Esame Metodologie di Programmazione

13 Novembre 2019.

Il documento mostra le regole di base per l'esame di *Metodologie di Programmazione*.

ATTENZIONE: per comprendere i contenuti del documento, soprattutto dalla seconda sezione in poi, è necessario che abbiate già seguito buona parte del corso o che abbiate già studiato buona parte del materiale del corso.

1 Regole di base

L'esame consiste di

- Progetto da svolgere a casa (individuale, NON in gruppi), da consegnare prima dell'orale (le varie finestre di consegna saranno fornite via via sul sito Moodle – la consegna del progetto avverrà tramite il sito Moodle);
- Orale (solo se il progetto è stato considerato sufficiente) sugli argomenti del corso.

Maggiori dettagli sul progetto saranno mostrati nelle sezioni successive.

Se il progetto consegnato è sufficiente sarà discusso durante l'orale. Durante l'orale, si deve essere in grado di discutere ogni singolo aspetto e dettaglio di quello che si è implementato nel progetto e di saper commentare in modo appropriato tutte le scelte effettuate nell'implementazione del progetto. Inoltre, durante l'orale, ci si aspetta che gli studenti siano in grado di rispondere alle domande sugli argomenti del corso in modo formale e con la dovuta proprietà di linguaggio tecnico.

Il voto finale, che sarà assegnato solo se entrambe le parti sono sufficienti, consisterà della media dei risultati delle due parti.

Se il progetto è sufficiente, potete sostenere l'orale anche in un appello successivo a quello della consegna del progetto, anche se si consiglia di svolgere entrambe le parti (progetto e orale) nello stesso appello (infatti, per svolgere il progetto si devono già conoscere tutti gli argomenti del corso che saranno chiesti all'orale, quindi per svolgere correttamente il progetto dovrete comunque studiare gli argomenti dell'orale).

Se il progetto è sufficiente, potete sostenere l'esame orale in un certo appello solo dopo esservi iscritti sul sito apposito dell'ateneo. Ovviamente, i risultati dei progetti saranno resi disponibili su Moodle in tempo utile per iscriversi eventualmente all'appello per sostenere l'esame orale.

Un progetto sufficiente vale quindi per tutto l'anno accademico. Tuttavia, il progetto viene annullato se:

- Lo studente non supera l'esame orale (cioè non prende la sufficienza all'orale) oppure
- Lo studente si prenota per l'esame orale ma non si presenta senza darne preavviso oppure
- Lo studente rifiuta il voto finale

Se il progetto viene annullato, lo studente dovrà farne uno completamente nuovo.

Se il progetto consegnato è sufficiente sarà discusso SOLO durante l'orale; non potete venire a ricevimento per conoscere le motivazioni del voto di un progetto sufficiente. Se non ritenete il voto del progetto soddisfacente, potete annullare il progetto (inviando un'email al docente e assicurandovi di ricevere la conferma da parte del docente) e in tal caso venire a ricevimento per conoscere le motivazioni del voto (vale quanto detto sopra riguardo l'annullamento del progetto).

Gli studenti il cui progetto è stato valutato insufficiente sono vivamente invitati a venire a ricevimento per conoscere le motivazioni che hanno portato all'insufficienza.

2 Il progetto

Il progetto consiste in un sistema software a vostra scelta che dovete implementare in Java, utilizzando i vari principi mostrati nel corso (inclusi leggibilità e pulizia di codice) e applicando qualche pattern. Il progetto NON deve essere incentrato tanto sulla parte computazionale quanto sulla corretta (secondo quanto visto nel corso) strutturazione in interfacce e classi e sulla collaborazione fra tali interfacce e classi e sulla collaborazione tra oggetti. Il progetto deve essere corredato da dei test JUnit (eventualmente usando librerie aggiuntive come AssertJ – librerie aggiuntive devono essere parte del progetto). Tali test devono dare garanzie sulla correttezza dell'implementazione nel modo più esaustivo possibile e devono essere anche loro scritti bene, leggibili e facilmente comprensibili. Il progetto non deve essere necessariamente un programma utilizzabile in pratica: deve essere un prototipo che mostra le vostre capacità di design e implementazione usando l'OOP e i principi e pattern visti a lezione. Quindi deve essere più una modellazione di un sistema software che un software realmente utilizzabile in pratica. Ovviamente, anche tale modellazione deve essere corretta e concreta, e testabile tramite test JUnit. Una classe col metodo *main* NON è necessaria.

Siccome il progetto deve solo modellare un prototipo di sistema software, potete avere parti non concretamente implementate, se non sono rilevanti ai fini degli argomenti del corso. Tali parti saranno accessibili tramite un'astrazione. Per esempio, ricordate l'interfaccia *FileSystemPrinter* che abbiamo usato alle esercitazioni che modellavano un file system: la usavamo per "delegare" le funzionalità di *print*, astraendo dall'implementazione effettiva della stampa (su schermo, su stampante, su file, ecc.). Non abbiamo mai implementato tale interfaccia con una classe concreta. Abbiamo creato un'implementazione fittizia ("mock") nella directory dei test che fosse sufficiente a permetterci di testare le funzionalità delle altre classi concrete del sistema che abbiamo implementato. In quel particolare contesto, non ci interessava avere un'implementazione effettiva di *FileSystemPrinter*.

Per la scelta del software da implementare potete ispirarvi a qualche esempio visto a lezione e a esercitazione, ma non potete riusare lo stesso codice già mostrato a lezione e a esercitazione.

Altri possibili esempi di sistemi software da implementare per il progetto potrebbero essere (la lista che segue mostra solo alcuni possibili esempi e non è assolutamente esaustiva): Shop online, Noleggio (libri e video), Agenzia viaggi, Gestione libro paga, Compagnia assicurazioni, Ristorante, ecc. Un esempio leggermente più dettagliato sarà mostrato alla fine di questo documento.

Dovreste quindi trovare un contesto per applicare i principi visti nel corso e applicare qualche pattern. ATTENZIONE: non ha assolutamente senso applicare TUTTI i pattern visti a lezione. I pattern devono essere applicati solo nel contesto giusto e quando e se servono.

Eventualmente, potete anche utilizzare altri pattern non visti al corso (ma presenti in letteratura); in tal caso, dovreste descrivere brevemente tali pattern nella relazione (vedere più avanti per quanto riguarda la relazione).

Si ricorda comunque che ai fini dell'esame si fa riferimento al materiale mostrato a lezione e disponibile sul sito Moodle, ed eventualmente a libri o articoli consigliati nel corso. Fare riferimento ad altro materiale, soprattutto trovato online, può essere rischioso, in quanto non necessariamente affidabile. Il materiale online non sarà automaticamente ritenuto autorevole e accreditato. La decisione finale a tal proposito spetterà solo al docente.

Se applicate un pattern basato su astrazione e classe concreta, cercate di avere almeno due o tre classi concrete. Per esempio, se usate il pattern *Decorator*, implementate almeno due o tre decorator concreti. Stessa cosa se usate ad esempio il pattern *Strategy*, *Visitor*, ecc.

Il progetto deve essere un progetto Eclipse, consegnato zippato, senza i file *.class*, ma con eventuali librerie aggiuntive (ad es., AssertJ). Tale zip contenente il progetto Eclipse deve essere importabile dal docente in Eclipse tramite "Import" → "Existing Projects into Workspace" → "Select archive file" senza ulteriori operazioni e aggiustamenti da parte del docente.

Il progetto, una volta importato, deve compilare senza errori e senza warning. In particolare,

- Se ci sono errori di compilazione il progetto sarà automaticamente insufficiente
- Se non ci sono test JUnit il progetto sarà automaticamente insufficiente
- Se ci sono test JUnit che falliscono il progetto sarà automaticamente insufficiente
- Se ci sono dei warning il voto del progetto subirà delle penalizzazioni

Il progetto si intende implementato in Java 8; se volete usare Java 11, assicuratevi che il progetto sia configurato in modo appropriato. Non usate altre versioni di Java che non siano la 8 o la 11.

Il nome del progetto Eclipse deve essere della forma "progetto.mp.*nome*.*cognome*" dove dovete sostituire *nome* e *cognome* col vostro nome e cognome (in caso di doppi nomi, usatene solo uno). Il nome del file zip deve essere "progetto.mp.*nome*.*cognome*.zip". I nomi dei package Java sono a vostra libera scelta. Progetti consegnati che non rispettano tali specifiche non saranno presi in considerazione.

All'interno del progetto Eclipse, nella cartella principale, deve essere inclusa anche una relazione (indicativamente di circa 10 pagine), in formato PDF. La relazione deve includere

- una descrizione delle funzionalità del sistema implementato
- diagrammi UML delle classi e interfacce e, se necessario o opportuno, eventuali diagrammi UML di interazione fra gli oggetti (vedere la prossima sezione per quanto riguarda l'UML)
- una descrizione sufficientemente dettagliata delle scelte di design e implementative e in particolare le motivazioni per i pattern applicati e le strategie per testare.

I progetti e le implementazioni non devono essere simili a quelli di altri studenti, quindi si consiglia di NON lavorare al progetto insieme ad altri colleghi e di NON confrontarvi tra di voi riguardo al progetto. Progetti ritenuti troppo simili saranno annullati.

NOTA. Lo studente deve lavorare in modo autonomo al progetto, senza confrontarsi continuamente con il docente sulle scelte di progettazione e implementazione, che sono oggetto della valutazione.

I progetti saranno valutati solo dopo che la consegna dei suddetti sarà stata chiusa su Moodle. Quindi, finché la finestra di consegna è aperta, potete modificare la vostra consegna. Tenete però conto che sarà valutata solo l'ultima versione consegnata.

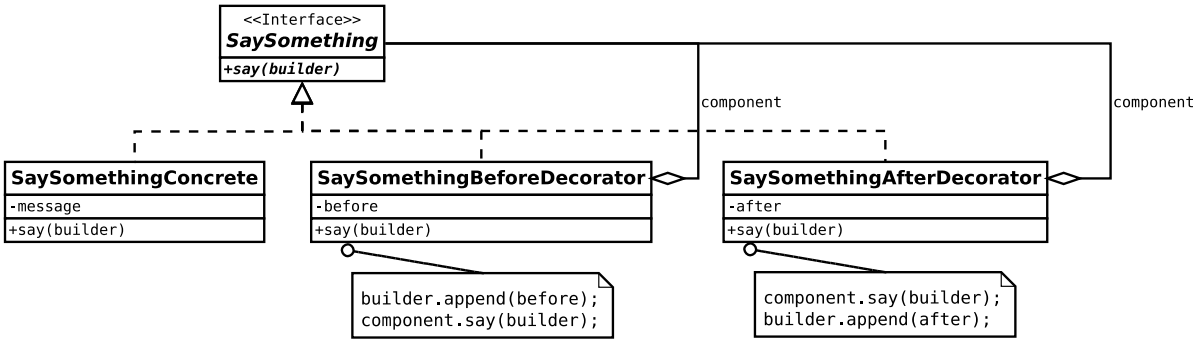
3 Software UML

Per creare i diagrammi UML nella vostra relazione si consiglia l'uso di software appropriato. Siete liberi di usare il software che più preferite e siete invitati a valutare diversi software fino a che non trovate quello che fa per voi. Di seguito sono riportati alcuni software che ho personalmente usato (tutti gratuiti e open-source), che permettono di esportare in tantissimi formati:

- *Dia*, software per diagrammi vari, che include anche una sezione apposita per UML. Leggermente spartano ma vi lascia molta libertà ed è relativamente semplice da utilizzare. https://wiki.gnome.org/Apps/Dia
- *ArgoUML*, specifico per UML; un po' "vetusto" ma ancora funzionante. http://argouml.tigris.org/
- *Eclipse Papyrus*, plugin Eclipse per diverse tipologie di diagrammi, tra cui UML; il più potente e completo dei 3, ma anche il più complesso da usare. https://www.eclipse.org/papyrus/

Tenete conto che per i diagrammi che dovrete disegnare voi, non avrete bisogno di tutte le funzionalità complete di UML, e senz'altro non avrete bisogno delle caratteristiche delle versioni più recenti di UML.

Per esempio, il diagramma mostrato di seguito (visto a lezione) è stato fatto con Dia e poi esportato in formato SVG (nel documento è stato poi incluso il file in formato SVG):



4 Possibili esempi di progetto

Quello che segue serve solo a darvi qualche spunto o idea sulla tipologia di progetto da realizzare per l'esame. Negli esempi dovrebbero risultarvi evidenti alcuni possibili pattern da utilizzare per gestire una particolare funzionalità. Il sistema da modellare descritto di seguito potrebbe anche

essere implementato solo concentrandosi su alcune parti che possono eventualmente essere ulteriormente estese.

Shop online

Un negozio online offre diversi prodotti (catalogo): il cliente può selezionare alcuni articoli, controllare il carrello degli articoli scelti, chiedere una fattura, effettuare il pagamento. La fattura descrive i prodotti acquistati, con il loro prezzo, e il totale.

Ci sono un certo numero di prodotti di base ma anche pacchetti di prodotti (che, a loro volta, possono essere formati sia da prodotti semplici che da pacchetti).

Il negozio può applicare dinamicamente particolari sconti (uno o più sconti) su alcuni articoli. Eventualmente, se viene applicato un certo tipo di sconto non si dovrebbero applicare ulteriori sconti.

In alternativa, si può modellare una possibile strategia di determinazione del prezzo di una vendita, che può essere modificata periodicamente (ad es, sconto giornaliero del 10 % oppure sconto per anziani, ecc) oppure dipendere da una Tessera-Cliente.

Nel caso un prodotto non sia al momento disponibile i clienti possono chiedere di essere notificati appena il prodotto torni disponibile.

Vengono effettuati periodicamente dei report, per controllare il prezzo più alto o più basso degli articoli in catalogo, quanti sono gli oggetti con il prezzo più alto, ecc.

Si può considerare più in dettaglio il problema della visualizzazione della fattura. Ad esempio, la fattura viene aggiornata man mano che si aggiorna il carrello. La fattura può essere costituita da diverse parti (intestazione, footer, ecc.) eventualmente personalizzabili.

Si può considerare in dettaglio il problema del pagamento, prevedendo diverse opzioni, Carta di credito, Bonifico, ecc.

In questo sistema dovrebbero venirvi già in mente alcuni pattern da utilizzare.