Aula 9

Introdução às Utilidades de Concorrência em Java

Concorrência vs Multithreading

- Multithreading
  - Suporte nativo desde o início: `Thread`, `Runnable`, `synchronized`, `wait()/notify()`.
  - Limitações do modelo tradicional:
    - Falta de ferramentas como semáforos, pools de threads e gerenciadores de execução.
    - Ineficiente para programas intensivamente concorrentes.
- Concurrency Utilities (JDK 5+)
  - Benefícios:
    - Controle avançado
    - Escalabilidade em sistemas multicore

Visão Geral da API

- **java.util.concurrent:**
  - Sincronizadores, Executores, Coleções Concorrentes, Fork/Join Framework.
- **java.util.concurrent.atomic:**
  - Operações atômicas sem locks (ex.: `AtomicInteger`).
- **java.util.concurrent.locks:**
  - Alternativa ao `synchronized` com maior flexibilidade (ex.: Lock).

Principais Sincronizadores

Classe	Finalidade
Semaphore	Controla acesso com contador de permissões
CountDownLatch	Espera por um número fixo de eventos
CyclicBarrier	Ponto de espera para grupo de threads
Exchanger	Troca de dados entre duas threads
Phaser	Sincronização de fases múltiplas

Semaphore - Controle de Acesso

- **Conceito:** Usa contador de permissões para gerenciar acesso a recursos.
- **Métodos principais:**
  - `acquire()`: Obtém permissão (bloqueia se contador = 0).
  - `release()`: Libera permissão.
- **Exemplo:** SemDemo
  - 1 thread incrementa, outra decrementa, acesso sincronizado.

```
Semaphore sem = new Semaphore(1);
sem.acquire(); // Bloqueia se não disponível
Shared.count++;
sem.release(); // Libera permissão
```

CountDownLatch - Espera por Eventos

- **Conceito:** Bloqueia até que um número de eventos ocorra.
- **Métodos principais:**
  - `await()`: Espera até contador chegar a 0.
  - `countDown()`: Decrementa contador.
- **Exemplo:** CDLDemo
  - Thread principal espera 5 eventos.

```
CountDownLatch cdl = new CountDownLatch(5);
cdl.await(); // Espera até 5
```

CyclicBarrier - Ponto de Sincronização

- **Conceito:** Threads esperam até que todas cheguem ao ponto.
- **Métodos principais:**
  - `await()`: Pausa até número de threads ser atingido.
- **Exemplo:** BarDemo
  - 3 threads aguardam barreira.

```
CyclicBarrier cb = new CyclicBarrier(3, new BarAction());
cb.await(); // Espera 3 threads
```

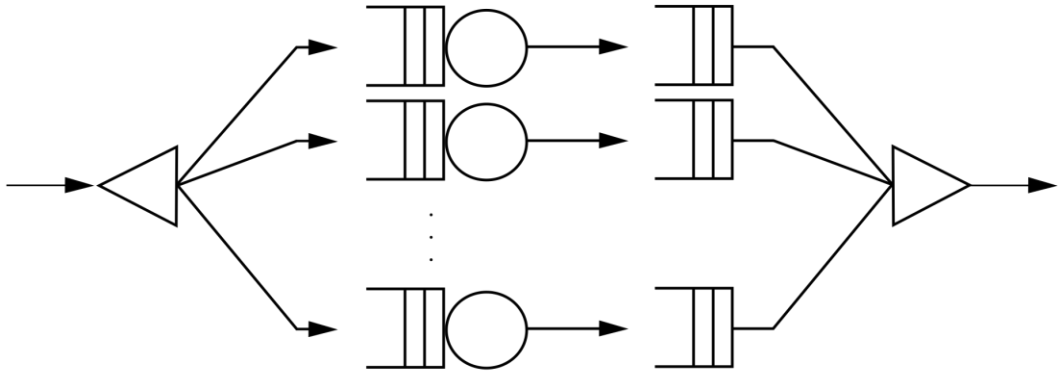
Executors - Gerenciamento de Threads

- **Conceito:** Alternativa ao controle manual de threads.
- **Classes principais:**
  - `ThreadPoolExecutor`: Pool gerenciado.
  - `ScheduledThreadPoolExecutor`: Agendamento.
- **Métodos de fábrica:**
  - `newFixedThreadPool(n)`: Pool fixo.
  - `newCachedThreadPool()`: Pool dinâmico.
- **Exemplo:** SimpExec
  - 4 tarefas em pool de 2 threads.

```
ExecutorService es = Executors.newFixedThreadPool(2);
es.execute(new MyThread(cdl, "A"));
```

Fork/Join Framework - Programação Paralela

- **Conceito:** Divide-e-conquiste para sistemas multicore.
- **Classes principais:**
  - `ForkJoinPool`: Gerencia tarefas.
  - `RecursiveAction`: Tarefa sem retorno.
  - `RecursiveTask`: Tarefa com retorno.
- **Estratégia:** Divisão recursiva até limiar sequencial.



Exemplos de Fork/Join

- **Exemplo 1: SqrtTransform (RecursiveAction)**
  - Transforma array em raízes quadradas.
  - `invokeAll()` para subtarefas.

```
if ((end - start) < seqThreshold) {
  for (int i = start; i < end; i++) data[i] = Math.sqrt(data[i]);
} else {
  invokeAll(new SqrtTransform(data, start, middle), ...);
}
```

- **Exemplo 2: Sum (RecursiveTask)**
  - Soma array com `fork()` e `join()`

```
Sum task = new Sum(nums, 0, nums.length);
double summation = fjp.invoke(task);
```

Conclusão

- **Comparação:** `synchronized` para simplicidade; Concurrency Utilities para controle avançado.
- **Dicas:**
  - Evitar bloqueios excessivos.
  - Escolher limiares adequados.
- **Aplicações reais:**
  - Processamento paralelo (ex.: big data).
  - Otimização em multicore.