Relazione Progetto Domo Educational

Corso di Programmazione ad oggetti 2014/15

Falzaresi Stefano

28 maggio 2015

Sommario

L’obiettivo di questo progetto è fornire al docente un insieme di package che utilizzati durante lo svolgimento delle lezioni permettano di illustrare le principali peculiarità e metodologie della programmazione ad oggetti, nello specifico applicata al linguaggio Java.

Indice

Capitolo 1

Analisi

1.1 Requisiti

I package generati dovranno permettere al docente di poter formare gli studenti sulle più importanti funzionalità/metodologie della buona programmazione ad oggetti.
L'intero progetto dovrà crescere lezione dopo lezione aumentando in modo lineare le conoscenze dello studente portandolo ad apprendere la programmazione ad oggetti in modo costruttivo permettendogli di vedere i risultati di quanto appreso con il formarsi sempre più di un vero e proprio applicativo per arrivare poi al termine del corso con un applicazione similare a quella sviluppata con il "Progetto Domo" originale.

1.2 Problema

Le difficoltà principali incontrate nello sviluppo di questo progetto sono state quelle legate al mantenimento di un filo conduttore di quanto appreso tra una lezione e l'altra.
È stato inoltre molto difficile riuscire a trasferire quanto sviluppato con il "Progetto Domo" originale dentro dei sottoprogetti che richiedevano come requisito fondamentale l'utilizzo di concetti già acquisiti in quel momento dallo studente.

Capitolo 2

Sviluppo

2.1 Approccio al problema

Per lo sviluppo di questo progetto non si sono utilizzati pattern salvo ove espressamente richiesto (Parte legata al pattern MVC legato all'interfaccia grafica) in quanto le necessità di questo progetto erano di tipo differente.
Per risolvere i problemi indicati precedentemente si è scelto di procedere valutando tutta la documentazione fornita dal docente e procedendo poi con lo sviluppo di nuove classi che fossero comprensibili dagli studenti valutando volta per volta quelle che erano le capacità acquisite nel momento in cui quella data lezione sarebbe stata svolta (si è seguito il programma per l'anno accademico 2014/15).
Un esempio di questo è visibile nelle prime classi in cui non sono stati utilizzati dei modificatori di accesso per le variabili.
Nelle pagine seguenti verranno mostrati i vari sotto progetti sviluppati con una breve spiegazione degli obbiettivi che si intendevano raggiungere ed uno schema UML delle classi inserite.
Per evitare il copia incolla di informazioni già date non verranno mostrate classi presente già nei package precedenti (ma comunque presenti all'interno dei package che le necessitano).
Ogni classe è stata commentata in modo meticoloso ed il maggior completa possibile per poterne comprendere le funzionalità.

2.2 Inheritance

2.2.1 Funzionalità:

Questo package mostra il concetto di ereditarietà tra classi attraverso una interfaccia ed una sua implementazione generale da cui poi si estendono due tipologie di sensori che aggiungono funzionalità alla classe estesa.

2.2.2 Classi e Interfacce presenti:

- Sensor: questa interfaccia definisce i metodi basilari di un sensore generico.
- SensorImpl: questa classe costruisce un oggetto che implementa tutti i metodi descritti nell'interfaccia Sensor.
- ResettableSensor: questa classe estende la classe SensorImpl creando un nuovo oggetto che aggiunge funzionalità alla classe estesa (possibilità di decrementare una variabile) senza pero modificando i parametri del costruttore.
- CamSensor: questa classe estende la classe SensorImpl creando un nuovo oggetto che aggiunge funzionalità alla classe estesa (possibilità di muovere la telecamera) e con un costruttore con maggiori parametri rispetto al costruttore esteso.
- InheritanceTestClass: classe generica in cui vengono instanziati i vari oggetti creati.

2.2.3 UML

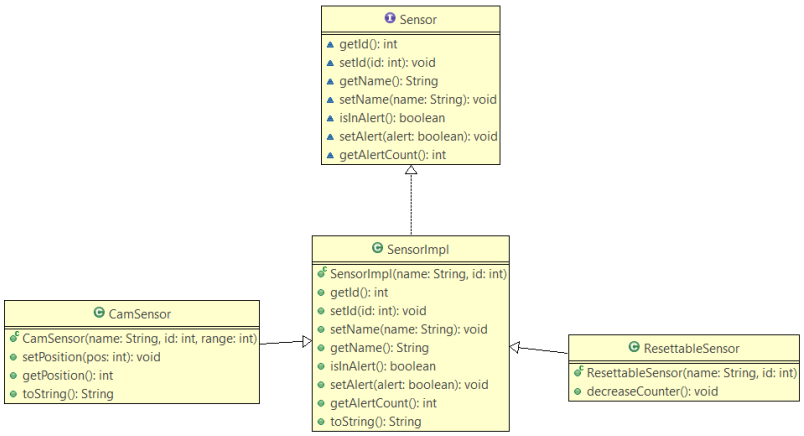

Figura 2.2.3.1 Schema UML del concetto di ereditarietà

2.3 Advanced Mechanisms

2.3.1 Funzionalità:

Questo package mostra alcuni meccanismi avanzati della programmazione ad oggetti.

Gli aspetti affrontati sono:

- Nested Class
- Inner Class
- Enumerator

2.3.2 Classi e Interfacce presenti:

- Sensor: classe simile alla precedente “SensorImpl” ma con l’aggiunta di una variabile “type” di tipo stringa.
- SensorTipology: questo enumerator definisce le principali tipologie di sensori.
- SensorWithEnum: questa classe è simile a Sensor ma utilizza un enum per definire il concetto di “type”.

- SensorWithNested: questa classe descrive un elemento inserendo al suo interno una classe innestata di tipo “nested” che aggiunge delle funzionalità alla classe sensor standard.
- SensorWithInner: questa classe al pari della precedente mostra il funzionamento di una classe innestata in questo caso una “Inner Class”
- Room: questa classe modella un elemento di tipo “stanza” che contiene due liste di sensori, una di tipologia Sensor ed una di tipologia SensorWithEnun.
- EnumeratorTestClass: questa classe di test mostra il diverso approccio nell’utilizzo di parametri di tipo enum, applicandolo all’oggetto “Room” mostra la maggior fruibilità di una variabile di tipo enum a differenza di una variabile stringa.
- AdvancedMechanismsTestClass: questa classe mostra come devono essere istanziati elementi che contengono classi innestate e mostra le differenze di approccio tra inner class e nested class.

2.3.3 UML

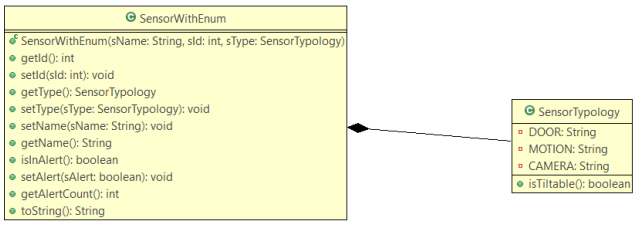

Figura 2.3.3.1 UML rappresentante l'interazione tra classe e relativo enumerator

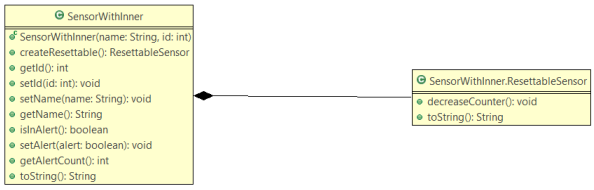

Figura 2.3.3.2 UML raffigurante il concetto di inner class

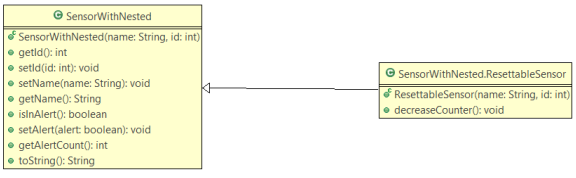

Figura 2.3.3.3 UML raffigurante il concetto di nested class

2.4 Exceptions

2.4.1 Funzionalità:

Questo package esplicita il concetto di eccezioni nella programmazione ad oggetti mostrando inoltre allo studente come è possibile gestire delle eccezioni personalizzate.

2.4.2 Classi e Interfacce presenti:

- CustomExceptions: questa classe che estende la classe standard "Exceptions" mostra come sia possibile creare delle proprie eccezioni che gestiscano il funzionamento del software sviluppato.
- FlatExceptions: questa classe crea una nuova tipologia di eccezioni personalizzate legate all'oggetto Flat.
- Room: questa classe modella un elemento di tipo "stanza" che contiene una lista di sensori e definisce un metodo che lancia delle eccezioni personalizzate nel caso il dato inserito dall'utente risulti scorretto.
- Flat: questa classe modella un oggetto di tipo "appartamento" contenente una lista di stanze e gestendo delle eccezioni in caso di dati non congruenti.
- ExceptionsTestClass: questa classe di test mostra le differenze di approccio tra metodi in cui è esplicitato il lancio di una eccezione (tramite throws) e metodi in cui è presente un try catch, mostra inoltre il funzionamento del costrutto try catch e try catch finally.

2.4.3 UML

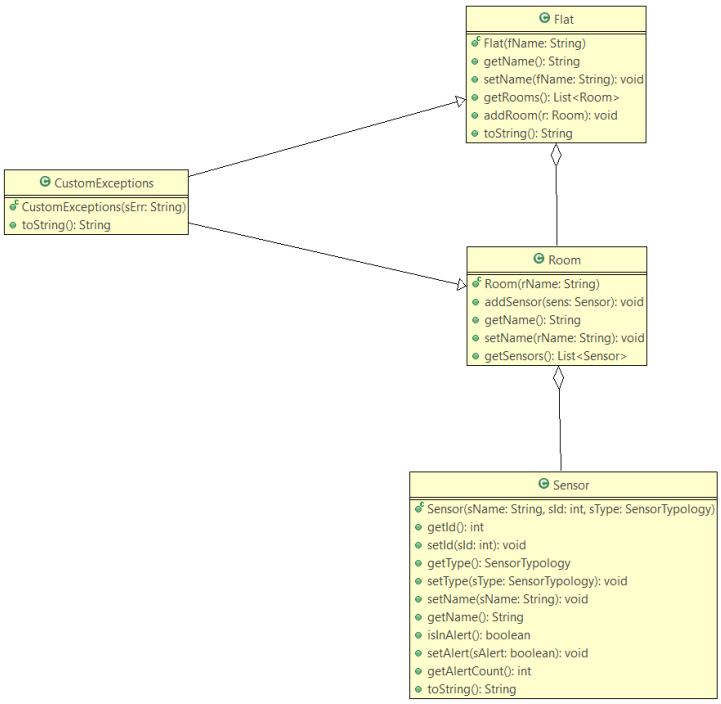

Figura 2.4.3.1 UML raffigurante le interazioni tra le varie classi e le eccezioni

2.5 Input Output

2.5.1 Funzionalità:

Questo package mostra come può essere gestito un file esterno all'applicazione partendo dal salvataggio per arrivare fino alla scrittura. Tutto ciò viene fatto attraverso due metodi che permetto di salvare parte degli oggetti creati (nella fattispecie il nome) per poi ripristinarli.

Viene inoltre mostrato come è possibile rendere l'applicazione più "universale" utilizzando le proprietà del sistema su cui viene eseguita.

2.5.2 Classi e Interfacce presenti:

- Backup: questa classe istanziata con un elemento "Flat" ne effettua il backup del nome dello stesso e delle stanze in esso contenute iterando ogni stanza e salvando poi il contenuto all'interno di un file attraverso un outputstream non bufferizzato.
- Restore: al contrario della classe di backup questa classe effettua il restore degli elementi all'interno del file fornito dall'utente e ricreando la struttura dell'elemento flat.
- InputOutputTestClass: questa classe, gestendo le eccezioni generate dai metodi applicati, crea un elemento di tipo room, gli aggiunge delle stanze e, dopo averne effettuato il backup, ne fa il ripristino.

2.5.3 UML

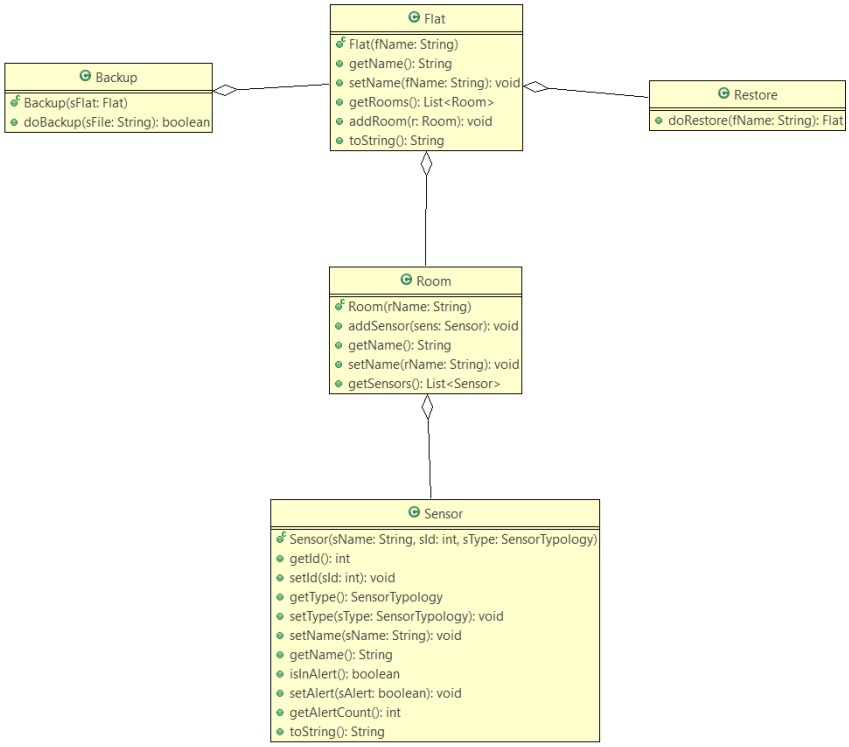

Figura 2.5.3.1 UML rappresentante l'interazione tra i vari oggetti in una struttura di backup e restore

2.6 GUI

2.6.1 Funzionalità:

Questo package vuole indicare i passi fondamentali per creare una semplice interfaccia grafica utilizzando la libreria "javax.swing" utilizzandone i seguenti elementi:

- JFrame
- JPanel
- JButton

2.6.2 Classi e Interfacce presenti:

- EasyGUI: questa classe mostra l'utilizzo base degli elementi forniti dalla libreria swing, mostra come sia possibile gestire gli eventi associati al click su un JButton identificando un action listener per ogni tasto inserito. Da inoltre un'indicazione all'utente su come sia possibile fare in modo che la grafica si adatti alle dimensioni dello schermo su cui viene visualizzata
- ImageView: questo oggetto che tramite ereditarietà estende un elemento JPanel permette di modellare un componente grafico al cui interno possa essere caricata una immagine in modo che la stessa venga poi visivamente ridimensionata in funzione al frame in cui viene inserita.
- GuiTestClass: questa classe semplicemente avvia un oggetto di tipo "EasyGUI".

2.6.3 UML

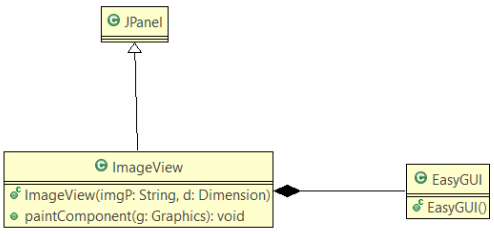

Figura 2.6.3.1 raffigurazione dell'interazione tra l'oggetto Image

2.7 MVC

2.7.1 Funzionalità:

Questo ultimo progetto cerca di unire tutti gli elementi visti nelle parti precedenti e, tramite il pattern MVC, porta l'utente a creare un applicativo che attraverso una interfaccia grafica vede l'interazione tra gli elementi dell'MVC.

In questo esempio il pattern è stato così suddiviso:

2.7.2 Classi e Interfacce presenti:

MODEL

Fanno parte del model tutte quelle classi che creano l'appartamento, e che ne permettono il salvataggio ed il ripristino, gli elementi utilizzati sono stati:

- Flat
- Room
- Sensor
- SensorTypology
- Backup
- Restore

Tutti questi elementi sono stati elaborati nelle precedenti parti del progetto e ne rispecchiano le complete funzionalità.

VIEW

La parte grafica di questo package è stata suddivisa in:

- ViewInterface: una interfaccia che definisce i metodi della view che dovranno poi essere utilizzati dal controller per lo scambio di informazioni tra i due elementi.
- ViewImpl: questa classe costruisce la view dell'applicazione, è simile a SimpleGui visto in precedenza ma in aggiunta vengono gestiti tutti i

metodi per lo scambio di dati con il controller, gli action listener sono stati modificati per implementare le funzionalità del pattern MVC.

CONTROLLER

Il controller in questa implementazione è definito dalla classe “Controller” e, attraverso i metodi descritti nel interfaccia observer che implementa permette l’interazione tra la parte view e i dati del model, in questa implementazione i metodi che sono gestiti riguardano la creazione di un nuovo progetto, il salvataggio ed il caricamento dello stesso (associati ai tre tasti presenti nell’interfaccia grafica).

In questo package si è inoltre scelto di creare un interfaccia observer e la relativa classe astratta che lo implementa, così come dagli standard del pattern MVC.

Questa classe implementa i metodi necessari a View e Controller per scambiare dati tra di loro in modo indiretto (ad esempio per effettuare il salvataggio o il ripristino di un file).

2.7.3 UML

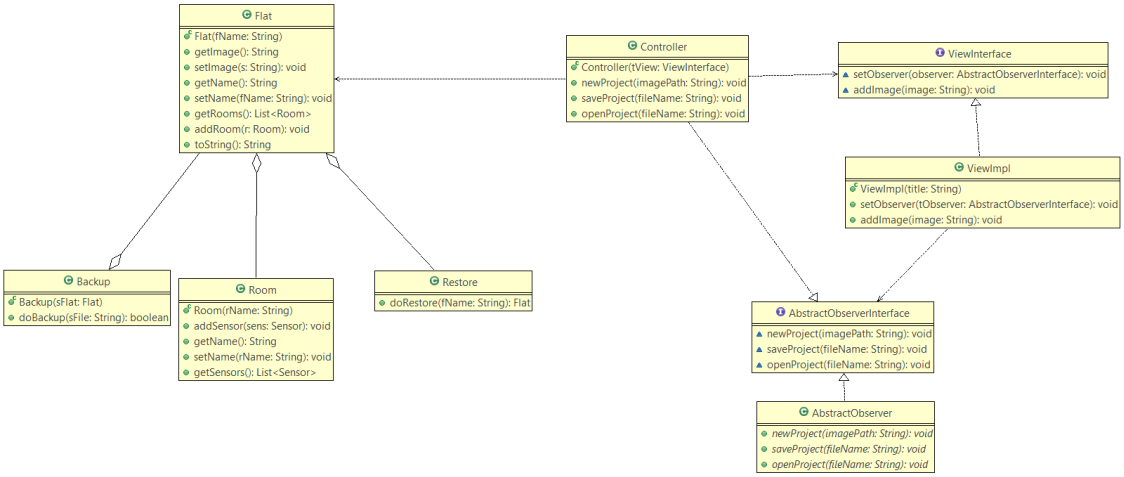

Figura 2.7.3.1 Schema UML raffigurante il concetto di MVC e da cui è possibile evincere l’interconnessione tra i vari elementi

Capitolo 3

Sviluppo

3.1 Testing automatizzato

I seguenti package sono stati testati su una macchina Windows 10 Insider Preview e si intendono per un utilizzo esclusivo tramite IDE Eclipse.

3.2 Divisione dei compiti e metodologia di lavoro

L'intero progetto è stato svolto in totale autonomia ma con il supporto sia del docente Marco Viroli per la scelta degli argomenti da trattare che di Simone De Mattia e Marco Versari per consigli su come affrontare alcune parti del progetto originale "Domo Project"

3.3 Note di sviluppo

Durante la creazione delle classi ho cercato di prestare molta attenzione a quanto fatto in modo da rendere più semplice la comprensione da parte dell'utente e di chiunque avesse utilizzato i package creati, in ogni classe è sono stati inseriti i commenti per poter in futuro generare una Javadoc.

Tra gli obbiettivi c'era la creazione anche di un package riguardante l'utilizzo di elementi "generici" ma purtroppo non è stato possibile crearlo in quanto nel progetto originale venivano utilizzati dei metodi che non permettevano la generalizzazione se non per quanto riguardava l'utilizzo della classe Pair<X, Y> già affrontata a lezione.

Capitolo 4

Commenti finali

4.1 Conclusioni e lavori futuri

Lo sviluppo di questo progetto visto il livello sperimentale che assume in alcune sue parti è risultato alquanto difficile, nonostante ciò è stato per me molto stimolante e di sicuro interesse vista la particolarità dell'obbiettivo che si voleva cercare di raggiungere.
Ritengo quindi interessante la possibilità di poter svolgere ulteriori progetti di questo tipo.