

Relazione del progetto di
“Programmazione ad Oggetti”

Giacomo Amadio, Daniel Pellanda e Davide Zandonella

16 maggio 2022

Indice

1	Analisi	2
1.1	Requisiti	2
1.2	Analisi e modello del dominio	2
2	Design	4
2.1	Architettura	4
2.2	Design dettagliato	5
3	Sviluppo	7
3.1	Testing automatizzato	7
3.2	Metodologia di lavoro	7
3.3	Note di sviluppo	7
4	Commenti finali	9
4.1	Autovalutazione e lavori futuri	9
4.2	Difficoltà incontrate e commenti per i docenti	9
A	Guida utente	11
B	Esercitazioni di laboratorio	12

Capitolo 1

Analisi

1.1 Requisiti

Nell'analisi dei *requisiti* dell'applicazione si dovrà spiegare cosa l'applicazione dovrà fare. Non ci si deve concentrare sui particolari problemi, ma esclusivamente su cosa si desidera che l'applicazione faccia. È consigliato descrivere separatamente i requisiti funzionali (quelli che descrivono l'effettivo comportamento dell'applicazione) da quelli non funzionali (requisiti che non riguardano direttamente aspetti comportamentali, come sicurezza, performance, eccetera).

1.2 Analisi e modello del dominio

In questa sezione si descrive il modello del *dominio applicativo*, descrivendo le *entità* in gioco ed i rapporti fra loro. Si possono sollevare eventuali aspetti particolarmente impegnativi, descrivendo perché lo sono, senza inserire idee circa possibili soluzioni, ovvero sull'organizzazione interna del software. Infatti, la fase di analisi va effettuata **prima** del progetto: né il progetto né il software esistono nel momento in cui si effettua l'analisi. La discussione di aspetti propri del software (ossia, della *soluzione* al problema e non del problema stesso) appartengono alla sfera della progettazione, e vanno discussi successivamente.

È obbligatorio fornire uno schema UML del dominio, che diventerà anche lo scheletro della parte “entity” del modello dell'applicazione, ovvero degli elementi costitutivi del modello (in ottica MVC - Model View Controller): se l'analisi è ben fatta, dovrete ottenere una gerarchia di concetti che rappresentano le entità che compongono il problema da risolvere. Un'analisi ben svolta **prima** di cimentarsi con lo sviluppo rappresenta un notevole aiuto per

le fasi successive: è sufficiente descrivere a parole il dominio, quindi estrarre i sostantivi utilizzati, capire il loro ruolo all'interno del problema, le relazioni che intercorrono fra loro, e reificarli in interfacce.

Capitolo 2

Design

In questo capitolo si spiegano le strategie messe in campo per soddisfare i requisiti identificati nell'analisi.

Si parte da una visione architetturale, il cui scopo è informare il lettore di quale sia il funzionamento dell'applicativo realizzato ad alto livello. In particolare, è necessario descrivere accuratamente in che modo i componenti principali del sistema si coordinano fra loro. A seguire, si dettagliano alcune parti del design, quelle maggiormente rilevanti al fine di chiarificare la logica con cui sono stati affrontati i principali aspetti dell'applicazione.

2.1 Architettura

Questa sezione spiega come le componenti principali del software interagiscono fra loro. In particolare, qui va spiegato **se** e **come** è stato utilizzato il pattern architetturale model-view-controller (e/o alcune sue declinazioni specifiche, come entity-control-boundary).

Se non è stato utilizzato MVC, va spiegata in maniera molto accurata l'architettura scelta, giustificandola in modo appropriato.

Se è stato scelto MVC, vanno identificate con precisione le interfacce e classi che rappresentano i punti d'ingresso per modello, view, e controller. Raccomandiamo di sfruttare la definizione del dominio fatta in fase di analisi per capire quale sia l'entry point del model, e di non realizzare un'unica macro-interfaccia che, spesso, finisce con l'essere il prodromo ad una "God class". Consigliamo anche di separare bene controller e model, facendo attenzione a non includere nel secondo strategie d'uso che appartengono al primo.

In questa sezione vanno descritte, per ciascun componente architetturale che ruoli ricopre (due o tre ruoli al massimo), ed in che modo interagisce (os-

sia, scambia informazioni) con gli altri componenti dell'architettura. Raccomandiamo di porre particolare attenzione al design dell'interazione fra view e controller: se ben progettato, sostituire in blocco la view non dovrebbe causare alcuna modifica nel controller (tantomeno nel model).

2.2 Design dettagliato

In questa sezione si possono approfondire alcuni elementi del design con maggior dettaglio. Mentre ci attendiamo principalmente (o solo) interfacce negli schemi UML delle sezioni precedenti, in questa sezione è necessario scendere in maggior dettaglio presentando la struttura di alcune sottoparti rilevanti dell'applicazione. È molto importante che, descrivendo la soluzione ad un problema, quando possibile si mostri che non si è re-inventata la ruota ma si è applicato un design pattern noto. Che si sia utilizzato (o riconosciuto) o meno un pattern noto, è comunque bene definire qual è il problema che si è affrontato, qual è la soluzione messa in campo, e quali motivazioni l'hanno spinta. È assolutamente inutile, ed è anzi controproducente, descrivere classe-per-classe (o peggio ancora metodo-per-metodo) com'è fatto il vostro software: è un livello di dettaglio proprio della documentazione dell'API (deducibile dalla Javadoc).

È necessario che ciascun membro del gruppo abbia una propria sezione di design dettagliato, di cui sarà il solo responsabile. Ciascun autore dovrà spiegare in modo corretto e giustamente approfondito (non troppo in dettaglio, non superficialmente) il proprio contributo. È importante focalizzarsi sulle scelte che hanno un impatto positivo sul riuso, sull'estensibilità, e sulla chiarezza dell'applicazione. Esattamente come nessun ingegnere meccanico presenta un solo foglio con l'intero progetto di una vettura di Formula 1, ma molteplici fogli di progetto che mostrano a livelli di dettaglio differenti le varie parti della vettura e le modalità di connessione fra le parti, così ci aspettiamo che voi, futuri ingegneri informatici, ci presentiate prima una visione globale del progetto, e via via siate in grado di dettagliare le singole parti, scartando i componenti che non interessano quella in esame. Per continuare il parallelo con la vettura di Formula 1, se nei fogli di progetto che mostrano il design delle sospensioni anteriori appaiono pezzi che appartengono al volante o al turbo, c'è una chiara indicazione di qualche problema di design.

Si divida la sezione in sottosezioni, e per ogni aspetto di design che si vuole approfondire, si presenti:

1. : una breve descrizione in linguaggio naturale del problema che si vuole risolvere, se necessario ci si può aiutare con schemi o immagini;

2. : una descrizione della soluzione proposta, analizzando eventuali alternative che sono state prese in considerazione, e che descriva pro e contro della scelta fatta;
3. : uno schema UML che aiuti a comprendere la soluzione sopra descritta;
4. : se la soluzione è stata realizzata utilizzando uno o più pattern noti, si spieghi come questi sono reificati nel progetto (ad esempio: nel caso di Template Method, qual è il metodo template; nel caso di Strategy, quale interfaccia del progetto rappresenta la strategia, e quali sono le sue implementazioni; nel caso di Decorator, qual è la classe astratta che fa da Decorator e quali sono le sue implementazioni concrete; eccetera);

La presenza di pattern di progettazione *correttamente utilizzati* è valutata molto positivamente. L'uso inappropriato è invece valutato negativamente: a tal proposito, si raccomanda di porre particolare attenzione all'abuso di Singleton, che, se usato in modo inappropriato, è di fatto un anti-pattern.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il testing automatizzato è un requisito di qualunque progetto software che si rispetti, e consente di verificare che non vi siano regressioni nelle funzionalità a fronte di aggiornamenti. Per quanto riguarda questo progetto è considerato sufficiente un test minimale, a patto che sia completamente automatico. Test che richiedono l'intervento da parte dell'utente sono considerati *negativamente* nel computo del punteggio finale.

3.2 Metodologia di lavoro

Ci aspettiamo, leggendo questa sezione, di trovare conferma alla divisione operata nella sezione del design di dettaglio, e di capire come è stato svolto il lavoro di integrazione. **Andrà realizzata una sotto-sezione separata per ciascuno studente** che identifichi le porzioni di progetto sviluppate, separando quelle svolte in autonomia da quelle sviluppate in collaborazione. Diversamente dalla sezione di design, in questa è consentito elencare package/classi, se lo studente ritiene sia il modo più efficace di convogliare l'informazione. Si ricorda che l'impegno deve giustificare circa 40-50 ore di sviluppo (è normale e fisiologico che approssimativamente la metà del tempo sia impiegata in analisi e progettazione).

3.3 Note di sviluppo

Questa sezione, come quella riguardante il design dettagliato va svolta **singolarmente da ogni membro del gruppo**.

Ciascuno dovrà mettere in evidenza eventuali particolarità del suo metodo di sviluppo, ed in particolare:

- **Elencare** (fare un semplice elenco per punti, non un testo!) le feature *avanzate* del linguaggio e dell’ecosistema Java che sono state utilizzate. Le feature di interesse sono:
 - Progettazione con generici, ad esempio costruzione di nuovi tipi generici, e uso di generici bounded. Uso di classi generiche di libreria non è considerato avanzato.
 - Uso di lambda expressions
 - Uso di **Stream**, di **Optional** o di altri costrutti funzionali
 - Uso della reflection
 - Definizione ed uso di nuove annotazioni
 - Uso del Java Platform Module System
 - Uso di parti di libreria non spiegate a lezione (networking, compressione, parsing XML, eccetera...)
 - Uso di librerie di terze parti (incluso JavaFX): Google Guava, Apache Commons...
 - Uso di build systems

Si faccia molta attenzione a non scrivere banalità, elencando qui features di tipo “core”, come le eccezioni, le enumerazioni, o le inner class: nessuna di queste è considerata avanzata.

- Descrivere *molto brevemente* le librerie utilizzate nella propria parte di progetto, se non trattate a lezione (ossia, se librerie di terze parti e/o se componenti del JDK non visti, come le socket). Si ricorda che l’utilizzo di librerie è valutato *positivamente*.
- Sviluppo di algoritmi particolarmente interessanti *non forniti da alcuna libreria* (spesso può convenirvi chiedere sul forum se ci sia una libreria per fare una certa cosa, prima di gettarvi a capofitto per scriverla voi stessi).

In questa sezione, *dopo l’elenco*, è anche bene evidenziare eventuali pezzi di codice “riadattati” (o scopiazzati...) da Internet o da altri progetti, pratica che tolleriamo ma che non raccomandiamo. I pattern di design, invece **non** vanno messi qui. L’uso di pattern di design (come suggerisce il nome) è un aspetto avanzato di design, non di implementazione, e non va in questa sezione.

Capitolo 4

Commenti finali

In quest'ultimo capitolo si tirano le somme del lavoro svolto e si delineano eventuali sviluppi futuri.

Nessuna delle informazioni incluse in questo capitolo verrà utilizzata per formulare la valutazione finale, a meno che non sia assente o manchino delle sezioni obbligatorie. Al fine di evitare pregiudizi involontari, l'intero capitolo verrà letto dai docenti solo dopo aver formulato la valutazione.

4.1 Autovalutazione e lavori futuri

È richiesta una sezione per ciascun membro del gruppo, obbligatoriamente. Ciascuno dovrà autovalutare il proprio lavoro, elencando i punti di forza e di debolezza in quanto prodotto. Si dovrà anche cercare di descrivere *in modo quanto più obiettivo possibile* il proprio ruolo all'interno del gruppo. Si ricorda, a tal proposito, che ciascuno studente è responsabile solo della propria sezione: non è un problema se ci sono opinioni contrastanti, a patto che rispecchino effettivamente l'opinione di chi le scrive. Nel caso in cui si pensasse di portare avanti il progetto, ad esempio perché effettivamente impiegato, o perché sufficientemente ben riuscito da poter esser usato come dimostrazione di esser capaci progettisti, si descriva brevemente verso che direzione portarlo.

4.2 Difficoltà incontrate e commenti per i docenti

Questa sezione, **opzionale**, può essere utilizzata per segnalare ai docenti eventuali problemi o difficoltà incontrate nel corso o nello svolgimento del

progetto, può essere vista come una seconda possibilità di valutare il corso (dopo quella offerta dalle rilevazioni della didattica) avendo anche conoscenza delle modalità e delle difficoltà collegate all'esame, cosa impossibile da fare usando le valutazioni in aula per ovvie ragioni. È possibile che alcuni dei commenti forniti vengano utilizzati per migliorare il corso in futuro: sebbene non andrà a vostro beneficio, potreste fare un favore ai vostri futuri colleghi. Ovviamente *il contenuto della sezione non impatterà il voto finale*.

Appendice A

Guida utente

Capitolo in cui si spiega come utilizzare il software. Nel caso in cui il suo uso sia del tutto banale, tale capitolo può essere omesso. A tal riguardo, si fa presente agli studenti che i docenti non hanno mai utilizzato il software prima, per cui aspetti che sembrano del tutto banali a chi ha sviluppato l'applicazione possono non esserlo per chi la usa per la prima volta. Se, ad esempio, per cominciare una partita con un videogioco è necessario premere la barra spaziatrice, o il tasto “P”, è necessario che gli studenti lo segnalino.

Appendice B

Esercitazioni di laboratorio

In questo capitolo ciascuno studente elenca gli esercizi di laboratorio che ha svolto (se ne ha svolti), elencando i permalink dei post sul forum dove è avvenuta la consegna.