Progettazione del Software

Simulazione di esame

Domenico Fabio Savo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti

Sapienza Università di Roma

Requisiti

L'applicazione da progettare riguarda la gestione di un'agenzia immobiliare che si occupa della vendita di immobili siti in città facendo da intermediaria fra proprietari ed acquirenti, che sono entrambi clienti dell'agenzia. Di ciascun immobile interessa l'indirizzo (via e numero civico), il piano, i metri quadri, la scala (opzionale), il prezzo di vendita, ed il proprietario (uno solo). Ciascun proprietario è caratterizzato dal nome, dal cognome, dalla data di nascita, dal numero di telefono e dagli immobili che possiede (almeno uno). L'agenzia classifica come acquirenti tutti i clienti che si rivolgono ad essa per acquistare un immobile. Si noti che un proprietario di un immobile può risultare anche acquirente se interessato all'acquisto di altri immobili. Degli acquirenti interessa il nome, il cognome ed il numero di telefono.

Requisiti (cont.)

Gli acquirenti visitano gli immobili su appuntamento, di cui interessa la data, l'ora, l'acquirente per il quale è stato fissato (uno solo) e l'immobile oggetto dell'appuntamento (uno solo). Ovviamente, un acquirente deve poter visitare anche più di una volta lo stesso immobile. Sugli immobili gli acquirenti possono presentare offerte di acquisto. Di ciascuna offerta interessa la data, l'importo, l'immobile a cui si riferisce e l'acquirente che la effettua. Ovviamente, un acquirente deve poter effettuare anche più di una offerta per lo stesso immobile. Delle offerte accettate interessa inoltre la data in cui si stabilisce di stipulare il rogito.

Requisiti (cont.)

L'agente immobiliare utente dell'applicazione è interessato ad effettuare alcuni controlli. In particolare:

- dato un insieme I di offerte di acquisto, restituire l'insieme degli acquirenti che hanno presentato offerte fra quelle contenute in I che risultano accettate;

- data un'offerta o per un immobile i, restituire l'insieme di appuntamenti che l'acquirente che presenta l'offerta o ha preso per visionare l'immobile i.

Requisiti (cont.)

Domanda 1. Basandosi sui requisiti riportati sopra, svolgere la fase di analisi producendo lo schema concettuale in UML per l'applicazione e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

Domanda 2. Svolgere la fase di progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

È obbligatorio solo progettare gli algoritmi e definire le responsabilità sulle associazioni.

Domanda 3. Svolgere la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

Requisiti (cont.)

È obbligatorio realizzare in Java solo i seguenti aspetti dello schema concettuale:

- le classi `Offerta` ed `Acquirente`, le associazioni che legano `Offerta` ed `Acquirente`, e le loro eventuali superclassi e/o sottoclassi. Nella realizzazione si ignorino tutte le altre associazioni a cui le classi menzionate partecipano;

- la prima operazione dello use case;

- il codice di classi `eccezione` eventualmente usate per la realizzazione dei punti precedenti.

SOLUZIONE

Fase di analisi

Diagramma delle classi

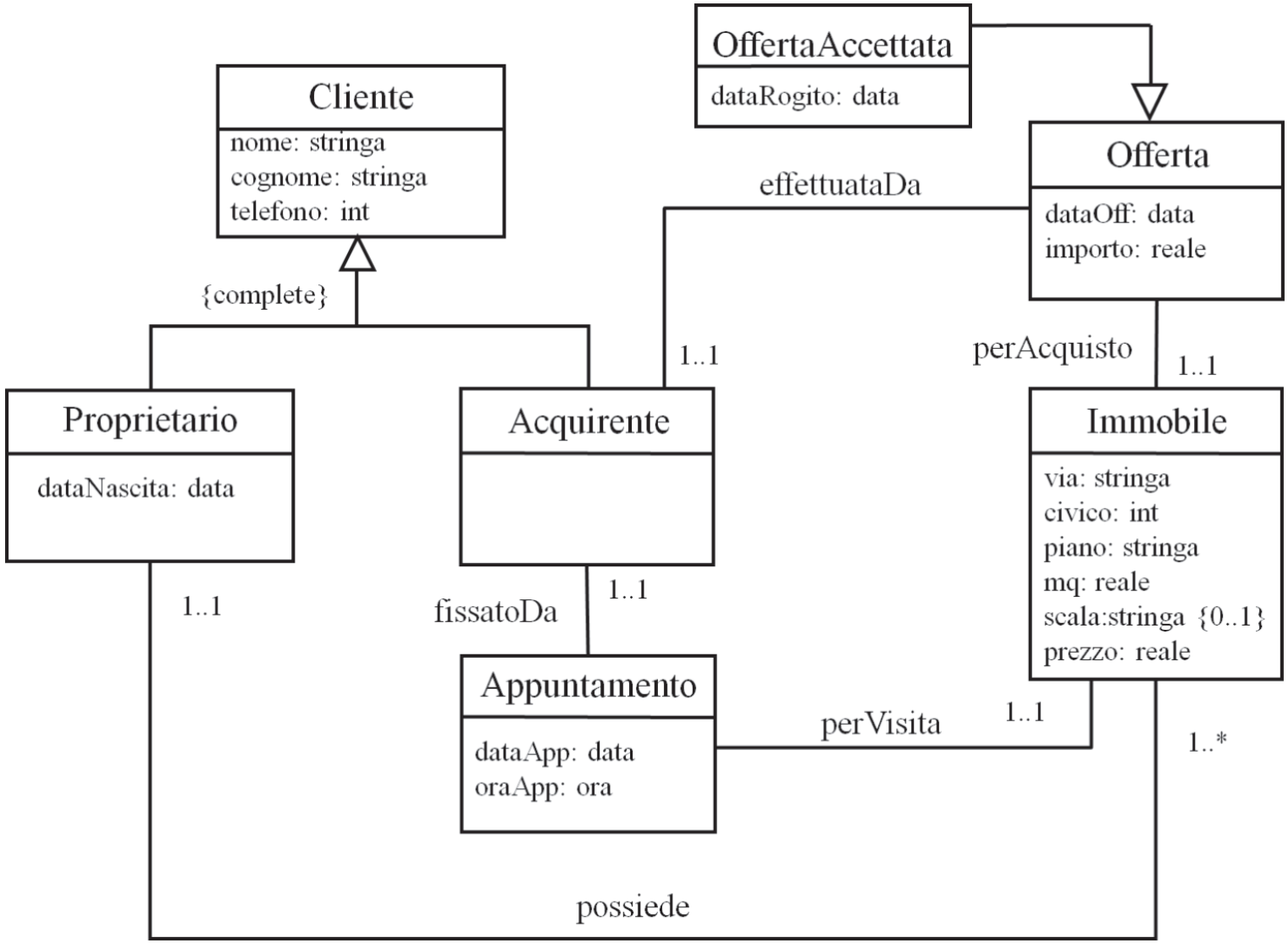
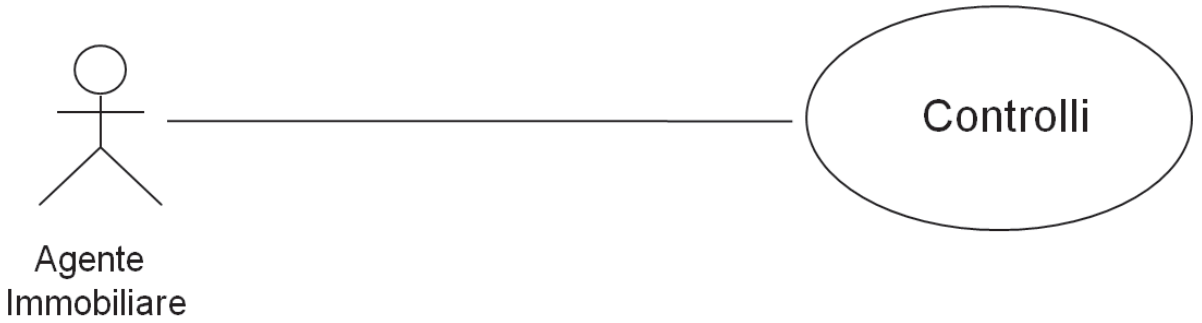


Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase Operazioni

acquirentiOfferteAccettate *(I: Insieme⟨Offerte⟩): Insieme⟨Acquirenti⟩*

pre: nessuna

post: result è l'insieme degli acquirenti che hanno presentato offerte fra quelle contenute in I che sono istanze di *OffertaAccettata*

appuntamentiImmobileConOfferta *(o: Offerta): Insieme⟨Appuntamenti⟩*

pre: nessuna

post: sia i l'immobile a cui o si riferisce, ed a l'acquirente che presenta o, result è l'insieme di appuntamenti che a ha preso per vedere i

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **acquirentiOfferteAccettate**:

```
result = new Insieme<Acquirenti>;
per ogni offerta o contenuta in I
 se o appartiene a OffertaAccettata
  sia l il link di tipo effettuataDa in cui o è coinvolta
  aggiungi l.Acquirente a result;
return result;
```

- Per l'operazione **appuntamentiImmobileConOfferta**:

```
result = new Insieme<Appuntamenti>;
sia l il link di tipo perAcquisto in cui o è coinvolta
Immobile i = l.Immobile;
sia ll il link di tipo effettuataDa in cui o è coinvolta
Acquirente a = ll.Acquirente;
per ogni link k di tipo fissatoDa in cui a è coinvolto
  sia kk il link di tipo perVisita in cui k.Appuntamento è coinvolto
  se kk.Immobile=i
    aggiungi k.Appuntamento a result;
return result;
```

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

1. i requisiti,
2. la specifica degli algoritmi per le operazioni di classe e use-case,
3. i vincoli di molteplicità nel diagramma delle classi.

Associazione	Classe	ha resp.
possiede	*Proprietario*	SÌ[3]
	Immobile	SÌ[3]
perVisita	*Appuntamento*	SÌ[2,3]
	Immobile	NO
perAcquisto	*Offerta*	SÌ[2,3]
	Immobile	NO
effettuataDa	*Offerta*	SÌ[2,3]
	Acquirente	NO
fissatoDa	*Appuntamento*	SÌ[3]
	Acquirente	SÌ[2]

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa:

- dei vincoli di molteplicità 0..* delle associazioni,
- delle variabili locali necessarie per vari algoritmi.

Per fare ciò, utilizzeremo l'interfaccia `Set`, e la classe `HashSet` del collection framework di Java 1.5.

Corrispondenza fra tipi UML e Java

Riassumiamo le nostre scelte nella seguente tabella di corrispondenza dei tipi UML.

Tipo UML	Rappresentazione in Java
intero	`int`
reale	`float`
Stringa	`String`
Insieme	`HashSet`
data	`Data`
ora	`Ora`

Si assume di aver definito dei tipi di dato Java `Data` ed `Ora`.

Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*.

Classe UML	**Proprietà immutabile**
Cliente	nome
Cliente	cognome
Offerta	dataOff
Offerta	importo

	Proprietà	
Classe UML	**nota alla nascita**	**non nota alla nascita**

Si ricorda che di default

- tutte le proprietà sono **mutabili**;
- le proprietà singole sono **note alla nascita**
- le proprietà multiple **non sono note alla nascita**.

Altre considerazioni

Valori alla nascita: Non sembra ragionevole assumere che per qualche proprietà esistano valori di default validi per tutti gli oggetti.

Fase di realizzazione

Considerazioni iniziali

Il compito richiede di realizzare solo quanto segue:

1. la classe UML *Offerta*;
2. la classe UML *OffertaAccettata*;
3. la classe UML *Acquirente*;
4. la classe UML *Cliente*;
5. l'associazione UML *effettuataDa* su cui ha responsabilità *Offerta* ma non *Acquirente*;
6. la prima operazione dello use case;
7. la classe Java `EccezioneMolteplicita`.

Procederemo quindi ignorando tutte le altre classi, entità e casi d'uso che non sono fra quelli menzionati sopra.

Struttura dei file e dei package

```
\---AppAgenziaImm
    |   Appuntamento.java
    |   Immobile.java
    |   EccezionePrecondizioni.java
    |   EccezioneMolteplicita.java
    |   Operazioni.java
    |
    +---Cliente
    |       Cliente.java
    |
    +---Acquirente
    |       Acquirente.java
    |
    +---Proprietario
    |       Proprietario.java
    |
    +---Offerta
    |       Offerta.java
    |
    |---OffertaAccettata
    |       OffertaAccettata.java
    \---TipiDato
            Data.java
            Ora.java
```

La classe Java Cliente

```
package AppAgenziaImm.Cliente;
import AppAgeziaImm.*;
import java.util.*;

public abstract class Cliente {
  protected final String nome;
  protected final String cognome;
  protected String telefono;

  public Cliente(String n, String c) {
    nome = n;
    cognome = c;
  }
  public String getNome() { return nome; }
  public String getCognome() { return cognome; }
  public void setTelefono(String t) { telefono = t; }
  public String getTelefono() { return telefono; }
}
```

La classe Java Acquirente

```
package AppAgenziaImm.Acquirente;
import AppAgenziaImm.Cliente.*;
import AppAgenziaImm.*;
import java.util.*;

public class Acquirente extends Cliente {
  public Acquirente(String n, String c) {
    super(n,c);
  }
}
```

La classe Java Offerta

```
package AppAgenziaImm.Offerta;
import AppAgenziaImm.*;
import AppAgenziaImm.TipiDato.*;
import java.util.*;

public class Offerta {
  protected final Data dataOff;
  protected final float importo;
  protected Acquirente offerente;
  protected static final int MOLT_MIN = 1;

  public Offerta(Data d, float i) {
    dataOff = d;
    importo = i;   }

  public Data getData() { return data; }
  public float getImporto() { return importo; }
  public void setOfferente(Acquirente a) {  offerente = a;   }

  public Acquirente getOfferente() throws EccezioneMolteplicita {
   if (offerente == NULL)
      throw new EccezioneMolteplicita("molteplicita' minima violata");
   return offerente;   }
}
```

La classe Java OffertaAccettata

```
package AppAgenziaImm.OffertaAccettata;
import AppAgenziaImm.Offerta.*;
import AppAgenziaImm.*;
import java.util.*;
import AppAgenziaImm.TipiDato.*;

public class OffertaAccettata extends Offerta {
  protected Data dataRogito;

  public OffertaAccettata(Data d, float i) {
    super(d,i);
  }
  public void setDataRogito(Data d){
   dataRogito =d;
  }
  public Data getDataRogito(){
   return dataRogito;
  }
}
```

Realizzazione in Java degli use case

```
package AppAgenziaImm;
import AppAgenziaImm.Offerta.*;
import AppAgenziaImm.OffertaAccettata.*;
import AppAgenziaImm.Acquirente.*;
import AppAgenziaImm.Proprietario.*;
import java.util.*;

public final class Operazioni {
  private Operazioni() { }

  public static Set<Acquirenti> acquirentiOfferteAccettate(Set<Offerte> I)
              throws EccezioneMolteplicita{
    HashSet<Acquirenti> result = new HashSet<Acquirenti>();
    Iterator<Offerte> it = I.iterator();
    while(it.hasNext()) {
      Offerta o = it.next();
      if (o instanceof OffertaAccettata)
        result.add(o.getOfferente);
      }
    return result;
  }

//...
}
```

La classe EccezioneMolteplicita

```
package AppAgenziaImm;

public class EccezioneMolteplicita extends Exception {
  private String messaggio;
  public EccezioneMolteplicita(String m) {
    messaggio = m;
  }
  public EccezioneMolteplicita() {
    messaggio = "Si e' verificata una violazione delle molteplicità";
  }
  public String toString() {
    return messaggio;
  }
}
```