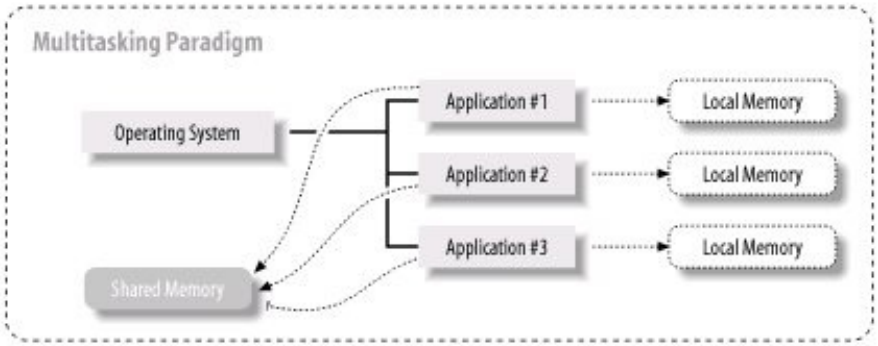
Multi-Thread

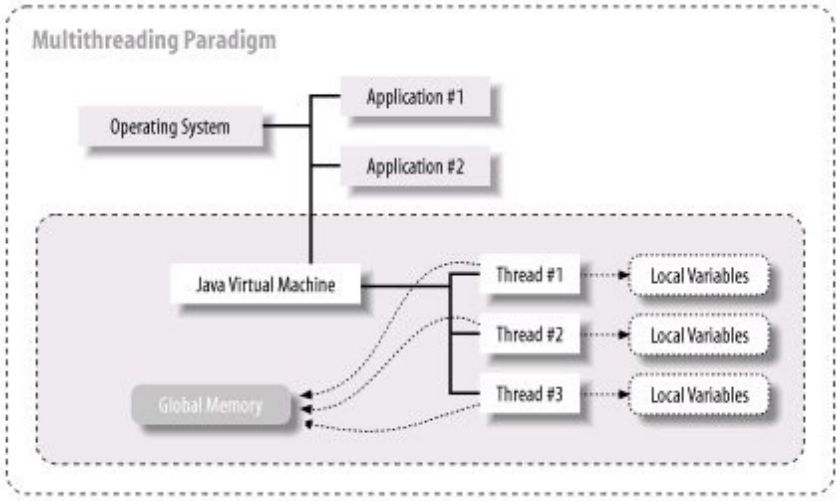
Multi-Thread

- **Multiprocessamento**
  - A capacidade de ter mais de um programa funcinando no que parece ser simultâneo
- Duas maneiras
  - O Sistema Operacional interromper o programa sem consultar primeiro
  - Os programas são interrompidos apenas quando estão querendo produzir controle



Multi-Thread

- **Linhas de Execução (Multi-Thread)**
  - Num mesmo processo cada tarefa é chamada de Thread (linha de execução).
- Qual é a diferença?
  - Cada processo tem um conjunto completo de variáveis próprias, as threads compartilham os mesmos dados.
  - É necessário muito menos sobrecarga para criar e destruir threads individuais.



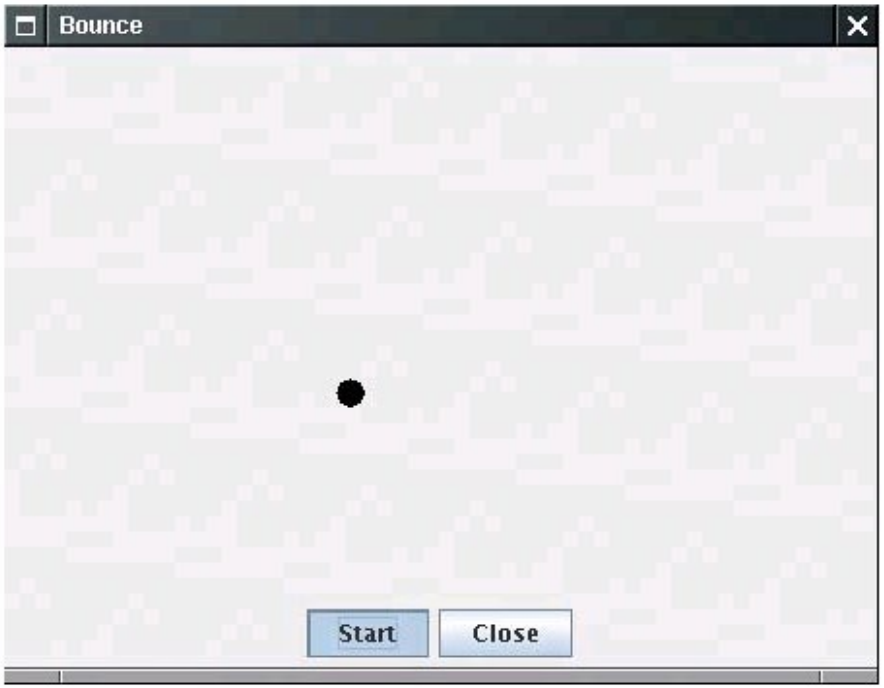
Multi-Thread

Exemplos

- Um navegedor deve tratar com vários hosts, abrir uma janela de correio eletrônico ou ver outra página, enquanto descarrega dados.
- A própria linguagem de programação Java usa uma thread para fazer coleta de lixo em segundo plano evitando assim o proglema de gerenciar memória!
- Os programas GUI têm uma thread separada para reunir eventos da interface com o usuário do ambiente operacional hospedeiro.

Multi-Thread

- **Usando Threads**
- Duas Threads:
  - Uma para a bola que quica
  - Outra para a thread principal, que cuida da interface com o usuário
- Para fazer nosso programa da bola que quica em uma thread separada:
  - Precisamos apenas de estender a classe Thread ou implementar a interface Runnable.



Multi-Thread

- **Classes MultiThread**
  - Estender Thread ou implementar a interface Runnable
  - Colocar o código que precisar ser executado no método run.

```
public class NovaTread extends Thread {
  // Outros atributos e métodos

  @Override
  public void run() {
    // Executa um conjunto de operações
  }
}

public class NovaClasse implements Runnable
  // Outros atributos e métodos

  @Override
  public void run() {
    // Executa um conjunto de operações
  }
}
```

Multi-Thread

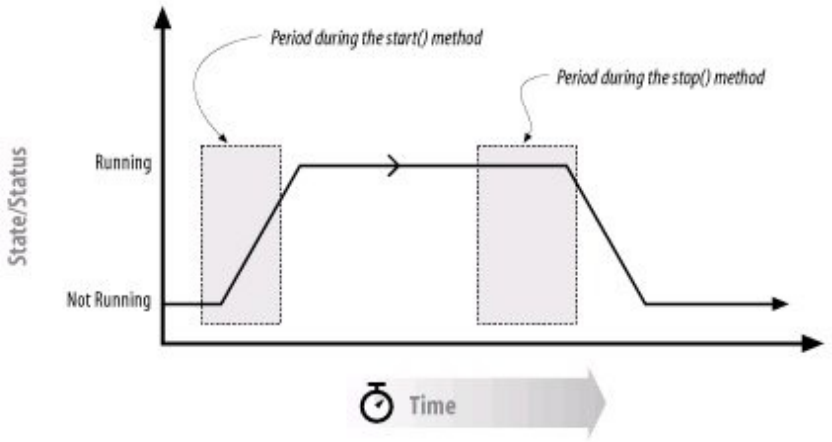
- **Executanto a Thread**
  - Criar um objeto Thread no caso das classes que implementam Runnable
  - Ativar a thread através da chamada do método start()

```
public class OutraClasse {
  public static void main(String[] args) {
    // Criando e iniciando uma Thread
    NovaThread umaThread = new NovaThread();
    umaThread.start();

    // Criando e iniciando um Objeto Runnable
    NovaClasse umRunnable = new NovaClasse();
    Thread aThread = new Thread(umRunnable);
    aThread.start();
  }
}
```

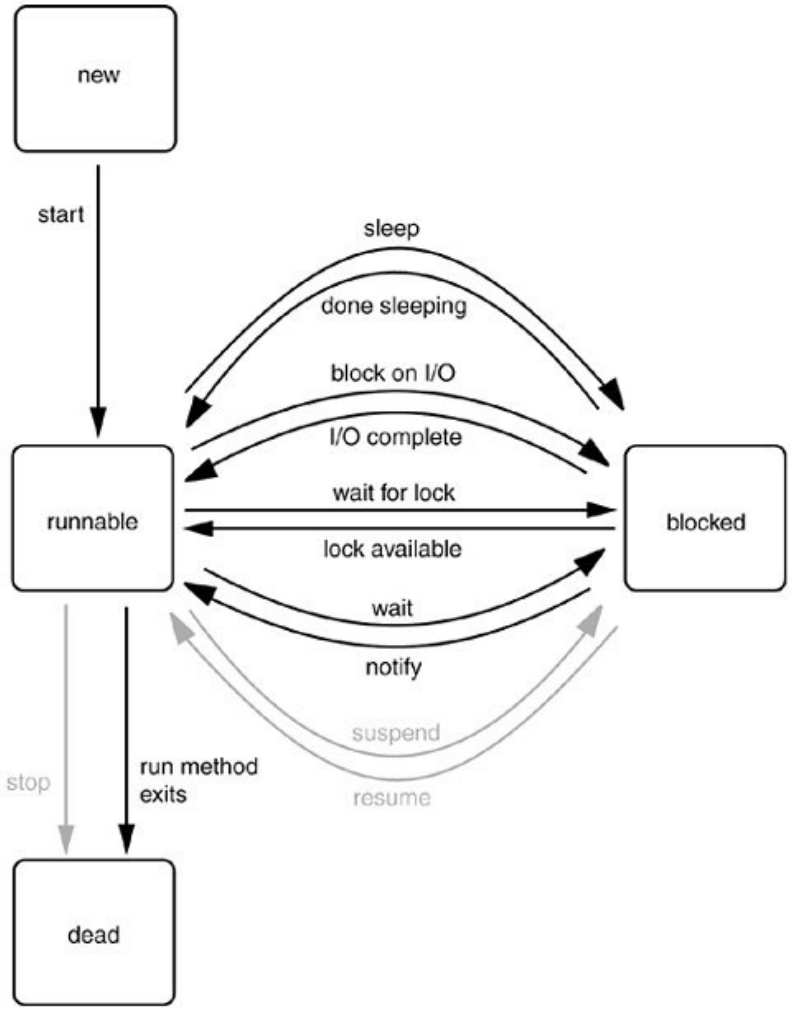
Multi-Thread

- **Estados de linha de execução**
  - Novo
  - Passível de execução
  - Bloqueada
  - Morta
- **Threads novas**
  - Quando você cria uma thread com o operador new, neste estado o método run() não começou a executar.



Multi-Thread

- **Threads passível de execução**
  - Quando você chama o método start, a thread é passível de execução (runnable).
  - *Uma* thread *passível de execução, não significa estar em execução (ela pode estar aguardando na fila de execução de threads).*



Multi-Thread

- **Threads bloqueadas**
  - Alguém chama o método sleep() da thread.
  - A thread chama uma operação que está bloqueando entrada/ Saída (I/O).
  - A thread chama o método wait()
  - A thread tenta bloquear um objeto que está bloqueado por outra execução.
  - Quando uma thread está bloqueada, outra thread é escalada para execução.
- **Threads Mortas**
  - Ela morre de morte natural, pois o método run encerra normalmente.
  - Ela morre abruptamente, pois uma exceção não capturada encerra o método run.
  - Para descobrir se uma thread está viva, use o método isAlive().

Multi-Thread

- **Interrompendo uma Thread**
  - O método run da thread deve testar pela interrupção de sua execução com o utilizando o método isInterrupted()
  - Chamar o metodo interrupted() da thread para solicitar a sua interrupção

```
public void run() {
 for (;;) {
  // Executa algumas operações

  // Testa se a Thread foi interrompida
  if (Thread.interrupted()) {
   // Thread Interrompida, retorne
   return;
  }
 }
}
```

```
public void run() {
 for(;;) {
  try {
   // Executa algumas operações

   // Espera 5 segundos
   Thread.sleep(5000);
  } catch (InterruptedException ex) {
   // Thread Interrompida, retorne
   return;
  }
 }
}
```

Multi-Thread

- **Prioridades da Thread**
  - MIN_PRIORITY (configurada como 1 na classe Thread)
  - NORM_PRIORITY (configurada como 5)
  - MAX_PRIORITY (configurada como 10)
  - A prioridade é atribuida através do método setPriority() da Thread

Multi-Thread

- **Grupo de linhas de Execução**
  - Útil no gerenciamento de múltiplas Linhas de execução com a mesma funcionalidade.
  - É necessária a instanciação da classe ThreadGroup e a adição das linhas de execução a esta instância.

```
// Criando um ThreadGroup
ThreadGroup grupo = new ThreadGroup("serviço");

// Criando e iniciando Treads
for (int i = 0; i < 5; i++) {
  NovaClasse run = new NovaClasse();
  Thread t = new Thread(grupo, run);
  t.start();
}
```

Multi-Thread

- **Grupo de linhas de Execução**
  - É possível parar todas as linhas de execução de um Grupo de linhas de execução simplemente invocando o método interrupt do objeto ThreadGroup

```
if(interrompe)
  grupo.interrupt();
```

  - Para obter o número de Threads ativas num GroupThread invoque o método activeCount()

```
int threadsAtivas = grupo.activeCount();
```

Sincronismo

- Na maioria dos aplicativos com múltiplas linhas de execução duas ou mais linhas precisam compartilhar o acesso aos mesmos objetos.
- Se duas linhas de execução têm acesso ao mesmo objeto pode acontecer a corrupção dos dados alterados, resultando em objetos danificados

Multi-Thread

- Para resolver este problema existem duas maneiras:
  - Sincronizar o método

```
public synchronized void calcula() {
  // executa operações
}
```

  - Sincronizar o acesso ao objeto

```
public void executa() {
  synchronized (objeto) {
    // executa operações
  }
}
```