Esercizio Composite:

un file system

Anche questa esercitazione ha lo stesso schema della volta scorsa. L'esercizio è suddiviso in diversi passi, da svolgere in sequenza. Inoltre, l'esercizio ha i seguenti obiettivi:

- Implementare il programma gradualmente, una classe alla volta, e passare alla successiva solo dopo aver testato la classe appena implementata. Per questo motivo, NON si deve implementare tutti i punti insieme e testarli solo alla fine.
- Familiarizzare con alcuni strumenti di refactoring.
- NON creare immediatamente astrazioni, ma implementare prima una classe concreta, testarla e poi tramite refactoring creare delle astrazioni (classi astratte e/o interfacce) per un possibile riuso futuro (cioè in vista del prossimo passo dell'esercizio).
- Familiarizzare con gli strumenti dell'IDE per essere produttivi durante l'implementazione, il testing e il refactoring.

Ricordate: abituatevi a rilanciare i test dopo ogni modifica (memorizzate le shortcut di Eclipse). Ricordate che i test andrebbero scritti in una directory sorgente separata, es., *tests*.

In questa esercitazione modelleremo un *file system*, composto quindi da *file* (oggetti foglia) e *directory* (oggetti composti), usando il pattern *Composite*. In particolare, applicheremo il pattern nella **forma type safe**. Quindi la gestione dei *children* sarà presente solo nel componente composto (in questo esempio, la directory). In questo modo, staticamente, non sarà possibile commettere l'errore di aggiungere un figlio in un componente foglia (in questo esempio, il file).

Ovviamente il file system sarà solo "modellato": gli oggetti che creeremo (e le loro classi) non implementeranno effettivamente le funzionalità per scrivere e leggere da un file system vero e proprio. Cioè, non scriveremo, ne' leggeremo, direttamente file o directory.

Per evitare ambiguità con le classi di Java, come *File*, useremo sempre il prefisso *FileSystem* per le classi e tipi che creeremo, es. *FileSystemFile*, *FileSystemDirectory*, ecc. Nelle soluzioni useremo come nome di pacchetto *mp.exercise.filesystem*. Ovviamente potete scegliere un altro nome di pacchetto. Il nome del package verrà esplicitamente mostrato nello svolgimento solo quando è strettamente necessario.

Sempre per semplicità, non modelleremo altre informazioni tipiche in un file system, come data di modifica, dimensione del file, ecc, ma solo il nome.

Questa volta avremo una classe di test separata per ogni classe concreta che implementeremo, quindi *FileSystemFileTest* per *FileSystemFile*, ecc.

1 File

Implementare una classe *FileSystemFile* che rappresenta una *foglia* del pattern *Composite*. La classe ha un campo *name* e un metodo *void ls(FileSystemPrinter*) per "listare" il file usando

un’interfaccia, che deve essere creata, *FileSystemPrinter* che contiene un solo metodo *void print(String)*. Ad esempio, quando si chiama il metodo *ls* su un file col nome “pippo”, la stringa passata al metodo *print* di *FileSystemPrinter* dovrebbe essere “File: pippo”. Per i test ci servirà un’implementazione di *FileSystemPrinter*. Che implementazione di tale interfaccia potremmo creare? E dove? Tenete conto che questa implementazione di *FileSystemPrinter* è al momento fittizia e ci serve solo per testare la classe *FileSystemPrinter*. Cioè, ci serve per esser sicuri che quando si chiama il metodo *ls* su un file, ad esempio, col nome “pippo”, la stringa passata al metodo *print* di *FileSystemPrinter* dovrebbe essere “File: pippo”. Notare che il metodo *ls* è void, quindi la fase di verifica non potrà controllare un valore di ritorno, ma dovrà controllare che si verifichino certi effetti collaterali sull’argomento passato (*FileSystemPrinter*).

2 Estrarre classe base astratta

Vogliamo una classe base astratta sia per le foglie (file) che per gli oggetti composti (directory). La vogliamo creare partendo da *FileSystemFile*. Tale classe base, che chiameremo *FileSystemResource*, contiene il campo comune *name*, e un *getter* al momento *protected* e la dichiarazione (astratta) del metodo *ls* visto prima. Usare le tecniche viste alla scorsa esercitazione per effettuare tale refactoring. I test non devono richiedere modifiche e devono continuare a passare.

3 Directory

Implementiamo adesso il componente *Composite*: la *directory*. La classe *FileSystemDirectory*, eredita da *FileSystemResource*, quindi dovrà avere un costruttore appropriato. Dovrà avere anche un campo per memorizzare i “contenuti”, i “figli”, ad es., *contents*. Al momento non abbiamo richieste particolari sulla struttura della collezione quindi usiamo *Collection*<…>. Che tipo mettiamo al posto di “…”? Ricordate che stiamo implementando il pattern *Composite*. Al momento dobbiamo implementare solo il metodo *ls*: ancora non implementiamo i metodi per gestire i “figli”, basandosi sul pattern *Composite*. Ricordate inoltre che nel pattern il composto deve delegare l’operazione ai figli, eventualmente facendo qualcosa prima e/o dopo. Per distinguere il fatto che stiamo “stampando” una directory, la prima stringa passata al metodo *print* dovrebbe essere “Directory: <nome>”. Poi, come appena detto, l’operazione dovrà essere delegata ai figli. Siccome implementeremo al momento solo *ls*, e non le operazioni sui figli (aggiunta, rimozione, ecc.), come potremmo fare per testare *ls*?

In particolare, dobbiamo testare diverse situazioni:

1. caso in cui la directory è vuota
2. caso in cui contiene almeno un file
3. caso in cui contiene sottodirectory

Come possiamo testare la classe se non possiamo usare le operazioni per la gestione dei figli?

4 Gestione contenuti: add

Implementare nella classe *FileSystemDirectory* il metodo *void add* per aggiungere contenuti nel *Composite*. Quindi che tipo deve avere il parametro? Per testarne la correttezza, NON usiamo *ls* (quindi come possiamo testare *add*?).

5 Remove

Similmente, implementare anche il metodo *void remove*, col parametro appropriato. Nei test per *remove*, NON dobbiamo usare *add*.

6 Ricerca per nome

In questo esempio non ha molto senso implementare l'accesso a un elemento tramite indice (come invece viene fatto a volte col pattern *Composite*). È meglio implementare una ricerca per nome della risorsa:

Optional<FileSystemResource> findByName(String name)

Questo va implementato solo nella classe del *Composite*, quindi in *FileSystemDirectory*. Per il momento, NON implementeremo una ricerca ricorsiva nelle sottodirectory, ma solo sul contenuto della directory su cui invochiamo il metodo (ci occuperemo della ricerca ricorsiva in un'eventuale altra esercitazione). La ricerca non distingue fra file e directory ovviamente. È importante testarla sia quando ci si aspetta di trovare un file sia quando ci si aspetta di non trovarlo.

7 Metodi di Object

Implementare *toString*, *equals* e *hashCode*, secondo le norme viste a "Programmazione" nelle varie classi. Potete usare i meccanismi di Eclipse per farveli generare automaticamente. Attenzione però a come Eclipse vi genera questi metodi: in alcuni casi non tiene conto dell'ereditarietà, quindi dovrete aggiustarli in modo opportuno (dobbiamo implementare *toString*, *equals*, e *hashCode* proprio in tutte le classi o possiamo riusare qualcosa in qualche classe?). Nel caso della directory si può generare anche la stringa dei contenuti in *toString*. Non è strettamente necessario testare questi metodi se li generate automaticamente. Comunque, a livello didattico, conviene scrivere qualche test per questi metodi.

8 File copy

Nella classe *FileSystemFile* implementiamo il metodo *createCopy* con tipo di ritorno *FileSystemResource* che crea un nuovo oggetto *FileSystemFile* con lo stesso nome del file su cui viene invocato. Nei test ci dobbiamo assicurare che l'oggetto restituito

1. NON sia lo stesso oggetto di quello originale e
2. sia "uguale" (tramite *equals*, implementato al passo precedente).

9 Directory copy

Dopo aver portato la dichiarazione astratta di *createCopy* nella superclasse *FileSystemResource*, implementare la copia ricorsiva anche in *FileSystemDirectory*. È poi possibile, nel metodo *createCopy*, avere un tipo di ritorno più specializzato in *FileSystemFile* e *FileSystemDirectory* in modo da sapere staticamente che la copia di un file è un file e la copia di una directory è una directory (invece di una generica risorsa)?