Atividade 13
(Depuração e *Profiling* em java)

Pedro Oliveira de Sousa

4 de novembro de 2025

0.1 DEPURAÇÃO (*DEBUGGING*)

0.1.1 O QUE É DEPURAÇÃO E QUAL É SEU PRINCIPAL OBJETIVO NO DESENVOLVIMENTO DE SOFTWARE?

A depuração consiste no procedimento de identificar e remediar falhas ou defeitos presentes no código-fonte de um programa de software. Sempre que um software apresenta um funcionamento diferente do planejado, os programadores analisam o seu código com o objetivo de descobrir a origem dessas incorreções. Para isso, eles se valem de ferramentas específicas de depuração, que lhes permitem rodar o programa em um ambiente monitorado, examinar o código minuciosamente, e então estudar e solucionar a falha identificada.

A ocorrência de bugs e erros é inerente à programação de computadores, uma vez que se trata de uma tarefa intrinsecamente abstrata e conceptual. Os computadores processam dados através de sinais eletrônicos, e as linguagens de programação atuam criando uma abstração dessas informações para possibilitar uma interação mais eficiente dos seres humanos com as máquinas. Qualquer software é composto por inúmeras camadas de abstração, contendo diversos componentes que interagem entre si para o correto funcionamento da aplicação. Quando surgem falhas, localizá-las e resolvê-las pode ser uma tarefa complexa. As ferramentas e táticas de depuração existem para auxiliar na correção desses problemas de maneira mais ágil, aumentando a produtividade dos desenvolvedores. Consequentemente, essa prática resulta na melhoria da qualidade do software e na otimização da experiência do utilizador final.

0.1.2 QUAIS SÃO AS ETAPAS BÁSICAS PARA DEPURAR UM PROGRAMA JAVA USANDO UMA IDE(COMO ECLIPSE, INTELLIJ IDEA OU VS CODE)?

1. Defina um breakpoint na linha onde você suspeita que o problema começa.
2. Inicie o programa em modo Debug.
3. Quando o programa pausar, use Step Over e Step Into para avançar pelo código.
4. Observe os valores no painel Variables para ver onde eles divergem do esperado.
5. Use Resume para pular para o próximo ponto problemático.
6. Repita os passos até encontrar e entender a causa do bug.

0.1.3 CITE DUAS BOAS PRÁTICAS AO USAR O MODO DE DEPURAÇÃO E EXPLIQUE POR QUE ELAS AJUDAM O PROGRAMADOR

Definir Breakpoints Estratégicos e com um Propósito Claro: Isso evita que você se perca em partes irrelevantes do código, direcionando a atenção diretamente para a área problemática. Você passa a "conversar"com o código, fazendo perguntas específicas como "o que está entrando neste método?"ou "qual é o estado do sistema antes desta operação crítica?".

Inspecionar o Estado das Variáveis e a Pilha de Chamadas (Call Stack): A Call Stack responde à pergunta "como cheguei aqui?". Isso é crucial para entender o fluxo de execução que levou ao erro, especialmente em cenários complexos com muitas chamadas de métodos ou quando o problema se origina em um método diferente daquele onde o sintoma aparece. Ela permite que você "viaje no tempo"pela execução para encontrar a origem real do problema.

0.2 PROFILING (ANÁLISE DE DESEMPENHO)

0.2.1 O QUE É PROFILING E COMO ELE DIFERE DA DEPURAÇÃO?

A prática de profiling consiste no procedimento de analisar o tempo de execução de métodos, com o objetivo de identificar e solucionar pontos de estrangulamento (gargalos) que impactam o desempenho.

0.2.2 QUAIS MÉTRICAS DE DESEMPENHO UM PROFILER PODE MEDIR EM UM PROGRAMA JAVA (EXEMPLOS)?

Complexidade de tempo e espaço(memória) com o intuito de examinar a utilização da memória heap e a periodicidade das operações de coleta de lixo, estudar a alocação de objetos e suas referências.

0.2.3 CITE DUAS FERRAMENTAS DE PROFILING COMPATÍVEIS COM JAVA E DESCREVA BREVEMENTE O QUE CADA UMA FAZ.

O JProbe Suite é composto fundamentalmente por quatro ferramentas: o JProbe Memory Debugger, o JProbe Profiler, o JProbe Threadalyzer e o JProbe Coverage. O JProbe Memory Debugger auxilia o programador a erradicar vazamentos de memória, diminuir uma coleta de lixo excessiva e reconhecer objetos que mantêm referências a outros objetos no heap. O JProbe Profiler integra uma interface com gráfico de chamadas e a captura de dados para oferecer um diagnóstico sobre o desempenho do aplicativo. O JProbe Threadalyzer constitui um instrumento robusto para a identificação de complicações relacionadas a threads, tais como deadlocks, condições de corrida (race conditions) e outros. Por último, o JProbe Coverage serve de suporte para as equipes de desenvolvimento e de garantia de qualidade no mapeamento de seções de código que não são executadas, facilitando a avaliação da confiabilidade e da exatidão dos testes de unidade.

O Optimizeit é uma ferramenta que permite aos desenvolvedores otimizar o desempenho de uma ampla gama de componentes Java, como aplicações, applets, servlets e EJBs. Sua análise foca na JVM para identificar problemas de alocação de memória e uso ineficiente da CPU. Sua principal vantagem é a facilidade de uso: a configuração é feita através de um assistente simples, sem a necessidade de recompilar o programa com um compilador personalizado ou modificar classes previamente. O processo se inicia simplesmente executando o Optimizeit Profiler, que dá acesso a dois módulos principais:

1. Memory Profiler: Oferece uma visualização em tempo real das classes e instâncias alocadas. É utilizado para analisar referências de objetos e conta com funcionalidades como filtros para classes específicas, controle da coleta de lixo, grafos de referência e detecção de vazamentos de memória.

2. CPU Profiler: Apresenta os resultados de desempenho para threads, exibindo diversas estatísticas que podem ser filtradas.