Introdução

Programação procedural, programação estruturada. Programas estruturados possuem funções e dados que são aplicados globalmente no programa inteiro. Em C, os programas são escritos de forma que sua execução é sequencial, e por isso muito eficiente, além de ser uma linguagem de baixo nível e permitir interação com o hardware. No paradigma atual, entretanto, os programas atuais não necessitam tanto desse diferencial do desempenho devido a capacidade de processamento dos computadores. A programação orientada a objetos permite que boa parte do código seja reutilizado e que o sistema seja representado com elementos que sejam comparados a coisas do mundo real.

Se temos um programa em C que fica muito complexo, isto é, temos muitas funções que dependem uma da outra, então a manutenção e escalabilidade do sistema se torna inviável, pois mudar algo em uma função impacta diversas outras. Programação orientada a objeto soluciona esse problema: Funções e variáveis (dados) que são relacionados são agrupados em “Objetos”, sendo que podemos ter várias instâncias desses objetos.

Exemplo spaghetti a bolonhesa: Se você faz um jantar comemorativo para seus amigos onde foi preparado um spaghetti a bolonhesa e infelizmente você percebe que esqueceu que metade dos seus amigos não comem carne. Assim, igual um código spaghetti, você não pode simplesmente lavar o macarrão para tirar o molho, mas sim fazer do começo tudo de novo corretamente.

A programação orientada a objeto foi sendo desenvolvida e aprimorada junto com a evolução dos computadores, mesmo na época dos computadores mainframes, já haviam paradigmas de conceitos de POO. A primeira linguagem que oferece os conceitos de POO como conhecemos hoje era SIMULA que era usada para simulações. Conceitos do SIMULA foram sendo desenvolvidos e levaram a criação do C++, a primeira linguagem orientada a objeto que obteve um sucesso comercial.

E porque “orientada a objeto”? Pois os modelos os objetos do mundo real de forma que os objetos do nosso programa correspondam a eles. A ideia é que os objetos ofereçam os tipos/dados que eles permitam usar e ofereçam os métodos para que esses dados possam ser manipulados (Exemplo: Radio de mesa).

Conceitos de POO

O **Encapsulamento** seria uma prática em que funcionalidades ou entidades sejam definidas cada uma como um objeto. Em suma, significa dividir a funcionalidade do programa em partes. Se temos dados e métodos que trabalham para para calcular o IMC do corpo, faz sentido estes façam parte de um único objeto/classe. O encapsulamento também se diz em relação ao nível de acesso às propriedades e métodos.

Abstração - Objetos com poucas propriedades (variáveis) e métodos são simples de entender. Mudanças em um objeto não causam impacto em outros objetos, e internamente você não precisa entender como o objeto funciona. Exemplo: Uma máquina de café tem uma funcionalidade bem definida: colocar água, o pó, apertar o botão. Mas internamente você não precisa saber como é a lógica dos circuitos integrados para fazer seu cafezinho.

A **Herança**, na minha opinião, é um dos conceitos mais importantes da programação orientada a objeto considerando o desenvolvimento de jogos. Ela permite que uma classe herde a funcionalidade da outra (considerando os níveis de acesso, explicados mais à frente). Se estamos criando um programa sobre o reino animal e temos classes (explicado a frente) que definem Répteis e Pássaros. Seria possível escrever o código para alimentação e descanso em cada uma das classes, mas usando herança é possível reduzir a quantidade de código redundante nas classes. Assim, coisas que possuem características em comum podem herdar de uma classe que define essas características. As classes filho possuem toda a funcionalidade da classe pai (métodos e propriedades) e podem definir características únicas cada uma. Como já dito, se não queremos que algo seja visto pelas classes exteriores, ou talvez nem pelas classes filho, os modificadores de acesso podem ser usados, mas os veremos mais à frente. Dando um exemplo em desenvolvimento de jogos, se tanto o inimigo quanto o player precisam se mover pelo mapa, tomar dano, tem vida etc eles podem herdar de uma mesma classe.

O **Polimorfismo** também é importante para o desenvolvimento de jogos e está de certa forma atrelado à herança. Temos um componente, um programa, um objeto, uma lousa que permite que as pessoas desenhem, mas para cada tipo de ferramenta, uma caneta, um lápis, um giz, se tivéssemos que escrever o código dentro da lousa que interprete o que cada objeto é e realize o procedimento para desenhar diferente para cada talvez ficasse muito redundante ou pouco eficiente de escrever. O conceito do polimorfismo permite que isso fique abstraído. A lousa só precisa cuidar do procedimento de desenhar, então cada objeto poderia definir seu próprio procedimento de desenho. Dessa forma, a lousa só recebe o objeto e sabe que precisa chamar a função desenho do próprio objeto, ou seja, o objeto fica abstraído da lousa. O conceito pode ser um pouco melhor exemplificado quando se tratando de interfaces.

Classes. Uma classe pode ser definida como sendo uma diagrama, uma esquemática. Descreve toda a funcionalidade e tem todas informações sobre um conceito ou objeto. Por exemplo, vendo um manual de instruções para montagem de uma mesa, este fornece informações sobre seu formato, dimensões etc. Do mesmo jeito, um manual de instruções de um controle remoto explica a funcionalidade de variados botões. Em C#, as variáveis de uma classe, por padrão, só podem ser acessadas dentro dela própria. De uma forma concisa, uma classe é onde as funções e variáveis do programa vão.
Um **objeto** é a concretização do conceito. O objeto é o que é usado para interagir na classe e vários objetos podem ser criados a partir de uma única classe.

Criar um objeto a partir de uma classe é chamado de instanciação. Com o objeto é possível acessar seus métodos (de acordo com o modificador de acesso) contanto que ele esteja instanciado, ou seja, não é possível acessar um método de um objeto, ou uma variável se ele não for instanciado. Podemos também definir que classes são o tipo (type) de objetos. Essencialmente objetos funcionam muito como ponteiros de memória, mas isso já é um conceito mais detalhado.

Um programa orientado a objeto também possui um método Main, igual em C, e é por ele que o programa começa.

Dicas sobre programação

É muito importante ler os erros. Muita gente não presta realmente atenção no que o erro fala, e qualquer linguagem sempre vai apontar a linha que tal erro está. Se na linha em questão aparenta estar tudo correto, pode ser algum escopo que não foi fechado, ou até mesmo erros de lógica de programação. A técnica do debug com pato de borracha ajuda muito a resolver erros de lógica. O nome é uma referência a uma história do livro O Programador Pragmático em que um programador explica seu problema para um pato de borracha, descrevendo linha a linha o código que já foi escrito. Essa técnica, ao observar efetivamente o que o código faz, auxilia a identificar o problema. O ensino de um assunto muitas vezes força o seu entendimento através de uma mudança de perspectiva e auxilia sua compreensão mais profunda. Por fim, é também sempre interessante consultar o Stack Overflow se seu erro é algo mais técnico de algo que você realmente desconhece. É relevante falar que não é interessante apenas copiar e colar a solução do SO e sim entender do que se trata e como isso resolve seu problema. Tanto para a questão dos erros quanto o SO é muito importante que o programador tenha um conhecimento de inglês. Apesar de existir conteúdo bom em português sobre programação na internet, tal conteúdo não é nem 1% do que você poderia estar absorvendo lendo em inglês.

Links relevantes

https://www.youtube.com/watch?v=XT_CtYN1OOU

https://inventwithpython.com/makinggames.pdf

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/object-oriented-programming#Classes