INGEGNERIA DEL SOFTWARE

Relazione Progetto

UNIVERSITÀ degli STUDI di CATANIA

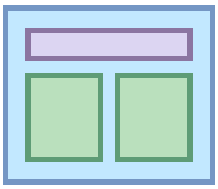
A cura di *Maccarrone Alessia*, studentessa del secondo anno al C.d.L. di Informatica all'Università di Catania.

Sommario

Analisi 3
Richiesta 3
1. 1 Requisiti e Specifiche 3
1. 2 Analisi e modello del dominio 4
Design 5
2. 1 Architettura 5
2. 2 Design Pattern dettagliato 5
Diagramma di stato 6
Sviluppo 6
3. 1 Implementazione 6
3. 2 testing 7
Repository github con corrispondente codice 7
Note di sviluppo 7

Analisi

Richiesta



Il progetto deve riguardare la *ricerca contemporanea di un certo numero di parole* all'interno di **file di testo** presenti in una **cartella**.
Le parole vengono date come **input** insieme, quindi si genera un corrispondente file di testo (**output**), in cui le *parole trovate sono evidenziate* (con doppi asterischi o con un altro marker).
Si potrebbero usare un *Observer*, uno *State* o un *Factory Method*.

1. 1 Requisiti e Specifiche

Il software si dovrà comporre delle seguenti entità:

- in primis il cliente (**client**) il quale si occuperà di prendere una cartella e un numero n di parole in input che passerà e verranno analizzati tutti i file di testo all'interno di un'altra gerarchia;
- un'entità Cartella (**folder**) che si occuperà di salvare al suo interno il numero m di file di testo, questa classe farà da interfaccia per il client nascondendo il processo di ricerca e la sotto-gerarchia;
- un'altra entità dovrà invece occuparsi della ricerca contemporanea delle n parole all'interno tutti i file di testo, e l'entità che ha questo compito è la classe **MyFile**. Questa classe è la classe Contesto del Design Pattern State;
- l'ultima entità dovrà occuparsi di creare un file corrispondente per tutti i file di testo che contengono almeno una parola ed evidenziare le parole trovate nel file, cioè la classe **Find**, ConcrateState che rappresenta lo stato in cui nel file abbiamo trovato una o più parole.

I seguenti punti sulla panoramica generale del software possono essere descritti più dettagliatamente elencando le caratteristiche:

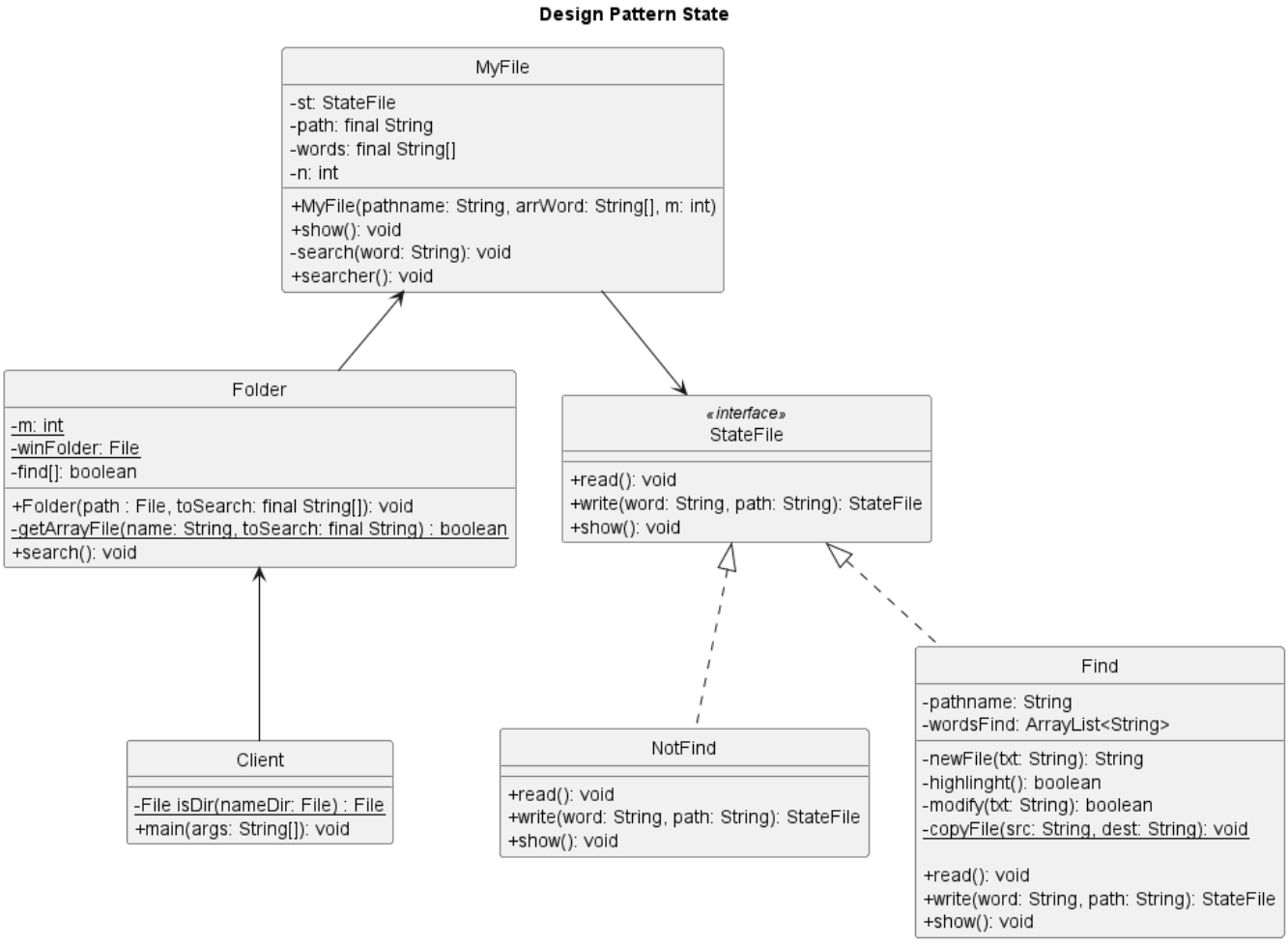
- il client ha bisogno di istanziare solo l'oggetto Folder;
- la responsabilità della classe Folder è quella di istanziare oggetti MyFile, salvarseli e avviare il processo di ricerca attraverso un metodo che il client potrà chiamare, il metodo search();
- il Contesto è rappresentato dall'oggetto MyFile, che analizza un singolo file di testo con l'ausilio delle classi State;
- la classe State è un'interfaccia che è poi implementata dalle due sottoclassi Find e NotFind, classi che rappresentano il comportamento dell'oggetto MyFile che varia in base al suo stato. Cioè deve creare un file con le dovute modifiche se il suo stato è Find oppure non fare nulla se il suo stato è NotFind.

Le classi sono descritte insieme alle loro dipendenze nel diagramma UML della *Figura 1.1*

1. 2 Analisi e modello del dominio

Per quanto riguarda la stesura delle classi come si può costatare dal diagramma UML:

- il Client interagisce solo con la classe Folder, istanziandola
- la classe Folder è puntata dalla classe Client e a sua volta interagisce con la classe MyFile che instanzierà al suo interno con tutti i file che contiene la cartella passata
- la classe MyFile è la classe che rappresenta il sotto-problema, cioè quello che riguarda il singolo file e non ogni singolo File della cartella. Questa classe ha al suo interno un riferimento alla classe StateFile che rappresenta il suo stato e il suo cambiamento di stato al variare di quest'ultimo
- così come da design pattern poi abbiamo un'interfaccia StateFile che è poi implementata dai Concrate State: Find e NotFind.
- La classe Find rappresenta il comportamento del contesto quando nel file troviamo le parole cercate, cioè quello che ne consegue dalla richiesta, l'evidenziare le parole cercate; quindi, è l'unica classe che ha questa responsabilità
- La classe NotFind non muta il suo cambiamento perché nel caso in cui la condizione di aver trovato delle parole nel file non si avvera allora non è richiesto nessun comportamento particolare, si notifica soltanto nel metodo show di non aver trovato nessuna parola cercata.



Design

2. 1 Architettura

Per questo progetto ho deciso di utilizzare il Design Pattern State, un Design Pattern comportamentale orientato sugli oggetti.

Suggeriti insieme al design pattern State, erano il Factory Method e l'Observer.

- Il Factory Method, un design pattern creazione e orientato alle classi, è utile per fornire un'interfaccia per creare oggetti nella superclasse, ma lascia alle sottoclassi la possibilità di cambiare il tipo dell'oggetto creato. È un design pattern interessante e sicuramente applicabile al problema per astrarre la creazione dell'oggetto ma ho preferito un atteggiamento diverso al problema in modo che il problema si incentrasse sul cambiamento di stato.
- L'Observer, un design pattern comportamentale anch'esso orientato agli oggetti, permette di definire una dipendenza uno a molti fra oggetti, così quando un oggetto cambia stato i suoi oggetti dipendenti sono notificati e aggiornati automaticamente. Questo design pattern è sicuramente un alternativa molto valida, soprattutto considerando la sua versione con reactive stream, ma, per questo progetto di piccole dimensioni, ho pensato fosse più opportuno un design pattern State. Pensandoci il file poteva essere il Subject e la cartella l'unico Observer, ma ho preferito separare il comportamento del cambio di stato in dei Concrete State così che solo la classe Find si sporcasse dei metodi che riguardassero la parte implementativa del creare un File, copiarci il File nel quale abbiamo trovato le parole cercate ed evidenziarle.

2. 2 Design Pattern dettagliato

Lo state, come già preannunciato, è un design pattern comportamentale e orientato sugli oggetti e permette di cambiare il comportamento di un oggetto quando il suo stato cambia, come se avesse cambiato classe.

La soluzione prevede:

- La classe Context che salva una referenza a un oggetto state che rappresenta il suo stato corrente, e delega tutti i "comportamenti che dipendono dallo stato" dentro queste classi Concrete State attraverso il suo riferimento;
- State definisce l'interfaccia per incapsulare il comportamento di uno specifico stato del Context
- I Concrete State rappresentano al suo interno il comportamento concreto che cerchiamo quando a run time si avrà lo stato Find o NotFind
- Per cambiare stato in questo caso appena il context vorrà scrivere nel file allora lo stato da NotFind diventerà Find, essendo un operazione che viene richiesta soltanto nel caso in cui abbiamo trovato una parola nel file, e questo lo possiamo dedurre anche dal diagramma di stato.

Diagramma di stato

Sviluppo

3. 1 Implementazione

Per quello che riguarda la stesura del codice, ho cercato di utilizzare la programmazione funzionale quando l'ho trovata utile e questi sono alcuni esempi:

```
String[] words = new String[n]; // array delle parole da cercare
IntStream.rangeClosed(1, n - 1).forEach(i -> words[i - 1] = args[i]);
```

nel Client e nella classe Folder:

```
IntStream.rangeClosed(1, m).forEach(i -> find[i] = false);
```

Nel costruttore

```
IntStream.rangeClosed(0, m - 1)
         .forEach(i -> theFile.add(new MyFile(supp[i], toSearch, n)));
```

Nel metodo getArrayFile

```
IntStream.rangeClosed(0, m - 1).forEach(i -> theFile.get(i).searcher());
IntStream.rangeClosed(0, m - 1).forEach(i -> theFile.get(i).show());
```

Nel metodo search

E anche nella classe MyFile nel metodo show

```
 IntStream.rangeClosed(0, n - 1)
          .forEach(i -> System.out.print(words[i]));
```

Per quanto riguarda le librerie usate sono diverse, ma essendo una manipolazione dei file sicuramente ho trovato utile alcune librerie come:

```
import java.io.File;
import java.util.List;
```

presenti nella classe Fonder

```
import java.util.ArrayList;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

nella classe Find

così come tutte le altre librerie per la gestione delle eccezioni ecc.

3. 2 testing

È stata implementata una classe Test che verificasse il reale comportamento della classe Context, MyFile, passandogli un file con all'interno almeno una parola che dovevamo cercare e un file con tutte parole diverse da quelle che dovevamo cercare.

In particolare, la classe Test è stata implementata così come descritto nel seguente diagramma UML.

Test
-prove: int -words: String[ ] -m: int
+main(String[ ] args):void -testSearcher(): void -newFile(boolean b): String -setNewFile(String path, boolean b): void -setFind(String path): void -setNotFind(string path): void -copyWrite(String path, String[ ] arr): void -copyRead(String path): void

Repository github con corrispondente codice

https://github.com/MaccarroneAlessia/IngegneriaDelSoftware-Progetto

Note di sviluppo

Per l'implementazione di questo software ho provato a seguire il **processo di sviluppo a spirale**: ripetendo le attività ciclicamente in modo da migliorare il software iniziale fino al suo completamento.