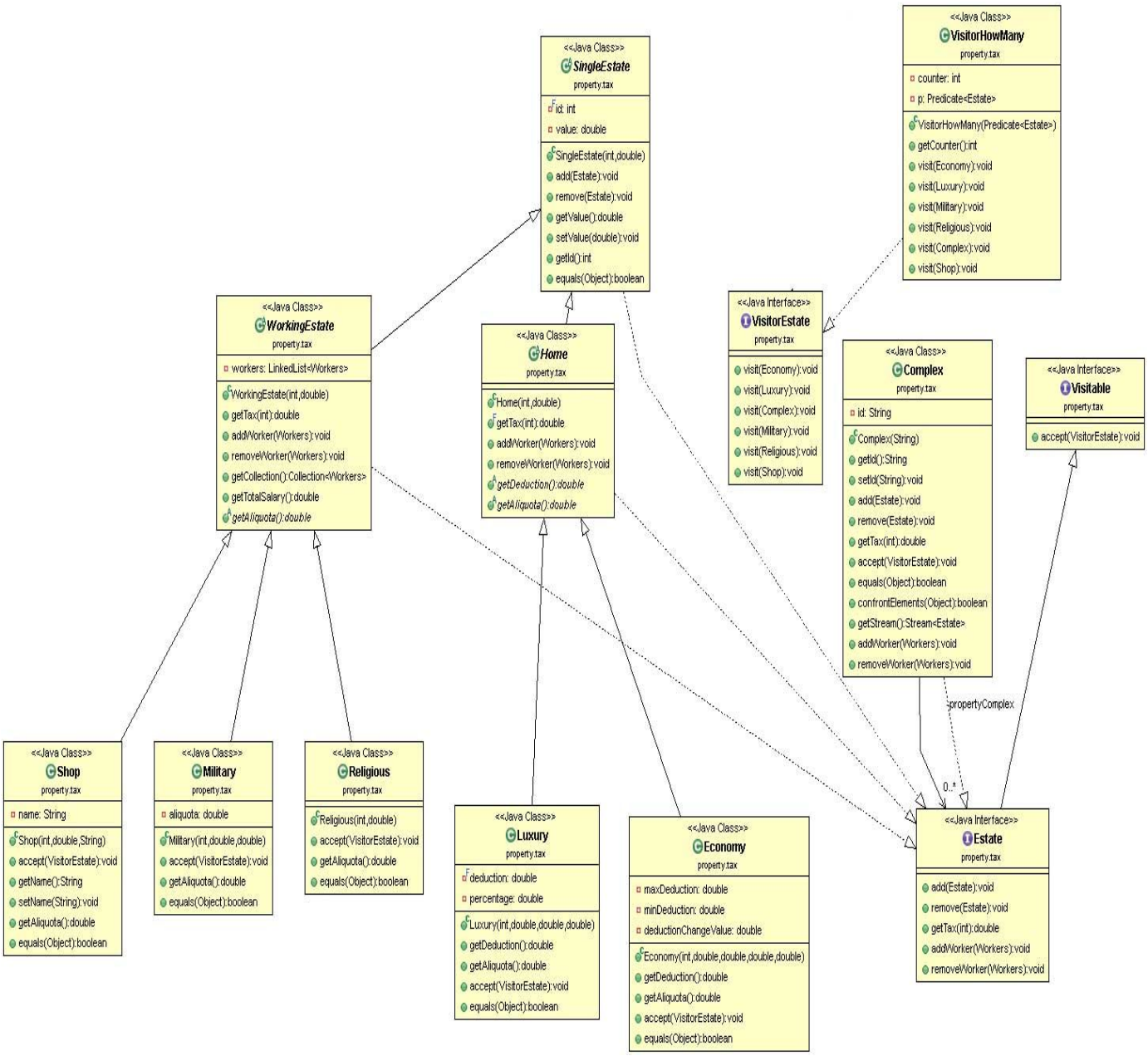
RELAZIONE DI METODOLOGIE DI PROGRAMMAZIONE

Studenti:
Gabriele Boddi: gabriele.boddi@stud.unifi.it, 6358510
Massimiliano Sirgiovanni: massimiliano.sirgiovanni@stud.unifi.it, 6357449
Anno iscrizione studenti: anno accademico 2017-2018

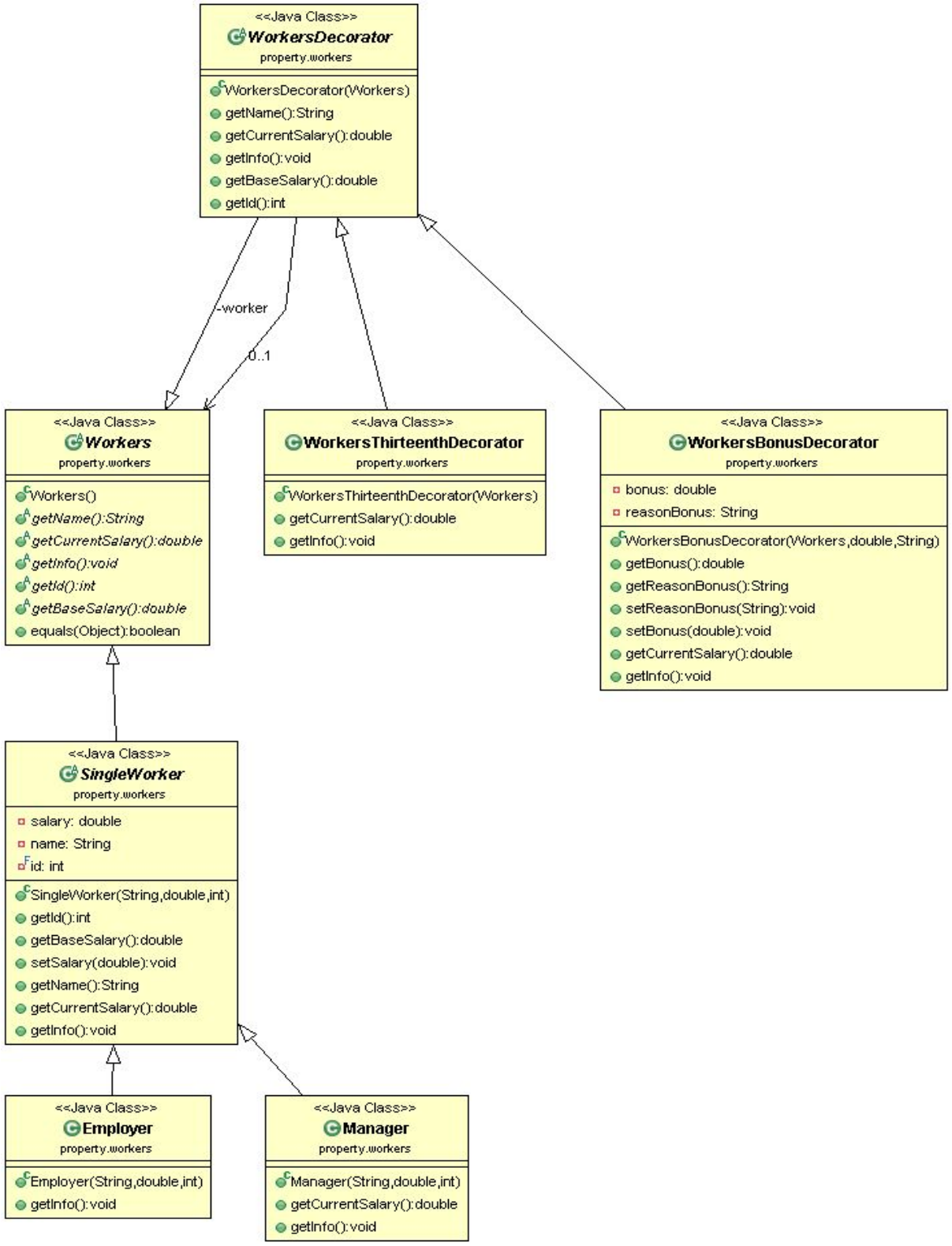
DESCRIZIONE DELL'ESERCIZIO PORTATO COME PROGETTO

L'esercizio che abbiamo scelto richiede di gestire **il pagamento delle tasse** imposte da uno stato sui suoi stabilimenti edili, che esso rende disponibili per abitarvi o per ricevere servizi pubblici. Con il nostro progetto intendiamo presentare una soluzione gestita all'interno di stati differenti tra di loro (Italia, Russia, etc..), ognuno dei quali avrà una propria *legge fiscale* per gli edifici lavorativi , come i complessi militari, e gli appartamenti privati dove abitano i cittadini. Pertanto, presentiamo una simulazione che consente di verificare il completamento di tali pagamenti e di sfruttare le risorse degli edifici disponibili (dal semplice pagamento delle tasse, alla possibilità di ricevere le loro informazioni). Di seguito vi riportiamo il *diagramma UML* dei singoli pacchetti del nostro programma per un'analisi più accurata:

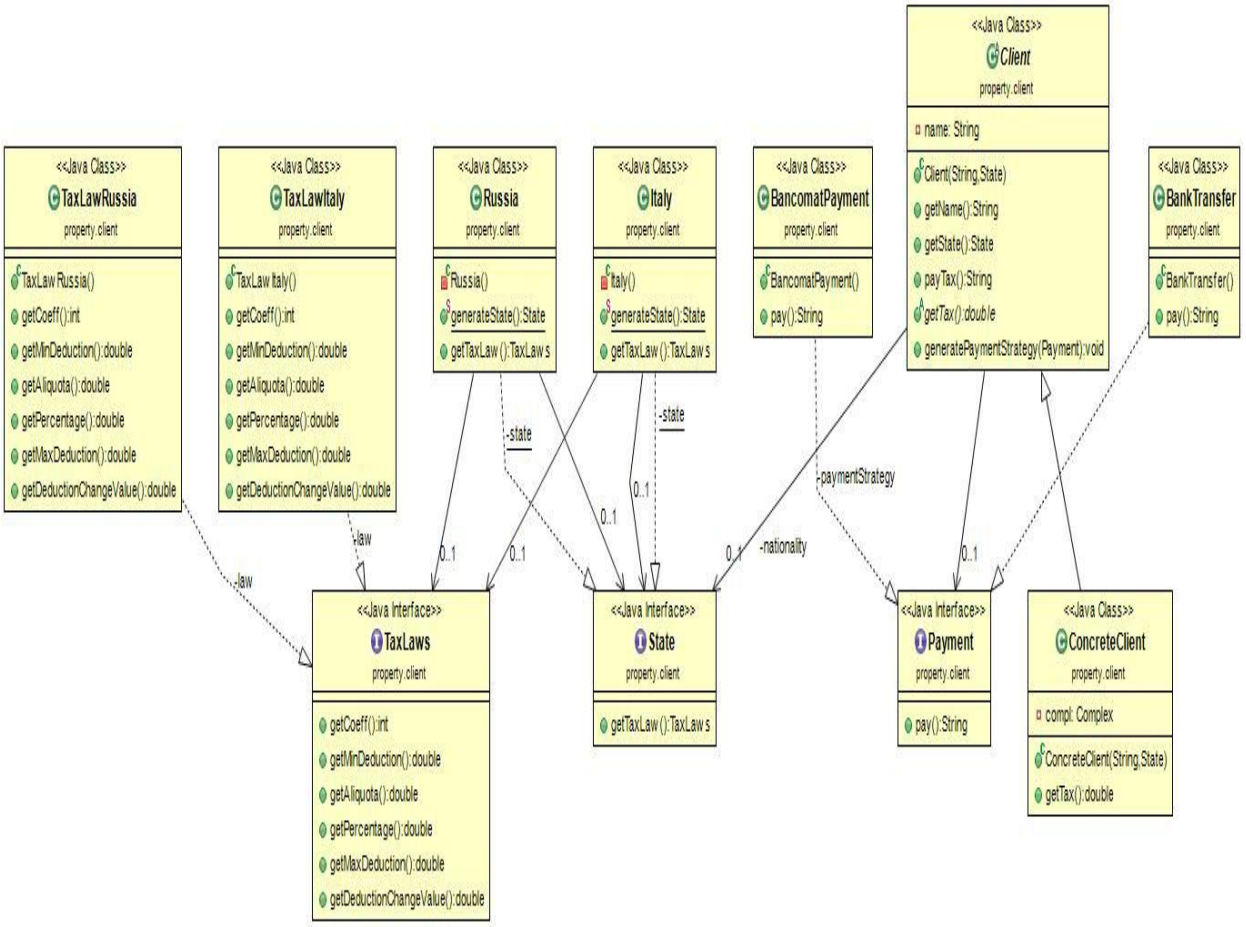
Package: property.tax



Package: property.workers



Package: property.client



SCELTE DEL DESIGN

Suddivisione del progetto:

Abbiamo suddiviso il progetto in tre pacchetti per offrire una disposizione migliore delle classi:

- Pacchetto client: nel quale definiamo gli stati, i clienti che vi appartengono, le leggi fiscali da rispettare a seconda nazionalità ed il metodo di notifica dei pagamenti tramite Carta di credito o transazioni bancarie.

- Pacchetto tax: utilizzato per suddividere i diversi tipi di stabilimenti disponibili (Case o Edifici Lavorativi), con le loro rispettive proprietà. Inoltre implementiamo operazioni esterne ad essi per poter stampare le loro informazioni e per poter essere a conoscenza del numero di edifici con una determinata caratteristica.

- Pacchetto workers: nel quale abbiamo definito dei lavoratori che sono necessari per edifici come negozi e complessi militari. E' così possibile simulare la gestione di enti pubblici in maniera più realistica, definendo per ogni lavoratore un salario ed un numero di identificazione che lo rappresenti. Inoltre è possibile eseguire delle modifiche, temporanee, ai lavoratori.

Utilizzo dei pattern:

Singleton: Utilizzato per poter definire gli Stati in cui vige una certa regola fiscale. Ogni stato dovrà essere unico, perciò abbiamo usato questo pattern per evitare che, ad esempio, *lo stato italiano avesse più copie*.

Composite: Utilizzato per rappresentare i complessi (all'interno della classe Complex) che conterranno un insieme di stabilimenti, sia pubblici che privati. In questo modo potremo confrontare gli stabilimenti presenti in un gruppo (tasse, tipo di abitazioni…) e potremo anche controllare tutti i lavoratori che sono presenti dentro ad un ente pubblico come un edificio militare, verificando i loro salari e le loro mansioni.

Decorator: Sfruttato per promuovere dei lavoratori, incrementando il loro salario tramite dei bonus, oppure per consegnare loro la tredicesima. Abbiamo deciso di usare questo pattern più che altro per poterlo provare dato che il nostro obiettivo principale erano le tasse sugli edifici.

Template: Utilizzato per restituire le informazioni degli edifici analizzati, tra cui le tasse da pagare per gli stessi, e le informazioni dei lavoratori presenti negli stabilimenti, descrivendo il loro salario, la loro identificazioni e se hanno usufruito di alcune promozioni.

Visitor: Usato per l'aggiunta di operazioni esterne agli stabilimenti come la verifica del numero di edifici che hanno una certa condizione (per esempio con il valore delle tasse al di sotto di un certo limite, se comprendono lavoratori ecc..).

Factory Method: Utilizzato per la creazione delle leggi fiscali presenti in uno stato: sarà quest'ultimo a preoccuparsi della gestione delle leggi tramite l'utilizzo delle sue sottoclassi (in questo caso rappresentanti lo stato italiano e lo stato russo).

Strategy: Utilizzato per gestire il pagamento dei Client tramite delle notifiche da parte dei servizi di Bancomat e delle transazioni bancarie. Inoltre, lo abbiamo sfruttato anche per il corretto pagamento delle tasse imposte dallo stato in cui si trova il cliente, potendo così ottenere la sua legge fiscale e di conseguenza il coefficiente che impone nei suoi edifici.

Observer: Viene utilizzato esclusivamente per il controllo ed il supporto dell'interfaccia grafica.

DESCRIZIONE DELL'IMPLEMENTAZIONE E DEL TESTING EFFETTUATI

Partendo dalla semplice **gestione del pagamento delle tasse**, di una singola abitazione (economica o lussuosa) e di vari complessi immobiliari, abbiamo deciso di ampliare il funzionamento del programma in modo da effettuare una simulazione su scala più ampia.
Il primo passo verso la realizzazione del nostro obiettivo è stato quello di aggiungere *nuovi tipi di immobili*. In questo modo abbiamo reso le due gerarchie, descritte sopra, sottoinsiemi di uno dei due gruppi principali di stabilimenti: **Working Estate**, che offrono un servizio ai cittadini residenti nella loro zona tramite dei lavoratori (Negozi, complessi Militari, Chiese) e **Home,** che sono le abitazioni di ogni cittadino.
Per poter leggere le informazioni relative agli immobili presenti, abbiamo sfruttato un *Visitor* che ci consente di verificare il numero di edifici presenti di una specifica categoria. Inoltre, abbiamo implementato **un insieme di classi per i lavoratori degli stabilimenti privati**, generando quindi delle componenti (sfruttando il pattern Composite), che potranno essere ***decorate*** aumentando i rispettivi salari, indicando i motivi di tali promozioni.
Sfruttando il pattern *Singleton*, abbiamo gestito il costo dei singoli edifici in Russia ed in Italia *(Con la possibilità di **aggiungere un qualsiasi stato** in caso di bisogno)*, entrambi con una propria legge dalla quale si determina il coefficiente di tassazione: i **Clienti** che saranno interessati ad acquistare degli stabilimenti dovranno attenersi alla legge del proprio stato per sapere quanto dovranno spendere. Infine, per simulare il pagamento, stampiamo semplicemente una stringa, dalle classi **CreditCardPayment** e **BankTransfer**, che ci indicano rispettivamente il pagamento effettuato con Carta di Credito e il pagamento effettuato con Bonifico. Per semplicità la payment strategy restituisce solo una stringa che conferma il pagamento, senza fare alcun controllo.
Nella cartella *testing* sono riportati i test dei rispettivi pacchetti elencati nella presentazione del diagramma uml. Al loro interno, testiamo la correttezza dei confronti tra oggetti (attraverso l'override del metodo equals), la correttezza del pagamento delle tasse, del conteggio di una certa categoria di stabilimenti presenti e se le decorazioni, di uno più lavoratori, sono state portate a buon fine.