Progetto per il Corso di
MP
–A.A. 2023/2024 –

Libro-paga degli impiegati di un’azienda

Autore: Nabiollah Tavakkoli
E-mail: nabiollah.tavakkoli@edu.unifi.it
Numero di matricola: 5953800
Data di consegna: 29/08/2024

Payroll

Ogni giorno/mese in un'azienda bisogna calcolare (e inviare) la paga ad ogni impiegato.
Ci sono diversi tipi di impiegati:

1. impiegati pagati ad ora, (inviano a fine giornata il loro tesserino su cui sono indicate le ore (aggiuntive) di lavoro)
2. impiegati con salario fisso
3. impiegati pagati con commissione sulle vendite effettuate
4. impiegati che ricadono nell'intersezione di due categorie, ad esempio, di (2) e (3).

I dati suddetti possono essere modificati: ad esempio, la classificazione di un impiegato.
Ogni impiegato può scegliere un particolare metodo per ricevere il pagamento (assegno, bonifico, ecc.)
C'è uno scheduling dei pagamenti: ogni impiegato è pagato con tempistiche diverse, ad es.,

- quelli ad ora e sono pagati ogni settimana,
- quelli a stipendio sono pagati ogni mese, ecc.

Il progetto è stato organizzato in

- Due source folder: *src* and *tests* che contengono rispettivamente il codice sorgente ed i vari test per testare tale codice sorgente.
- Tre pacchetti: *contracts, employee, salarySchedual* che contengono rispettivamente le seguenti classi con le loro proprietà

La classe:

- Commission:
  - È stato pensato per gestire gli impiegati pagati con commissione sulle vendite effettuate
  - Il costruttore della classe è privato e si può creare istanze di classe usando i due metodi statici *commissionsSalaryWithDixedMaxedSalesAndCommission, commissionSalary* e i parametri passati a questi metodi devono essere maggiore di zero
  - Contiene i metodi
    - *getFinalSalary*,
      - ♦ per ottenere il salario finale di un impiegato
    - *setInterval*
      - ♦ per specificare lo scheduling di pagamento dell'impiegato

- FixedSalary
  - È stato pensato per gestire gli impiegati pagati con salario fisso in base a loro orario di lavoro oppure con orario di lavoro fisso.
  - Il costruttore della classe è privato e si può creare istanze di classe usando i due metodi statici *fixedSalaryWithFixedHour, fixedSalaryWithVariableHour* e i parametri passati a questi metodi devono essere maggiore di zero
  - Contiene i metodi
    - *getFinalSalary*,
      - ♦ per ottenere il salario finale di un impiegato
    - *setInterval*
      - ♦ per specificare lo scheduling di pagamento dell'impiegato

- Hourly
  - È stato pensato per gestire gli impiegati pagati in base a loro orario di lavoro.
  - Contiene i metodi
    - *getFinalSalary*,
      - per ottenere il salario finale di un impiegato
    - *setInterval*
      - per specificare lo scheduling di pagamento dell'impiegato

- MultiContractExchange
  - È stato pensato per gestire i contratti multipli degli impiegati
  - Contiene i metodi:
    - addContract
      - per associare un nuovo tipo di contratto ad un impiegato,
      - prende in input un nuovo tipo di contratto che non dovrebbe essere *null*
    - modifyAContract
      - modifica un contratto in una posizione specifica associata all'impiegato con un nuovo contratto
      - il contratto passato dovrebbe essere di tipo *TypeOfContract* e non dovrebbe essere null

- MultiContract
  - È stato pensato per gestire gli impiegati che ricadono nell'intersezione di diverse categorie di contratti (es. *FixedSalary e Hourly*)
  - Contiene i metodi
    - *getFinalSalary*,
      - per ottenere il salario finale di un impiegato,
    - *setInterval*
      - per specificare lo scheduling di pagamento dell'impiegato

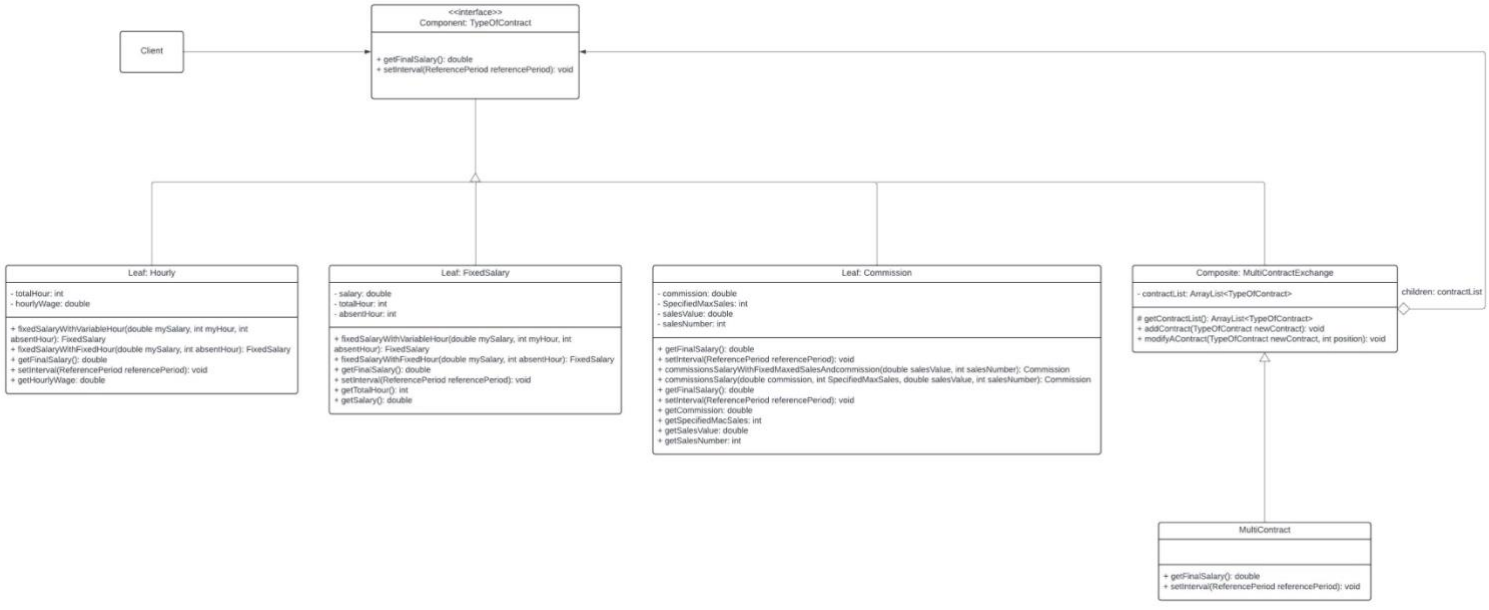
- WeeklyInterval / MonthlyInterval
  - È stato pensato per specificare lo scheduling dei pagamenti:
    - ogni impiegato è pagato con tempistiche diverse, ad es.,
      - quelli ad ora e sono pagati ogni settimana (7 giorni),
      - quelli a stipendio sono pagati ogni mese (31 giorni).
  - Ogni stipendio ha associato il suo periodo di riferimento, cioè un intervallo settimanale o mensile a cui lo stipendio si riferisce.
    - Il periodo di riferimento dipende dal tipo di contratto assegnato all'impiegato.
    - Per la gestione dei periodi di riferimento associati a ciascuno stipendio in base al tipo di contratto dell'impiegato abbiamo utilizzato il design pattern Visitor. In particolare:
      - La classe *ReferencePeriod* è l'interfaccia Visitor del pattern,
      - La classe *WeeklyInterval, MonthlyInterval* (che implementa *ReferencePeriod*) sono i *ConcreteVisitor*,
      - mentre dall'interfaccia *TypeOfContract* viene richiesto di implementare il metodo che accetta il visitatore.

- AccountingDate
  - È stato pensato per determinare la data dell'inizio e della fine del lavoro e per modificare tale data
  - Contiene i metodi:
    - isLeapYear
      - controlla se l'anno corrente è bisestile oppure meno
    - addDay
      - viene usato per aggiungere dei giorni ad una data e modificarla, tenendo conto dei gruppi specifici dei mesi
        - mese bisestile: 2' mese
        - mesi di 30 gioni: 4', 6', 9', 11'
        - mesi di 31 gioni: tutti altri mesi
      - se al 12' mese dell'anno corrente, la somma del giorno dato e dei giorni lavorati risulta maggiore di 31, allora i giorni aggiunti non vengono considerati
    - toggleMonth
      - viene usato per modificare il giorno di una data determinata nel caso in cui la somma del giorno dato e dei giorni lavorati risulta maggiore dell'ultimo giorno dei gruppi di mesi specifici.

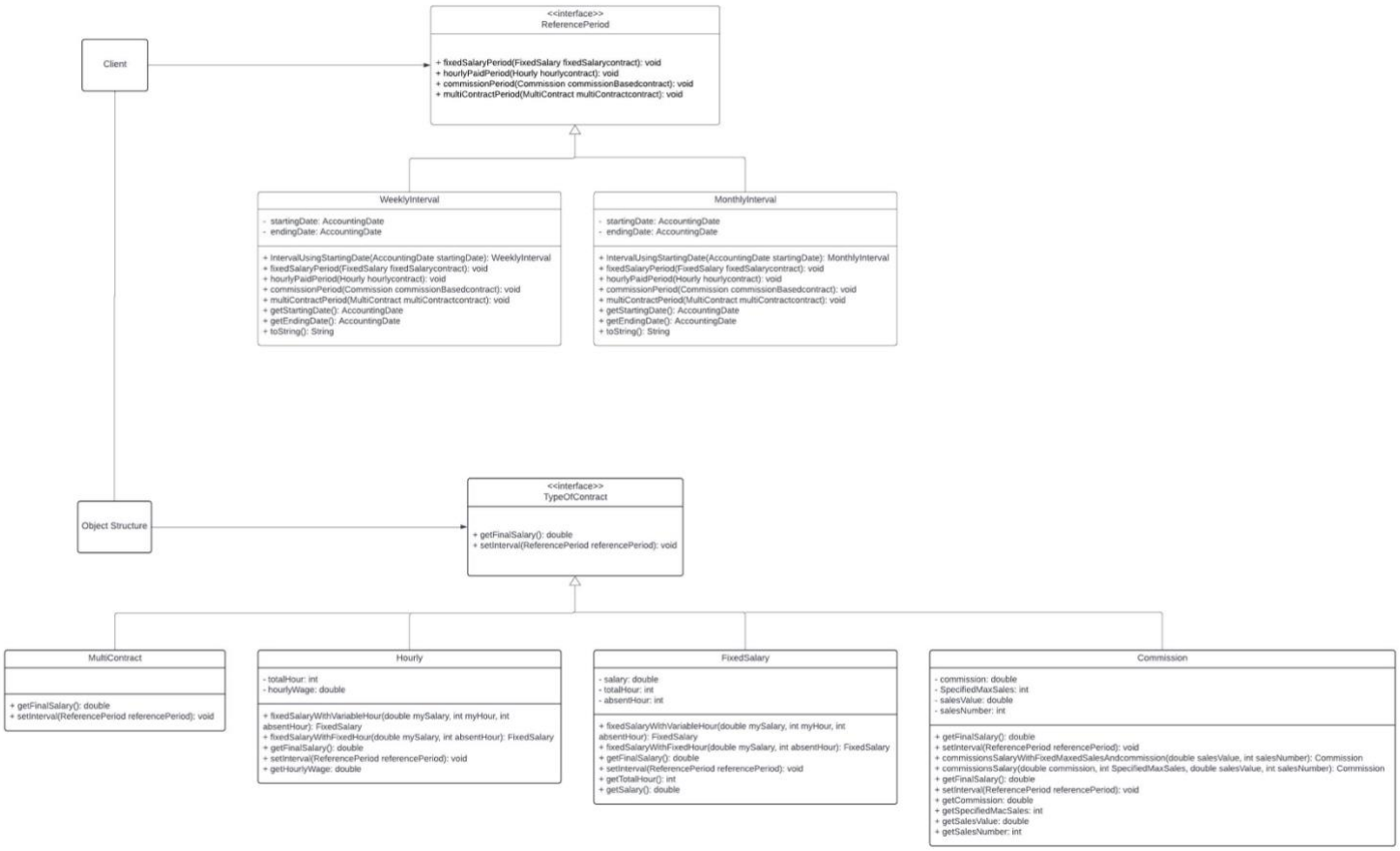
- Salary:
  - È stato pensato per ottenere lo stipendio di un *Employee* che ha inizializzato il suo lavoro in una data specifica, considerando un intervallo di lavoro mensile per *Employee*
    - I parametri passati al costruttore della classe durante l'inizializzazione non devono essere null
  - Contiene il metodo *getEmployeeSalary* che dato un *Employee* e il suo tipo di contratto ci restituisce il suo stipendio totale

Pattern applicati

- Composite:
  - Il pattern viene applicato nella **forma type safe**, quindi la gestione dei children sarà presente solo nel componente composto (cioè *MultiContractExchange*)
    - Componente è *TypeOfContract*
    - Oggetto composto è *MultiContractExchange*
    - Oggetti foglia sono *Hourly, FixedSalary, Commission*
  - Le classi del pattern hanno il metodo *getFinalSalary* in comune che viene definito nella classe *TypeOfContract* e implementato nelle classi *Hourly, FixedSalary, Commission, MultiContract*
  - *MultiContractExchange* ha una collezione di oggetti (di tipo statico *TypeOfContract*) detto *contractList*



- Visitor:
  - È stato utilizzato la versione del pattern Visitor coi metodi Void, allora
    - Dobbiamo mantenere uno stato
    - Un oggetto Visitor deve essere utilizzato una ed una sola volta
  - Abbiamo:
    - Il visitor astratto: ReferencePeriod
    - Due visitor concreti: WeeklyInterval e MonthlyInterval
    - Un elemento astratto TypeOfContract
    - Quattro elementi concreti *Hourly, FixedSalary, Commission, MultiContract*

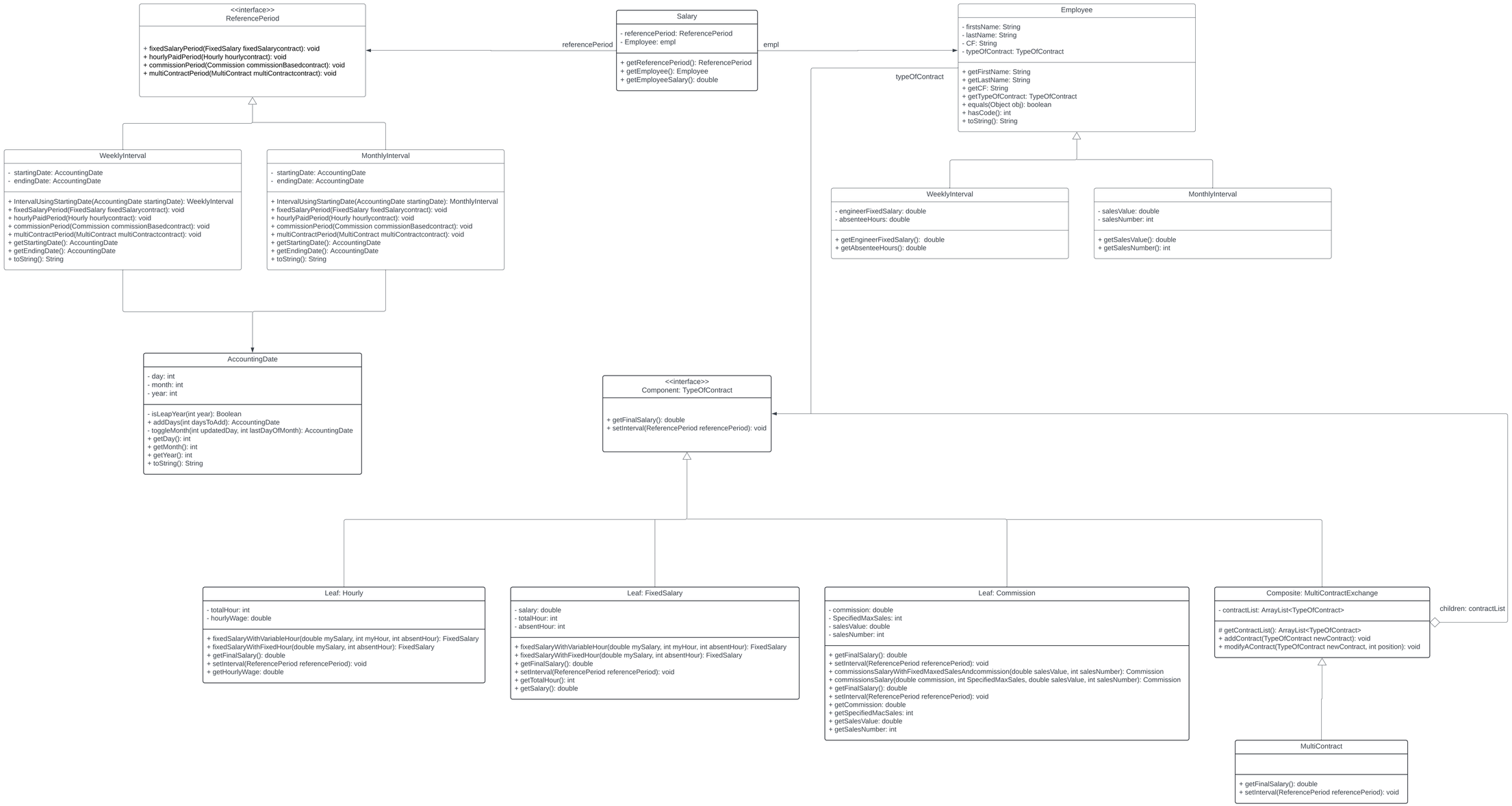


- Static Factory Method:

  - Poiché nelle classi *FixedSalary, Commission* abbiamo bisogno di più costruttori con vari parametri, allora invece di avere più costruttori in overloading che non sembra leggibile e pulito abbiamo usato il pattern Static Factory Method
    - Dichiarando il costruttore delle classi come *private* ed implementando i metodi statici con parametri specifici che invocano i costruttori privati.

  - Static Factory Method della classe *FixedSalary* viene utilizzato all'interno della classe *Engineer* nel pacchetto *employee,*

  - Static Factory Method della classe *Commission* viene utilizzato all'interno della classe *Sale* nel pacchetto *employee.*


<<interface>>
ReferencePeriod
+ fixedSalaryPeriod(FixedSalary fixedSalarycontract): void
+ hourlyPaidPeriod(Hourly hourlycontract): void
+ commissionPeriod(Commission commissionBasedcontract): void
+ multiContractPeriod(MultiContract multiContractcontract): void
Salary
- referencePeriod: ReferencePeriod
- Employee: empl
+ getReferencePeriod(): ReferencePeriod
+ getEmployee(): Employee
+ getEmployeeSalary(): double
referencePeriod
empl
Employee
- firstsName: String
- lastName: String
- CF: String
- typeOfContract: TypeOfContract
+ getFirstName: String
+ getLastName: String
+ getCF: String
+ getTypeOfContract: TypeOfContract
+ equals(Object obj): boolean
+ hasCode(): int
+ toString(): String
typeOfContract
WeeklyInterval
- startingDate: AccountingDate
- endingDate: AccountingDate
+ IntervalUsingStartingDate(AccountingDate startingDate): WeeklyInterval
+ fixedSalaryPeriod(FixedSalary fixedSalarycontract): void
+ hourlyPaidPeriod(Hourly hourlycontract): void
+ commissionPeriod(Commission commissionBasedcontract): void
+ multiContractPeriod(MultiContract multiContractcontract): void
+ getStartingDate(): AccountingDate
+ getEndingDate(): AccountingDate
+ toString(): String
MonthlyInterval
- startingDate: AccountingDate
- endingDate: AccountingDate
+ IntervalUsingStartingDate(AccountingDate startingDate): MonthlyInterval
+ fixedSalaryPeriod(FixedSalary fixedSalarycontract): void
+ hourlyPaidPeriod(Hourly hourlycontract): void
+ commissionPeriod(Commission commissionBasedcontract): void
+ multiContractPeriod(MultiContract multiContractcontract): void
+ getStartingDate(): AccountingDate
+ getEndingDate(): AccountingDate
+ toString(): String
WeeklyInterval
- engineerFixedSalary: double
- absenteeHours: double
+ getEngineerFixedSalary(): double
+ getAbsenteeHours(): double
MonthlyInterval
- salesValue: double
- salesNumber: int
+ getSalesValue(): double
+ getSalesNumber(): int
AccountingDate
- day: int
- month: int
- year: int
- isLeapYear(int year): Boolean
+ addDays(int daysToAdd): AccountingDate
- toggleMonth(int updatedDay, int lastDayOfMonth): AccountingDate
+ getDay(): int
+ getMonth(): int
+ getYear(): int
+ toString(): String
<<interface>>
Component: TypeOfContract
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void
Leaf: Hourly
- totalHour: int
- hourlyWage: double
+ fixedSalaryWithVariableHour(double mySalary, int myHour, int absentHour): FixedSalary
+ fixedSalaryWithFixedHour(double mySalary, int absentHour): FixedSalary
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void
+ getHourlyWage: double
Leaf: FixedSalary
- salary: double
- totalHour: int
- absentHour: int
+ fixedSalaryWithVariableHour(double mySalary, int myHour, int absentHour): FixedSalary
+ fixedSalaryWithFixedHour(double mySalary, int absentHour): FixedSalary
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void
+ getTotalHour(): int
+ getSalary(): double
Leaf: Commission
- commission: double
- SpecifiedMaxSales: int
- salesValue: double
- salesNumber: int
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void
+ commissionsSalaryWithFixedMaxedSalesAndcommission(double salesValue, int salesNumber): Commission
+ commissionsSalary(double commission, int SpecifiedMaxSales, double salesValue, int salesNumber): Commission
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void
+ getCommission: double
+ getSpecifiedMacSales: int
+ getSalesValue: double
+ getSalesNumber: int
Composite: MultiContractExchange
- contractList: ArrayList<TypeOfContract>
getContractList(): ArrayList<TypeOfContract>
+ addContract(TypeOfContract newContract): void
+ modifyAContract(TypeOfContract newContract, int position): void
children: contractList
MultiContract
+ getFinalSalary(): double
+ setInterval(ReferencePeriod referencePeriod): void