Programação Orientada a Objetos

Desenvolvendo um jogo de estratégia em C++ – Dia 1

Programa de Educação Tutorial do Curso de Ciência da Computação da UFRN

Universidade Federal do Rio Grande do Norte - UFRN
Pró-Reitoria de Graduação - PROGRAD

30 de Julho de 2018

Roteiro do curso

Dia 1 Introdução à POO;
Classes;
Objetos;
Construtores;
Destrutores.

Dia 2 Encapsulamento;
Modificadores de acesso;
Classes contêiner;
Sobrecarga de métodos.

Roteiro do curso

Dia 3 Modificador estático;
Herança;
Relação de Objetos.

Dia 4 Polimorfismo;
Classe Abstrata;
Interface.

Dia 5 Interface Gráfica;
Conclusão do Jogo de Estratégia.

Projeto

O projeto será um jogo de estratégia

Nele, o jogador irá gerenciar uma aldeia e terá o objetivo de conquistar outras aldeias. Além disso, o jogo contará com as seguintes mecânicas:

- Coletar recursos;
- Construir e aprimorar estruturas;
- Treinar unidades de coleta e guerreiros;
- Atacar aldeias inimigas.

Roteiro do primeiro dia

Introdução à POO

- Programação Imperativa
  - Uma forma de pensar o programa sequencialmente;
  - Mais próximo das instruções de processador;
  - O controle do programa é feito apenas através de estruturas condicionais e estruturas de repetição;
  - Soluciona a complexidade com a modularização.
- Vantagens:
  - Maior desempenho computacional;
  - Mais direto para programas simples.
- Desvantagens:
  - Pode ser difícil de ler em programas mais complexos;
  - Mais complicado de fazer modularização se comparado a POO.

Introdução à POO

- Programação Orientada à Objetos (POO):
  - Baseia-se no conceito de abstração de objetos reais, o que faz desse paradigma bom de trabalhar com simulações.
- Vantagens:
  - Maior nível de abstração;
  - Mais facilidade em representar entidades do mundo real;
  - Maior reusabilidade de código;
  - Menor custo de desenvolvimento e de manutenção (escalabilidade).
- Desvantagens:
  - Maior uso de recursos computacionais (memória, processamento, etc.).

Quando usar POO?

- Quando é preciso abstrair modelos computacionalmente;
- Quando a aplicação necessita de escalabilidade.

O que é abstração?

- É a extração do que é relevante em um problema/aplicação;
- Pensando em objetos:
  - É olhar para um problema e identificar as características relevantes para a representação do objeto do mundo real em uma determinada aplicação.

Classes

- Uma classe define um tipo e representa uma coleção de objetos;
- Possuem nome, dados e operações.

```
1  class Classe {
2      // Atributos (Dados)
3      // Métodos (Operações)
4  };
```

Exemplo de classe em C++


```
1  class Dog {
2      public:
3          char* name;
4          float weight;
5          float height;
6          int age;
7
8          void bark() { /* Metodo latir */ }
9          void walk() { /* Metodo andar */ }
10         void eat() { /* Metodo comer */ }
11         void sleep() { /* Metodo dormir */ }
12 };
```

Structs × Classes

- Structs em C:
  - Permite colocar variáveis de tipos diferentes dentro da mesma estrutura.
  - Todos as variáveis dentro da struct são acessíveis.
- Classes em C++:
  - Permite colocar variáveis e funções dentro de uma mesma estrutura.
  - Permite **encapsular** variáveis e/ou funções para que sejam acessíveis apenas dentro da classe ou a "quem" for dado acesso.

Exemplo de struct em C


```c
struct Dog {
    char* name;
    int age;
};

int get_human_age(Dog dog) {
    return (dog.age * 4 + 12);
}
```

Mesmo Exemplo com Classe em C++


```
1  class Dog {
2    public:
3      char* name;
4      int age;
5
6      int get_human_age() {
7          return (age * 4 + 12);
8      }
9  };
```

Objetos

- São a instanciação do que foi declarado na classe;
- Podem ser chamados de instâncias;
- Uma classe é um modelo que representa uma coleção de objetos.

Classe × Objeto

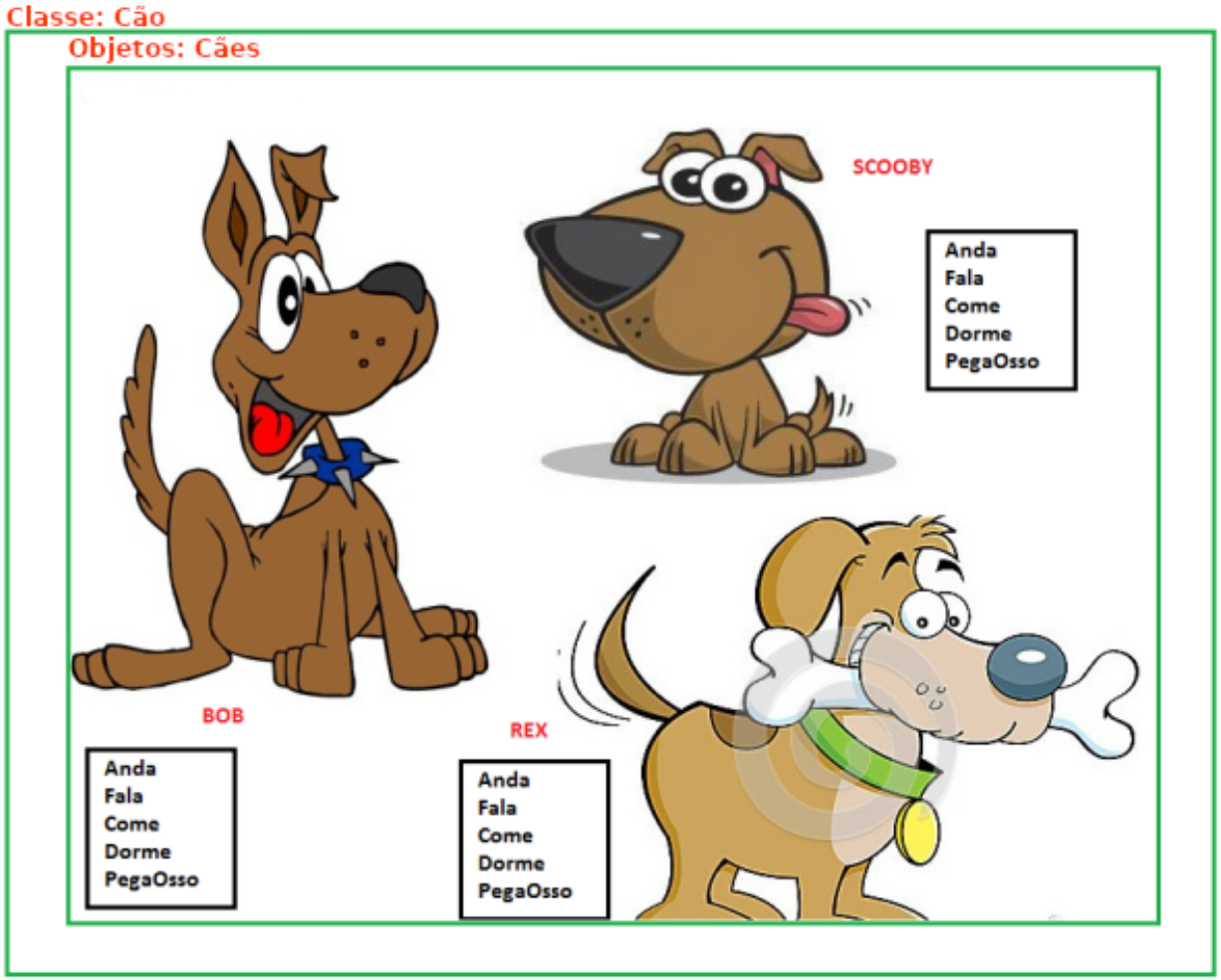

Figura: Analogia para classe e objeto. Fonte: DevMedia.

Exemplo de objeto em C++


```
g++ nome_da_clase.cpp -o nome_do_executavel
```

```
1  class Dog {
2    public:
3      char* name;
4      int age;
5  };
6
7  int main(){
8      Dog d1, d2;
9      d1.name = "Rex";
10     d1.age = 1;
11     d2.name = "Roberto Carlos";
12     d2.age = 5;
13 }
```

Exercício #1

Pense em um modelo para a classe `Player` para um sistema de combate simples.

Figura: *Dark Souls*.

Exercício #1

Modele e implemente a classe `Player` de um jogo de luta com os seguintes atributos: `health`, `speed`, `attack` e `defense`. Implemente os seguintes métodos: `attack(Player* p)` e `is_dead()`.

Construtores

- Métodos especiais que são chamados para instanciar um objeto;
- Em C++ um construtor possui o mesmo nome da classe;
- Tipos de Construtores:
  - Construtor padrão;
  - Construtor parametrizado;
  - Construtor cópia.

Exemplo de construtor padrão I


```
1  class Dog {
2    public:
3      Dog() {
4          this->name = "Billy";
5          this->weight = 10.0;
6          this->height = 0.39;
7          this->age = 8;
8      }
9  };
10
11 Dog dog; // Instância de construtor padrão
12 std::cout << dog.name; // Irá imprimir "Billy"
```

Exemplo de construtor padrão II


```
1  class Dog {
2    public:
3      Dog() : name("Billy"), weight(10.0),
4          height(0.39), age(8) { /* Empty */ }
5  };
6
7  Dog dog; // Instância de construtor padrão
8  std::cout << dog.name; // Irá imprimir "Billy"
```

Exemplo de construtor parametrizado I


```
1  class Dog {
2    public:
3      // Construtor com passagem de parâmetros
4      Dog(string _name, float _weight, float _height)
5              : name(_name), weight(_weight), height(_height)
6              { /* Empty */ }
7  };
8
9  Dog dog ("Billy", 10.0, 0.5); // Instanciando a classe
10 cout << dog.nome; // Irá imprimir "Billy"
11 cout << dog.height; // Irá imprimir "0.5"
```

Exemplo de construtor parametrizado II


```
1  // Construtor com passagem de parâmetros opcionais
2  class Dog {
3    public:
4      Dog(string _nome, float _weight, float _height=0.39)
5              : nome(_nome), weight(_weight), height(_height)
6              { /* Empty */ }
7  };
8
9  Dog dog ("Billy", 10.0); // Instanciando a classe
10 cout << dog.nome; // Irá imprimir "Billy"
11 cout << dog.height; // Irá imprimir "0.39"
```

Exercício #2

Implemente o construtor para a classe `Player` do exercício anterior.

Destrutor

- É um método chamado antes de um objeto ser removido da memória;
- São utilizados para evitar vazamento de memória;
- Um objeto é removido da memória quando:
  - O programa sai do escopo (exemplo: na saída de um laço se um objeto foi declarado apenas dentro do laço, ele será destruído);
  - O programa termina a execução;

Exemplo de destrutor


```
1  class Dog {
2    public:
3      // Destrutor
4      ~Dog() {
5          std::cout << this->name << "morreu\n";
6          delete [] this->name;
7      }
8  };
9  int main(){
10     Dog dog ("Billy", 10.0, 0.5);
11     if (true){
12         Dog dog ("Magrelinho", 5.0);
13         // ... Umas linhas de código
14     } // Vai imprimir "Magrelinho morreu\n" ao sair do if;
15     // ... Mais linhas de código
16 } // Vai imprimir Billy morreu
```

Exercício #3

Figura: Campo minado.

Implemente a classe `Map` como uma matriz de inteiros com um construtor e um destrutor.

Atividade de casa

Utilize a classe `Player` desenvolvida nos exercícios #1 e #2 para criar uma batalha entre dois players utilizando o método `attack`. A batalha deve ser baseada em turnos e o player mais rápido deve realizar o primeiro ataque.

Dúvidas

Jogo de Estratégia

O jogo consistirá em uma batalha de turnos entre aldeias.

- Cada aldeia pertence a somente um jogador e possui um conjunto de estruturas e unidades;
- Existem recursos pré-definidos, coletados pelo jogador, que são utilizados para realizar ações no jogo;
- Cada estrutura possui uma finalidade: Aprimorar unidades, aumentar capacidade de estoque de recursos, treinar unidades;
- As unidades podem ser do tipo trabalhador ou guerreiro;
- O objetivo do jogador é aprimorar seu vilarejo para derrotar seu inimigo por meio de ataques;

Modelagem inicial das entidades

Com base na descrição do projeto e no que você aprendeu até agora, pense nas possíveis entidades que irão compor o jogo e as implemente.

Mãos à obra!

Solução dos exercícios

Solução dos exercícios:
`https://github.com/brenov/petcc-poo-course`.