

Compito d'esame del 12 settembre 2005

SOLUZIONE

Requisiti

Requisiti. L'applicazione di interesse riguarda la progettazione di appartamenti. Di ogni appartamento interessano l'indirizzo, una descrizione testuale, i locali che lo compongono e come questi sono connessi tra loro (si noti che se il locale A è connesso con il locale B, allora anche B è connesso con A). Di ciascun locale interessano i metri quadri ed una descrizione testuale. I locali sono suddivisi in vani generici di cui interessa il tipo ("singolo", "doppio", "corridoio"), bagni di cui interessa il numero di punti acqua, e cucine di cui interessa il numero di punti gas.

Un appartamento è inizialmente in preparazione, alla fine dei lavori diviene pronto per la consegna, a questo punto possono essere richiesti ulteriori lavori (e quindi torna ad essere in preparazione) o può essere consegnato. Solo quando un appartamento è in preparazione si possono aggiungere ed eliminare locali da esso.

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 2

Requisiti (cont.)

L'utente dell'applicazione vuole poter effettuare dei controlli sugli appartamenti. A questo scopo, si faccia riferimento ad uno use case che prevede che, dato un appartamento, si possa:

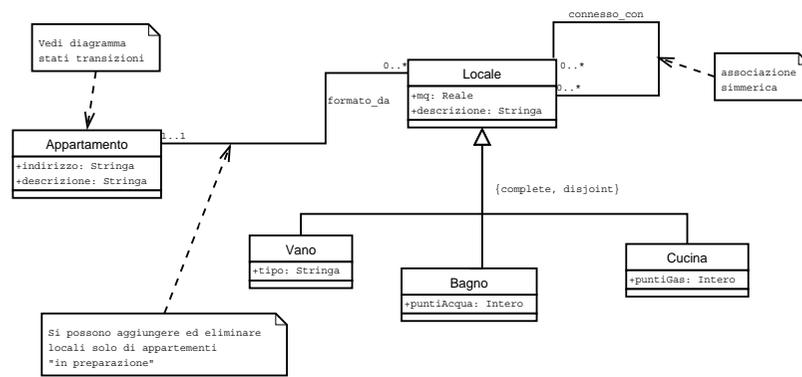
- verificare che sia presente almeno un bagno ed una cucina tra i locali di cui è composto;
- restituire l'insieme di tutti i vani generici che lo compongono.

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 3

Fase di analisi

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 4

Diagramma delle classi



Commento sul diagramma delle classi

Nelle successive fasi di progetto realizzazione dovremo tenere presente che solo se un appartamento è in preparazione si possono aggiungere ed eliminare locali dallo stesso. Quindi dovremo permettere ai clienti della classe *Appartamento* di tenere traccia dello stato in cui si trovano le sue istanze.

Diagramma degli stati e delle transizioni classe Appartamento

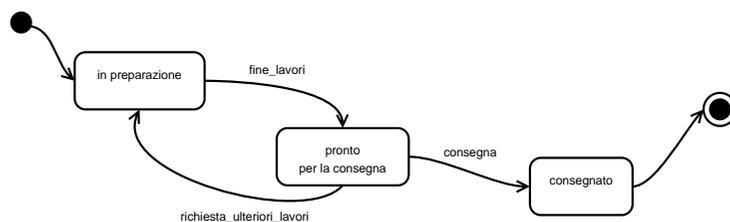
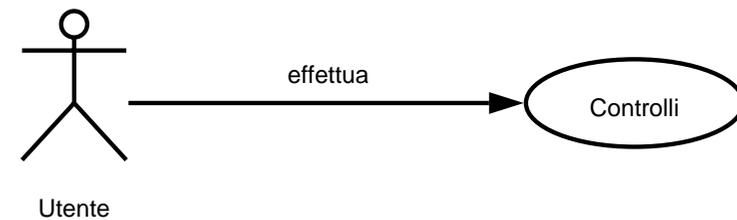


Diagramma degli use case



Specifica dello use case

InizioSpecificaUseCase Controlli

verificaBagnoCucina (*a: Appartamento*): *Booleano*

pre: true // nessuna preconditione

post:

$result = \exists b, c (b \in Bagno \wedge c \in Cucina \wedge \langle a, b \rangle \in formato_da \wedge \langle a, c \rangle \in formato_da)$

insiemeVani (*a: Appartamento*): *Insieme(Locale)*

pre: true // nessuna preconditione

post: $result = \{v \in Vano \mid \langle a, v \rangle \in formato_da\}$

FineSpecifica

Fase di progetto

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **verificaBagnoCucina**:

```
bool haBagno = false;
bool haCucina = false;
per ogni link l formato_da di a
  if l.locale è istanza di Bagno
    haBagno == true;
  else if l.locale è istanza di Cucina
    haCucina == true;
return haBagno AND haCucina;
```

Algoritmi per le operazioni degli use case (cont.)

- Per l'operazione **insiemeVani**:

```
Insieme(vani) result = new Insieme(Vani);
per ogni link l di formato_da di a
  se l.Locale è istanza Vani
    aggiungi l.locale 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.
<i>formato_da</i>	<i>Appartamento</i> <i>Locale</i>	$S^{1,2}$ S^3
<i>connesso_con</i>	<i>Locale</i> <i>Locale</i>	S^1 S^1

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 gli algoritmi.

Per fare ciò, utilizzeremo Set e 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
Booleano	boolean
Intero	int
Stringa	String
Insieme	HashSet

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
<i>Appartamento</i>	<i>indirizzo</i>
<i>Locale</i>	<i>metriQuadri</i>

Classe UML	Proprietà	
	nota alla nascita	non nota alla nascita

Altre considerazioni

Sequenza di nascita degli oggetti: Non è necessario assumere un particolare ordine nella nascita degli oggetti.

Valori alla nascita: Non esistono valori di default validi per tutti gli oggetti di ciascuna classe.

Rappresentazione degli stati in Java

Per la classe UML *Appartamento*, ci dobbiamo occupare della rappresentazione in Java del diagramma degli stati e delle transizioni.

Scegliamo di rappresentare gli stati mediante una variabile `int`, secondo la seguente tabella.

Stato	Rappresentazione in Java	
	tipo var.	<code>int</code>
	nome var.	<code>stato</code>
in preparazione	valore	1
pronto per la consegna	valore	2
consegnato	valore	3

API delle classi Java progettate

A titolo di esempio, viene fornita la API della classe *Appartamento*:

```
public class Appartamento {
// COSTRUTTORE
    public Appartamento(String indirizzo)
// GESTIONE ATTRIBUTI
    public String getIndirizzo()
    public String getDescrizione()
    public void setDescrizione(String d)
// GESTIONE ASSOCIAZIONI
// - formatoDa
    public void inserisciLinkFormatoDa(AssociazioneFormatoDa a)
    public void eliminaLinkFormatoDa(AssociazioneFormatoDa a)
    public Set<TipoLinkFormatoDa> getLinkFormatoDa()
// GESTIONE STATI E TRANSIZIONI
    public bool estInPreparazione()
    public void fineLavori()
    public void richiestaUlterioriLavori()
    public void consegna()
}
```

Fase di realizzazione

Struttura dei file e dei package

```
+---appartamenti
|   Appartamento.java
|   Locale.java
|   AssociazioneFormatoDa.java
|   AssociazioneConnessoCon.java
|   TipoLinkFormatoDa.java
|   TipoLinkConnessoCon.java
|   EccezionePrecondizioni.java
|   EccezioneCardMin.java
|   Controlli.java
|
+---vano
|   Vano.java
|
+---bagno
|   Bagno.java
|
\---cucina
    Cucina.java
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 21

La classe Java Appartamento

```
// File appartamenti/Appartamento.java
package appartamenti;

import java.util.*;

public class Appartamento {
    private final String indirizzo;
    private String descrizione;
    private HashSet<TipoLinkFormatoDa> insiemeFormatoDa;
    private int stato;
    private static final int INPREPARAZIONE = 1;
    private static final int PRONTO = 2;
    private static final int CONSEGNATO = 3;
    public Appartamento(String i, String d) {
        indirizzo = i;
        descrizione = d;
        insiemeFormatoDa = new HashSet<TipoLinkFormatoDa>();
        stato = INPREPARAZIONE;
    }
    public String getIndirizzo() { return indirizzo; }
    public String getDescrizione() { return descrizione; }
    public void setDescription(String d) { descrizione = d; }
    public void inserisciLinkFormatoDa(AssociazioneFormatoDa a) {
        if (a != null) insiemeFormatoDa.add(a.getLink());
    }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 22

```
}
public void eliminaLinkFormatoDa(AssociazioneFormatoDa a) {
    if (a != null) insiemeFormatoDa.remove(a.getLink());
}
public Set<TipoLinkFormatoDa> getLinkFormatoDa() {
    return (HashSet<TipoLinkFormatoDa>)insiemeFormatoDa.clone();
}
public boolean estInPreparazione() {
    return stato == INPREPARAZIONE;
}
public void fineLavori() {
    if (stato == INPREPARAZIONE) stato = PRONTO;
}
public void consegna() {
    if (stato == PRONTO) stato = CONSEGNATO;
}
public void richiestaUlterioriLavori() {
    if (stato == PRONTO) stato = INPREPARAZIONE;
}
}
```

La classe Java Locale

```
// File appartamenti/Locale.java
package appartamenti;

import java.util.*;

public abstract class Locale {
    private double mq;
    private String descrizione;
    private TipoLinkFormatoDa linkFormatoDa;
    private HashSet<TipoLinkConnessoCon> insiemePrimo;
    private HashSet<TipoLinkConnessoCon> insiemeSecondo;
    public static final int MIN_LINK_FORMATODA = 1;

    public Locale(double m, String d) {
        mq = m;
        descrizione = d;
        insiemePrimo = new HashSet<TipoLinkConnessoCon>();
        insiemeSecondo = new HashSet<TipoLinkConnessoCon>();
    }

    public String getDescrizione() { return descrizione; }
    public void setDescription(String d) { descrizione = d; }
    public int quantiFormatoDa() {
        if (linkFormatoDa == null)

```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 23

```

        return 0;
    else return 1;
}

public void inserisciLinkFormatoDa(AssociazioneFormatoDa a) {
    if (a != null) linkFormatoDa = a.getLink();
}

public void eliminaLinkFormatoDa(AssociazioneFormatoDa a) {
    if (a != null) linkFormatoDa = null;
}

public TipoLinkFormatoDa getLinkFormatoDa() throws EccezioneCardMin {
    if (linkFormatoDa == null)
        throw new EccezioneCardMin("Cardinalita' minima violata");
    else
        return linkFormatoDa;
}

public boolean haLinkFormatoDa() {
    return linkFormatoDa != null;
}

public void inserisciLinkPrimo(AssociazioneConnessoCon a) {
    if (a != null) insiemePrimo.add(a.getLink());
}

public void eliminaLinkPrimo(AssociazioneConnessoCon a) {
    if (a != null) insiemePrimo.remove(a.getLink());
}

public void inserisciLinkSecondo(AssociazioneConnessoCon a) {

```

```

        if (a != null) insiemeSecondo.add(a.getLink());
    }
    public void eliminaLinkSecondo(AssociazioneConnessoCon a) {
        if (a != null) insiemeSecondo.remove(a.getLink());
    }
    public Set<TipoLinkConnessoCon> getLinkConnessoCon() {
        // ConnessoCon è simmetrica
        HashSet<TipoLinkConnessoCon> ris =
            (HashSet<TipoLinkConnessoCon>) insiemePrimo.clone();
        Iterator <TipoLinkConnessoCon> it = insiemeSecondo.iterator();
        while(it.hasNext())
            ris.add(it.next());
        return ris;
    }
}

```

La classe Java AssociazioneFormatoDa

// File appartamenti/AssociazioneOrdine.java

```

package appartamenti;

public final class AssociazioneFormatoDa {
    private AssociazioneFormatoDa(TipoLinkFormatoDa t) { link = t; }
    private TipoLinkFormatoDa link;
    public TipoLinkFormatoDa getLink() { return link; }
    public static void inserisci(TipoLinkFormatoDa y) {
        if (y != null &&
            y.getAppartamento().estInPreparazione() && //NB stato
            !(y.getLocale().haLinkFormatoDa()) ) { //NB molteplicita
            AssociazioneFormatoDa k = new AssociazioneFormatoDa(y);
            k.link.getAppartamento().inserisciLinkFormatoDa(k);
            k.link.getLocale().inserisciLinkFormatoDa(k);
        }
    }
    public static void elimina(TipoLinkFormatoDa y) throws EccezioneCardMin{
        if (y != null &&
            y.getAppartamento().estInPreparazione() && //NB stato
            y.getLocale().getLinkFormatoDa().equals(y)) { //NB levo link giusto
            AssociazioneFormatoDa k = new AssociazioneFormatoDa(y);
            k.link.getAppartamento().eliminaLinkFormatoDa(k);
            k.link.getLocale().eliminaLinkFormatoDa(k);
        }
    }
}

```

```

    }
}
}

```

La classe Java AssociazioneConnessoCon

```
// File appartamenti/AssociazioneConnessoCon.java
package appartamenti;

//NOTA: questa associazione deve essere simmetrica

public final class AssociazioneConnessoCon {
    private AssociazioneConnessoCon(TipoLinkConnessoCon t) { link = t; }
    private TipoLinkConnessoCon link;
    public TipoLinkConnessoCon getLink() { return link; }
    public static void inserisci(TipoLinkConnessoCon y) {
        if (y != null) {
            AssociazioneConnessoCon k = new AssociazioneConnessoCon(y);
            k.link.getPrimo().inserisciLinkPrimo(k);
            k.link.getSecondo().inserisciLinkSecondo(k);
            k.link.getPrimo().inserisciLinkSecondo(k); //NOTA
            k.link.getSecondo().inserisciLinkPrimo(k); //manteniamo la simmetria
        }
    }
    public static void elimina(TipoLinkConnessoCon y) {
        if (y != null) {
            AssociazioneConnessoCon k = new AssociazioneConnessoCon(y);
            k.link.getPrimo().eliminaLinkPrimo(k);
            k.link.getSecondo().eliminaLinkSecondo(k);
            k.link.getPrimo().eliminaLinkSecondo(k); //NOTA
        }
    }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 25

```
        k.link.getSecondo().eliminaLinkPrimo(k); //manteniamo la simmetria
    }
}
```

La classe Java TipoLinkFormatoDa

```
// File appartamenti/TipoLinkFormatoDa.java
package appartamenti;

public class TipoLinkFormatoDa {
    private final Appartamento lAppartamento;
    private final Locale ilLocale;
    public TipoLinkFormatoDa(Appartamento a, Locale l)
        throws EccezionePrecondizioni {
        if (a == null || l == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        lAppartamento = a; ilLocale = l;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkFormatoDa t = (TipoLinkFormatoDa)o;
            return t.lAppartamento == lAppartamento &&
                t.ilLocale == ilLocale;
        }
        else return false;
    }
    public hashCode() {
        return lAppartamento.hashCode() + ilLocale.hashCode();
    }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 26

```
public Appartamento getAppartamento() { return lAppartamento; }
public Locale getLocale() { return ilLocale; }
}
```

La classe Java TipoLinkConnessoCon

```
// File appartamenti/TipoLinkConnessoCon.java
package appartamenti;

public class TipoLinkConnessoCon {
    private final Locale primo;
    private final Locale secondo;
    public TipoLinkConnessoCon(Locale l1, Locale l2)
        throws EccezionePrecondizioni {
        if (l1 == null || l2 == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        primo = l1; secondo = l2;
    }
    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkConnessoCon t = (TipoLinkConnessoCon)o;
            return t.secondo == secondo && t.primo == primo;
        }
        else return false;
    }
    public hashCode() {
        return primo.hashCode() + secondo.hashCode();
    }
    public Locale getPrimo() { return primo; }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 27

```
    public Locale getSecondo() { return secondo; }
}
```

La classe Java Vano

```
// File appartamenti/vano/Vano.java
package appartamenti.vano;
import appartamenti.*;

public class Vano extends Locale {
    private String tipo;
    public Vano(double mq, String d, String t) {
        super(mq, d);
        tipo = t;
    }
    public String getTipo() { return tipo; }
    public void setTipo(String t) { tipo = t; }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 28

La classe Java Bagno

```
// File appartamenti/bagno/Bagno.java
package appartamenti.bagno;
import appartamenti.*;

public class Bagno extends Locale {
    private int puntiAcqua;
    public Bagno(double mq, String d, int p) {
        super(mq, d);
        puntiAcqua = p;
    }
    public int getPuntiAcqua() { return puntiAcqua; }
    public void setPuntiAcqua(int p) { puntiAcqua = p; }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 29

La classe Java Cucina

```
// File appartamenti/cucina/Cucina.java
package appartamenti;
import appartamenti.*;

public class Cucina extends Locale {
    private int puntiGas;
    public Cucina(double mq, String d, int p) {
        super(mq,d);
        puntiGas = p;
    }
    public int getPuntiGas() { return puntiGas; }
    public void setPuntiGas(int p) { puntiGas = p; }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 30

Le classi Java per le eccezioni

```
// File appartamenti/EccezionePrecondizioni.java
package appartamenti;

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

// File appartamenti/EccezioneCardMin.java
package appartamenti;

public class EccezioneCardMin extends Exception {
    private String messaggio;
    public EccezioneCardMin(String m) {
        messaggio = m;
    }
    public String toString() {
        return messaggio;
    }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 31

Realizzazione in Java dello use case

```
// File appartamenti/Controlli.java
package appartamenti;

import java.util.*;
import appartamenti.vano.*;
import appartamenti.bagno.*;
import appartamenti.cucina.*;

public final class Controlli {
    private Controlli(){}

    public static boolean verificaBagnoCucina(Appartamento a) {
        boolean haBagno = false;
        boolean haCucina = false;
        Set<TipoLinkFormatoDa> locali = a.getLinkFormatoDa();
        Iterator <TipoLinkFormatoDa>it = locali.iterator();
        while(it.hasNext() && (!haBagno || !haCucina)) {
            Locale l = it.next().getLocale();
            if (l instanceof Bagno) haBagno = true;
            else if (l instanceof Cucina) haCucina = true;
        }
        return haBagno && haCucina;
    }
}
```

U. "La Sapienza". Fac.Ingegneria. Progettazione del Software I. Soluzione compito 2005-09-12 32

```
public static Set<Vano> insiemeVani(Appartamento a) {
    Set<Vano> result = new HashSet<Vano>();
    Set<TipoLinkFormatoDa> locali = a.getLinkFormatoDa();
    Iterator <TipoLinkFormatoDa> it = locali.iterator();
    while(it.hasNext()) {
        Locale l = it.next().getLocale();
        if (l instanceof Vano) result.add((Vano)l);
    }
    return result;
}
```