Relazione della prova di progettazione relativa al corso di METODOLOGIE DI PROGRAMMAZIONE

A.A. 2018/2019

"PayRoll"

1. Componenti del gruppo in ordine alfabetico

Jebali Mohamed Salah

- **Matricola:** 5968114
- **Email:** mohamed.jebali@stud.unifi.it
- **Anno di iscrizione:** SECONDO anno
- **Superamento prove in itinere:** sì

Manfriani Bernardo

- **Matricola:** 6344894
- **Email:** bernardo.manfriani@stud.unifi.it
- **Anno di iscrizione:** SECONDO anno
- **Superamento prova in itinere:** no

Santi Andrea

- **Matricola:** 6014033
- **Email:** andrea.santi@stud.unifi.it
- **Anno di iscrizione:** SECONDO anno
- **Superamento prova in itinere:** sì

2. Introduzione al progetto

La traccia che abbiamo scelto è **PayRoll**, ovvero un libro paga che si occupa della gestione degli stipendi dei dipendenti di un'azienda.

Il software da noi creato offre diverse funzioni alle aziende che desiderino automatizzare la gestione degli stipendi dei propri dipendenti. Le funzioni offerte sono:

- Possibilità di creare un profilo per ogni dipendente dell'azienda con un portafoglio per depositare la propria remunerazione e un cartellino per segnare ore lavorate e/o commissioni svolte.
- Possibilità di **inserire** nel libro paga un dipendente appena assunto, e di **eliminarlo** dal libro paga qualora il rapporto di lavoro dovesse terminare.
- Possibilità di creare profili dipendente con diversi contratti:
    - Stipendio fisso.
    - Stipendio a provvigione.

    - Stipendio ad ore.

- Possibilità di offrire diversi metodi di erogazione dello stipendio ai dipendenti:
    - Pagamento tramite contanti.
    - Pagamento tramite bonifico.
    - Pagamento tramite assegno.
- Possibilità di inserire detrazioni fisse allo stipendio come:
    - Donazione in beneficienza pari all' "8xmille".
    - Donazione ad enti governativi pari al "5xmille".
    - Fondo risparmio personalizzato.
- Possibilità di schedulare e rendere automatico il rilascio degli stipendi:
    - Pagamento mensile per gli stipendi fissi.
    - Pagamento settimanale per gli stipendi orari e a commissione.

Il software è stato implementato/costruito in modo tale da lasciare aperte future estensioni e aggiunte sui metodi di pagamento, detrazioni allo stipendio lordo e tipologie di contratto.

3. Dettagli implementativi

A livello implementativo, possiamo visualizzare il software in gerarchie di oggetti, per rendere più semplice e intuitiva la spiegazione del loro funzionamento. Inoltre, per ogni gruppo di oggetti abbiamo applicato alcuni dei ***Design Patterns*** visti a lezione per risolvere problemi di design e di implementazione. Di seguito sono elencati i gruppi di oggetti con a seguito la lista dei *patterns* utilizzati. Verranno poi commentate e approfondite le scelte implementative attuate.

1) **Employee**
    - Decorator Pattern.
    - Strategy Pattern.
2) **Payment Method**
    - Strategy Pattern.
3) **NetSalaryCalculator**
    - Decorator Pattern.
    - Template Method Pattern.
4) **PayRoll**
    - Observer Pattern.
    - Singleton Pattern.

4. Discussione delle scelte implementative

1) EMPLOYEE

Il blocco di classi **Employee** è costituito da un'interfaccia *DecorableEmployee,* una classe concreta *BasicEmployee* che implementa l'interfaccia, una classe astratta *EmployeeDecorator* che implementa l'interfaccia, e una serie di sottoclassi concrete di *EmployeeDecorator*, che sono *FixedSalaryEmployee, CommissionSalaryEmployee e HourSalaryEmployee.* È presente anche un oggetto di tipo *Wallet* che ha come scopo quello

di tenere il denaro guadagnato dal dipendente. Inoltre, un oggetto di tipo *BasicEmployee* possiede un oggetto *Workcard.*

Abbiamo scelto di usare il *DecoratorPattern* in questo caso per permettere di creare oggetti employee con più forme di contratto contemporaneamente. Infatti, grazie alle classi di decorazione un employee può avere sia un contratto a stipendio fisso che un contratto a provvigione insieme.

Invece, per i metodi di pagamento abbiamo optato per uno *StrategyPattern* per delegare ad un oggetto *PaymentMethod* di occuparsi del rilascio del pagamento. In questo modo, la scelta del metodo di pagamento è più flessibile e lascia anche la possibilità di creare un nuovo metodo di pagamento.

Inoltre, abbiamo deciso di delegare la responsabilità del calcolo dello stipendio netto ad un oggetto *NetSalaryCalculator*. Questo perché le detrazioni possono essere molteplici e diventare complesse, quindi ci è sembrato più opportuno utilizzare un oggetto per non caricare di responsabilità l'employee. Inoltre, così facendo, è possibile decorare un oggetto *NetSalaryCalculator* con varie detrazioni (maggiori dettagli più avanti).

Dunque, il *Client* che decide di creare un oggetto employee istanzia un *BasicEmployee*. Un oggetto di questo tipo dispone di alcuni campi riconoscitivi e di vari metodi *getter* e *setter*. Quando sarà definito una tipologia di contratto per il *BasicEmployee* creato, basterà decorarlo con la classe scelta (esempio: DecorableEmployee fixed = new FixedSalaryEmployee (basic, 1000)).

L'oggetto *WorkCard* è pensato a immagine e somiglianza ad un cartellino da lavoro. All'interno dell'oggetto vengono aggiornate le ore lavorate e/o le commissioni assolte. In questo modo deleghiamo la responsabilità della gestione delle ore e delle commissioni a terzi. È importante osservare che questo oggetto è detenuto da un *BasicEmployee e* che il suo utilizzo dipende dalla decorazione di *BasicEmployee*. Infatti, se la decorazione è un *FixedSalaryEmployee* la presenza del cartellino non influenzerà il suo stipendio. Ma se la decorazione è un *CommissionSalaryEmployee* il cartellino servirà per sapere quante commissioni sono state assolte, e dunque calcolare lo stipendio lordo del dipendente. L'esigenza di questo oggetto è nata da una complicanza nella quale ci siamo imbattuti durante l'implementazione del progetto. Infatti, il problema che avevamo è che i metodi *updateHours* e *updateCommission* si trovavano solo nelle rispettive classi, e una volta decorati questi metodi diventavano "invisibili".

2) **PaymentMethod**

Il blocco di oggetti *PaymentMethod* è costituito da un'interfaccia *PaymentMethod* e da tre classi concrete che implementano l'interfaccia, una per ogni tipo di pagamento. Le classi sono *CashPayment, TransferPayment, CheckPayment.* Quest'ultima ha un'istanza di un oggetto *Check* che viene generato quando richiedo di inviare il pagamento.

3) **NetSalaryCalculator**

Il blocco di oggetti *NetSalaryCalculator* è costituito da un'interfaccia *NetSalaryCalculator*, una classe concreta *BasicNetSalaryCalculator* che implementa l'interfaccia, una classe

astratta *NetSalaryCalcDecorator* che implementa l'interfaccia e una serie di sottoclassi concrete che estendono la classe astratta.

In questo caso abbiamo scelto di utilizzare il *DecoratorPattern* per decorare la classe *BasicNetSalaryCalculator* per dare la possibilità di aggiungere anche più detrazioni contemporaneamente. Ad esempio, un *employee* può decidere di devolvere dei soldi in beneficienza e, allo stesso tempo, di destinare una parte del proprio stipendio ad un fondo risparmio.

Un oggetto di tipo *BasicNetSalaryCalculator* offre come detrazione base una tassazione standard al 20% dello stipendio lordo. Infatti, un dipendente può non volere altri tipi di detrazione, ma le tasse allo stato non sono esenti.

Inoltre, ci è parso opportuno utilizzare un *Template Method Decorator* per le classi *DeductionDecorator, GovernmentDeduction, BeneficalDeduction* perché le ultime due classi differivano solo per la percentuale da detrarre dallo stipendio lordo, e dunque abbiamo evitato duplicazione di codice.

4) **PayRoll**

Il blocco di oggetti *PayRoll* è costituito dalle interfacce *ObservablePayRoll* e *Observer*, e dalle classi concrete *Payroll, MonthlyPayrollObserver, WeeklyPayrollObserver, ResetWeeklyObserver* e *ResetMonthlyObserver*.

Un oggetto di tipo *Payroll* è "contenitore" di oggetti *DecorableEmployee*. Si comporta esattamente come se fosse un libro paga aziendale: contiene le informazioni dei propri dipendenti e all'occorrenza si occupa del rilascio degli stipendi.

Proprio per automatizzare il rilascio degli stipendi abbiamo optato per l'*Observer Pattern*. Infatti, l'oggetto Payroll possiede un contatore di giorni che viene aggiornato da un Timer della libreria *java.util.* Ogni volta che viene aggiornato il giorno, vengono notificati i due *Observer: MonthlyPayRollObserver* e *WeeklyPayrollObserver*. Il primo si occupa del rilascio degli stipendi ogni 28 giorni, mentre il secondo esegue il rilascio degli stipendi ogni 7 giorni. Infatti, ad ogni tipologia di contratto è associata una diversa tempistica di pagamento, che può essere mensile per i dipendenti a stipendio fisso, e settimanale per gli altri.

Gli altri due *observer* concreti, *ResetWeeklyObserver* e *ResetMonthlyObserver*, servono per resettare le eventuali ore lavorate e/o le commissioni assolte. Il primo li resetta ogni 7 giorni, mentre il secondo ogni 28. Mentre il primo si occuperà degli oggetti *CommissionSalaryEmployee*e *HourSalaryEmployee* (e delle loro combinazioni), il secondo avrà la responsabilità di resettare le ore e/o le commissioni degli oggetti *CommissionSalaryEmployee* e/o *HourSalaryEmployee* decorati dal *FixedSalaryEmployee.*

Infine, abbiamo utilizzato il *Singleton Pattern* perché abbiamo pensato che ogni azienda avesse un unico libro paga per gestire i propri dipendenti.

5. Descrizione sintetica del funzionamento del software

Di seguito verrà offerta una descrizione sintetica del funzionamento del software. Per maggiori dettagli si rimanda al diagramma UML allegato e al codice Java, con relativi test, del progetto in questione.

Innanzitutto, si crea un oggetto concreto *Payroll* e si creano gli oggetti *Observer* che guardano l'oggetto *Payroll*. Prima di aggiungere un impiegato nel libro paga questo va costruito. Per farlo, vanno prima costruiti un oggetto *Workcard* (cartellino da lavoro) e poi un *BasicEmployee*, passandogli tra i parametri del costruttore l'oggetto *WorkCard.* Non appena il *BasicEmployee* viene costruito, ad esso vengono associati un metodo di pagamento standard (*CashPaymentMethod),* un oggetto base *BasicNetSalaryCalculator*, che servirà per calcolare lo stipendio netto dell'impiegato, e un oggetto *Wallet* nel quale verrò depositato lo stipendio dell'impiegato.

Dunque, all'impiegato dovrà essere assegnata una tipologia di contratto. Per farlo basterà decorarlo con uno dei tre decoratori adibiti ai contratti o con una loro combinazione.

Se decidiamo di decorarlo con un *FixedSalaryEmployee* dobbiamo indicare l'importo fisso dello stipendio. Per il *CommissionSalaryEmployee*, dobbiamo passargli la provvigione a commissione, mentre per l'*HourSalaryEmployee* sarà necessario assegnare la paga oraria.

A questo punto, sarà possibile aggiungere al *Payroll* creato il dipendente e avviare lo *scheduler* dei giorni, che si occuperà tenere aggiornato il calendario per effettuare i pagamenti.

Se un impiegato decidesse di cambiare metodo di pagamento, sarà possibile modificarlo passandogli attraverso l'apposito metodo un nuovo oggetto di pagamento.

Inoltre, se un impiegato volesse devolvere parte del proprio stipendio in beneficienza o in un fondo risparmio, occorrerebbe prima di tutto decorare l'oggetto *BasicNetSalaryCalculator* con le decorazioni opportune, e dopodiché passarlo all'*employee* tramite l'apposito metodo.

6. Osservazioni e note implementative

L'utilizzo del *DecoratorPattern* ha lo svantaggio di rendere complessa la costruzione di un oggetto e di lasciare molte responsabilità al *Client*. Infatti, nel *Client*, senza le dovute accortezze è possibile decorare un oggetto già decorato dalla stessa decorazione, ad esempio decorare un *CommissionEmployee* con un *CommissionEmployee.*

Abbiamo deciso di erogare i pagamenti ogni 30 giorni, in un'ottica in cui un anno è composto da 12 mesi di 30 giorni, per rendere più omogeneo il meccanismo. Inoltre, abbiamo deciso per comodità dell'azienda, che se un *CommissionSalaryEmployee* o un *HourSalaryEmployee* vengono decorati con un *FixedSalaryEmployee,* questo verrà pagato mensilmente e non più settimanalmente.

Per testare gli *Observer* concreti abbiamo utilizzato un *setDays(int days)* per settare i giorni e testare il loro funzionamento. Questo metodo e i test relativi si trovano nel codice, ma sono commentati perché non sono utili al funzionamento del software, ed il metodo *setDays(int days),* poiché public, potrebbe essere pericoloso.

7. Codice Java, Test e UML

Il codice Java del progetto e i relativi test si trovano in allegato nel file UML_PAYROLL.png.

Di seguito sarà possibile visionare un'immagine dell'UML semplificata e in allegato si trova un'immagine del diagramma UML più dettagliata.