La estructura del programa sigue el siguiente flujo:

El programa se divide en 3 clases, La principal es javaPhil.java. Esta se encarga de crear a los filósofos y añadirlos a un pool de hilos para ser ejecutados y finalmente utiliza el método shutdown con el cual espera a que estos hilos finalicen para terminar la ejecución del programa.

La segunda clase Philosopher.java define un id para cada filosofo, además de un booleano para saber si esta comiendo y otro para verificar la cantidad de veces que ha comido. En este ejemplo del problema de los filósofos se considera que después de haber tenido 3 comidas los filósofos terminan su día. Esto por simplicidad al analizar la ejecución final del programa. Se definen métodos para acceder a estas variables y en particular un método que define su comportamiento run(). Este método primero verifica si aun no ha comido sus 3 comidas y después si ya esta comiendo o si tiene hambre. Si tiene hambre Llama a una clase estática monitor para pedir permiso para comenzar a comer. Si obtiene permiso procede a comer por 500ms después notifica al monitor que va a retornar sus cubiertos y piensa por 500ms antes de tener hambre otra vez. En caso de que no haya obtenido permiso el hilo queda en espera del monitor, quien lo pone en una cola de prioridad.

La tercera clase define a la clase Monitor.java esta se encarga de monitorear el acceso a los cubiertos que son un recurso compartido. Para esto implementa las funciones sincronizadas requestToEat() y returnForks() así como una lista de booleanos que representan los cubiertos y . La primera función ve si el cubierto en la posición del id del filosofo que pide comer esta libre, así como el siguiente cubierto que seria el de su derecha. Si lo esta los cambia a false para mostrar que están ocupados y retorna 0 para darle saber al filosofo que tiene permiso para comer. En caso de que no estén disponibles revisa la lista de espera para comer y si no esta añade al filosofo al final. De esta forma se puede después de que se liberen los cubiertos dar permiso de comer al filosofo que lleva mas tiempo esperando. Una vez añadidos a la lista se pone el hilo en espera.

Finalmente la función returnForks() es llamada por el filosofo después de terminar de comer. Esta comienza por poner en true a los cubiertos que había utilizado y después revisa si este cambio permite comer a alguno de los filósofos en la lista de espera, esta revisión se realiza en el orden del que lleva mas tiempo esperando, de forma que sea lo mas justo posible. Una seleccionado el próximo filosofo que puede comer se elimina al mismo de la lista de espera y se notifica a todos los hilos para continuar su ejecución. Para asegurarnos que continue el correcto se guarda en un dequeue el id del filosofo que obtuvo permiso y cada hilo revisa si son el correcto.

Así todo los filósofos se ordenan mientras comen y se imprimen en consola los diferentes estados de cada uno para poder analizar el proceso por el cual interactúan hasta terminar sus 3 comidas diarias.