5. POO, Concorrência e Boas Práticas

O paradigma da Programação Orientada a Objetos (POO) e as aplicações em Sistemas Operacionais (concorrência e escalonamento) são tópicos avançados que visam a construção de programas mais complexos, estruturados e alinhados ao mundo real.

Programação Orientada a Objetos (POO):

POO modela o mundo real através da **abstração**, definindo características (atributos) e comportamentos (métodos) relevantes para um contexto específico.

• **Classe:** É o **molde** ou estrutura para a criação de objetos. Em Python, é definida pela palavra-chave **class** e deve ter seu nome em **CamelCase**.

• **Objeto (Instância):** É a concretização da classe, com valores específicos para seus atributos.

• **self:** O primeiro parâmetro obrigatório de todos os métodos em uma classe. Ele referencia o objeto que está sendo operado ou criado.

• **Método Construtor (__init__)**: Um método especial invocado na criação do objeto, usado para configurar seus atributos iniciais.

Aplicações em Sistemas Operacionais:

• **Concorrência (Threads):** Refere-se à execução de trabalhos em paralelo. O módulo threading é usado para criar essas unidades de execução.

◦ **Sincronização:** É vital para gerenciar o acesso a recursos compartilhados. O método **join()** força o programa a esperar pela conclusão de uma *thread*.

◦ **Produtor vs. Consumidor:** Um problema clássico de concorrência que ilustra a necessidade de sincronização. Usa-se o objeto **Lock** para garantir o acesso exclusivo a uma região crítica de dados (evitando corrupção) e **Condition** para notificar e fazer as threads "dormirem" ou "acordarem" dependendo das condições do armazém.

◦ **Deadlock:** Uma condição grave em SO, exemplificada pelo "Jantar dos Filósofos", onde processos (filósofos) ficam em espera circular por recursos (talheres) que nunca são liberados.

• **Escalonamento:** Algoritmos que determinam a ordem em que os processos são executados pelo sistema operacional.

◦ **FIFO (First-in, First-out):** O algoritmo mais simples, onde o primeiro processo a chegar é o primeiro a ser executado.

◦ **SJF (Shortest Job First):** Prioriza a execução dos processos com menor **duração** (*burst time*), o que geralmente resulta em um **tempo médio de espera e retorno menor**.

◦ **Variáveis de Tempo:** Conceitos chave para avaliar a eficiência do escalonamento incluem *tempo de conclusão*, *tempo de chegada*, *duração*, *tempo de retorno* e *tempo de espera*.

Boas Práticas de Programação:

Para garantir a qualidade, legibilidade e manutenibilidade do código, algumas práticas são essenciais:

• **Clareza de Nomes:** Usar nomes de variáveis, funções e classes que sejam claros e descritivos (ex: preco_total em vez de x).

• **Comentários:** Comentar para explicar o *porquê* de uma decisão lógica complexa no código, e não apenas o que o código está fazendo.

• **Organização:** Modularizar o código em funções e módulos.
• **Padrão de Estilo:** Seguir o guia de estilo oficial da linguagem, o **PEP 8**