Programação Orientada a Objetos

Módulo 2

Vaux Gomes [1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará
Campus Jaguaribe

7 de Agosto de 2022

Sumário

Threads

Introdução

- As threads correspondem a linhas de controle independentes no âmbito de um mesmo processo.
- No caso da linguagem Java, é precisamente o conceito de *thread* que é utilizado como modelo para a **programação concorrente**, permitindo que uma aplicação em execução numa máquina virtual Java possa ter várias linhas de execução concorrentes em que cada uma corresponde a uma *thread*.

Threads

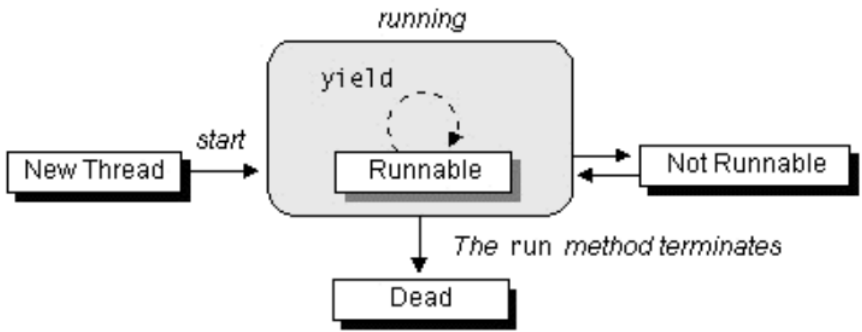
Threads em Java

- O suporte para threads na linguagem Java é realizado através da classe `java.lang.Thread` cujos métodos definem o API disponível para a gestão de *threads*.
- Entre as operações disponíveis incluem-se métodos para:
  - iniciar,
  - executar,
  - suspender e
  - gerir a prioridade de uma *thread*.
- O código que define o que uma *thread* vai fazer é o que estiver no método `run`.
- A classe Thread é uma implementação genérica de uma *thread* com um método run vazio, cabendo ao programador definir esse código para uma *thread* em particular.

Threads

Threads em Java

- O ciclo de vida de uma thread **começa com a criação** do respectivo objeto, **continua com a execução do método** `run` (iniciada através da invocação do método `start`), e irá **terminar quando terminar a execução do método** `run`.



Threads

Threads em Java

- Umas das técnicas para criar uma nova *thread* de execução numa aplicação Java é criar uma subclasse de Thread e re-escrever o método `run`.
- A outra forma de criar uma Thread é declarar uma classe que implementa o interface Runnable e que implementa o método `run`

Criando Threads

Exemplo 1

```
class Assobiar extends Thread {
    long frequencia;

    Assobiar(long frequencia) {
        this.frequencia = frequencia;
    }

    public void run() {
        // codigo a executar pela thread
    }
}

// MAIN
Assobiar a = new Assobiar(10);
a.start();
```

Criando Threads

Exemplo 2

```
class Assobiar implements Runnable {
    long frequencia;

    Assobiar(long frequencia) {
        this.frequencia = frequencia;
    }

    public void run() {
        // codigo a executar pela thread
    }
}

// MAIN
Assobiar a = new Assobiar(10);
new Thread(a).start();
```

Demonstracao

Aplicações concorrentes

- Em algumas aplicações encontraremos casos de utilização de um mesmo recurso pelas *threads*.
- Mais que isso, pode acontecer deste recurso não poder ser utilizado por mais de uma *thread* por vez.
- Assim verificamos que neste cenário temos a existência de *threads* concorrentes.
- Exemplos:
  - Sistema de semáforos do SO
  - *Buffer* de uma impressora
  - etc.

Prática

Sincronização de threads

- A existência de Threads concorrentes levanta a necessidade de sincronização.
- Em Java, cada objeto tem associado um monitor através do qual é possível garantir o acesso exclusivo às seções críticas desse mesmo objeto.
- O controle de acesso às seções críticas de um objeto é feito de forma automática pelo sistema de run-time do Java . O que o programador precisa de fazer é apenas assinalar as seções críticas usando para tal a palavra reservada `synchronized`.

Sincronização de threads

- Um bloco de código sincronizado é uma região de código que apenas pode ser executado por uma *thread* de cada vez.
- Pode-se declarar um método como sendo *synchronized*:
  - `synchronized int pull () {...}`
- Ou pode-se declarar apenas uma parte de um método como sendo synchronized:
  - `synchronized (this) {...}`

Demonstracao

Dependência de estado e guardas de métodos

- A invocação de um método num objeto pode depender de condições associadas ao estado desse objeto.
- As ações dependentes do estado são métodos cuja execução pode não ser possível num determinado momento mas cuja viabilidade futura depende de algum evento externo que poderá ocorrer a qualquer momento.

Dependência de estado e guardas de métodos

- Qual a atitude a tomar quando o método é invocado mas a pré-condição necessária à sua execução[1] não se verifica?
- Para decidir o que fazer nesses casos podem ser seguidas essencialmente duas abordagens:
  - Uma abordagem optimista (*liveness first*) é assumir que mais tarde ou mais cedo a pré-condição há de ser verdadeira e por isso espera-se.
  - Uma abordagem conservadora (*safety first*) admite que a pré-condição pode nunca vir a ser verdadeira (pelo menos em tempo útil) e por isso se num determinado momento não é possível executar o método então mais vale devolver uma indicação de erro.
  - Uma abordagem intermédia consiste em esperar mas definir um tempo limite.

Dependência de estado e guardas de métodos

- Estas várias abordagens correspondem então às seguintes políticas:
  - **Balking**: Caso a invocação do método não seja possível a sua execução é recusada
  - **Guarded Suspension (guarded waits)**: Execução é suspensa até que a pré-condição seja verdadeira. Implica que outras *threads* alterem o estado do objeto!
  - **Timed Waits**: Execução é suspensa até que a pré-condição seja verdadeira ou decorra um tempo máximo definido.

[1] por exemplo existirem caixas no depósito.

Prática