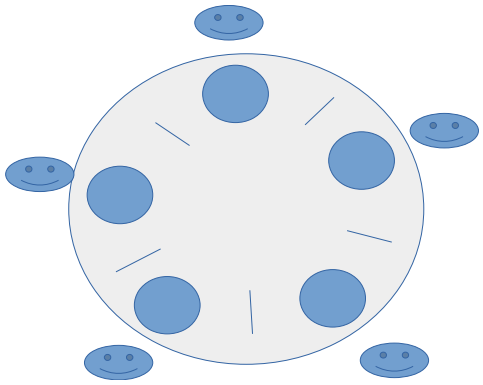
1 El problema de los filósofos

El problema de los filósofos se enuncia de la siguiente forma:

Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando. Cada filósofo tiene un plato de fideos y un palillo a la izquierda de su plato. Para comer los fideos son necesarios dos palillos y cada filósofo sólo puede tomar los que están a su izquierda y derecha. Si cualquier filósofo toma un palillo y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.

El problema consiste en encontrar un algoritmo que permita que los filósofos nunca se mueran de hambre.



Este problema sirve para representar situaciones en las que varios hilos compiten por conseguir varios recursos. Es fácil darse cuenta que se pueden dar situaciones de interbloqueo.

Existen varias soluciones a este problema, como puede ser asignar turnos; o si no pueden tener los dos palillos después de un tiempo, se libera el palillo; limitar el número de filósofos en la mesa de forma que no puedan estar los 5 filósofos sentados a la vez en la mesa; poner un semáforo binario a la mesa;...

Una de las soluciones más sencillas es que los filósofos pares tomen primero el palillo de la derecha y luego el de la izquierda y los filósofos que ocupan puestos impares tomen primero el palillo de la izquierda y luego el de la derecha. En este caso los palillos serán semáforos binarios:

```
import java.util.concurrent.Semaphore;

class Compartido {
	public Semaphore palillo[];

	public Compartido() {
		palillo = new Semaphore[5];
		for(int i = 0; i < palillo.length; i++)
			palillo[i] = new Semaphore(1);
	}
}

class FilosofoPar extends Thread {
	private Compartido m;
	private int mPuesto;

	public FilosofoPar(Compartido compartido, int puesto) {
		m = compartido;
		mPuesto = puesto;
	}
```

```
	public void run() {
		try {
			while(true) {
				// El filosofo piensa
				// ...
				// Quiere comer
				m.palillo[(mPuesto+1) % 5].acquire();
				m.palillo[mPuesto].acquire();

					// El filósofo tiene los dos palillos y puede
comer
					System.out.println("El filósofo " + mPuesto + "
come");

				m.palillo[mPuesto].release();
				m.palillo[(mPuesto+1) % 5].release();
			}
		} catch(InterruptedException err) {
			System.err.println(err);
		}
	}
}


class FilosofoImpar extends Thread {
	private Compartido m;
	private int mPuesto;

	public FilosofoImpar(Compartido compartido, int puesto) {
		m = compartido;
		mPuesto = puesto;
	}

	public void run() {
		try {
			while(true) {
				// El filosofo piensa
				// ...
				// Quiere comer

				m.palillo[mPuesto].acquire();
				m.palillo[(mPuesto+1) % 5].acquire();

				 // El filósofo tiene los dos palillos y puede comer
				 System.out.println("El filósofo " + mPuesto + " come");

				m.palillo[(mPuesto+1) % 5].release();
				m.palillo[mPuesto].release();

			}
		} catch(InterruptedException err) {
			System.err.println(err);
		}
	}
}



class Filosofos {
	public static void main(String args[]) {
		Compartido compartido = new Compartido();
		Thread filosofos[] = new Thread[5];
		for(int i = 0; i < filosofos.length; i++) {
```

```
                  if((i % 2) == 0)
                        filosofos[i] = new FilosofoPar(compartido, i);
                  else
                        filosofos[i] = new FilosofoImpar(compartido, i);

                  filosofos[i].start();
            }
            for(int i = 0; i < filosofos.length; i++) {
                  try {
                        filosofos[i].join();
                  } catch(InterruptedException err) {
                        System.err.println(err);
                  }
            }
      }
}
```