

Parte III

Lógica de Hoare

Lógica de Hoare

- ◆ Sistema Lógico Formal para raciocinar sobre programas, formulado por Hoare em 1972.
- ◆ Simplificação do método (pioneiro) das asserções devido a Floyd, concebido para diagramas de fluxo, tendo em vista linguagens estruturadas.

$P ::= \text{skip} \mid x := E \mid \text{while } C \text{ do } P \text{ end} \mid P;P \mid$
 $\text{if } C \text{ then } P \text{ else } P \text{ end} \mid$

Lógica de Hoare

- ◆ Asserções da Lógica de Hoare:

$\{ P \} S \{ Q \}$
 $\{ P \}$ Pré-condição
 S Programa
 $\{ Q \}$ Pós-condição

- ◆ Significado: qualquer execução do programa S num estado que satisfaz P , se terminar, termina num estado que satisfaz Q .

Lógica de Hoare

- ◆ Asserções da Lógica de Hoare:

$\{ P \} S \{ Q \}$
 $\{ P \}$ Pré-condição
 S Programa
 $\{ Q \}$ Pós-condição

- ◆ As condições P e Q são expressas em lógica de primeira ordem, podendo mencionar variáveis do programa.

Lógica de Hoare

- ◆ Exemplo de asserção válida

$\{ x = 0 \} x := x + 1 \{ x > 0 \}$
 $\{ x = 0 \}$ Pré-condição
 $x := x + 1$ Programa
 $\{ x > 0 \}$ Pós-condição

- ◆ As condições P e Q são expressas em lógica de primeira ordem, podendo mencionar variáveis do programa.

Lógica de Hoare

- ◆ Exemplo de asserção válida

$\{ x = 0 \ \& \ y = x \} x := x + 1 \{ x > 0 \ \& \ y = x - 1 \}$
 $\{ x = 0 \ \& \ y = x \}$ Pré-condição
 $x := x + 1$ Programa
 $\{ x > 0 \ \& \ y = x - 1 \}$ Pós-condição

- ◆ A pré e pós condição pode mencionar outras variáveis que não surgem no programa.

Regras de Inferência

- ◆ A lógica de Hoare introduz uma regra de inferência para cada construção da linguagem.
- ◆ As regras de inferência são da forma
$$\frac{\{ P_1 \} S_1 \{ Q_1 \} \{ P_2 \} S_2 \{ S_2 \} \dots \{ P_n \} S_n \{ Q_n \}}{\{ P \} S \{ Q \}}$$
onde o número n de premissas pode ser ≥ 0 . Os programas S_i são elementos constituintes de S .
- ◆ Se todas as premissas são válidas então a conclusão é válida (consistência).

© Luis Caires

TALMP 2004/2005

121

Implicação lógica

$$\frac{P \Rightarrow P_1 \quad \{ P_1 \} S \{ Q_1 \} \quad Q_1 \Rightarrow Q}{\{ P \} S \{ Q \}}$$

- ◆ Permite usar todo o poder de raciocínio lógico no meio de uma prova.

© Luis Caires

TALMP 2004/2005

122

Sequência

$$\frac{\{ P \} S_1 \{ R \} \quad \{ R \} S_2 \{ Q \}}{\{ P \} S_1 ; S_2 \{ Q \}}$$

© Luis Caires

TALMP 2004/2005

123

if then else

$$\frac{\{ P \ \& \ C \} S_1 \{ Q \} \quad \{ P \ \& \ \neg C \} S_2 \{ Q \}}{\{ P \} \text{ if } C \text{ then } S_1 \text{ else } S_2 \{ Q \}}$$

© Luis Caires

TALMP 2004/2005

124

skip

$$\frac{}{\{ P \} \text{ skip } \{ P \}}$$

© Luis Caires

TALMP 2004/2005

125

Afectação

$$\frac{}{\{ P \{ x / E \} \} x := E \{ P \}}$$

- ◆ Exemplos:

$\{ y > 0 \ \& \ y > 0 \} x := y \{ x > 0 \ \& \ y > 0 \}$

$(y > 0) \Rightarrow (y > 0 \ \& \ y > 0) = (x > 0 \ \& \ y > 0) \{ x / y \}$

$\{ y > 0 \} x := y \{ x > 0 \ \& \ y > 0 \}$

© Luis Caires

TALMP 2004/2005

126

while

$$\frac{\{ P \& C \} S \{ P \}}{\{ P \} \text{ while } C \text{ do } S \{ P \& \neg C \}}$$

- ◆ A condição P chama-se também "invariante" do ciclo (while) pois é mantida válida ao longo das várias iterações. O corpo S do ciclo while deve preservar o invariante P, tendo em conta que a condição C é verdadeira no início de cada iteração.

© Luis Caires

TALMP 2004/2005

127

Exemplo (if then else)

```
{ } if (a > b) then
  { a > b } ⇒ { a > b & a = max(a,b) }
  m := a { a > b & m = max(a,b) } ⇒ { m = max(a,b) }
else
  { a ≤ b } { a ≤ b & b = max(a,b) }
  m := b { a ≤ b & m = max(a,b) } ⇒ { m = max(a,b) }
end { m = max(a,b) }
```

© Luis Caires

TALMP 2004/2005

128

Exemplo (while)

```
{ } i := 0; s := 0; { i = 0 & s = 0 } ⇒ { s = Σj=0i j }
while ( i <> N ) do
  { s = Σj=0i j & i <> N }
  { s = (Σj=0i+1 j) - (i+1) }
  i := i + 1;
  { s = (Σj=0i j) - i } ⇒ { s + i = (Σj=0i j) }
  s := s + i;
  { s = (Σj=0i j) }
end { s = (Σj=0i j) & i = N } ⇒ { s = (Σj=0N j) }
```

© Luis Caires

TALMP 2004/2005

129

Parte III Monitores (Hoare)

© Luis Caires

TALMP 2004/2005

130

Monitores (Exemplo 1)

```
monitor single_resource:
begin busy:Boolean;
  nonbusy:condition;
  procedure acquire;
    begin if busy then nonbusy.wait;
      busy := true
    end;
  procedure release;
    begin busy := false;
      nonbusy.signal
    end;
  busy := false; comment inital value;
end single_resource;
```

© Luis Caires

TALMP 2004/2005

131

Monitores (Exemplo 2a)

```
monitor bounded_buffer;
begin if count = N then nonfull.wait;
  lastpointer := 0..N - 1;
  count := 0..N;
  nonempty,nonfull:condition;

  procedure append(x:portion);
    begin if count = N then nonfull.wait;
      buffer[lastpointer] := x;
      lastpointer := lastpointer + 1;
      count := count + 1;
      nonempty.signal
    end append;

  count := 0; lastpointer := 0;
end bounded_buffer;
```

© Luis Caires

TALMP 2004/2005

132

Monitores (Exemplo 2b)

```
bounded buffer:monitor
begin buffer:array 0..N - 1 of portion;
  lastpointer: 0..N - 1;
  count: 0..N;
  nonempty,nonfull:condition;

procedure remove(result x: portion);
begin if count = 0 then nonempty.wait;
  x := buffer[lastpointer (-) count];
  count := count - 1;
  nonfull.signal
end remove;

count := 0; lastpointer := 0;
end bounded buffer;
```

© Luis Caires

TALMP 2004/2005

133

Monitores (Exemplo 3)

```
alarmclock:monitor
begin now:integer;
  wakeup:condition;

procedure wakeme(n:integer);
begin alarmsetting:integer;
  alarmsetting := now + n;
  while now < alarmsetting do wakeup.wait(alarmsetting);
  wakeup.signal;
  comment In case the next process is due to wake up at the same time;
end;

procedure tick;
begin now := now + 1;
  wakeup.signal
end;

now := 0;
end alarmclock;
```

© Luis Caires

TALMP 2004/2005

134

Monitores (Exemplo 4a)

```
class readers_and_writers: monitors
begin readercount:integer;
  busy:Boolean;
  OKtoread, OKtowrite:condition;

procedure startread;
begin if busy or OKtowrite.queue then OKtoread.wait;
  readercount := readercount + 1;
  OKtoread.signal;
  comment Once one reader can start, they all can;
end startread;

procedure endread;
begin readercount := readercount - 1;
  if readercount = 0 then OKtowrite.signal
end endread;

readercount := 0;
busy := false;
end readers_and_writers;
```

© Luis Caires

TALMP 2004/2005

135

Monitores (Exemplo 4b)

```
class readers_and_writers: monitors
begin readercount:integer;
  busy:Boolean;
  OKtoread, OKtowrite:condition;

procedure startwrite;
begin
  if readercount > 0 or busy then OKtowrite.wait
  busy := true
end startwrite;

procedure endwrite;
begin busy := false;
  if OKtoread.queue then OKtoread.signal
  else OKtowrite.signal
end endwrite;

readercount := 0;
busy := false;
end readers_and_writers;
```

© Luis Caires

TALMP 2004/2005

136

Análise de Objectos

- ◆ Estamos interessados em analisar o comportamento de objecto, em termos das suas solicitações exteriores,
- ◆ Qualquer chamada de método deve preservar a consistência do objecto, expressa sob a forma de uma condição invariante *Inv*.
- ◆ Assim, para todo o corpo de método *S*, deve ter-se
 $\{ Inv \} S \{ Inv \}$

© Luis Caires

TALMP 2004/2005

137

Análise de Objectos

- ◆ Para cada método deve determinar-se a sua semântica em termos de pré e pós condições. Por exemplo:
 $\{ queue(q) \ \& \ q = v \ @ \ q' \}$
 $x = q.dequeue()$
 $\{ x = v \ \& \ q = q' \ \& \ queue(q') \}$
- ◆ A condição *Inv* deve ser estabelecida pelo estado inicial do objecto (ser pós condição do construtor).
- ◆ Para todo o método *m* definido pelo programa *S*, deve ter-se que
 $\{ Inv \ \& \ PS \} S \{ Inv \}$
onde *PS* é a pré condição do método.

© Luis Caires

TALMP 2004/2005

138

Exemplo (Bounded Buffer)

```
class Buffer {
  Queue queue; int N;
  sync void write(Val v) {
    if (N < Max) { queue.enq(v); N++; }
    else ERR }
  sync Val read() {
    if (N > 0) { v = queue.deq(); N--; }
    else ERR }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

139

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ Max)
  Queue queue; int N;
  sync void write(Val v) {
    if (N < Max) { queue.enq(v); N++; }
    else ERR }
  sync Val read() {
    if (N > 0) { v = queue.deq(); N--; }
    else ERR }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

140

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ Max)
  Queue queue; int N;
  Buffer() {
    { }
    queue = new Queue();
    { |queue| = 0 }
    N = 0
    { |queue| = 0 & N = 0 } ⇒
    { INV }
  }
}
```

© Luis Caires

TALMP 2004/2005

141

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ Max)
  Queue queue; int N;
  sync void write(Val v) {
    { INV & |queue| < Max }
    { INV & N < Max }
    if (N < Max) {
      { INV & (N < Max) }
      queue.enq(v); N++;
      { INV }
    }
    else { INV & False } ERR { INV } }
}
```

© Luis Caires

TALMP 2004/2005

142

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ Max)
  Queue queue; int N;
  sync Val read() {
    { INV & |queue| > 0 }
    { INV & N > 0 }
    if (N > 0) {
      { INV & (N > 0) }
      v = queue.deq(); N--;
      { INV }
    }
    else { INV & False } ERR { INV } }
}
```

© Luis Caires

TALMP 2004/2005

143

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ MAX)
  Queue queue; int N;
  sync void { pre:|queue| < MAX } write(Val v) {
    if (N < Max) { queue.enq(v); N++; }
    else ERR }
  sync Val { pre:|queue| > 0 } read() {
    if (N > 0) { v = queue.deq(); N--; }
    else ERR }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

144

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: | queue | = N & (0 ≤ N ≤ MAX)
  Queue queue; int N; Condition nonfull, nonempty;
  sync void write(Val v) {
    wait(nonfull);
    queue.enq(v); N++;
    signal(nonempty); }
  sync Val read() {
    wait(nonempty);
    v = queue.deq(); N--;
    signal(nonfull); }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

145

Regra de Hoare (wait)

$$\frac{}{\{ \text{Inv} \} \text{wait}(C) \{ \text{Inv} \ \& \ \text{cond}(C) \}}$$

◆ Cond(C) é a propriedade denotada pela (variável de) condição C.

◆ No exemplo em discussão (Bounded Buffer):

- Cond(nonempty) = (N > 0)
- Cond(nonfull) = (N < MAX)

© Luis Caires

TALMP 2004/2005

146

Regra de Hoare (wait)

$$\frac{}{\{ \text{Inv} \} \text{wait}(C) \{ \text{Inv} \ \& \ \text{Cond}(C) \}}$$

- ◆ Inv é a condição invariante do objecto.
- ◆ É importante verificar que Inv é válido em todas as situações em que possa existir uma mudança de contexto de execução (ex: outro thread tome posse do objecto).

© Luis Caires

TALMP 2004/2005

147

Regra de Hoare (signal)

$$\frac{}{\{ \text{Inv} \ \& \ \text{cond}(C) \} \text{signal}(C) \{ \text{Inv} \}}$$

◆ Cond(C) é a propriedade denotada pela (variável de) condição C.

◆ No exemplo em discussão (Bounded Buffer):

- Cond(nonempty) = (N > 0)
- Cond(nonfull) = (N < MAX)

© Luis Caires

TALMP 2004/2005

148

Regra de Hoare (signal)

$$\frac{}{\{ \text{Inv} \ \& \ \text{Cond}(C) \} \text{signal}(C) \{ \text{Inv} \}}$$

- ◆ Inv é a condição invariante do objecto.
- ◆ É importante verificar que Cond(C) é válida antes de provocar o “acordar” potencial de um thread em espera na condição C.
- ◆ Reflecta na consistência das regras wait e signal.

© Luis Caires

TALMP 2004/2005

149

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: | queue | = N & (0 ≤ N ≤ MAX)
  Queue queue; int N; Condition nonfull, nonempty;
  sync void write(Val v) {
    wait(nonfull);
    queue.enq(v); N++;
    signal(nonempty); }
  sync Val read() {
    wait(nonempty);
    v = queue.deq(); N--;
    signal(nonfull); return v; }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

150

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: | queue | = N & (0 ≤ N ≤ MAX)
  Queue queue; int N; Condition nonfull, nonempty;
  sync void write(Val v) {
    { INV }
    wait(nonfull);
    { INV & (N < MAX) }
    queue.enq(v); N++;
    { INV & (N ≤ MAX) & (N > 0) }
    signal(nonempty);
    { INV }
  }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

151

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: |queue| = N & (0 ≤ N ≤ MAX)
  Queue queue; int N; Condition nonfull, nonempty;
  sync Val read() {
    { INV }
    wait(nonempty);
    { INV & (N > 0) }
    v = queue.deq(); N--;
    { INV & (N ≥ 0) & (N < MAX) }
    signal(nonfull);
    { INV }
    return v; }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

152

Exemplo (Bounded Buffer)

```
class Buffer {
  // INVARIANT: | queue | = N & (0 ≤ N ≤ MAX)
  Queue queue; int N; Condition nonfull, nonempty;
  sync void { pre: True } write(Val v) {
    wait(nonfull);
    queue.enq(v); N++;
    signal(nonempty); }
  sync Val { pre: True } read() {
    wait(nonempty);
    v = queue.deq(); N--;
    signal(nonfull); }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

153

Implementação Java

```
class Buffer {
  // INVARIANT: | queue | = N & (0 ≤ N ≤ MAX)
  Queue queue; int N;
  sync void { pre: True } write(Val v) {
    while(N ≥ MAX) wait();
    queue.enq(v); N++;
    notifyAll(); }
  sync Val { pre: True } read() {
    while(N == 0) wait();
    v = queue.deq(); N--;
    notifyAll(); }
  Buffer() { queue = new Queue(); N = 0 }
}
```

© Luis Caires

TALMP 2004/2005

154

N Readers & 1 Writer

```
class NR1W {
  int readercount;
  Bool busy;
  Condition OKtoread, OKtowrite;

  void Startread ()
  {
    if (busy) wait(OKtoread);
    readercount = readercount + 1;
    signal(OKtoread);
  }

  void Endread ()
  {
    readercount = readercount - 1;
    if (readercount == 0) signal(OKtowrite)
  }

  RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

155

N Readers & 1 Writer

```
class NR1W {
  int readercount;
  Bool busy;
  Condition OKtoread, OKtowrite;

  void StartWrite ()
  {
    if ((readercount > 0) || busy) wait(OKtowrite);
    busy = true
  }

  void EndWrite ()
  {
    busy = false;
    signal(OKtoread);
  }

  RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

156

N Readers & 1 Writer

```
class NRLW {
    int readercount;
    Bool busy;
    Condition OKtoread, OKtowrite;
    // { INV = busy => readercount = 0 }
    // { OKtoread = ¬ busy }
    // { OKtowrite = ¬ busy & (readercount = 0) }

    RW() {
        readercount = 0;
        busy = false;
        // { INV }
    }
}
```

© Luis Caires

TALMP 2004/2005

157

N Readers & 1 Writer

```
class NRLW {
    int readercount; Bool busy;
    Condition OKtoread, OKtowrite;
    // { INV = busy => readercount = 0 }
    // { OKtoread = ¬ busy }
    // { OKtowrite = ¬ busy & (readercount = 0) }
    void Startread () { POST: (readercount > 0) }
    { { INV }
        if (busy) { INV & busy } wait(OKtoread);
        { INV & ¬ busy }
        signal(OKtoread);
        { INV }
        readercount = readercount + 1;
        { INV & (readercount > 0) }
    }
}
```

© Luis Caires

TALMP 2004/2005

158

N Readers & 1 Writer

```
class NRLW {
    int readercount; Bool busy;
    Condition OKtoread, OKtowrite;
    // { INV = busy => readercount = 0 }
    // { OKtoread = ¬ busy }
    // { OKtowrite = ¬ busy & (readercount = 0) }
    void {pre: readercount > 0 } Endread ()
    { { INV & (readercount > 0) }
        { INV & (readercount > 0) & ¬busy }
        readercount = readercount - 1;
        { INV & (readercount >= 0) & ¬busy }
        if (readercount == 0)
            { INV & (readercount = 0) & ¬busy }
            signal(OKtowrite) { INV }
    }
}
```

© Luis Caires

TALMP 2004/2005

159

N Readers & 1 Writer

```
class NRLW {
    int readercount; Bool busy;
    Condition OKtoread, OKtowrite;
    // { INV = busy => readercount = 0 }
    // { OKtoread = ¬ busy }
    // { OKtowrite = ¬ busy & (readercount = 0) }
    void StartWrite () { POST: (readercount = 0 & busy) }
    { { INV }
        if ((readercount <> 0) || busy) {
            { INV & (readercount ≠ 0 ∨ busy) }
            wait(OKtowrite); }
        { INV & ¬ busy & (readercount = 0) } }
        { ¬ busy & (readercount = 0) }
        busy = true
        { INV & busy & (readercount = 0) } }
    }
}
```

© Luis Caires

TALMP 2004/2005

160

N Readers & 1 Writer

```
class NRLW {
    int readercount; Bool busy;
    Condition OKtoread, OKtowrite;
    // { INV = busy => readercount = 0 }
    // { OKtoread = ¬ busy }
    // { OKtowrite = ¬ busy & (readercount = 0) }
    void EndWrite ()
    {
        { INV }
        busy = false;
        { INV & ¬ busy }
        signal(OKtoread);
        { INV }
    }
}
```

© Luis Caires

TALMP 2004/2005

161

N Readers & 1 Writer

```
class NRLW {
    int readercount;
    Bool busy;
    Condition OKtoread, OKtowrite;

    void Startread ()
    {
        if (busy) wait(OKtoread);
        readercount = readercount + 1;
        signal(OKtoread);
    }

    void {PRE: readercount > 0 } Endread ()
    {
        readercount = readercount - 1;
        if (readercount == 0) signal(OKtowrite)
    }

    RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

162

N Readers & 1 Writer

```
class NR1W {
    int readercount;
    Bool busy;
    Condition OKtoread, OKtowrite;

    void Startread () {POST: (readercount > 0) & ¬ busy}
    {
        if (busy) wait(OKtoread);
        readercount = readercount + 1;
        signal(OKtoread);
    }

    void {PRE: (readercount > 0) } Endread ()
    {
        readercount = readercount - 1;
        if (readercount == 0) signal(OKtowrite)
    }

    RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

163

N Readers & 1 Writer

```
class NR1W {
    int readercount;
    Bool busy;
    Condition OKtoread, OKtowrite;

    void StartWrite () {POST: busy & (readercount = 0) }
    {
        if ((readercount > 0) || busy) wait(OKtowrite);
        busy = true
    }

    void EndWrite ()
    {
        busy = false;
        signal(OKtoread);
    }

    RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

164

Concurrent HashTable

```
class ConcurrentHashTable {
    int readercount;
    Bool busy;

    void StartRead () {POST: (readercount > 0) & ¬ busy}
    void {PRE: (readercount > 0) } EndRead ()

    void StartWrite () {POST: busy & (readercount = 0) }
    void EndWrite ()

    Val Find(Key k) {
        StartRead(); { (readercount > 0) & ¬ busy } /* retrieve value from HT */;
        EndRead();
    }

    Void Insert(Key k) {
        StartWrite(); { busy & (readercount = 0) } /* insert value from HT */;
        EndWrite();
    }
}
```

© Luis Caires

TALMP 2004/2005

165

Implementação Java

```
class NR1W {
    int readercount;
    Bool busy;

    synchronized void Startread () {PRE: (readercount > 0) & ¬ busy }
    {
        while (busy) wait();
        readercount = readercount + 1;
        notifyAll();
    }

    synchronized void {PRE: (readercount > 0) } Endread ()
    {
        readercount = readercount - 1;
        if (readercount == 0) notifyAll();
    }

    RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

166

Implementação Java

```
class NR1W {
    int readercount;
    Bool busy;

    synchronized void StartWrite () {POST: busy & (readercount = 0) }
    {
        while ((readercount > 0) || busy) wait();
        busy = true
    }

    synchronized void {PRE: busy} EndWrite ()
    {
        busy = false;
        notifyAll();
    }

    RW() { readercount = 0; busy = false; }
}
```

© Luis Caires

TALMP 2004/2005

167

Discussão

- ◆ As várias filas de espera associadas a cada variável-condição na implementação sobre monitores são substituídas por uma única fila na implementação "directa" em Java.
- ◆ O teste centrado nas variáveis-condição é substituído por "busy-waiting" global, e.g.,

```
while ((readercount > 0) || busy) wait();
```
- ◆ Global, pois a cada objecto a linguagem Java (e C#) associa um único conjunto de threads em espera.
- ◆ Solução ☹ : implementar explicitamente os monitores e as variáveis condição em Java, usando semáforos primitivos.

© Luis Caires

TALMP 2004/2005

168

Implementação Java (2)

```
class NR1W {
    int readercount;
    Bool busy;
    Monitor mon;
    Monitor.Condition OKtoread, OKtowrite;
    void Startread ()
    {
        mon.enter();
        if (busy) OKtoread.await();
        readercount = readercount + 1;
        OKtoread.signal();
        mon.leave();
    }
    RW() {
        readercount = 0; busy = false;
        mon = new Monitor();
        OKtoread = new Condition();
        OKtowrite = new Condition();
    }
}
```

© Luis Caires

TALMP 2004/2005

169

Problema 1

- ◆ Defina uma classe Java para implementar semáforos contadores, com a seguinte estrutura:

```
class Semaphore {
    int nwaits;

    acquire() { ... }
    release() { ... }
}
```

- ◆ Mostre, usando a lógica de Hoare, que a sua implementação respeita a condição invariante:

INV: $(0 \leq nwaits)$

© Luis Caires

TALMP 2004/2005

170

Problema 2

- ◆ Defina uma classe Java para implementar certo tipo de contas bancárias, com a seguinte estrutura:

```
class Account {
    int balance;

    debit(int val) { ... }
    withdraw(int val) { ... }
}
```

- ◆ Pretende-se que qualquer pedido de levantamento recebido seja suspenso até que possa ser processados (exista saldo).
- ◆ Assim, poderão existir vários pedidos (levantamento / depósito) em execução concorrente.
- ◆ Mostre, usando a lógica de Hoare, que a sua implementação mantém o invariante INV: $(0 \leq \text{balance})$

© Luis Caires

TALMP 2004/2005

171