Programação Orientada a Objetos (PO24CP-4CP)

Aula #14 Threads em Java

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Threads - Programação Concorrente

Aplicação com uma única Thread (linha de execução)

```
public class Fluxo1{
  public void disparar(){
    for(int i=0; i<10;i++){
      System.out.println("Fluxo 1");
    }
  }
}
public class Fluxo2{
    public void disparar(){
      for(int i=0; i<10;i++){
      System.out.println("Fluxo 2");
      }
    }
}
public class Principal{
  public static void main(String[] args){
    Fluxo1 f1 = new Fluxo1(); Fluxo2 f2 = new Fluxo2();
      f1.disparar();
      f2.disparar();
      System.out.println("Fim do programa");}
}
```

Fluxo 1
Fluxo 1
⋮
Fluxo 1
Fluxo 1
Fluxo 2
Fluxo 2
⋮
Fluxo 2
Fluxo 2
Fim do programa

Programação concorrente

Onde está presente?

Sistemas operacionais modernos são caracterizados como **multitarefas**

- Executam **diversos processos simultaneamente**
- E quando esses processos precisam trocar informações?

Comunicação entre processos

Cada processo possui suas próprias variáveis e a troca de informações entre processos é feita através de **arquivos em disco** ou através de **sockets de rede**

E o que são *threads*? (1/2)

São linhas de execução ou processos leves que permitem a um **aplicativo** realizar diversas tarefas de forma concorrente.

E o que são *threads*? (2/2)

Permite que um processo realize diversas tarefas de forma concorrente

- Uma *thread* fica responsável por interagir com o usuário (leitura de teclas) e
- Outra *thread* fica responsável por escrever em um *sockets* de rede

Vantagem

Por estarem dentro de um mesmo processo, compartilhando variáveis, a comunicação entre *threads* tende a ser mais eficiente e mais fácil de programar

Desenvolvendo aplicação *multithread* em Java

- Para desenvolver uma aplicação *multithread* é necessário:
  1. Escrever o código que será executado pela *Thread*
  2. Escrever o código que irá disparar a *Thread*

Desenvolvendo aplicação *multithread* em Java

- Para desenvolver uma aplicação *multithread* é necessário:
  1. Escrever o código que será executado pela *Thread*
  2. Escrever o código que irá disparar a *Thread*

Em Java é possível criar uma Thread de duas formas:

1. Criar uma classe que estenda a classe **Thread**
   - Deve-se sobrescrever o método `public void run()`
2. Criar uma classe que implemente a interface **Runnable**
   - Deve-se implementar o método `public void run()`
   - Opção interessante já que Java não possui o conceito de herança múltipla

Threads em Java - Exemplo com Thread e Runnable

- Herança

```
public class Fluxo1 extends Thread{
    public void run(){
      for(int i=0; i<1000;i++){
       System.out.println("Fazendo uso de heranca");
      }
    }
}
```

- Interface

```
public class Fluxo2 implements Runnable{
  public void run(){
    for(int i=0; i<1000;i++){
      System.out.println("Fazendo uso de interface");
    }
  }
}
```

Exemplo de uso com herança e interface

```
public static void main(String[] args){
  Thread comHeranca = new Fluxo1();
  Thread comInterface = new Thread(new Fluxo2());

  //executando as threads
  comHeranca.start();
  comInterface.start();

  System.out.println("Fim do programa");
}
```

Qual será a saída do programa acima?

1. 1000 linhas com herança + 1000 linhas com interface + Fim do programa
2. Fim do programa + 1000 linhas com herança + 1000 linhas com interface
3. Não tenho como prever

Exemplo de uso com herança e interface

```
public static void main(String[] args){
  Thread comHeranca = new Fluxo1();
  Thread comInterface = new Thread(new Fluxo2());

  //executando as threads
  comHeranca.start();
  comInterface.start();

  System.out.println("Fim do programa");
}
```

Qual será a saída do programa acima?

1. 1000 linhas com herança + 1000 linhas com interface + Fim do programa
2. Fim do programa + 1000 linhas com herança + 1000 linhas com interface
3. Não tenho como prever

Exemplo de uso com herança e interface

```
public static void main(String[] args){
  Thread comHeranca = new Fluxo1();
  Thread comInterface = new Thread(new Fluxo2());

  //executando as threads
  comHeranca.start();
  comInterface.start();

  System.out.println("Fim do programa");
}
```

Qual será a saída do programa acima?

1. 1000 linhas com herança + 1000 linhas com interface + Fim do programa
2. Fim do programa + 1000 linhas com herança + 1000 linhas com interface
3. Não tenho como prever

Exemplo - Carro de Corrida (sem *Threads*)

```
package threads;
public class CarroDeCorrida{
 private String nome;
 private int distância;
 private int velocidade;
 public CarroDeCorrida(String n, int vel){
  nome = n;
  distância = 0;
  velocidade = vel;
 }
 public void executa(){
  while (distância <= 1200)  {
   System.out.println(nome + " rodou " + distância + " km.");
   distância += velocidade;
   // Causa um delay artificial.
   for (int sleep = 0; sleep < 1000000; sleep++)
   double x = Math.sqrt(Math.sqrt(Math.sqrt(sleep)));
  }
 } }
```

Exemplo - Carro de Corrida (sem *Threads*)

```
package threads;
public class SimulacaoSemThreads {
 public static void main(String[] args){
  // Criamos instâncias da classe
       CarroDeCorrida.
  CarroDeCorrida penélope =
  new CarroDeCorrida("Penélope
       Charmosa",60);
  CarroDeCorrida dick =
  new CarroDeCorrida("Dick Vigarista"
       ,100);
  CarroDeCorrida quadrilha =
  new CarroDeCorrida("Quadrilha da
       Morte",120);
  // Criados os carros, vamos executar as
       simulações.
  penélope.executa();
  dick.executa();
  quadrilha.executa();
 }
}
```

```
Penélope Charmosa rodou 0 km.
Penélope Charmosa rodou 60 km.
Penélope Charmosa rodou 120 km.
...
Penélope Charmosa rodou 1140 km.
Penélope Charmosa rodou 1200 km.
Dick Vigarista rodou 0 km.
Dick Vigarista rodou 100 km.
Dick Vigarista rodou 200 km.
...
Dick Vigarista rodou 1100 km.
Dick Vigarista rodou 1200 km.
Quadrilha da Morte rodou 0 km.
Quadrilha da Morte rodou 120 km.
...
Quadrilha da Morte rodou 1080 km.
Quadrilha da Morte rodou 1200 km.
```

Exemplo - Carro de Corrida com *Threads*

```
package threads;
public class CarroDeCorridaComThreads extends Thread{
 private String nome;
 private int distância;
 private int velocidade;
 public CarroDeCorridaComThreads(String n, int vel){
  nome = n;
  distância = 0;
  velocidade = vel;
 }
 public void run(){
  while (distância <= 1200){
   System.out.println(nome + " rodou " + distância + " km.");
   distância += velocidade;
   // Causa um delay artificial.
   for (int sleep = 0; sleep < 1000000; sleep++)
   double x = Math.sqrt(Math.sqrt(Math.sqrt(sleep)));
  }
 }
}
```

Exemplo - Carro de Corrida com *Threads*

```
package threads;
public class SimulacaoComThreads
{
 public static void main(String[] args){
  // Criamos instâncias da classe CarroDeCorrida.
  CarroDeCorridaComThreads penelope =
  new CarroDeCorridaComThreads("Penelope
      Charmosa",60);
  CarroDeCorridaComThreads dick =
  new CarroDeCorridaComThreads("Dick Vigarista
      ",100);
  CarroDeCorridaComThreads quadrilha =
  new CarroDeCorridaComThreads("Quadrilha da
      Morte",120);
  // Criados os carros, vamos executar as
      simulacoes.
  penelope.run();
  dick.run();
  quadrilha.run();
 }
}
```

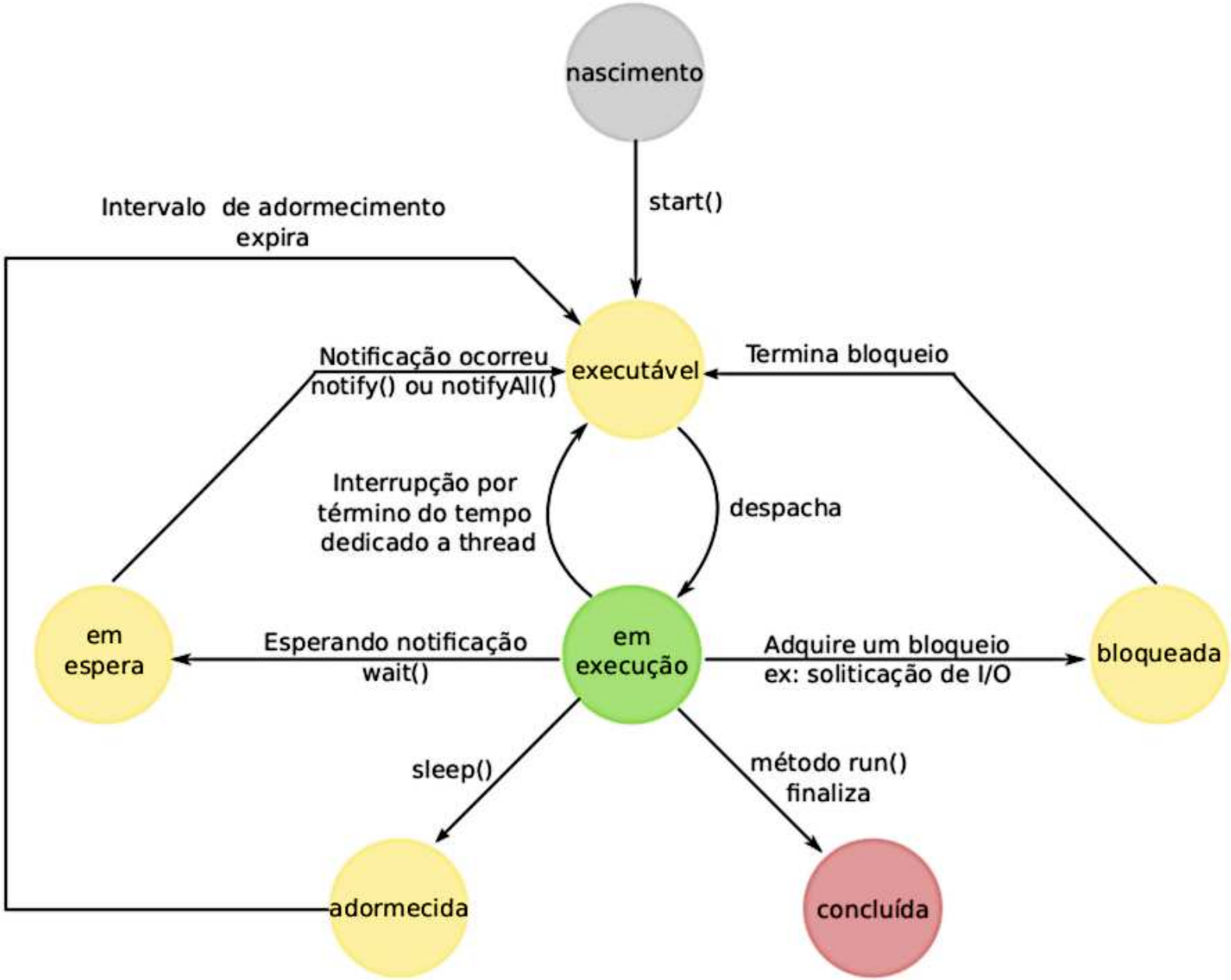
```
Penélope Charmosa rodou 0 km.
Penélope Charmosa rodou 60 km.
Penélope Charmosa rodou 120 km.
...
Penélope Charmosa rodou 1140 km.
Penélope Charmosa rodou 1200 km.
Dick Vigarista rodou 0 km.
Dick Vigarista rodou 100 km.
Dick Vigarista rodou 200 km.
...
Dick Vigarista rodou 1100 km.
Dick Vigarista rodou 1200 km.
Quadrilha da Morte rodou 0 km.
Quadrilha da Morte rodou 120 km.
...
Quadrilha da Morte rodou 1080 km.
Quadrilha da Morte rodou 1200 km.
```

Exemplo - Carro de Corrida com *Threads*

```
package threads;
public class SimulacaoComThreads
{
 public static void main(String[] args){
  // Criamos instâncias da classe CarroDeCorrida.
  CarroDeCorridaComThreads penelope =
  new CarroDeCorridaComThreads("Penelope
      Charmosa",60);
  CarroDeCorridaComThreads dick =
  new CarroDeCorridaComThreads("Dick Vigarista
      ",100);
  CarroDeCorridaComThreads quadrilha =
  new CarroDeCorridaComThreads("Quadrilha da
      Morte",120);
  // Criados os carros, vamos executar as
      simulacoes.
  penelope.start();
  dick.start();
  quadrilha.start();
 }
}
```

```
Penélope Charmosa rodou 0 km.
Dick Vigarista rodou 0 km.
Quadrilha da Morte rodou 0 km.
Penélope Charmosa rodou 60 km.
Dick Vigarista rodou 100 km.
Quadrilha da Morte rodou 120 km.
...
Quadrilha da Morte rodou 1080 km.
Penélope Charmosa rodou 600 km.
Dick Vigarista rodou 1000 km.
...
Penélope Charmosa rodou 1140 km.
Penélope Charmosa rodou 1200 km.
```

Ciclo de vida de uma thread



Principais métodos para trabalhar com threads

start	Ocorre a invocação do método **run** da Thread. • Após disparar a *thread*, o fluxo de execução retorna para o seu chamador imediatamente
run	Onde é colocada a lógica do fluxo • Ao finalizar este método, a *thread* morre
sleep	Faz com que a *thread* durma por alguns milisegundos • **Importante:** Enquanto uma *thread* dorme, ela não disputa o processador • Exemplo de uso dentro do método **run**: `Thread.sleep( 1000 );`
join	Espera que a *thread* que fora invocada morra antes de retornar para a *thread* que a invocou

Fazendo a thread dormir por 1000 milisegundos

```
public class Fluxo3 extends Thread{

  public Fluxo3 (String nome){
     super(nome);
  }

  public void run( ){
    try{
       System.out.println(this.getName( ) + " vair dormir... ") ;
        Thread.sleep(1000) ;
      }
     catch (InterruptedException e ){
        System.err.println(e.toString( )) ;
      }
      System.out.println(this.getName( ) + " acordou...");
  }
}
```

Exemplo de uso do método join

```
public static void main(String[] args){
  Thread f3 = new Fluxo3("fluxo 3");

  //disparando a thread
  f3.start();
  System.out.println("Depois do start e antes do join");
  try{
   f3.join();
   //a linha abaixo e' executada somente depois
   //de finalizar o metodo run do objeto f3
   System.out.println("Depois do join");

   }catch(InterruptedException ex) {
        System.err.println(ex.toString());
   }

   System.out.println("Fim do programa");
}
```

Concorrência e sincronismo entre threads

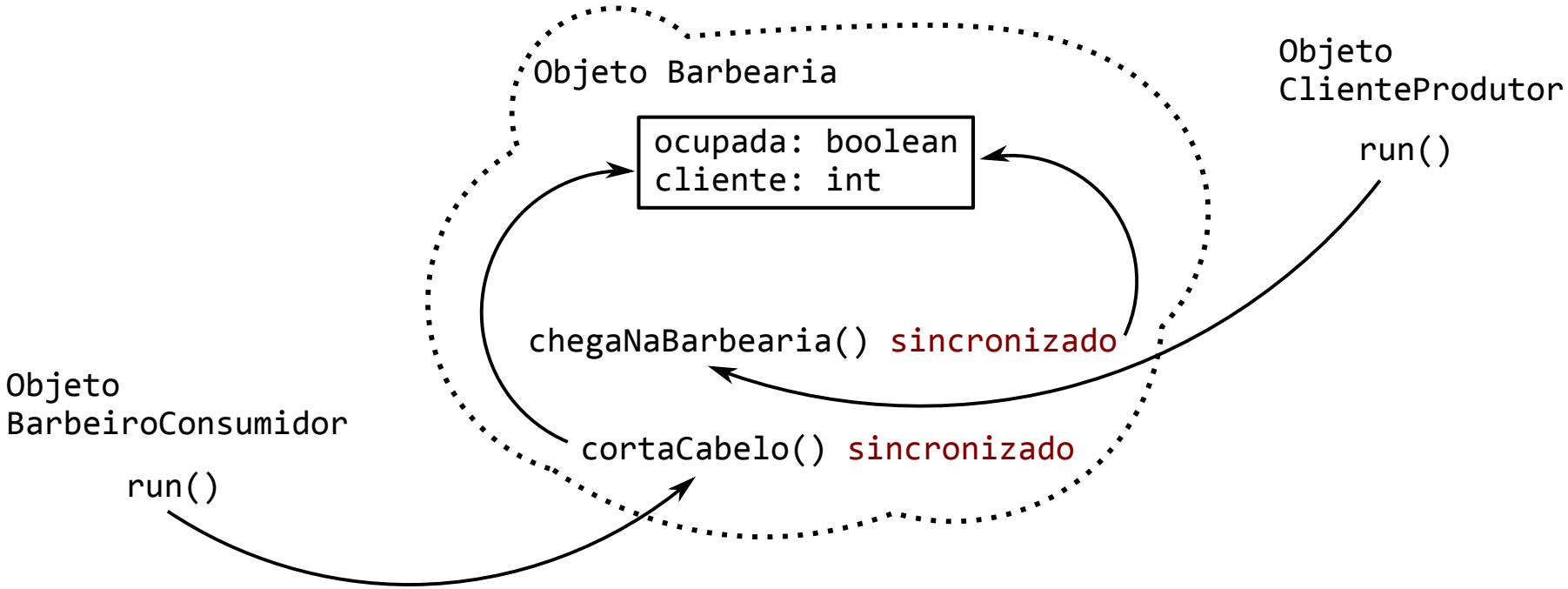
Memória compartilhada entre as *threads*

- Requer a **sincronização** das ações que serão executadas sobre essa memória.
  - Somente uma *thread* por vez pode acessar essa memória compartilhada

- Java implementa o conceito de `monitor` para impor o acesso mutuamente exclusivo aos métodos
  - Tais métodos devem apresentar a palavra **synchronized**
- Quando um método sincronizado é executado o monitor é consultado
  - **Se** não existir outro método sincronizado em execução, então **continua**; **Senão**, aguarde pela **noticação**
- Métodos para trabalhar com sincronismo:
  - **wait, notify** e **notifyAll;**

Exemplo Barbearia

- Simulação: barbeiro chega na barbearia para atender no máximo 10 clientes. Enquanto a cadeira esta vazia, o barbeiro tira um cochilo...



Exercício: Olá mundo reverso

- Escreva um programa chamado **OlaMundoReverso** o qual deverá disparar uma *thread*, chamada Ola-01. A *thread* Ola-01 deve criar a *thread* Ola-02. A *thread* Ola-02 deve criar a *thread* Ola-03 e assim sucessivamente até criar a *thread* Ola-10.

  Cada *thread* deverá imprimir "Olá mundo, sou a *thread* Ola-XX", contudo essa mensagem de saudação deve ser impressa na ordem inversa da criação da *thread*. Ou seja, a *thread* Ola-10 deve ser a primeira a imprimir a mensagem e a *thread* Ola-01 deverá ser a última a imprimir.