

# Prácticas de laboratorio

## Los cinco filósofos comensales

### (1 sesión)

## Concurrencia y Sistemas Distribuidos

### **Introducción**

---

El objetivo de esta práctica es diseñar e implementar distintas soluciones al problema del interbloqueo (deadlock). Cuando haya concluido sabrá:

- compilar y ejecutar programas concurrentes.
- detectar interbloques en un programa concurrente.
- diseñar soluciones que resuelvan el problema del interbloqueo.
- implementar dichas soluciones.

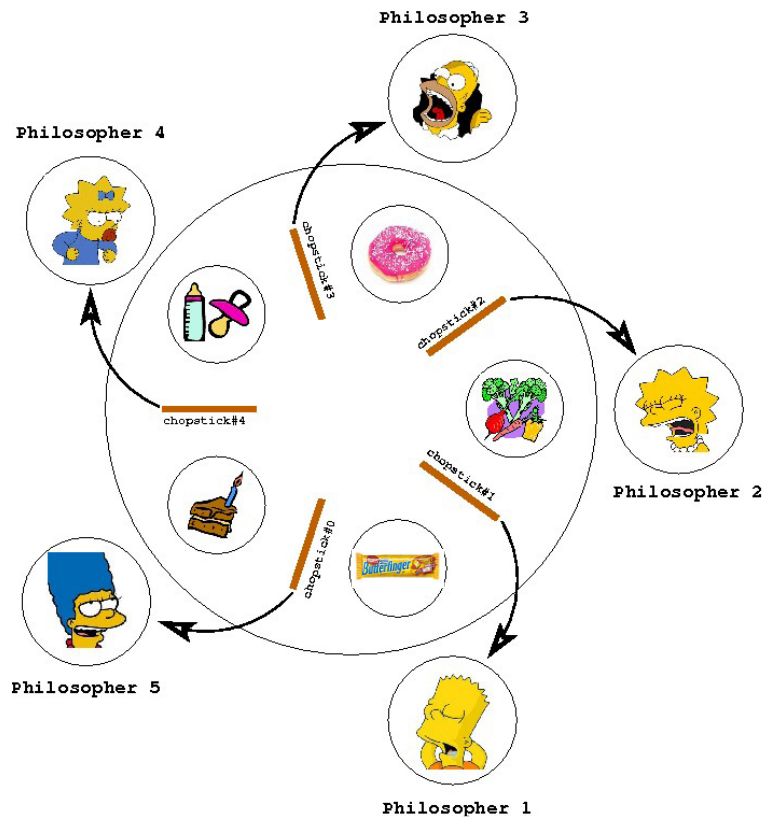
Como lenguaje de programación utilizará Java. La duración estimada de esta práctica es de una semana. En la sesión de laboratorio podrá presentar al profesorado de prácticas sus progresos y resolver las dudas que le surjan. Tenga en cuenta no obstante que necesitará destinar algo de tiempo de su trabajo personal para concluir la práctica.

A lo largo de la práctica verá que hay una serie de ejercicios a realizar. Se recomienda resolverlos y anotar sus resultados para facilitar el estudio posterior del contenido de la práctica.

### **El problema de los cinco filósofos comensales**

---

Para ilustrar el problema de los interbloques utilizamos el conocido ejemplo de los cinco filósofos (se ha trabajado sobre el mismo en el tema de "Interbloques").



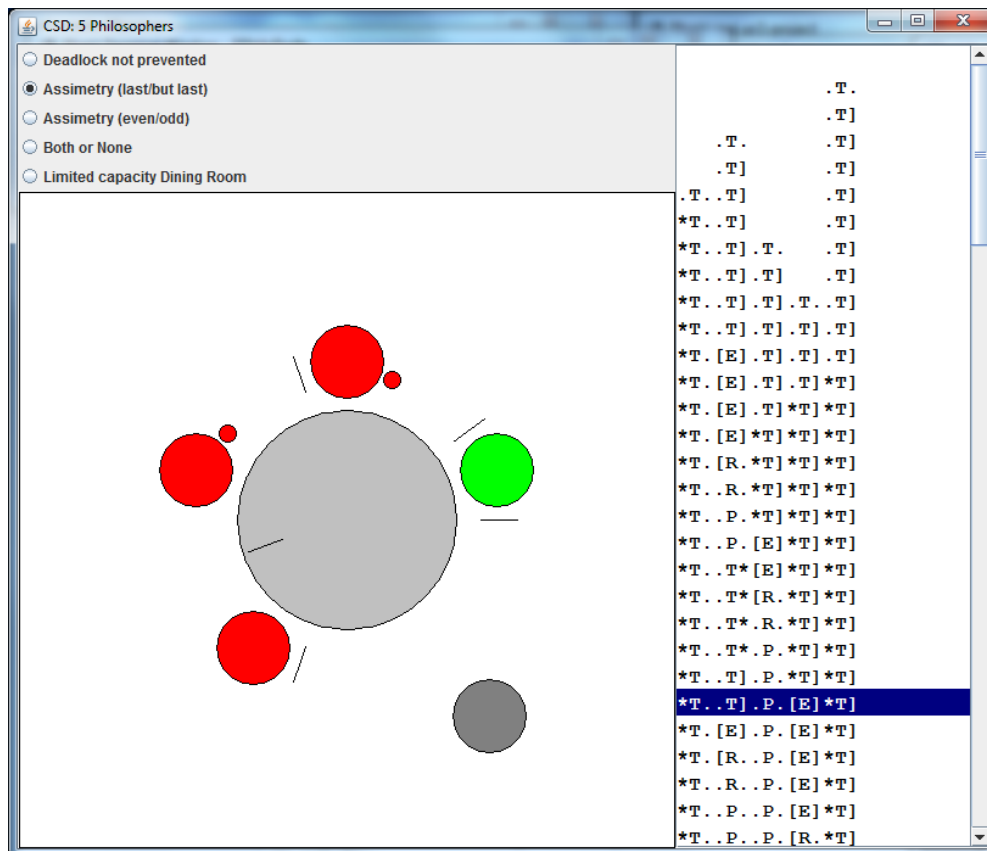
Decimos que puede existir interbloqueo si hay alguna posibilidad de que se produzca, independientemente de que dicha probabilidad sea grande o pequeña. Por ejemplo, si un filósofo toma los dos tenedores en un intervalo de tiempo corto es menos probable llegar a un interbloqueo que si pasa mucho tiempo entre la acción de coger un tenedor y el siguiente. Por esto, el hecho de no observar un interbloqueo no nos asegura que no se pueda llegar a producir.

## Descarga y utilización de la solución parcial al problema

Para realizar esta práctica le proporcionamos una solución parcial al problema de los cinco filósofos. Es parcial porque no garantiza que no se produzca un interbloqueo. Puede descargar el código desde el Lessons de la práctica 2 en el sitio PoliformaT de la asignatura (archivo "PPhilo.jar").

El interfaz y mecanismo de funcionamiento es similar al utilizado en la práctica anterior (Piscina Compartida). En la parte superior izquierda aparecen las distintas versiones seleccionables, de forma que al pulsar sobre una opción se ejecuta la simulación correspondiente. En la parte derecha aparece la secuencia de estados correspondientes a dicha simulación, y la parte inferior izquierda muestra la representación gráfica del estado seleccionado.

En el código proporcionado únicamente está completa la versión 0 (*Deadlock not prevented*). El resto de versiones son inicialmente equivalentes a la básica, y la tarea del alumno es completar el código de las mismas hasta obtener la funcionalidad solicitada.



La siguiente tabla ayuda a interpretar los estados posibles de un filósofo y su representación textual y gráfica. En el caso del texto se utilizan 3 caracteres para indicar el estado de cada filósofo.

Estado	Txt	Gráfico
Inactivo (ya ha terminado)		
Esperando a sentarse a la mesa (entrar)	.*	Círculo rojo alejado de la mesa
Sentado a la mesa (talking)	.T.	Círculo rojo junto a la mesa
Sentado a la mesa, con el tenedor derecho en la mano	.T]	Círculo rojo junto a la mesa, línea en la parte derecha indicando tenedor
Sentado a la mesa, con el tenedor izquierdo en la mano	[T.	Círculo rojo junto a la mesa, línea en la parte izquierda indicando tenedor
Comiendo (eating)	[E]	Círculo verde junto a la mesa, líneas en ambos lados indicando tenedores
Reflexionando (pondering)	.P.	Círculo gris alejado de la mesa
Sentado a la mesa, sin tenedores y esperando el tenedor derecho	.T*	Círculo rojo junto a la mesa, círculo rojo pequeño en la parte derecha indicando que espera ese tenedor
Sentado a la mesa, sin tenedores y esperando el tenedor izquierdo	*T.	Círculo rojo junto a la mesa, círculo rojo pequeño en la parte izquierda indicando que espera ese tenedor
Sentado a la mesa, con el tenedor derecho y esperando el tenedor izquierdo	*T]	Círculo rojo junto a la mesa, línea en la parte derecha y círculo rojo pequeño en la parte izquierda

Sentado a la mesa, con el tenedor izquierdo y esperando el tenedor derecho	[T*	Círculo rojo junto a la mesa, línea en la parte izquierda y círculo rojo pequeño en la parte derecha
Sentado a la mesa, esperando ambos tenedores	*T*	Círculo rojo junto a la mesa, círculos rojos pequeños en ambos lados
Sentado a la mesa, descansando tras comer (resting), pero todavía con el tenedor izquierdo	[R.	Círculo blanco junto a la mesa, línea en la parte izquierda. NOTA.- se debe dejar siempre primero el tenedor derecho, y luego el izquierdo
Sentado a la mesa, descansando (tras comer, y antes de volver a meditar)	.R.	Círculo blanco junto a la mesa

Se asume que el filósofo de la parte superior es el 0, y los siguientes en sentido horario se numeran consecutivamente (1,2,3,4). En los mensajes de texto los filósofos están ordenados del 4 a 0. El estado representado en la imagen utilizada como ejemplo de ejecución indica:

Filósofo	4	3	2	1	0
Estado	*T.	.T]	.P.	[E]	*T]

Que se interpreta como:

- El filósofo 4 está charlando (“talking”, esperando comer): espera el tenedor izquierdo (lo está usando el filósofo 0), y no tiene el tenedor derecho.
- El filósofo 3 está charlando (“talking”, esperando comer), y dispone del tenedor derecho.
- El filósofo 2 está meditando (“pondering”).
- El filósofo 1 está comiendo (“eating”).
- El filósofo 0 está charlando (“talking”, esperando comer): dispone del tenedor derecho, y está esperando el izquierdo (porque lo usa el filósofo 1).

Desde programa debe hacerse la llamada correspondiente a *state* (objeto de la clase *StateManager*) si se desea cambiar el estado del filósofo (ver cómo se utiliza en la clase *RegularTable* del código proporcionado). La siguiente tabla muestra los métodos de *StateManager* que permiten cambiar el estado. Hay que tener en cuenta que algunas acciones son sólo válidas para algunos tipos de mesa.

Método de la clase <i>StateManager</i>	Acción del filósofo
begin	Inicio
end	Fin
wenter	Espera para sentarse (entrar) a la mesa
enter	Se sienta (entra) en la mesa
exit	Se levanta (sale) de la mesa
takeR	Coge tenedor derecho
takeL	Coge tenedor izquierdo
takeLR	Coge ambos tenedores
eat	Empieza a comer
ponder	Empieza a pensar
wtakeR	Espera para coger tenedor derecho
wtakeL	Espera para coger tenedor izquierdo
wtakeLR	Espera para coger ambos tenedores
dropR	Suelta tenedor derecho
dropL	Suelta tenedor izquierdo

## Actividad 0

Esta actividad tiene como objetivo constatar que se puede producir un interbloqueo al ejecutar la solución suministrada y también que estos tipos de error son difíciles de detectar porque en situaciones reales la probabilidad de interbloqueo es muy baja.

En primer lugar, dedique unos minutos a explorar la solución. Para ello, analice el contenido de la clase *Philo* y la clase *RegularTable* (que implementa el interfaz *Table*):

archivo	descripción
Philo	Los objetos de esta clase son hilos que realizan las acciones de los filósofos.
RegularTable	Esta clase implementa un monitor que controla el uso que hacen los filósofos de los tenedores. Resuelve correctamente el uso de los tenedores en exclusión mutua y la suspensión y reactivación de los filósofos en función del estado de cada tenedor. No garantiza en cambio la ausencia de interbloqueos.

A continuación, preste atención a este fragmento de código del archivo *Philo.java*, método *run()*.

```
table.takeR(id); delay(msegDelay); table.takeL(id);
```

Para aumentar la probabilidad de que se produzca el interbloqueo, se ha añadido un retardo desde que el filósofo coge el tenedor derecho hasta que coge el tenedor izquierdo. El valor de dicho retardo se puede indicar como argumento al lanzar la ejecución del programa. Se admiten valores entre 1 y 10 (milisegundos), y el valor por defecto (si no se indica argumento o se indica un valor no válido) es 10.

**Ejercicio 0.1:** Ejecute 10 veces el programa actual (opción *Deadlock not prevented*) utilizando como argumento del programa cada uno de los valores de N de la tabla siguiente, y anote cuántos interbloqueos se producen en cada caso.

N	Número de interbloqueos
1	10
5	10
10	10

NOTA.- Tras la ejecución, en consola aparece la frase DEADLOCK o bien OK. También se puede observar analizando el estado final (si todo ha ido bien los filósofos han desaparecido y todos los tenedores están sobre la mesa, y en caso de interbloqueo todos los filósofos están esperando alrededor de la mesa).

**Ejercicio 0.2:** ¿Cómo influye el valor de N en la probabilidad de interbloqueo?

Aumenta la probabilidad de deadlock

## Actividades

Proponemos cuatro mecanismos para prevenir interbloqueos:

Versión	Descripción
1	Todos los filósofos cogen primero su tenedor derecho, excepto el número 4 (el último), que coge primero su tenedor izquierdo
2	Los filósofos pares cogen primero su tenedor derecho mientras que los filósofos impares cogen primero su tenedor izquierdo
3	Los filósofos, o bien cogen los dos tenedores o, si alguno está ocupado, no cogen ninguno
4	Como máximo pueden haber cuatro filósofos sentados a la mesa

El objetivo de la práctica es desarrollar y probar el funcionamiento de las distintas versiones.

Es importante destacar que deben realizarse las llamadas correspondientes a los métodos públicos de la clase `StateManager`.

La implementación de la clase `StateManager` se encarga de verificar que no se intentan operaciones ilegales (ej.- liberar un tenedor que no se tiene, coger un tenedor utilizado por otro filósofo, etc.). Si se intenta una acción ilegal, el programa indica el error y aborta.

## Versiones 1 y 2: Asimetría

Las versiones 1 y 2 de la tabla anterior se basan en asimetría, para lo que se ha previsto una clase **`LefthandedPhilo`** que corresponde a un filósofo que debe coger los tenedores en orden contrario al de `Philo`, pero mantiene el orden en que se liberan.

**Ejercicio 1.1:** Complete la clase **`LefthandedPhilo`** y realice las modificaciones oportunas en la clase `PPhilo`, para dar solución tanto a la versión 1 como a la versión 2. Tenga en cuenta que para que el código funcione de forma correcta, la clase `LefthandedPhilo` debe extender a `Philo`.

NOTA.- Recuerde que, con independencia del orden en el que se cojan los tenedores, siempre se debe dejar primero el tenedor derecho y luego el izquierdo, ya que esto es una restricción impuesta por la clase `StateManager` (es decir, por la interfaz gráfica).

**Ejercicio 1.2:** Compruebe que no se pueden producir ya interbloqueos, ejecutando varias veces el programa, tanto en la versión *Assimetry (last/but last)* como en la versión *Assimetry (even/odd)*.

### Preguntas:

1) ¿Qué condición o condiciones de Coffman se rompen con la solución propuesta, para las versiones 1 y 2?

Circular wait! Because each philosopher is waiting for a resource his partner is holding

2) ¿Tenemos garantía de que nunca se vayan a producir interbloqueos?      sí!

3) Desde el punto de vista de solucionar los interbloqueos, ¿es importante que los filósofos `LefthandedPhilo` ocupen determinadas posiciones o podrían estar en cualquier posición?

No, en principio da igual, ya que la espera circular lo permite (cualquier lado es bueno para romper el círculo)

### Versión 3: Todo o nada

---

En la versión 3 de los mecanismos propuestos en la tabla anterior, el mecanismo para prevenir el interbloqueo consiste en que “los filósofos, o bien cogen los dos tenedores o, si alguno está ocupado, no cogen ninguno”. En este caso, los filósofos deberán solicitar los tenedores con la operación **takeLR**. Esta operación (a implementar por el alumno) debe permitir al filósofo coger los dos tenedores a la vez, o bien ninguno (si alguno de ellos está ocupado).

**Ejercicio 2:** Realice las modificaciones oportunas para dar solución a la versión 3.

**Preguntas:**

1) ¿Qué condición o condiciones de Coffman se rompen con la solución propuesta para la versión 3? Se rompe la de espera circular

2) ¿Ha utilizado la misma mesa *RegularTable* que en las versiones anteriores o bien ha utilizado una mesa diferente? ¿Por qué? no, ya que se requiere la implementación de takeLR

3) ¿Ha utilizado el filósofo *Philo* de la versión 1? ¿Ha utilizado el filósofo *LefthandedPhilo*? ¿Ha necesitado implementar un nuevo tipo de filósofo? ¿Por qué?

Se ha implementado un nuevo filósofo que utilice la función takeLR

### Versión 4: capacidad mesa

---

Finalmente, la versión 4 propone como mecanismo de prevención de interbloqueos que como máximo pueda haber cuatro filósofos sentados a la mesa. Para ello, se propone hacer uso de los métodos (*enter/exit*) de la mesa, de modo que los filósofos deban solicitar entrar y salir (*enter/exit*) de la mesa en cada iteración.

**Ejercicio 3.1:** Realice las modificaciones oportunas para dar solución a la versión 4.

**Preguntas:**

1) ¿Qué condición o condiciones de Coffman se rompen con la solución propuesta para la versión 4?

2) ¿Ha utilizado la misma mesa *RegularTable* que en las versiones anteriores o bien ha utilizado una mesa diferente? ¿Por qué?

3) ¿Ha utilizado el filósofo *Philo* de la versión 1? ¿Ha necesitado implementar un nuevo tipo de filósofo? ¿Por qué?