POLITECNICO
DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione
Corso di Laurea in Ingegneria Gestionale
Classe L-8 Ingegneria dell'informazione

Strumento per la valutazione e la selezione d'investimenti nel mercato immobiliare

Relatore
Prof. Fulvio Corno

Candidato
Soru Alessio

A.A. 2019/2020

Sommario

1 - Proposta di progetto

Studente proponente

s235059 Soru Alessio

Titolo della proposta

Strumento per la valutazione e la selezione d'investimenti nel mercato immobiliare

Descrizione del problema proposto

L'applicazione si pone l'obiettivo di consentire all'utente immobiliarista di avere un quadro generale, attraverso la visione di precise statistiche, sulla situazione immobiliare nella città presa in considerazione. In seguito a una fase preliminare di analisi, consiglierà un elenco ottimale di immobili da acquistare che andranno a creare un "portafoglio immobiliare" personalizzato a seconda della forza economica posseduta e delle preferenze indicate.

Descrizione della rilevanza gestionale del problema

Il problema ha una forte rilevanza per soggetti operanti nella compravendita di beni abitativi, consistendo nell'assunzione di uno strumento che permetta una immediata visione della gestione del proprio capitale. In base ai propri requisiti e alle risorse finanziare disponibili, fornirà un elemento aggiuntivo di confronto che possa supportare la valutazione del mercato preso in esame, appoggiando o confutando le osservazioni qualitative ottenute durante una fase di analisi preliminare.

Descrizione dei data-set per la valutazione

https://www.kaggle.com/harry007/philly-real-estate-data-set-sample

Per la creazione dell'applicazione, verrà utilizzato un data-set di compravendita immobiliare della città di Philadelphia i cui dati sono ottenuti da phila.gov.
Ogni riga della tabella individua una proprietà immobiliare, identificata dal suo indirizzo. Per ogni immobile sono disponibili i dati di alcune caratteristiche peculiari, tra le quali si possono precisare le più notevoli ai fini dello sviluppo dell'applicazione:
o Address (indirizzo);
o Postal code (codice postale);
o Violent crime rate (tasso di criminalità);
o School score (punteggio medio delle scuole della zona);

o Zillow estimate (stima del valore di mercato dell'immobile fatta da Zillow, compagnia immobiliare di elevata rilevanza negli USA);
o Rent estimate (stima del possibile rendimento dell'affitto dell'immobile);
o Year built (anno di costruzione);
o Finished SqFt (piedi quadrati dell'immobile completato);
o Bedrooms (numero di camere da letto);
o Bathrooms (numero di bagni);
o PropType (tipologia di immobile);

Inoltre, per completare i dati, verrà utilizzato un ulteriore data-set che mette in relazione i nomi dei quartieri con i codici postali ad essi associati.

Descrizione preliminare degli algoritmi coinvolti

L'applicazione prevederà una prima parte di ricerca di statistiche estratte dal database tramite semplici algoritmi di selezione dei record richiesti.
Il principale algoritmo implementato sarà di tipo ricorsivo e svolgerà la funzione di ricerca di un "portafoglio immobiliare" ottimale personalizzato in base ai valori indicati dall'utente. In particolare, verrà generato un elenco di immobili da acquistare (sostanzialmente un problema dello zaino) in base ai seguenti parametri indicati dall'utente:

- Obbligatori:
  ¬ N. minimo di case che si vogliono acquistare (il massimo viene imposto dall'applicazione per motivi di complessità algoritmica);
  ¬ Budget disponibile;

- Opzionali (sarà possibile selezionare al massimo 1/2 vincoli opzionali, una volta selezionato/i il/i vincolo/i non sarà possibile selezionarne altri):
  ¬ Tipologia di immobili;
  ¬ Stima del rendimento degli affitti (media dell'elenco degli immobili selezionati);
  ¬ Crime rate;
  ¬ Anno di costruzione;
  ¬ Range tra massimo e minimo di metri quadri dell'immobile (convertiti dai piedi quadrati);
  ¬ Numero di camere da letto minime per ogni immobile;
  ¬ Numero di bagni minimo per ogni immobile;
  ¬ Quartiere in cui si vogliono acquistare gli immobili (o almeno un immobile se non sono presenti immobili in vendita a sufficienza);
  ¬ Immobile da inserire necessariamente nell'elenco.

Inoltre, si eseguirà l'algoritmo ricorsivo in modo da sfruttare al massimo il budget per ottenere il numero minimo di immobili con le caratteristiche richieste (acquisto il numero minimo di immobili ma con valore di mercato maggiore, privilegia la qualità) oppure in modo da ottimizzare l'elenco minimizzando la spesa dal budget e inserendo il maggior numero possibile di immobili con le caratteristiche richieste (acquisto di un numero elevato di immobili ma con basso valore di mercato, privilegia la quantità).

Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione sarà suddivisa in due sezioni:

- La prima permetterà di raccogliere una serie di record sul contesto immobiliare della città, individuati in modo da dare all'utente una panoramica della situazione generale. In particolare, l'utente avrà la possibilità di visualizzare le statistiche medie per ogni zona della città, osservando sia la zona con valori massimi e minimi per ogni caratteristica che decide di selezionare dal data-set, sia gli elenchi dei quartieri ordinati in base alla caratteristica in quel momento selezionata. Con caratteristiche si intendono le quantità medie per zona dei valori tabulari del data-set come il crime rate, lo school score, l'anno di costruzione medio degli immobili, affitti medi, valore medio di mercato, tipologia principale di immobili.

- La seconda richiederà i valori necessari alla ricerca di un "portafoglio immobiliare" ottimale, permettendo l'inserimento dei parametri obbligatori (budget disponibile, n. minimo di immobili che si vuole acquistare, scelta booleana tra privilegiare la quantità o la qualità degli immobili) e poi richiederà di selezionare una/due caratteristiche opzionali dalla lista estratta dal database, per la generazione del "portafoglio immobiliare" personalizzato (elenco ottimale che verrà visualizzato dall'utente in seguito alla selezione dei parametri e l'invio della richiesta) che, appunto, varierà in base alle caratteristiche selezionate dall'utente. In particolare, per esempio, l'utente avrà la possibilità di indicare un immobile che vuole necessariamente acquistare e che perciò dovrà essere obbligatoriamente inserito nel portafoglio, di conseguenza l'applicazione darà la possibilità di ricalcolare la soluzione ottima in base al nuovo parametro imposto.

2 - Descrizione del problema affrontato

Contesto operativo

Uno dei settori che ha maggiormente eco nel sistema economico e una forte risonanza nelle analisi finanziare è da sempre il mercato immobiliare.

Per questo, risulta particolarmente fondamentale che l'analisi sulla compravendita di beni abitativi si evolva di pari passo con lo sviluppo di competenza informatica, sfruttando in maniera consistente le capacità di quest'ultimo per un'applicazione interdisciplinare su aree differenti.

In tal senso, è fondamentale lo sviluppo di software che le aziende del settore immobiliare possano sfruttare per ricerche ad uso interno ma anche per presentare ai clienti una panoramica completa del mercato preso in esame.
L'obiettivo di tali software deve essere quello di migliorare la visione immediata del quadro generale, ma anche poter offrire soluzioni ottimizzate in base alle richieste del cliente.

Caratterizzazione del problema e obiettivo software

Andando a focalizzarsi su specifiche esigenze, emerge la necessità di uno strumento che possa supportare le decisioni d'investimento per gli immobiliaristi operanti nel settore.

Di conseguenza, risulta particolarmente rilevante lo sviluppo di un'applicazione che permetta di ottenere dei suggerimenti personalizzati per la gestione del proprio capitale con la finalità di raggiungere standard qualitativi e quantitativi più elevati.

Nel dettaglio e in prima istanza, l'applicazione si pone l'obbiettivo di permettere di ottenere un quadro generale sugli immobili nell'area presa in considerazione. Questo potrà avvenire attraverso la visione di precise statistiche prodotte in base alle caratteristiche che si ha intenzione di esaminare.

In secondo luogo, con lo scopo di filtrare le informazioni, verranno richiesti all'utente i parametri chiave per la valutazione di investimenti immobiliari (zona, tipologia di immobile, superficie, stima del rendimento degli affitti futuri, budget, ecc.) e l'applicazione genererà un "portafoglio immobiliare" che permetterà all'utente di avere un elenco consigliato di immobili da acquistare ottimizzato in base alle sue preferenze. Tra queste, una peculiarità notevole del software è quella di consultare il cliente sull'orientamento del suo portafoglio, permettendo una scelta, e conseguentemente una ricerca della lista ottimale, orientata su indici di qualità - sotto forma di punteggi basati su precise caratteristiche degli immobili - o su valori di quantità che privilegiano il numero di immobili da acquistare.

3 - Descrizione dei data-set utilizzati

Per lo sviluppo dell'applicazione, la scelta del data-set è caduta su un archivio di dati immobiliari della città di Philadelphia (Stati Uniti, Pennsylvania).

Da un punto di vista esemplificativo, la base dati risulta particolarmente funzionale ai propositi dell'applicazione. Infatti, illustra informazioni sulla compravendita di immobili a Philadelphia nel 2016/17. Tali dati hanno una notevole utilità in quanto presentano parametri fortemente pertinenti con l'obiettivo di consigliare un elenco ottimale di abitazioni da acquistare focalizzato sul rendimento degli investimenti nel settore.

Il dataset è disponibile all'indirizzo (dal sito di database kaggle e precedentemente ottenuto da phila.gov):
https://www.kaggle.com/harry007/philly-real-estate-data-set-sample

Tabella "properties"

I dati sono raccolti in un'unica tabella "properties" con 30 istanze.

Ogni riga della tabella individua un immobile, identificato univocamente dal suo indirizzo dal suo indirizzo "Address" (chiave primaria), per un totale di 612 elementi.
Per ogni immobile, sono disponibili i dati di alcune caratteristiche peculiari (corrispondenti ad alcune delle 33 colonne fruibili), tra le quali si possono precisare le più notevoli utilizzate per lo sviluppo dell'applicazione:

- "Address" (indirizzo);
- Postal_code" (codice postale);
- "Violent_Crime_Rate" (tasso di criminalità);
- "School_Score" (punteggio medio delle scuole della zona);
- "Zillow_Estimate" (stima del valore di mercato dell'immobile fatta da Zillow, compagnia immobiliare di elevata rilevanza negli USA);
- "Rent_Estimate" (stima del possibile rendimento dell'affitto dell'immobile);
- "Year built" (anno di costruzione);
- "PropType" (tipologia di immobile);
- "Finished_SqFt" (piedi quadrati dell'immobile completato che verranno sempre convertiti in metri quadrati – secondo la conversione mq = (ftq/10.764));
- "Bedrooms" (numero di camere da letto);
- "Bathrooms" (numero di bagni);

3 - Descrizione dei data-set utilizzati

Le colonne della tabella “properties” prese in esame sono rappresentati con i seguenti tipi di dati (schermata del programma per l’elaborazione di dati relazionali HeidiSql):

Nome	Tipo di dati
Address	VARCHAR
Postal_Code	INT
Violent_Crime...	DECIMAL
School_Score	DECIMAL

Nome	Tipo di dati
School_Score	DECIMAL
Zillow_Estimate	DECIMAL
yearBuilt	INT
finished_SqFt	INT

Nome	Tipo di dati
bathrooms	VARCHAR
bedrooms	VARCHAR
PropType	VARCHAR

Tabella “neighborhood”

Inoltre, per completare i dati, è stata inserita una seconda tabella “neighborhood” che permette di associare i codici postali con il nome delle rispettive zone (quartieri).

A ogni “Postal_Code” è univocamente associato un solo “Neighborhood”.

La tabella “neighborhood” è rappresentata come segue:

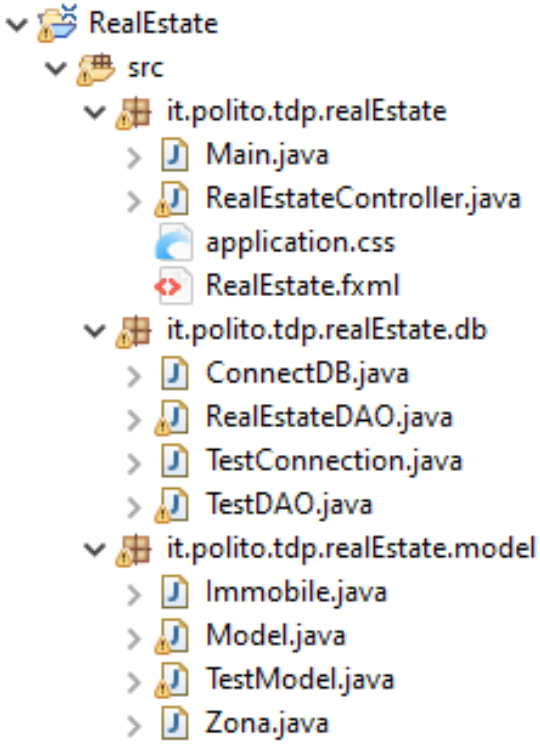
Nome	Tipo di dati
Neighborhood	VARCHAR
Postal_Code	INT

In questo modo, con una JOIN, è possibile associare il nome del quartiere (e non solo il codice postale) della tabella “neighborhood” con gli elementi (singoli immobili) della tabella “properties”.

4 - Descrizione delle strutture dati

L'applicazione è stata sviluppata in linguaggio Java, supportato dalle interfacce JavaFX. Per il suo sviluppo, è stato seguito il pattern MVC (Model View Controller) e quello DAO (Data Access Object). In questo modo vengono tenute separate le tre parti del programma: logica del programma, interfaccia utente, accesso al database.

In linea con quanto appena detto, la struttura dati dell'applicazione è conseguentemente suddivisa in tre differenti packages (come mostrato in figura):



- **it.polito.tdp.RealeEstate**
  È il package che contiene la classe Main, che serve per avviare l'applicazione, e il RealEstateController che collega il Main alla logica applicativa e definisce i metodi che permettono l'interazione tra la user interface – è la View nel pattern MVC - e i corrispondenti file FXML, contenuti nello stesso package, che ne definiscono la grafica;

- **it.polito.tdp.RealeEstate.db**
  È il package che consente di seguire il pattern DAO, contenendo la classe ConnectDB, per la connessione al database, la classe RealEstateDAO, che permette il caricamento dei record necessari per far funzionare l'applicazione attraverso query SQL; contiene anche le classi TestConnection e TestDAO che testano rispettivamente che la connessione effettivamente avvenga e che i metodi del DAO funzionino correttamente;

- **it.polito.tdp.RealeEstate.Model**
  È il package che contiene l'intera logica applicativa nella Classe Model, definendo i metodi e le funzionalità dell'applicazione; inoltre, contiene le classi Immobile e Zona, utili per salvare in degli oggetti a sé stanti le rispettive caratteristiche richieste al database.

5 - Descrizione degli algoritmi utilizzati

L'applicazione è suddivisa in due sezioni principali.
La prima sviluppa gli algoritmi di ricerca dei record che danno all'utente una panoramica generale sul contesto immobiliare, la seconda riguarda la ricerca del portafoglio immobiliare ottimizzato con l'utilizzo di algoritmi ricorsivi.

Algoritmi sezione A – statistiche sul contesto generale

La prima parte permette all'utente di selezionare una proprietà caratteristica degli immobili da un elenco di sei elementi. L'intento di tale selezione è finalizzato, in seguito al click di alcuni pulsanti impostati nel Controller, alla creazione di un elenco delle zone ordinato (crescente/decrescente) e della stampa del nome della zona migliore e peggiore, in linea con la caratteristica scelta.
Analizzando nel dettaglio il metodo utilizzato nel model per ottenere l'elenco ordinato, si può notare che l'elenco vien direttamente restituito invocando il metodo dal DAO – che esegue una query SQL con tutti i filtri - dopo aver passato la caratteristica selezionata e l'ordine scelto dall'utente:

metodo nel Model

```
public List<Zona> getElencoPerCarattristica(String caratteristica, String ordine) {
    return dao.getElencoPerCarattristica(caratteristica, ordine);
}
```

metodo nel DAO

```
public List<Zona> getElencoPerCarattristica(String caratteristica, String ordine) {
    if(ordine == "Crescente")
        ordine="ASC";
    else if(ordine == "Decrescente")
        ordine = "DESC";
    //ordino già nella query
    String sql = "SELECT p.Postal_Code AS postalCode, " +
            " n.Neighborhood AS neighborhoodName, " +
            " AVG(p.Violent_Crime_Rate) AS avgCrimeRate, " +
            " AVG(p.School_Score) AS avgSchoolRate, " +
            " AVG(p.Zillow_Estimate) AS marketValue, " +
            " AVG(p.Rent_Estimate ) AS avgRent, " +
            " AVG(p.yearBuilt) AS avgYear, " +
            " ((AVG(p.finished_SqFt))/10.764) AS avgMq " +
            "FROM properties AS p, neighborhood AS n " +
            "WHERE p.Postal_Code = n.Postal_Code AND " +
            "   p.Violent_Crime_Rate!=0 AND " +
            "   p.School_Score!=0 AND " +
            "   p.Zillow_Estimate!=0 AND " +
            "   p.Rent_Estimate!=0 AND  " +
            "   p.yearBuilt!=0 AND  " +
            "   ((p.finished_SqFt)/10.764)!=0  " +
            "GROUP BY postalCode, neighborhoodName " +
            "ORDER BY "+caratteristica+" "+ordine+" " ;
    try {
        Connection conn = ConnectDB.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet rs = st.executeQuery();
        List<Zona> zone = new ArrayList<Zona>();
        while (rs.next()) {
            Zona z = new Zona(rs.getInt("postalCode"), rs.getString("neighborhoodName"),
                    rs.getFloat("avgCrimeRate"), rs.getFloat("avgSchoolRate"),
                    rs.getFloat("marketValue"), rs.getFloat("avgRent"), rs.getInt("avgYear"),
                    rs.getFloat("avgMq"));
            zone.add(z); // id e oggetto
        }
        conn.close();
        return zone;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("SQL Query Error");
    }
}
```

5 - Descrizione degli algoritmi utilizzati

Per trovare la zona peggiore e quella migliore, vengono confrontati tutti i valori della caratteristica su una mappa contenente tutte le possibili zone (l'algoritmo usato è identico per migliore e

```
public String getZonaMigliorePerCaratteristica(String caratteristica) {
    Zona zonaBest=null;
    for(Zona z : this.zoneIdMap.values()) {

        if(zonaBest==null)
            zonaBest=z;

        switch(caratteristica) {
        case "Tasso di criminalità":
            // crime rate basso è migliore
            if(z.getAvgCrimeRate() < zonaBest.getAvgCrimeRate())
                zonaBest = z;
            break;
        case "Livello scolastico":
            //school rate alto è migliore
            if(z.getAvgSchoolRate()>zonaBest.getAvgSchoolRate())
                zonaBest = z;
            break;
        case "Valore di mercato medio":
            // valore di mercato alto è migliore perchè è una buona casa
            if(z.getMarketValue()>zonaBest.getMarketValue())
                zonaBest = z;
            break;
        case "Affitto medio stimato":
            // affitto alto è migliore perchè l'investitore deve affittare
            if(z.getAvgRent()>zonaBest.getAvgRent())
                zonaBest = z;
            break;
        case "Anno di costruzione":
            // anno di costruzione alto=recente è migliore
            if(z.getAvgYear()>zonaBest.getAvgYear())
                zonaBest = z;
            break;
        case "Metratura media":
            // casa grande migliore
            if(z.getAvgMq()>zonaBest.getAvgMq())
                zonaBest = z;
            break;
        }
    }
    // ho la zona migliore, ritorno la stringa con
    // nome zona e valore della caratteristica richiesta
    switch(caratteristica) {
    case "Tasso di criminalità":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getAvgCrimeRate();
    case "Livello scolastico":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getAvgSchoolRate();
    case "Valore di mercato medio":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getMarketValue();
    case "Affitto medio stimato":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getAvgRent();
    case "Anno di costruzione":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getAvgYear();
    case "Metratura media":
        return zonaBest.getneighborhoodName().toString()+" - "+ zonaBest.getAvgMq();
    }

    return null;
}
        return null;
    }
```

5 - Descrizione degli algoritmi utilizzati

Algoritmi sezione B – portafoglio immobiliare

Le istruzioni che portano alla creazione di un portafoglio immobiliare sono suddivisibili in tre sottosezioni.

1. Gestione Filtri: il metodo `gestioneFiltri()` riceve dal Controller tutti i filtri impostati dall'utente, alcuni dei quali opzionali - ma comunque utili per restringere il campo di ricerca nella ricorsione e ridurre i tempi -e altri obbligatori - senza i quali l'applicazione non permette di proseguire.
   Il filtro obbligatorio chiave per il corretto funzionamento del successivo algoritmo ricorsivo è quello che imposta una ricerca orientata sulla qualità o sulla quantità (valore booleano), a seconda della quale verranno chiamati due differenti algoritmi ricorsivi.
   A questo punto il metodo seleziona un elenco di immobili filtrati dalle variabile passate, richiamando il metodo `dao.getImmobiliFiltrati()` dal DAO, e setta i parametri utili alla ricerca: soglie di budget ed eventuali pesi riguardanti il valore che l'utente associa a differenti caratteristiche di qualità. Il metodo ritorna la lista di immobili filtrati, utile per l'aggiunta eventuale di un immobile, tra quelli possibili, che l'utente vuole necessariamente nel proprio portafoglio.

```
// GESTISCO PRIMA TUTTI I FILTRI
public List<Immobile> gestioneFiltri(Integer postalCode, String propType,
        Integer minMq, Integer maxMq, Integer minYear, Integer maxYear,
        float pesoMq, float pesoRent, float pesoCrimeRate, float pesoYear, float pesoSchoolRate){
    this.pesoMq = pesoMq;
    this.pesoRent = pesoRent;
    this.pesoCrimeRate = pesoCrimeRate;
    this.pesoYear = pesoYear;
    this.immobiliIdMap.clear();
    this.immobiliList.clear();
    this.massimoBudgetMinimo=0;
    this.massimoSforamento=0;
    this.immobiliIdMap.clear();
    this.immobiliList.clear();
    this.immobiliIdMap = dao.getImmobiliFiltrati(postalCode, propType,
         minMq, maxMq, minYear, maxYear);

    for(Immobile immobile : this.immobiliIdMap.values()) {
        immobile.setPunteggio(this.calcolaPunteggioImmobile(immobile));
        System.out.println(immobile+" punteggio "+immobile.getPunteggio());
        this.immobiliList.add(immobile);
        this.massimoBudgetMinimo+=immobile.getMarketValue();
    }
    Collections.sort(this.immobiliList);
    this.massimoSforamento = (this.massimoBudgetMinimo/this.immobiliList.size())/2;
    return immobiliList;

}
```

2. Richiesta del portafoglio immobiliare: il metodo `getPortafoglioImmobiliare()` riceve come parametri in ingresso il valore booleano sulla scelta dell'utente qualità/quantità, il budget scelto dall'utente associato al valore calcolato come massimo sforamento dal budget e l'eventuale immobile necessariamente voluto dall'utente – che viene inserito nel portafoglio se presente.
Il metodo inizializza le variabili per la ricorsione (livello, lista best di immobili che rappresenterà il portafoglio personalizzato trovato, costi e punteggi) e poi, a seconda della scelta, richiama l'algoritmo ricorsivo basato sulla qualità o quello basato sulla quantità.

```
// QUESTO SARA' IL METODO CHE RICHIAMA LA RICORSIONE
public List<Immobile> getPortafoglioImmobiliare(boolean qualita,
        float budgetScelto, float maxSforoBudget,
        Immobile immobileVoluto){
    this.budgetMin = budgetScelto-maxSforoBudget;
    this.budgetMax = budgetScelto+maxSforoBudget;
    if(this.budgetMin<0)
        this.budgetMin=0;
    // variabili per la ricorsione
    this.bestPunteggioPortafoglioMedio = - 1; // cos', anche se i pesi sono a zero, nel primo
    // giro della ricorsione entra di sicuro
    this.bestCostoPortafoglio = Float.MAX_VALUE;
    List<Immobile> parziale = new ArrayList<Immobile>();
    this.bestPortafoglio = new ArrayList<Immobile>();
    float punteggioPortafoglio = 0;
    float costoPortafoglio=0;
    Integer livello=0;
    if(immobileVoluto!=null) {
        this.immobileVoluto = immobileVoluto;
        if(this.immobiliIdMap.containsValue(immobileVoluto))
            this.immobiliIdMap.remove(immobileVoluto);
        // aggiungo nel best
        this.immobileVoluto.setPunteggio(this.calcolaPunteggioImmobile(immobileVoluto));
        this.bestPortafoglio.add(this.immobileVoluto);
        parziale.add(immobileVoluto);
        // non modifico il punteggio del best perchè rischio altrimenti nell'if di non entrare
        // e quindi, finchè non lo modifica la ricorsione, mantengo -1
            punteggioPortafoglio=this.calcolaPunteggioImmobile(immobileVoluto);
        // comunque modifico il punteggio portafoglio perchè altrimenti
        // avrei una media su un valore in meno
        costoPortafoglio=immobileVoluto.getMarketValue();
        punteggioPortafoglio=immobileVoluto.getPunteggio();
        livello++;
    }
    this.startTime = System.nanoTime();
    if(qualita)
        cercaQualita(parziale, livello, punteggioPortafoglio, costoPortafoglio);
    else
        cercaQuantita(parziale, livello, costoPortafoglio);
    return this.bestPortafoglio;
}
```

5 - Descrizione degli algoritmi utilizzati

3. RICORSIONE – Ricerca del portafoglio immobiliare: come già sottolineato, la ricerca del portafoglio immobiliare può variare a seconda della scelta tra un portafoglio ottimizzato sulla base di indici di qualità di un immobile o di quantità.

- `cercaQualita()`

Il metodo `cercaQualita()` elabora un algoritmo ricorsivo basato su un sistema di punteggi assegnati ad ogni immobile e ottenuti dalla combinazione dei pesi sulle differenti caratteristiche di qualità, che vengono selezionati dall'utente in input, e i valori - ottenuti da metodi che eseguono semplici query nel DAO - della determinata caratteristica per il determinato immobile.
Tutto questo, è possibile grazie ai singoli metodi `associaValoriX()` - seguono tutti lo stesso paradigma - che associano un grado (calcolato come la normalizzazione del rapporto tra il valore della caratteristica per un determinato immobile e il massimo presente nel database) ad ogni possibile valore di una caratteristica di qualità che, nel metodo `calcolaPunteggioImmobile(Immobile imm)`, viene moltiplicato per il peso percentuale selezionato dall'utente e poi sommato ai risultati ottenuti per tutte le altre caratteristiche, ottenendo così un punteggio finale per ogni immobile calcolato come la somma dei singoli punteggi di ogni caratteristica. Le caratteristiche di qualità sono: superficie, stima rendimento affitto mensile, anno di costruzione, tasso di criminalità e livello scolastico.

```
private void associaValoriMq() {
    // chiave mq, value valore
    this.valoriMq = dao.getAllMq(); //  nel dao sono a zero
    Integer max = dao.getMaxMq();
    Integer min = dao.getMinMq();
    for(Integer mq : valoriMq.keySet()) { // ciclo sulle chiavi
        float valore = 100*((float)(max-mq)/(float)(max-min));
        this.valoriMq.put(mq, valore);
    }
}
```

```
private float calcolaPunteggioImmobile(Immobile imm)
    // calcolo per ogni caratt punti
    float punteggioTotale=0;
    float punteggioMq = 0;
    float punteggioRent = 0;
    float punteggioYear = 0;
    float punteggioCrime = 0;
    float punteggioSchool = 0;
    //mq
    if(this.pesoMq!=0.0) {
        float valoreMq = this.valoriMq.get(imm.getMq());
        punteggioMq = valoreMq*this.pesoMq;
    }
    //rent
    if(this.pesoRent!=0.0) {
        float valoreRent = this.valoriRent.get(imm.getEstimatedRent());
        punteggioRent = valoreRent*this.pesoRent;
    }
    if(this.pesoYear!=0.0) {
        float valoreYear = this.valoriYear.get(imm.getYearBuilt());
        punteggioYear = valoreYear*this.pesoYear;
    }
    //crime
    if(this.pesoCrimeRate!=0.0) {
        float valoreCrime = this.valoriCrime.get(imm.getCrimeRate());
        punteggioCrime= valoreCrime*this.pesoCrimeRate;
    }
    //school
    if(this.pesoSchoolRate!=0.0) {
        float valoreSchool= this.valoriSchool.get(imm.getSchoolRate());
        punteggioSchool = valoreSchool*this.pesoSchoolRate;
    }
    // sommo tutti i punteggi
    punteggioTotale = punteggioMq + punteggioRent + punteggioYear + punteggioCrime + punteggioSchool;
    return punteggioTotale;
}
```

5 - Descrizione degli algoritmi utilizzati

`cercaQualita()` richiede in input:

- la lista "parziale" di immobili (settata inizialmente vuota o contenente l'immobile voluto dall'utente) che verrà riempita durante la ricorsione;
- il livello della ricorsione (inizialmente a zero o 1 a seconda dell'inserimento dell'immobile voluto);
- il punteggio del portafoglio attuale, ovvero il punteggio di "parziale";

Dopo essere stato chiamato, entra all'interno di un ciclo for per l'inserimento di immobili, inserisce un immobile nella lista parziale, richiama sé stesso e rimuove lo stesso immobile dalla lista parziale. Andrà così a generare delle liste di possibili candidati come miglior portafoglio immobiliare. Il ciclo viene eseguito sulla lista di immobili precedentemente filtrati. La condizione di terminazione per l'aggiunta di immobili a "parziale" consiste nella verifica che il costo del portafoglio "parziale" abbia raggiunto il budget minimo e quindi sia dentro il range massimo e minimo. Se è così verifico se il parziale ha un punteggio migliore dell'attuale portafoglio migliore. In tal caso, l'attuale parziale sarà il best, altrimenti il metodo ha un return e torna al livello di ricorsione precedente. Alla fine, il metodo genererà una lista di immobili ottimizzata da un punto di vista qualitativo e personalizzata in base alle richieste dell'utente.

```
// RICORSIONE CHE PRIVILEGIA LA qualità DI IMMOBILI NEL PORTAFOGLIO
private void cercaQualita(List<Immobile> parziale, int livello, float punteggioParziale,
        float costoParziale) {
    // CONDIZIONE DI TERMINAZIONE
    if(costoParziale>this.budgetMin && costoParziale<this.budgetMax) {
        float punteggioPortafoglioAttualeMedio = (punteggioParziale)/(livello);

        if(punteggioPortafoglioAttualeMedio>this.bestPunteggioPortafoglioMedio) {
            // svuoto il best
            this.bestPortafoglio.removeAll(bestPortafoglio);
            // inserisco la nuova best
            for(int i=0;i<livello;i++) {
                this.bestPortafoglio.add(parziale.get(i));
            }
            this.bestCostoPortafoglio = costoParziale;
            this.bestPunteggioPortafoglioMedio = punteggioPortafoglioAttualeMedio;
        }
        return;
    }
    // INSERIMENTO
    for(Immobile imm : this.immobiliIdMap.values()) {
        if((System.nanoTime()-this.startTime)/1000000000<60){
            if(!parziale.contains(imm)) {

                parziale.add(imm);
                punteggioParziale+=imm.getPunteggio();
                costoParziale+=imm.getMarketValue();

                cercaQualita(parziale, livello+1, punteggioParziale, costoParziale);

                parziale.remove(imm);
                costoParziale = costoParziale - imm.getMarketValue();
                punteggioParziale = punteggioParziale - imm.getPunteggio();

            }
        } else {
            this.timeout = true;
            return;
        }
    }

}
```

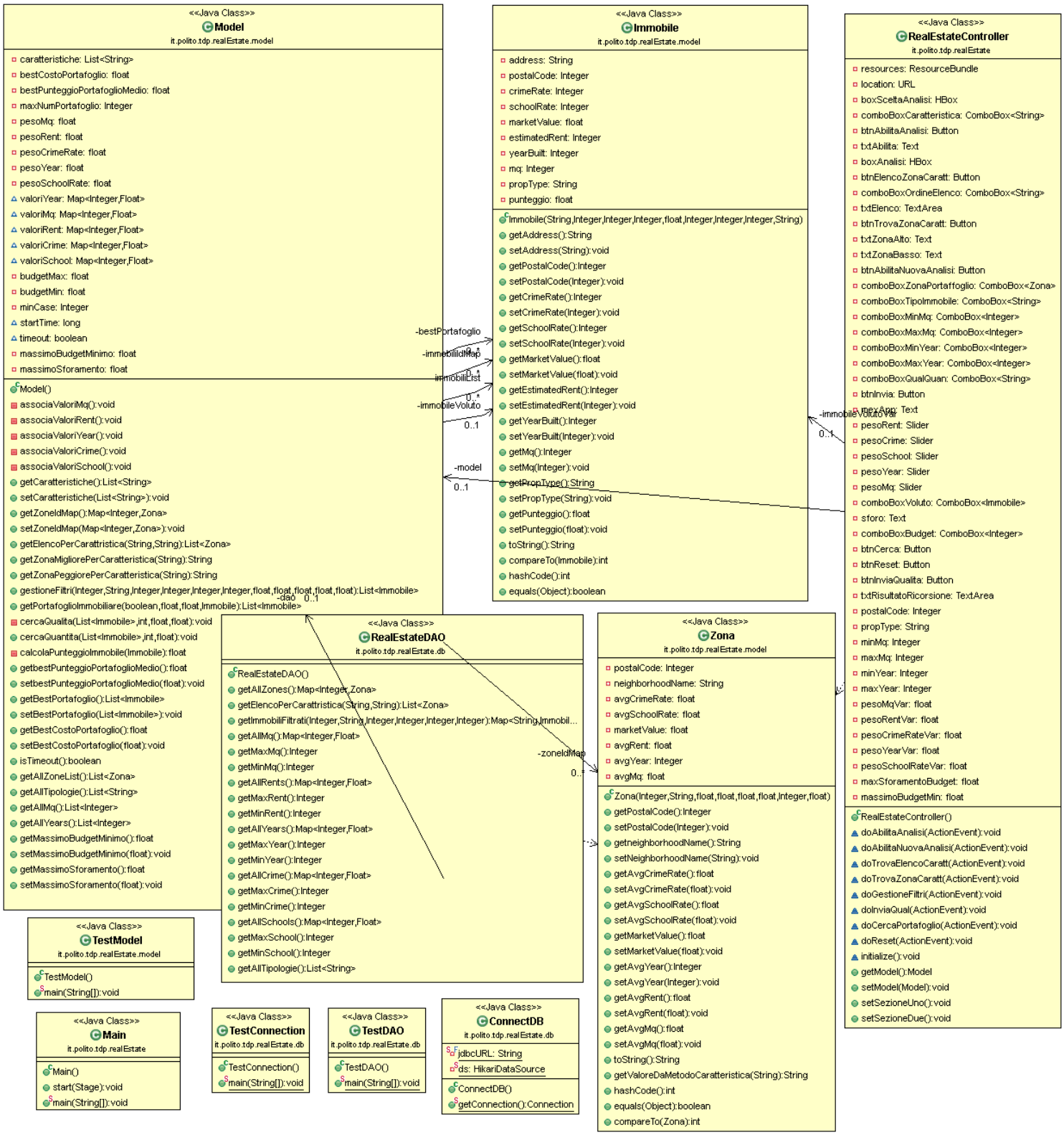
0

- `cercaQuantita()`

Il metodo `cercaQualita()` è simile al precedente ma poco più semplice. Elabora un algoritmo ricorsivo basato sulla quantità. Ovvero, a differenza del precedente, non calcola i punteggi e all'interno della condizione di terminazione verifica quale portafoglio, a parità di livello (e quindi dimensione del portafoglio candidato), sia meno costoso oppure se il parziale è più grande del best. Sostanzialmente favorisce l'inserimento di immobili meno cari, in modo da lasciare parte del budget per l'inserimento di altri e solo a parità di dimensione preferisce quello economicamente più conveniente: predilige la quantità.

```
// RICORSIONE CHE PRIVILEGIA LA quaNtità DI IMMOBILI NEL PORTAFOGLIO
// MOLTE CASE, PROBABIIOMENTE PICCOLE/SCCARSA QUALITA' A SECONDA DEI FILTRI, A COSTO INFERIORE
public void cercaQuantita(List<Immobile> parziale, int livello, float costoParziale) {

    // CONDIZIONE DI TERMINAZIONE
    //if(livello>=this.maxNumPortafoglio) condizione su limite immobili iseribili, la metto????
    if(costoParziale>this.budgetMin && costoParziale<this.budgetMax) { // termina ricerca parziale e tolgo l'ultimo

         if(costoParziale<this.bestCostoPortafoglio &&
                   (livello==this.bestPortafoglio.size())) { // a parità di size del portafoglio inserisco quello che costa meno
              // altrimenti scelgo quello che costa meno (a parità di size, numero di immobili -> vado su quantità)
              // questo mi permette comunque l'inserimento

              // svuoto il best
              this.bestPortafoglio.removeAll(bestPortafoglio);
              // inserisco la nuova best
              for(int i=0;i<livello;i++) { // inserisco tutti tranne l'ultimo perchè fuori budgetMax
                  this.bestPortafoglio.add(parziale.get(i));
              }
              this.bestCostoPortafoglio = costoParziale;

          } else if(livello>this.bestPortafoglio.size()) {
              // se non ho parità di prezzo metto quello che mi permette di avere più riempimento
              // svuoto il best
              this.bestPortafoglio.removeAll(bestPortafoglio);
              // inserisco la nuova best
              for(int i=0;i<livello;i++) { // inserisco tutti tranne l'ultimo perchè fuori budgetMax
                  this.bestPortafoglio.add(parziale.get(i));
              }
              this.bestCostoPortafoglio = costoParziale;

          }
    }

    // INSERIMENTO
    for(Immobile imm : this.immobiliIdMap.values()) {
        if((System.nanoTime()-this.startTime)/1000000000<60){
            if(!parziale.contains(imm)) {

                parziale.add(imm);

                costoParziale+=imm.getMarketValue();

                cercaQuantita(parziale, livello+1, costoParziale);

                parziale.remove(imm);
                costoParziale = costoParziale - imm.getMarketValue();
            }
        } else {
            this.timeout = true;
            return;
        }
    }
}
```

6 - Diagramma delle classi principali



7 - Interfaccia dell'applicazione e risultati sperimentali

SCREEN APP E UN PO' DI SPIEGAZIONE DI COME FUNZIONA

VEDI FOGLIO BOZZA PER VIDEO!

Spiegare l'importanza dei filtri per evitare che il programma non giri

timeout

fare prove applicazione, fare schermate e descrivere cosa faccio

1. Alcune videate dell'applicazione realizzata e link al video dimostrativo del software
2. Tabelle con risultati sperimentali ottenuti (risultati, valori, tempi, …)

8 - Valutazioni e Conclusioni

SCRIVERE MEZZA PAGINA

criticità timeout

e non poter operare su tutte le 612 istanze, l'algoritmo dovrebebe essere ulteriormente ottimizzato