Università degli Studi di Padova

# Un modello concreto

SCD

**Anno accademico 2003/4**
**Corso di Sistemi Concorrenti e Distribuiti**

**Tullio Vardanega,** tullio.vardanega@math.unipd.it





## Forma sintattica - 1

- **Tipo (classe) task come modello di processo**
  - **Deve essere dichiarato per poter essere istanziato**
  - **La dichiarazione consiste di specifica (*specification*) ed implementazione (*body*)**
  - **La specifica contiene**
    - **Nome del tipo**
    - **Parte (opzionale) discriminante**
      tramite la quale l'istanziazione può passare parametri (ma solo di tipi di dimensione determinabile) all'oggetto, esattamente come accade con un metodo costruttore
    - **Parte visibile**
      che definisce ciò che di quel processo occorre che gli altri sappiano, per esempio i punti di accesso (*entry*); include il discriminante
    - **Parte privata**
      che definisce aspetti di implementazione, che non riguardano il resto del sistema





## Forma sintattica - 2

- **Esempio di specifica di tipo task**

```
task type Controller;
```
← Parti discriminante, pubblica e privata vuote

```
task type Agent(Param : Integer);
```
← Parte discriminante definita

```
task type Cashier_Attendant(Post : Post_Number := 1);
```
Parte discriminante definita, con valore iniziale predefinito modificabile (*default*)

```
task type Yellow_Pages is
   entry Directory_Enquiry(
      Entity : in Name;
      Place  : in Address;
      Number : out Telephone_Number);
end Yellow_Pages;
```
← Parte pubblica definita con un punto di accesso per sincronizzazione





## Una dichiarazione completa - 1

```
task type Customer(My_Id : Positive) is
   pragma Priority(12);
end Customer;

task body Customer is
   Outcome : Boolean;
begin
   Status.Signal(Outcome);
   if Outcome then
      Service.Wait;
   else
      … -- alternate action
   end if;
end Customer;
```

```
protected Status is
   entry Wait;
   procedure Signal(Outcome: out Boolean);
private
   In_Line : Max_In_Line := 0;
   Present : Boolean := False;
end Status;
```

```
protected Service is
   entry Wait;
   procedure Signal;
private
   Calling_For_Service : Boolean := False;
end Service;
```

```
type Customer_Ref is access Customer;
Customer_1 : Customer_Ref :=
   new Customer(1);
```
❶

```
Customer_2 : Customer(2);
```
❷





## Una dichiarazione completa - 2

- **❶ e ❷ rappresentano distinte modalità dichiarative di oggetti task**
- **Il processo dichiarato tramite ❶ viene attivato solo alla fine dell'elaborazione della parte dichiarativa, all'ingresso della parte esecutiva del modulo dichiarante**
- **❷ invece crea dinamicamente un processo che viene attivato all'ingresso della parte esecutiva del modulo creatore**

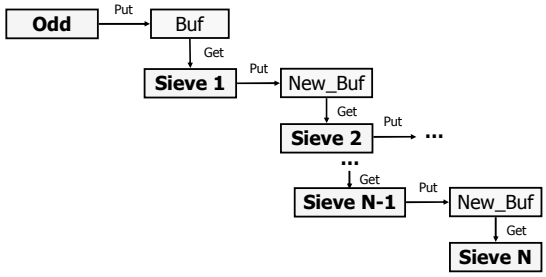




## Un esempio completo - 1

- **Il crivello di Eratostene**
  - **Un processo A seleziona tutti i numeri dispari in un intervallo fissato**
  - **Dato il numero primo 3, che è noto a priori, un altro processo B seleziona tra i numeri dispari rilevati quelli che non sono divisibili per il numero primo noto**
  - **I numeri sono poi passati ad un clone di B che opera la stessa verifica con il numero primo successivo**

## Un esempio completo - 2








## Esercizio 5

- **Replicare la struttura concorrente proposta nel riquadro precedente, utilizzando il linguaggio Java**
- **Raffrontare la soluzione proposta con la soluzione replicata**





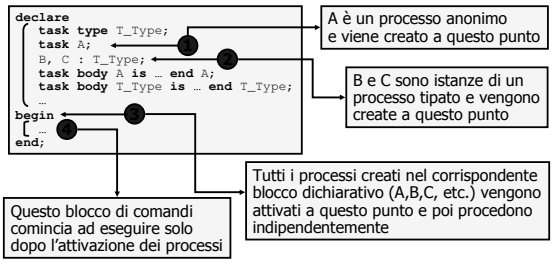
## Stati di vita di processo - 1

- **Un processo viene creato dall'elaborazione del modulo che ne contiene la dichiarazione**
- **L'esecuzione del processo attraversa 3 fasi**
  - **Attivazione : nella quale viene elaborata la parte dichiarativa della sua implementazione**
  - **Esecuzione : nella quale vengono eseguiti i comandi nella parte esecutiva della sua implementazione**
  - **Finalizzazione : nella quale viene eseguita il codice di finalizzazione di ogni oggetto creato dalla sua parte dichiarativa**





## Stati di vita di processo - 2








## Stati di vita di processo - 3






## Stati di vita di processo - 4

- **In un linguaggio a blocchi**
  - **I processi possono essere dichiarati in ciascun blocco**
  - **I blocchi possono essere annidati gerarchicamente**
  - **Un processo è esso stesso un blocco**
- **Ciò consente di realizzare gerarchie di processi**
  - **Il processo padre è soggetto alle stesse regole del blocco contenitore per quanto concerne attesa per l'attivazione dei processi figli**

## Stati di vita di processo - 5

- **L'attesa per la terminazione di un processo è responsabilità della regione dichiarativa nella quale è avvenuta la sua attivazione (*master*)**
  - **Il processo padre non è sempre *master* dei suoi processi figli, ma può esserlo un blocco (*scope*) interno ad esso**
  - **Nel caso di creazione dinamica di processo (come al punto xx di pagina 4) ne è *master* la regione dichiarativa che ne definisce il tipo**
  - **Formalmente, un *master* non ha figli ma dipendenti!**





## Stati di vita di processo - 6

```
task type Dependent;
task body Dependent is … end Dependent;
…
declare
   type Dependent_Ref is access Dependent;
   A : Dependent_Ref;
begin
   …
   declare
      B : Dependent;
      C : Dependent_Ref := new Dependent;
      D : Dependent_Ref := new Dependent;
   begin
      …
      A := C;
   end;
   …
end;
```

1 Quando può terminare 1?

2 Quando può terminare 2?

1 è *master* di tutti i processi creati a partire dal tipo `Dependent_Ref`

Quali sono?

2 è *master* solo del processo `B`





## Stati di vita di processo - 7

- **Nel programma che realizza il crivello di Eratostene (pagina 6, esercizio 4) il modulo *package SoE* definisce vari processi (quali?) senza esserne il *master***
- **In questo caso è il programma principale che ne diventa *master* mediante inclusione (`with`) del modulo creatore**





## Stati di vita di processo - 8