**Sistema de Gerenciamento de Conexões com Banco de Dados usando Singleton**

**Contexto:**  imagine que você está desenvolvendo uma aplicação que se conecta frequentemente a um banco de dados. A fim de otimizar o uso de recursos, você decidiu implementar um sistema de pool de conexões. Esse sistema de gerenciamento precisa assegurar que todas as partes do aplicativo utilizem as mesmas conexões, evitando a criação de conexões desnecessárias.

**Tarefa:** implemente uma classe **DatabaseConnectionPool** que gerencie um conjunto de conexões ao banco de dados usando o padrão **Singleton**. O pool de conexões deve ter uma quantidade máxima de conexões ativas (por exemplo, 5) e uma fila para requisições quando o limite é atingido. Quando uma conexão é liberada, ela deve ser reutilizada pelo próximo processo na fila.

**Requisitos Específicos:**

1. **Classe Singleton**: crie uma classe **DatabaseConnectionPool** que siga o padrão **Singleton**.

2. **Pool de Conexões**: a classe deve manter uma lista de conexões ativas, limitando a um número máximo de conexões configurado.

3. **Solicitação e Liberação de Conexões**: implemente métodos para solicitar e liberar uma conexão. Se o pool estiver cheio, as requisições de novas conexões devem aguardar até que uma conexão seja liberada.

4. **Logs de Conexões:** adicione logging para monitorar o número de conexões ativas e o status das solicitações (por exemplo, aguardando ou atendida).

5. **Teste:** escreva um programa de teste que crie múltiplas threads para simular a requisição simultânea de conexões, demonstrando o comportamento do sistema quando o pool atinge o limite.

**Detalhes para implementação:**

1. **Classe DatabaseConnection**: representa uma conexão individual, que possui um identificador único (**id**) e uma flag **in\_use** para rastrear se está em uso ou não.

2. **Classe DatabaseConnectionPool**: é um Singleton que possui um número máximo de conexões definidas na inicialização. Garante que apenas uma instância do pool seja criada, mesmo quando acessado por múltiplas threads.

3. **Métodos get\_connection e release\_connection**:

- **get\_connection**: procura uma conexão disponível. Se todas estiverem em uso, a thread entra na fila de espera.

- **release\_connectio**: libera a conexão e notifica a primeira thread na fila de espera, para que tente obter uma conexão novamente.

**4. Teste de Conexões com Threads:**

* cria várias threads para simular múltiplas requisições simultâneas.
* Cada thread solicita uma conexão, simula um tempo de uso e depois libera a conexão.
* O pool de conexões mantém o limite de conexões ativas e organiza as threads em espera quando o limite é atingido.