Prova de Programação Paralela e Distribuída

Jantar dos Filósofos

IFSP - Instituto Federal de São Paulo

12 de dezembro de 2025

Instruções Gerais

- Esta prova possui **5 tarefas** valendo 2,0 pontos cada, totalizando 10,0 pontos.
- **Duração:** Três dias a partir da data de publicação.
- **Recursos permitidos:** ChatGPT, Internet, livros, materiais de estudo e consulta a documentação.
- **Recursos vetados:** Cópia explícita de solução de colegas ou de repositórios de outros alunos.
- **Forma de entrega:** Todas as soluções devem ser submetidas via **GitHub** em um repositório público.
- O repositório deve conter:
  - Código-fonte completo e funcional
  - Documentação explicando as soluções implementadas
  - Testes que demonstrem o funcionamento correto
  - README.md com instruções de compilação e execução
- **Data limite de entrega:** Enviar o link do repositório até 14/12/2025 às 23h59.
- A avaliação considerará: correção da implementação, qualidade do código, documentação, testes e compreensão dos conceitos demonstrada através dos comentários e documentação.

1 Tarefa 1: Implementação Básica com Deadlock (2,0 pontos)

Objetivo

Implementar uma versão básica do problema do Jantar dos Filósofos que demonstre o problema de deadlock.

Requisitos

1. Implemente em Java uma solução para o problema do Jantar dos Filósofos com 5 filósofos e 5 garfos.
2. Cada filósofo deve ser uma thread que executa em loop infinito, alternando entre:
   - **Pensar:** Simular um tempo aleatório de pensamento (entre 1 e 3 segundos)
   - **Comer:** Pegar os dois garfos (esquerdo e direito), simular um tempo de alimentação (entre 1 e 3 segundos) e soltar os garfos
3. Use `synchronized` para garantir exclusão mútua nos garfos.
4. Implemente um mecanismo de log que registre:
   - Quando um filósofo começa a pensar
   - Quando um filósofo tenta pegar um garfo
   - Quando um filósofo consegue pegar ambos os garfos e começa a comer
   - Quando um filósofo termina de comer e solta os garfos
5. Execute o programa por pelo menos 30 segundos e documente a ocorrência de deadlock (se houver).
6. Inclua no README.md uma explicação de por que essa implementação pode resultar em deadlock.

Critérios de Avaliação

- Código funcional e bem estruturado (0,5)
- Sistema de logging adequado (0,5)
- Documentação explicando o deadlock (0,5)
- Evidência de execução e análise dos resultados (0,5)

2 Tarefa 2: Solução com Prevenção de Deadlock (2,0 pontos)

Objetivo

Modificar a implementação da Tarefa 1 para prevenir deadlock usando a técnica de um filósofo pegar os garfos em ordem diferente.

Requisitos

1. Modifique a implementação da Tarefa 1 para que um dos filósofos (por exemplo, o filósofo de ID 4) pegue primeiro o garfo direito e depois o esquerdo, enquanto os outros pegam primeiro o esquerdo e depois o direito.
2. Mantenha o sistema de logging da Tarefa 1.
3. Execute o programa por pelo menos 2 minutos e verifique que não ocorre deadlock.
4. Documente no README.md:
    - Por que essa solução previne deadlock
    - Se ainda existe possibilidade de starvation e por quê
    - Comparação dos resultados com a Tarefa 1
5. Inclua estatísticas de execução: quantas vezes cada filósofo comeu durante o período de teste.

Critérios de Avaliação

- Implementação correta da solução (0,7)
- Sistema de estatísticas funcionando (0,5)
- Documentação técnica adequada (0,5)
- Análise crítica dos resultados (0,3)

3 Tarefa 3: Solução com Semáforos (2,0 pontos)

Objetivo

Implementar uma solução usando semáforos para limitar o número de filósofos que podem tentar pegar os garfos simultaneamente.

Requisitos

1. Implemente uma nova solução usando `Semaphore` do Java.
2. Use um semáforo para limitar a 4 o número de filósofos que podem tentar pegar os garfos ao mesmo tempo.
3. Mantenha os garfos como objetos sincronizados ou use semáforos individuais para cada garfo.
4. Implemente o mesmo sistema de logging das tarefas anteriores.
5. Execute o programa por pelo menos 2 minutos e colete estatísticas.
6. Documente no README.md:
    - Como a solução funciona
    - Por que ela previne deadlock
    - Comparação de desempenho com a solução da Tarefa 2
    - Vantagens e desvantagens dessa abordagem

Critérios de Avaliação

- Implementação correta com semáforos (0,7)
- Prevenção de deadlock garantida (0,5)
- Documentação completa (0,5)
- Análise comparativa (0,3)

4 Tarefa 4: Solução com Monitores e Garantia de Fairness (2,0 pontos)

Objetivo

Implementar uma solução usando monitores (synchronized blocks/methods) que garanta fairness e previna tanto deadlock quanto starvation.

Requisitos

1. Implemente uma solução usando uma classe `Mesa` que atua como monitor.
2. A classe `Mesa` deve gerenciar o acesso aos garfos de forma centralizada.
3. Implemente um mecanismo de fila ou prioridade que garanta que todos os filósofos tenham oportunidade de comer (evitar starvation).
4. Use `wait()` e `notifyAll()` para coordenar os filósofos.
5. Mantenha o sistema de logging e estatísticas.
6. Execute o programa por pelo menos 2 minutos.
7. Documente no README.md:
    - Como o monitor garante fairness
    - Como deadlock e starvation são prevenidos
    - Comparação com as soluções anteriores
    - Trade-offs entre as diferentes abordagens

Critérios de Avaliação

- Implementação correta do monitor (0,7)
- Mecanismo de fairness funcionando (0,6)
- Documentação técnica detalhada (0,4)
- Análise comparativa completa (0,3)

5 Tarefa 5: Análise Comparativa e Relatório Final (2,0 pontos)

Objetivo

Criar um relatório comparativo das diferentes soluções implementadas e uma análise crítica.

Requisitos

1. Execute todas as soluções (Tarefas 2, 3 e 4) por 5 minutos cada, coletando as seguintes métricas:
   - Número total de vezes que cada filósofo comeu
   - Tempo médio de espera entre tentativas de comer
   - Taxa de utilização dos garfos
   - Distribuição justa de oportunidades (coeficiente de variação do número de refeições)
2. Crie um relatório em Markdown (`RELATORIO.md`) contendo:
   - **Introdução:** Breve descrição do problema do Jantar dos Filósofos
   - **Metodologia:** Descrição de como os testes foram realizados
   - **Resultados:** Tabelas e gráficos (se possível) comparando as métricas
   - **Análise:** Comparação crítica das três soluções em termos de:
     - Prevenção de deadlock
     - Prevenção de starvation
     - Performance/throughput
     - Complexidade de implementação
     - Uso de recursos
   - **Conclusão:** Qual solução é mais adequada para diferentes cenários e por quê
3. Inclua no README.md principal:
   - Instruções de compilação e execução
   - Estrutura do projeto
   - Como executar os testes
   - Link para o relatório comparativo

Critérios de Avaliação

- Coleta adequada de métricas (0,6)
- Relatório bem estruturado e completo (0,7)
- Análise crítica e fundamentada (0,5)
- Qualidade da apresentação (0,2)

Estrutura Esperada do Repositório

O repositório GitHub deve seguir a seguinte estrutura:

```
prova-jantar-filosofos/
 README.md                  # Instruções gerais e estrutura
 RELATORIO.md               # Relatório comparativo (Tarefa 5)
 src/
    tarefa1/              # Implementação com deadlock
    tarefa2/              # Solução com ordem diferente
    tarefa3/              # Solução com semáforos
    tarefa4/              # Solução com monitores
 test/                      # Testes unitários (opcional, mas recomendado)
 docs/                      # Documentação adicional (opcional)
```

Observações Finais

- O código deve ser bem comentado, especialmente explicando as decisões de design.
- Use nomes de variáveis e métodos descritivos.
- Siga boas práticas de programação Java.
- Commits frequentes no Git são recomendados para demonstrar o processo de desenvolvimento.
- Em caso de dúvidas, consulte o material de referência ou o professor.
- Lembre-se: o objetivo é demonstrar compreensão dos conceitos, não apenas entregar código funcionando.

Boa prova!