Comienzo el diseño planteando que estructura de datos usar para almacenar los programas y trabajos a ejecutar. Usaré un árbol donde cada nodo puede tener cero o más referencias a nodos padres y lo mismo para los nodos hijos. Cada nodo del árbol contendrá un objeto de la clase programa que almacenará un nombre, una duración y un indicador de si se ejecutó el programa. La clase program implementa Runnable, en el método run() incluimos la lógica del programa en concreto.

Siguiendo con la estructura de datos, creamos la clase ConfigReader que idealmente sería una interfaz que tiene varias posibles implementaciones entre ellas leer la configuración de un fichero. Por simplificar, ya que no se pide implementar la lectura de la configuración, ConfigReader tiene un único método getTree() que devuelve un árbol "harcodeado" dentro del propio método (*en caso de querer probar otras distribuciones de trabajos hay que cambiar el código de ese método*).

En este punto del diseño me encuentro con la necesidad de una función "ponte a funcionar" que mediante el algoritmo correcto recorra el árbol ejecutando los programas como esperamos. ThreadRunner es una clase Singleton que se encarga de ejecutar los programas que la clase Mediator detecta como válidos para lanzar. Es decir que el nodo que lo contiene no tenga ningún nodo padre con un programa cuyo método getIsDone() devuelva false.

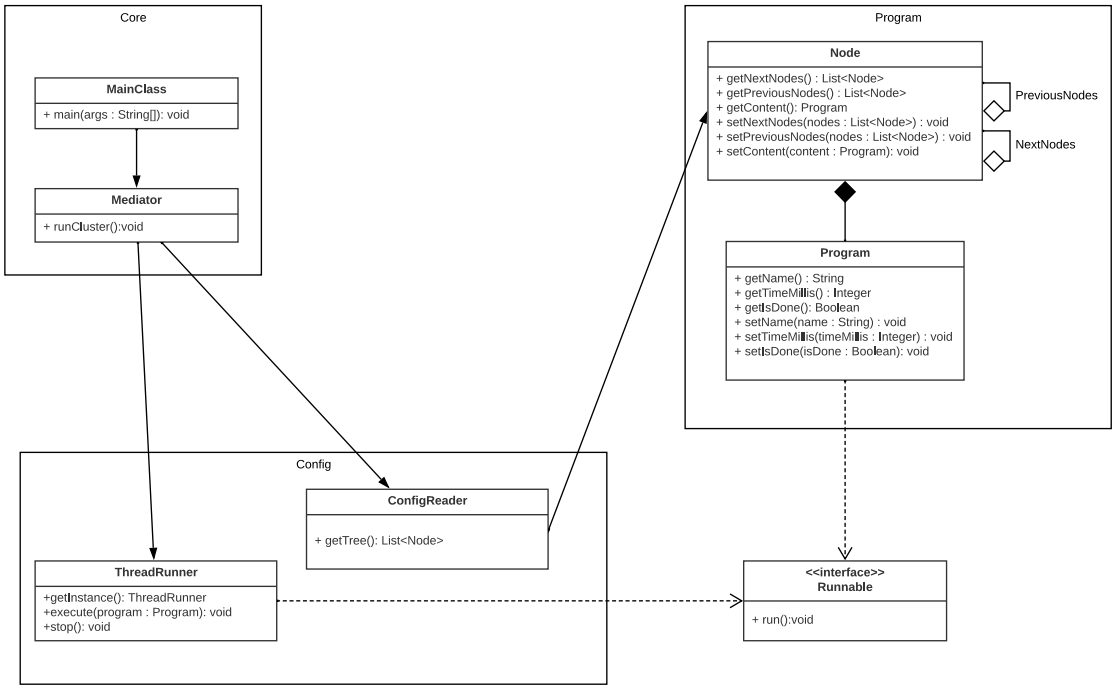
```
public void runCluster() {
    Boolean isProgramExecutable;
    Iterator<Node> iterator;
    do {
        iterator = this.nodesNoIterated.iterator();
        while (iterator.hasNext()) {
            Node aNode = iterator.next();
            isProgramExecutable = true;
            // Miro si el nodo tiene prerequisitos
            if (aNode.getPreviousNodes() != null) {
                // Miro si se cumplen los prerrequisitos
                for (Node aPreviousNodo : aNode.getPreviousNodes()) {
                    if (!aPreviousNodo.getContent().getIsDone()) {
                        // si no se cumplen los prerrequisitos no es ejecutable
                        isProgramExecutable = false;
                    }
                }
            }
            // Si es ejecutable se ejecuta
            if (isProgramExecutable) {
                this.threadRunner.execute(aNode.getContent());
                iterator.remove();
            }
            //  System.out.println(
            //  "\t Programa: " + aNode.getContent().getName() + " Executable:" +
            //  isProgramExecutable);
        }

    } while (!this.nodesNoIterated.isEmpty());
    this.threadRunner.stop();

}
```

Además ThreadRunner va a encargarse de detectar cuando un programa finaliza y usar Programa.setIsDone(true) para modificar la estructura de datos. Para eso se implementa la interfaz Runnable con un bucle que itera los programas que el propio ThreadRunner lanzó verificando su estado con thread.isAlive(). Para finalizar este hilo de ThreadRunner uso el método stop() al que llamará la clase Mediator cuando haya lanzado todos los programas del árbol.[1]

En conclusión tenemos tres clases (Node, ConfigReader y ThreadRunner) con una fuerte dependencia entre ellas. Uso un patrón mediador en la clase Mediator para que las llamadas entre estas clases ocurran en esta clase reduciendo el acoplamiento.



[1] Cuando se añaden los observadores en GOF2.B me doy cuenta de que al cerrar ThreadRunner sin verificar que todos los programas acabasen da pie a que no se notifique la finalización de los últimos programas. En esta versión no pasa porque la salida por consola se implementa en el propio programa. Para solucionarlo while (!this.stop) se convierte en while (!this.stop || this.map.keySet().size() != 0) donde this.map contiene los programas en ejecución.