**Monitores**

**->** Um monitor encapsula **dados privados e procedimentos** que os acessam (semelhante a uma classe)

**->** Apenas um processo pode estar ativo no monitor em um dado instante (exclusão mútua entre processos)

**->** Sincronização feita usando variáveis de condição: **wait() e signal().**

**Signal():**

* **Signal-and-exit:**
  + O processo atual deve sair do monitor após o signal().
  + O processo sinalizado entra no monitor.
* **Signal-and-continue:**
  + O processo atual continua executando
  + O processo sinalizado compete pelo monitor no próximo escalonamento.

Java tem suporte parcial a monitores através de métodos synchronized.

Exclusão mútua no acesso ao objeto.

* Operações: wait(), notify(), notifyAll().
* Semântica signal-and-continue

**Semáforos vs monitores:**

* Monitores são fáceis de programar, e reduzem a probabilidade de erros (já incorporam a exclusão mútua).
* Monitores dependem de linguagem de programação, enquanto semáforos são implementados pelo SO.
  + Podem ser usados com C, Java, BASIC, ASM,...
* Ambos podem ser usados apenas em sistemas centralizados (com memória compartilhada)
  + Sistemas distribuídos usam troca de mensagens.

**Barreiras**

-> Mecanismo usado para definir um ponto de sincronização para múltiplos processos/threads.