Métodos Sincronizados

Walter Fetter Lages

w.fetter@ieee.org

Universidade Federal do Rio Grande do Sul

Escola de Engenharia

Departamento de Engenharia Elétrica

ENG04008 Sistemas de Tempo Real

Copyright (c) Walter Fetter Lages – p.1

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Introdução

- O acesso à um método sincronizado apenas é permitido se o lock associado ao objeto é obtido
  - Métodos sincronizados garantem acesso em exclusão mútua aos dados encapsulados, desde que estes dados sejam acessado apenas através destes métodos
- Blocos de código também podem ser sincronizados
  - Pode destruir as vantagens de encapsular as restrições em um único lugar do programa

Copyright (c) Walter Fetter Lages – p.2

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Exemplo - Métodos

```
class SharedInteger
{
    private int data;
    public SharedInteger(int init)
    {
        data=init;
    }
    public synchronized int read()
    {
        return data;
    }
    ...
}
```

Copyright (c) Walter Fetter Lages – p.3

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Exemplo - Bloco

```
public int read()
{
    synchronized(this)
    {
        return data;
    }
}
```

Copyright (c) Walter Fetter Lages – p.4

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Dados Estáticos

- Métodos e blocos sincronizados funcionam com base no lock do objeto
- Dados estáticos são compartilhados entre todos os objetos da classe
  - A exclusão mútua não é garantida entre objetos
  - Em Java, a classe também é um objeto, portanto também possui um lock
    - Pode ser obtido através de um método estático sincronizado ou através de um bloco sincronizado

Copyright (c) Walter Fetter Lages – p.5

Exclusão Mútua entre Objetos

```
class SharedInteger
{
    private static int data;
    .
    .
    .
    public synchronized static void write(int newdata)
    {
        data=newdata;
    }
    public synchronized int read()
    {
        synchronized(this.getClass())
        {
            return data;
        }
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Sincronização Condicional

- `public void wait() throws IllegalMonitorStateException`
  - Bloqueia a thread e libera o lock
- `public void notify() throws IllegalMonitorStateException`
  Desbloqueia um thread, mas não libera o lock
- `public void notifyAll() throws IllegalMonitorStateException`
  Desbloqueia todos os threads, mas não libera o lock

Copyright (c) Walter Fetter Lages – p.7

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Herança e Sincronização

- Existe apenas um lock por classe
- Métodos definidos em uma classe derivada podem fazer sincronização entre sí por motivos diferentes da sincronização existente na classe base
  - Podem existir diferentes sincronizações independentes em uma classe
  - Uma chamada a `notify` pode acordar o método errado => anomalia de herança
    - Sempre encapsular o teste de condição em um loop e utilizar `notifyAll`, mesmo assim pode ocorrer bug

Anomalia de Herança

```
public synchronized void put(int item)
{
    while(prohibted) wait();
    super.put(item);
}
```

- Se o buffer estiver cheio `super.put` ficará bloqueado no `wait`
  - Acesso é proibido => `prohibted=0`
  - Chamadas à `get` e a um método que proíbe o acesso são bloqueadas

Copyright (c) Walter Fetter Lages – p.9

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Anomalia de Herança

- `put`, `get` e método que proíbe o acesso estão bloqueadas
  - É chamado um método que permite o aceso ao buffer
  - Se as threads são liberadas na ordem:
    1. Consumidor
       - Pega o dado e chama notifyAll, que é ignorada
    2. Thread que proíbe o acesso
       - Acesso ao buffer é proibido
    3. Produtor
       - `Put` coloca o dado no buffer apesar do acesso estar proibido

Copyright (c) Walter Fetter Lages – p.10