As opções abaixo são as formas para se trabalhar com Threads em Java. Todas elas são acessíveis em Java 21+

Tipo / Abordagem	Categoria	É uma thread?	Justificativa técnica	Por que usar no TCC
Thread (herança de Thread)	**Thread real (SO)**	Sim	Cada instância cria uma thread do sistema operacional. Código e controle ficam acoplados.	Representa a forma mais direta e básica de concorrência. Serve como baseline.
Runnable (com Thread)	**Thread real (SO)**	Sim	Runnable define apenas a tarefa. Ao passá-lo para uma Thread, cria-se uma thread de SO.	Mostra separação entre lógica (tarefa) e execução (thread).
Callable + Future	**Thread real (SO)**	Sim	Callable encapsula uma tarefa que retorna valor. Precisa rodar dentro de uma thread de SO (normalmente via Executor).	Relevante porque combina execução concorrente com **retorno de resultados**.
Virtual Threads (Project Loom, Java 21+)	**Thread virtual (JVM)**	Sim (mas não do SO)	Criadas e agendadas pela JVM, mapeadas sobre poucas threads do SO. API é idêntica a Thread.	Importante por mostrar a inovação recente em Java: **concorrência massiva com baixo custo**.
Executors (ThreadPoolExecutor, ScheduledExecutorService)	**Gerenciador**	**Não**	Framework que cria e reutiliza threads de SO em pools. Usuário lida com tarefas, não com threads diretamente.	Permite comparar reuso e escalabilidade em cenários concorrentes.
ForkJoinPool	**Gerenciador especializado**	**Não**	Executor com política de *work-stealing*; usa internamente ForkJoinWorkerThread (threads de SO).	Relevante para evidenciar ganhos em algoritmos recursivos e divisíveis.

Existem 3 classes de comparação:
- Threads reais: mostram o funcionamento básico e os custos da criação direta de threads.
- Threads virtuais: introduz o novo paradigma (milhões de threads baratas).
- Gerenciadores: demonstram como o Java oferece camadas de abstração para maior eficiência e organização.

Qual a pergunta central?
Existem diferenças de desempenho entre essas técnicas na JVM sob alta carga? (comparamos só threads reais com virtual ou coloca gerenciador junto? Dar uma olhada nos trabalhos relacionados da tese pra ver se tem um q faça só isso… eu acho que o beronic 2022 faz)

Objetivo

Comparar **desempenho** e **comportamento** de três classes de modelos de threads no Java 21+: threads reais, threads virtuais e gerenciadores.

Para responder a pergunta central, algumas perguntas menores precisam ser testadas:

- Existem diferenças perceptíveis de desempenho entre esses modelos em cenários CPU-bound e I/O-bound?

Implementação

- **Benchmark 1 – CPU-bound:** cálculo de números primos ou multiplicação de matrizes (simples de implementar e medir).
    - Hipótese: *threads virtuais não trarão ganhos relevantes*.
- **Benchmark 2 – I/O-bound:** servidor HTTP simples atendendo requisições que simulam espera com Thread.sleep() de 100 ms (base na tese de referência)
    - Hipótese: *threads virtuais terão melhor desempenho em cenários com alta concorrência*.

O **Benchmark 3 (recursivo com ForkJoinPool)** pode ser opcional (extra), apenas se a aluna tiver tempo.

Experimentos

Em ambiente controlado (todos os testes dos trabalhos relacionados da tese foram feitos em Máquina Virtual com configuração semelhante… já a tese usou dois macs… qual é o mais compatível pros testes deste tcc?).

- **Load ramping:** aumentar gradualmente o número de tarefas/conexões até saturar.
    - *A tese fez o aumento da* taxa de requisições até a saturação; incremento de 50 rps
- **Constant load:** rodar com carga fixa intermediária.
    - A tese fez com uma taxa fixa correspondente a ~70% uso do recurso limitante (memória/CPU) (
- **Repetições:** 5 execuções por cenário e coleta nos moldes da disciplina de SPD

Métricas coletadas (intuito semelhante da tese: avaliar a experiência do usuário e o dimensionamento de hardware)

- Tempo total de execução – dg
- Para os testes:
    - I/O-bound: latência média – eu
    - CPU-bound: speedup – ep
- Throughput (requisições/segundo ou tarefas/segundo) – eu
- Uso médio de CPU e memória – dh
- Número de threads criadas (via monitoramento do processo) – **pág. 57**- es

Avaliações

- desempenho geral (dg)

- experiência do usuário (eu)

- eficiência da paralelização (ep)

- dimensionamento de hardware (dh)

- escalabilidade (es)

Ferramentas sugeridas

- **JDK próprio** (usar System.nanoTime() para tempos básicos).

- **VisualVM** → monitorar CPU, heap, threads em tempo real.
- **Spring Boot** → falaram que é tranquilo.

Conceitos necessários

- Concorrência x paralelismo.
- Custo de criação e gerenciamento de threads.
- Diferença entre **modelo manual** (Thread, Runnable, Callable) e **modelo gerenciado** (Executors, ForkJoinPool).
- Papel das **virtual threads** como alternativa moderna.

Resultado esperado

- Relatório comparativo simples (tabelas e gráficos).
- Identificação de cenários em que cada modelo se sai melhor ou pior.
- Discussão crítica sobre vantagens, desvantagens e trade-offs.