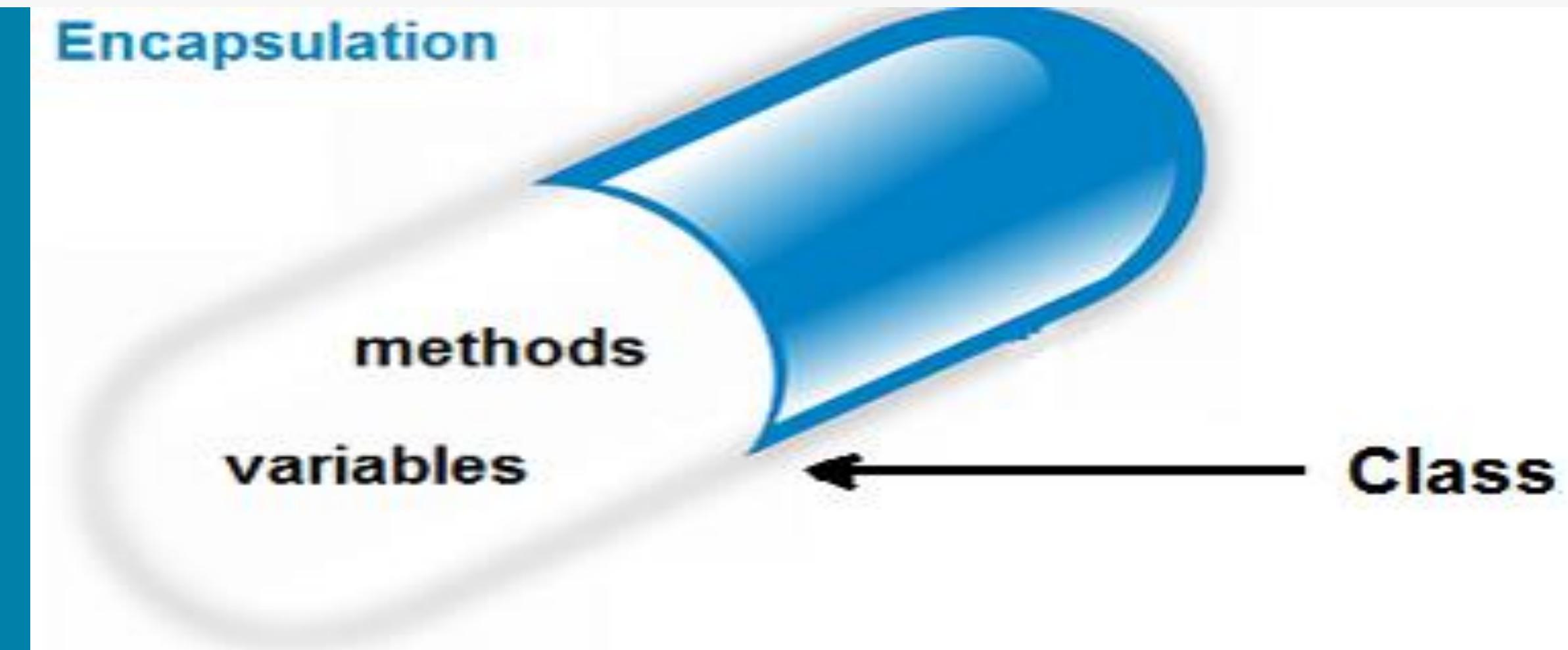
Exemplo

3

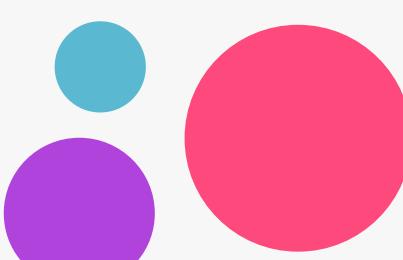
Encapsulamento

Encapsulamento em orientação à objetos

Permite ocultar ou restringir o acesso à implementação de componentes, a partir do fornecimento de uma interface ao mundo exterior



Os serviços fornecidos pelo componente são providos por meio de uma interface





Como encapsular?

Utilizar corretamente modificadores de acesso em atributos e métodos;

Criar métodos especiais, como getters e setters e um ou mais construtores para inicializar atributos de classe;

Nem sempre será necessário criar um método get e set para todos os atributos de classe.

Como encapsular?

Esses modificadores de acesso definem o nível de acesso ou visibilidade e irão definir as condições de acesso para os elementos internos (atributos e métodos) de uma classe, são esses:



- **private**: nível mais rígido, apenas a própria classe pode acessar;
- **protected**: nível intermediário, a própria classe e as subclasses podem acessar;
- **public**: nível sem restrições, equivalente a não encapsular.



Encapsulamento em orientação à objetos

Métodos Acessores (getters):

Permitem verificar o valor atribuído a um atributo, tendo, assim, sempre que possuir um tipo de retorno definido (i.e., int, boolean, float, uma Classe, etc.)

```
class Pessoa {  
    public:  
        //métodos ou funções  
        double calcularImc(double peso, double altura);  
        void setCPF(string novoCPF);  
        string getCPF();
```

-
- É considerada uma boa prática de programação seguir o padrão:
 - `get + NomeDoAtributo()`
 - Não possuem parâmetros



Encapsulamento em orientação à objetos

Métodos Modificadores (setters):

Permitem alterar o valor atribuído a um atributo, tendo, assim, sempre que possuir um parâmetro definido (i.e., int, boolean, float, uma Classe, etc.)

```
class Pessoa {  
    public:  
        //métodos ou funções  
        double calcularImc(double peso, double altura);  
        void setCPF(string novoCPF);  
        string getCPF();
```

-
- É considerada uma boa prática de programação seguir o padrão:
 - **set + NomeDoAtributo(tipo parametro)**
 - Geralmente, não retornam valor (métodos void).



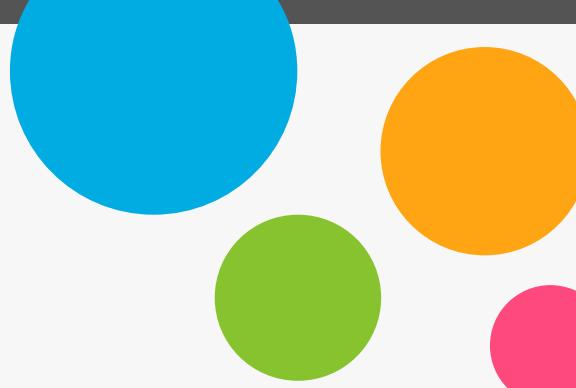
Encapsulamento em orientação à objetos

Exemplo: (produto.h)

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Produto {
6
7 private:
8     string nome;
9     double preco;
10
11 public:
12     string getName();
13     double getPreco();
14     void setName(string nome);
15     void setPreco(double preco);
16 };
```

Exemplo: (produto.cpp)

```
1 #include "produto.h"
2 #include <string>
3
4 string Produto::getName(){
5     return nome;
6 }
7
8 double Produto::getPreco(){
9     return preco;
10 }
11
12 void Produto::setName(string nome){
13     this->nome = nome;
14 }
15
16 void Produto::setPreco(double preco){
17     this->preco = preco;
18 }
```



Encapsulamento em orientação à objetos

Opa! Quem é atributo da classe e quem é parâmetro do método `setNome()`?

```
void Pessoa::setNome(string nome) {  
    nome = nome;  
}
```

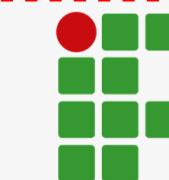
Encapsulamento em orientação à objetos

O this, em C++, serve para fazer uma autoreferência ao objeto atual que está chamando o método

Aqui, nos referimos ao atributo nome, a variável de instância, declarada na classe Produto

```
void Pessoa::setNome(string nome) {  
    this->nome = nome;  
}
```

E aqui nos referimos ao argumento nome, passado como parâmetro do método setNome()





Encapsulamento em orientação à objetos

Exemplo: main.cpp

```
1 #include <iostream>
2 #include "produto.h"
3 #include <string>
4
5 using namespace std;
6
7 int main() {
8     Produto p1 = Produto();
9     p1.setNome("Shampoo");
10    p1.setPreco(10.58);
11
12    cout << "Nome: " << p1.getNome() << endl;
13    cout << "Preco: " << p1.getPreco() << endl;
14
15    return 0;
16 }
```



4

Construtores

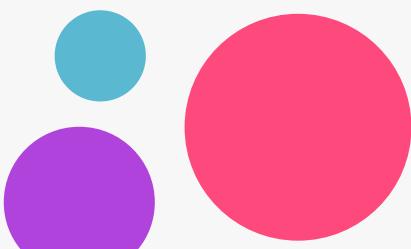


Construtores

Permitem estabelecer um conjunto de instruções que serão realizadas sempre que um objeto de uma determinada classe for instanciado

```
Classe nomeDoObjeto = Classe();  
Classe *nomeDoObjeto = new Classe();
```

São “métodos” especiais invocados no momento da criação dos objetos (inicialização):





Construtores

Existe um construtor padrão em todas as classes que não precisa ser programado, mas pode ser;

É o primeiro método que um objeto executa:

- O seu objetivo é garantir a inicialização correta do objeto.

Exemplo: Na criação de um objeto do tipo Cliente, pode-se definir: nome do cliente, cpf, etc.



Construtores

Eles devem ser públicos,
não possuir nenhum tipo
de retorno e ter o mesmo
nome da classe

```
Produto() {
```

Entenda por “ter o mesmo nome da classe”
que você deve utilizar exatamente a mesma
sintaxe que utilizou para o nome da classe!

```
}
```



Construtores

Eles podem possuir parâmetros, que, geralmente, são utilizados para inicializar atributos.

```
Pessoa(string cpf, string nome, int idade) {  
    this->cpf = cpf;  
    this->nome = nome;  
    this->idade = idade;  
}
```



Construtores

Produto.h

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Produto {
6
7     private:
8         string nome;
9         double preco;
10
11    public:
12        Produto(); // construtor padrao
13        Produto(string nome, double preco);
14
15        string getNome();
16        double getPreco();
17        void setNome(string nome);
18        void setPreco(double preco);|
19    };
```

Produto.cpp

```
1 #include "produto.h"
2
3 Produto::Produto(){
4     this->preco = 1.00;
5     this->nome = "";
6 }
7
8 Produto::Produto(string nome, double preco){
9     this->preco = preco;
10    this->nome = nome;
11 }
12
13 // outros metodos
14
15
16
```

Main.cpp

```
1 #include <iostream>
2 #include "produto.h"
3 using namespace std;
4
5 int main() {
6
7     Produto p1 = Produto();
8     p1.setNome("Shampoo");
9     p1.setPreco(10.58);
10
11    cout << "Nome: " << p1.getNome() << endl;
12    cout << "Preco: " << p1.getPreco() << endl;
13
14    Produto p2 = Produto("Sabonete", 2.50);
15    cout << "Nome: " << p2.getNome() << endl;
16    cout << "Preco: " << p2.getPreco() << endl;
17
18    return 0;
19 }
```



Construtores

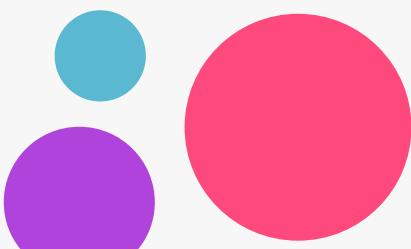
Exemplo:
(produto.cpp)

Outra forma de inicializar
os construtores

```
#include "produto.h"

Produto::Produto(){
    this->preco = 1.00;
    this->nome = "";
}

Produto::Produto(string n, double p): nome(n), preco(p){ }
// outros metodos
```



Main.cpp

outra forma de inicializar

```
1 #include <iostream>
2 #include "produto.h"
3 using namespace std;
4
5 int main() {
6
7     Produto p1 = Produto();
8     p1.setNome("Shampoo");
9     p1.setPreco(10.58);
10
11    cout << "Nome: " << p1.getNome() << endl;
12    cout << "Preco: " << p1.getPreco() << endl;
13
14    Produto p2("Sabonete", 2.50);
15    cout << "Nome: " << p2.getNome() << endl;
16    cout << "Preco: " << p2.getPreco() << endl;
17
18    return 0;
19
20 }
```



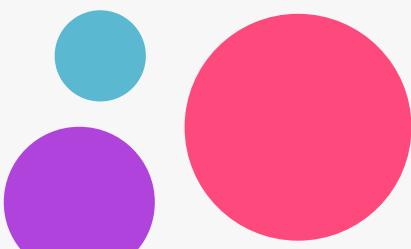
Construtores

Outra forma de inicializar



Dica de leitura

Pesquisar sobre destrutores



Alguma dúvida?

Não guardem dúvidas, perguntam

• • •

Referências

- 1 DA COSTA, Anderson Fabiano F. **Fundamentos de C++**. Instituto Federal da Paraíba. 2022.
- 2 Materiais de aula dos professores Guillermo Camara-Chavez, Tiago Maritan, Fred Guedes Pereira e Danielle Chaves.
- 3 DEITEL, **C++ Como Programar**, 5^a edição, Editora Prentice Hall, 2006
- 4
- 5