

Compito d'esame del 20 aprile 2007

## SOLUZIONE

### Requisiti (cont.)

Una costruzione può essere montata e poi esposta. Sia quando è montata che esposta può essere smontata. Inizialmente la costruzione è ovviamente smontata.

Il fruitore della applicazione è interessato ad effettuare diverse operazioni, in particolare:

- data una scatola di montaggio  $s$  ed una costruzione  $c$ , verificare se  $s$  contiene tutti i tipi di mattoncini richiesti per  $c$  in quantità sufficienti;
- dato un tipo di mattoncino elettrificato  $me$ , restituire l'insieme  $C$  delle costruzioni in cui  $me$  è richiesto.

### Requisiti

L'applicazione da progettare riguarda la gestione di costruzioni per bambini. Sono di interesse le scatole di montaggio, ciascuna caratterizzata da una descrizione testuale e dai tipi di mattoncini che essa contiene (almeno uno) con le rispettive quantità. Alcune scatole sono speciali e sono caratterizzate da un livello di difficoltà (un intero). Tali scatole speciali contengono almeno un tipo di mattoncini elettrificati (per esempio motorini elettrici, dispositivi luminosi, ecc.). I tipi di mattoncini sono caratterizzati dalle dimensioni (rappresentate da una stringa) e dal colore (una stringa). I tipi di mattoncini elettrificati sono inoltre caratterizzati da una specifica elettrica (una stringa). Oltre alle scatole di montaggio sono di interesse le costruzioni realizzabili con i mattoncini. Ogni costruzione è caratterizzata dalle istruzioni di montaggio (una stringa) e dai tipi di mattoncini richiesti, con le rispettive quantità. Alcune costruzioni sono speciali in quanto servono a illustrare un fenomeno elettrico, queste sono caratterizzate da una descrizione testuale del fenomeno (una stringa) e contengono esattamente un tipo di mattoncini elettrificati (in quantità arbitraria).

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 2

### Requisiti (cont.)

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare 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.** Effettuare 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.** Effettuare 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:

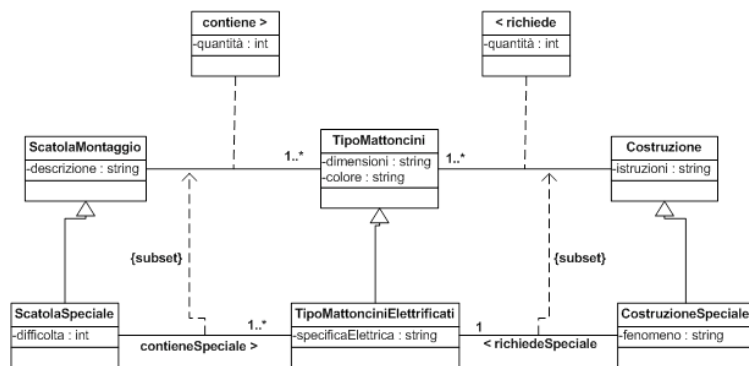
- le classi *Costruzione* e *TipoMattoncino*, e le eventuali associazioni tra di esse;
- il primo use case.

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 5

## Fase di analisi

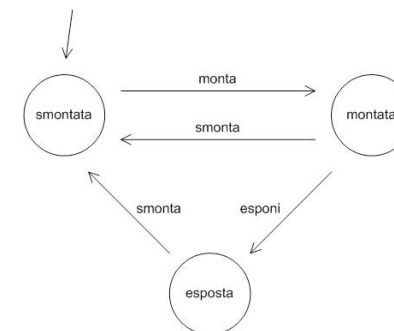
U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 6

## Diagramma delle classi



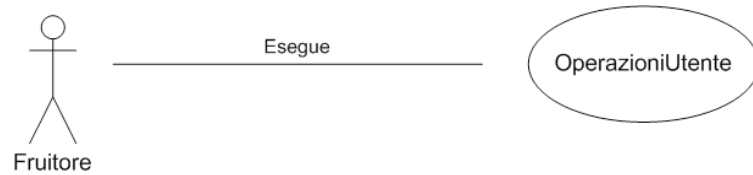
U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 7

## Diagramma degli stati e delle transizioni della classe Costruzione



U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 8

## Diagramma degli use case



U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 9

## Specifica dello use case

### InizioSpecificaUseCase OperazioniUtente

**mattonciniSuff** (*s*: ScatolaMontaggio, *c*: Costruzione): booleano

pre: true

post: Definiamo gli insiemi:

$$M_c \doteq \{m \mid m \in TipoMattoncini \wedge \langle c, m \rangle \in richiede\}$$

$$M_s \doteq \{m \mid m \in TipoMattoncini \wedge \langle s, m \rangle \in contiene\}$$

*result* = true se e solo se:

- $(M_c \subseteq M_s)$  e
- $\forall m \in M_c \rightarrow quantita(\langle s, m \rangle) \geq quantita(\langle c, m \rangle)$

...

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 10

## Specifica dello use case (cont.)

...

**costruzioniElett** (*me*: TipoMattonciniElettrificati):

*Insieme*< Costruzione >

pre: true

post: *result* =  $\{c \mid c \in Costruzione \wedge \langle c, me \rangle \in richiede\}$

### FineSpecifica

## Fase di progetto

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 11

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 12

## Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **mattonciniSuff**(*s: ScatolaMontaggio, c: Costruzione*): *booleano*

```
per ogni link l di tipo richiede in cui c è coinvolto {
    se (non esiste un link l' di tipo contiene in cui s è coinvolto tale
        che l.TipoMattoncini == l'.TipoMattoncini and l.quantita <= l'.quantita)
        return false;
}
return true;
```

- Per l'operazione **costruzioniElett**(*me: TipoMattonciniElettrificati*): *Insieme<Costruzione>*

```
result = new Insieme<Costruzione>;
per ogni link l di tipo richiede in cui me è coinvolto{
    result = result union l.Costruzione;
}
return result;
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 13

## Strutture di dati

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

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

Per fare ciò, utilizzeremo le classi del collection framework di Java 1.5: *Set*, *HashSet*.

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 15

## 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>contiene</i>	<i>ScatolaMontaggio</i> <i>TipoMattoncini</i>	$\bar{S}I^{2,3}$ NO
<i>contieneSpeciale</i>	<i>ScatolaSpeciale</i> <i>TipoMattonciniElettrificati</i>	$\bar{S}I^3$ NO
<i>richiede</i>	<i>Costruzione</i> <i>TipoMattoncini</i>	$\bar{S}I^{2,3}$ $\bar{S}I^2$
<i>richiedeSpeciale</i>	<i>CostruzioneSpeciale</i> <i>TipoMattonciniElettrificati</i>	$\bar{S}I^3$ NO

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 14

## Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
stringa	<i>String</i>
booleano	<i>boolean</i>
Insieme	<i>HashSet</i>

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 16

## Tablelle 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>ScatolaMontaggio</i>	<i>nome</i>
<i>ScatolaSpeciale</i>	<i>difficolta</i>
<i>TipoMattoncini</i>	<i>dimensioni</i>
	<i>colore</i>
<i>TipoMattonciniElettrificati</i>	<i>specificaElettrica</i>
<i>Costruzione</i>	<i>istruzioni</i>
<i>CostruzioneSpeciale</i>	<i>fenomeno</i>

Classe UML	Proprietà	
	nota alla nascita	non nota alla nascita
<i>CostruzioneSpeciale</i>	–	<i>richiedeSpeciale</i>

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 17

## Rappresentazione degli stati in Java

Per la classe UML *Costruzione*, 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>
<code>smontata</code>	<code>valore</code>	<code>1</code>
<code>montata</code>	<code>valore</code>	<code>2</code>
<code>esposta</code>	<code>valore</code>	<code>3</code>

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 19

## Altre considerazioni

**Sequenza di nascita degli oggetti:** Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

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

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 18

## API delle classi Java progettate

Omesse per brevità (si faccia riferimento al codice Java).

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 20

La traccia ci richiede di realizzare:

1. Le classi Costruzione e TipoMattoncini.
2. l'associazione UML *richiede* con responsabilità doppia e con vincoli di molteplicità 1..\* (molteplicità minima diversa da zero) e 0..\*;

Nel seguito verranno realizzate tutte le classi e gli use case individuati in fase di analisi.

## Fase di realizzazione

## Struttura dei file e dei package

```
+---AppCostruzioni
|   TipoLinkRichiede.java
|   AssociazioneRichiede.java
|   TipoLinkContiene.java
|   TipoLinkContieneSpeciale.java
|   OperazioniUtente.java
|   EccezioneSubset.java
|   EccezioneMolteplicita.java
|   EccezionePrecondizioni.java
|
+---Costruzione
|   Costruzione.java
|
+---CostruzioneSpeciale
|   CostruzioneSpeciale.java
|
+---TipoMattoncini
|   TipoMattoncini.java
|
+---TipoMattonciniElettrificati
|   TipoMattonciniElettrificati.java
|
+---ScatolaMontaggio
|   ScatolaMontaggio.java
|
\---ScatolaSpeciale
    ScatolaSpeciale.java
```

## La classe Java Costruzione

```
// File AppCostruzioni/Costruzione/Costruzione.java
package AppCostruzioni.Costruzione;

import AppCostruzioni.*;

import java.util.*;

public class Costruzione {
    private final int MOLT_MIN = 1;
    private final int SMONTATA = 1, MONTATA = 2, ESPOSTA = 3;

    private String istruzioni;
    private HashSet<TipoLinkRichiede> richiede;
    private int statoCorrente;

    public Costruzione(String istruzioni){
        this.istruzioni = istruzioni;
        statoCorrente = SMONTATA;
        richiede = new HashSet<TipoLinkRichiede>();
    }

    public String getIstruzioni(){
        return istruzioni;
    }
}
```

```

public void inserisciLinkRichiede (AssociazioneRichiede a){
    if (a != null)
        richiede.add(a.getLink());
}

public void eliminaLinkRichiede(AssociazioneRichiede a){
    if (a != null)
        richiede.remove(a.getLink());
}

public Set<TipoLinkRichiede> getLinkRichiede() throws EccezioneMolteplicita{
    if (richiede.size() < MOLT_MIN)
        throw new EccezioneMolteplicita("Molteplicità minima violata");
    return (HashSet<TipoLinkRichiede>) richiede.clone();
}

public void monta(){
    if (statoCorrente == SMONTATA)
        statoCorrente = MONTATA;
}

public void smonta(){
    if (statoCorrente == MONTATA || statoCorrente == ESPOSTA)
        statoCorrente = SMONTATA;
}

```

```

public void esponi(){
    if (statoCorrente == MONTATA)
        statoCorrente = ESPOSTA;
}
}

```

## La classe Java CostruzioneSpeciale

```

// File AppCostruzioni/CostruzioneSpeciale/CostruzioneSpeciale.java
package AppCostruzioni.CostruzioneSpeciale;
import AppCostruzioni.*;
import AppCostruzioni.Costruzione.*;
import AppCostruzioni.TipoMattoncini.*;
import AppCostruzioni.TipoMattonciniElettrificati.*;
import java.util.*;

public final class CostruzioneSpeciale extends Costruzione{
    private String fenomeno;
    private TipoMattonciniElettrificati richiedeSpeciale;

    public CostruzioneSpeciale(String istruzioni, String fenomeno){
        super(istruzioni);
        this.fenomeno = fenomeno;
        richiedeSpeciale = null;
    }

    public String getFenomeno(){
        return fenomeno;
    }

    public void inserisciTipoMattonciniElettrificati(TipoMattonciniElettrificati me){
        if (me != null)

```

```

        richiedeSpeciale = me;
    }

    public void eliminaTipoMattonciniElettrificati(){
        richiedeSpeciale = null;
    }

    public TipoMattonciniElettrificati getTipoMattonciniElettrificati()
        throws EccezioneSubset, EccezioneMolteplicita {
        if (richiedeSpeciale == null)
            throw new EccezioneMolteplicita("Molteplicità min/max violate");
        if (!getLinkRichiede().contains(new TipoLinkRichiede(this,
            richiedeSpeciale,
            0 /*non significativo*/))
        )
            throw new EccezioneSubset("Vincolo di subset violato");
        return richiedeSpeciale;
    }
}

```

## La classe Java TipoMattoncini

```
// File AppCostruzioni/TipoMattoncini/TipoMattoncini.java
package AppCostruzioni.TipoMattoncini;

import AppCostruzioni.*;

import java.util.*;

public class TipoMattoncini{
    private String dimensioni;
    private String colore;
    private HashSet<TipoLinkRichiede> richieste;

    public TipoMattoncini(String dimensioni, String colore){
        this.dimensioni = dimensioni;
        this.colore = colore;
        richieste = new HashSet<TipoLinkRichiede>();
    }

    public String getDimensioni(){
        return dimensioni;
    }

    public String getColore(){
        return colore;
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 26

## La classe Java TipoMattonciniElettrificati

```
// File AppCostruzioni/TipoMattonciniElettrificati/TipoMattonciniElettrificati.java
package AppCostruzioni.TipoMattonciniElettrificati;

import AppCostruzioni.*;
import AppCostruzioni.TipoMattoncini.*;

import java.util.*;

public final class TipoMattonciniElettrificati extends TipoMattoncini{
    private String specificaElettrica;

    public TipoMattonciniElettrificati
        (String dimensioni, String colore, String specificaElettrica){
        super(dimensioni, colore);
        this.specificaElettrica = specificaElettrica;
    }

    public String getSpecificaElettrica(){
        return specificaElettrica;
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 27

```
}

public void inserisciLinkRichiede (AssociazioneRichiede a){
    if (a != null)
        richieste.add(a.getLink());
}

public void eliminaLinkRichiede(AssociazioneRichiede a){
    if (a != null)
        richieste.remove(a.getLink());
}

public Set<TipoLinkRichiede> getLinkRichiede(){
    return (HashSet<TipoLinkRichiede>) richieste.clone();
}
}
```

## La classe Java ScatolaMontaggio

```
// File AppCostruzioni/ScatolaMontaggio/ScatolaMontaggio.java
package AppCostruzioni.ScatolaMontaggio;

import AppCostruzioni.*;

import java.util.*;

public class ScatolaMontaggio {
    private final int MOLT_MIN = 1;

    private String descrizione;
    private HashSet<TipoLinkContiene> contiene;

    public ScatolaMontaggio(String descrizione){
        this.descrizione = descrizione;
        contiene = null;
    }

    public String getDescrizione(){
        return descrizione;
    }

    public void inserisciLinkContiene (TipoLinkContiene c){
        if (c != null)

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 28



## La classe Java ScatolaSpeciale

```
    contiene.add(c);
}

public void eliminaLinkContiene(TipoLinkContiene c){
    if (c != null)
        contiene.remove(c);
}

public Set<TipoLinkContiene> getLinkContiene() throws EccezioneMolteplicita{
    if (contiene.size() < MOLT_MIN)
        throw new EccezioneMolteplicita("Molteplicità minima violata");
    return (HashSet<TipoLinkContiene>) contiene.clone();
}
}
```

```
// File AppCostruzioni/ScatolaSpeciale/ScatolaSpeciale.java
package AppCostruzioni.ScatolaSpeciale;

import AppCostruzioni.*;
import AppCostruzioni.ScatolaMontaggio.*;
import AppCostruzioni.TipoMattoncini.*;

import java.util.*;

public final class ScatolaSpeciale extends ScatolaMontaggio {
    private int MOLT_MIN;
    private int difficolta;
    private HashSet<TipoLinkContieneSpeciale> contieneSpeciale;

    public ScatolaSpeciale(String descrizione, int difficolta){
        super(descrizione);
        this.difficolta = difficolta;
        contieneSpeciale = new HashSet<TipoLinkContieneSpeciale>();
    }

    public int getDifficolta(){
        return difficolta;
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 29

```
public void inserisciLinkContieneSpeciale (TipoLinkContieneSpeciale c){
    if (c != null)
        contieneSpeciale.add(c);
}

public void eliminaLinkContiene(TipoLinkContieneSpeciale c){
    if (c != null)
        contieneSpeciale.remove(c);
}

public Set<TipoLinkContieneSpeciale> getLinkContieneSpeciale()
    throws EccezioneMolteplicita, EccezioneSubset{
    if (getLinkContiene().size() < MOLT_MIN)
        throw new EccezioneMolteplicita("Molteplicità minima violata");
    Set<TipoLinkContiene> c = getLinkContiene();
    Iterator<TipoLinkContieneSpeciale> it = contieneSpeciale.iterator();
    while (it.hasNext()) {
        TipoMattoncini m = (TipoMattoncini) it.next().getTipoMattonciniElettrificati();
        if (!c.contains(new TipoLinkContiene(this,m,0))) //Nota: la
            //quantità
            //(terzo
            //parametro)
            //non è significativa

        throw new EccezioneSubset("Vincolo di subset violato");
    }
}
```

```
        return (HashSet<TipoLinkContieneSpeciale>) contieneSpeciale.clone();
    }
}
```

## La classe Java TipoLinkRichiede

```
// File AppCostruzioni/TipoLinkRichiede.java
package AppCostruzioni;

import AppCostruzioni.TipoMattoncini.*;
import AppCostruzioni.Costruzione.*;
import java.util.*;

public class TipoLinkRichiede {
    private final Costruzione laCostruzione;
    private final TipoMattoncini ilTipoMattoncini;
    private final int quantita;

    public TipoLinkRichiede(Costruzione c, TipoMattoncini m, int quantita)
        throws EccezionePrecondizioni {
        if (c == null || m == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laCostruzione = c;
        ilTipoMattoncini = m;
        this.quantita = quantita;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 30

## La classe Java AssociazioneRichiede

```
// File AppLibrerie/AssociazioneRichiede.java
package AppCostruzioni;

public final class AssociazioneRichiede {
    private TipoLinkRichiede link;

    private AssociazioneRichiede(TipoLinkRichiede link){
        this.link = link;
    }

    public TipoLinkRichiede getLink(){
        return link;
    }

    public static void inserisci(TipoLinkRichiede y) {
        if (y != null) {
            AssociazioneRichiede k = new AssociazioneRichiede(y);
            y.getCostruzione().inserisciLinkRichiede(k);
            y.getTipoMattoncini().inserisciLinkRichiede(k);
        }
    }

    public static void elimina(TipoLinkRichiede y) {
        if (y != null) {
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 31

```
        TipoLinkRichiede l = (TipoLinkRichiede) o;
        return l.laCostruzione == laCostruzione &&
            l.ilTipoMattoncini == ilTipoMattoncini;
    }
    else return false;
}

public int hashCode() {
    return laCostruzione.hashCode() + ilTipoMattoncini.hashCode();
}

public Costruzione getCostruzione(){
    return laCostruzione;
}

public TipoMattoncini getTipoMattoncini(){
    return ilTipoMattoncini;
}

public int getQuantita(){
    return quantita;
}

public String toString() {
    return "<" + laCostruzione + ", " + ilTipoMattoncini + ">";
}
}
```

```
AssociazioneRichiede k = new AssociazioneRichiede(y);
y.getCostruzione().eliminaLinkRichiede(k);
y.getTipoMattoncini().eliminaLinkRichiede(k);
}
}
```

## La classe Java TipoLinkContiene

```
// File AppCostruzioni/TipoLinkContiene.java
package AppCostruzioni;

import AppCostruzioni.ScatolaMontaggio.*;
import AppCostruzioni.TipoMattoncini.*;
import java.util.*;

public class TipoLinkContiene {
    private final ScatolaMontaggio laScatolaMontaggio;
    private final TipoMattoncini ilTipoMattoncini;
    private final int quantita;

    public TipoLinkContiene(ScatolaMontaggio c, TipoMattoncini m, int quantita)
        throws EccezionePrecondizioni {
        if (c == null || m == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laScatolaMontaggio = c;
        ilTipoMattoncini = m;
        this.quantita = quantita;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 32

## La classe Java TipoLinkContieneSpeciale

```
// File AppCostruzioni/TipoLinkContieneSpeciale.java
package AppCostruzioni;

import AppCostruzioni.ScatolaSpeciale.*;
import AppCostruzioni.TipoMattonciniElettrificati.*;
import java.util.*;

public class TipoLinkContieneSpeciale {
    private final ScatolaSpeciale laScatolaSpeciale;
    private final TipoMattonciniElettrificati ilTipoMattonciniElettrificati;

    public TipoLinkContieneSpeciale(ScatolaSpeciale c, TipoMattonciniElettrificati m)
        throws EccezionePrecondizioni {
        if (c == null || m == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laScatolaSpeciale = c;
        ilTipoMattonciniElettrificati = m;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkContieneSpeciale l = (TipoLinkContieneSpeciale) o;
            return (l.laScatolaSpeciale == laScatolaSpeciale &&
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 33

```
        TipoLinkContiene l = (TipoLinkContiene) o;
        return l.laScatolaMontaggio == laScatolaMontaggio &&
            l.ilTipoMattoncini == ilTipoMattoncini;
    }
    else return false;
}

public int hashCode() {
    return laScatolaMontaggio.hashCode() + ilTipoMattoncini.hashCode();
}

public ScatolaMontaggio getScatolaMontaggio(){
    return laScatolaMontaggio;
}

public TipoMattoncini getTipoMattoncini(){
    return ilTipoMattoncini;
}

public int getQuantita(){
    return quantita;
}

public String toString() {
    return "<" + laScatolaMontaggio + ", " + ilTipoMattoncini + ">";
}
}
```

```
        l.ilTipoMattonciniElettrificati == ilTipoMattonciniElettrificati);
    }
    else return false;
}

public int hashCode() {
    return laScatolaSpeciale.hashCode() + ilTipoMattonciniElettrificati.hashCode();
}

public ScatolaSpeciale getScatolaSpeciale(){
    return laScatolaSpeciale;
}

public TipoMattonciniElettrificati getTipoMattonciniElettrificati(){
    return ilTipoMattonciniElettrificati;
}

public String toString() {
    return "<" + laScatolaSpeciale + ", " + ilTipoMattonciniElettrificati + ">";
}
}
```

## La classe Java OperazioniUtente

```
// File AppCostruzioni/OperazioniUtente.java
package AppCostruzioni;

import AppCostruzioni.Costruzione.*;
import AppCostruzioni.ScatolaMontaggio.*;
import AppCostruzioni.TipoMattoncini.*;
import AppCostruzioni.TipoMattonciniElettrificati.*;

import java.util.*;

public final class OperazioniUtente{
    public static boolean mattonciniSuff(ScatolaMontaggio s, Costruzione c)
        throws EccezioneMolteplicita {
        Iterator<TipoLinkRichiede> itr =
            c.getLinkRichiede().iterator(); // Throws EccezioneMolteplicita
        while(itr.hasNext()){
            TipoLinkRichiede richiede = itr.next();
            Iterator<TipoLinkContiene> itc =
                s.getLinkContiene().iterator(); // Throws EccezioneMolteplicita
            boolean found = false;
            while(itc.hasNext() && !found){
                TipoLinkContiene contiene = itc.next();
                if ((richiede.equals(new TipoLinkRichiede(c,
                    contiene.getTipoMattoncini(),
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 34

## Realizzazione in Java delle classi per eccezioni

```
// File AppCostruzioni/EccezioneMolteplicita.java
package AppCostruzioni;

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

// File AppCostruzioni/EccezioneSubset.java
package AppCostruzioni;

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

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-04-20 35

```
        0 /* non significativo*/))
        && (richiede.getQuantita() <= contiene.getQuantita()))
        found = true;
    } // while(itc.hasNext())
    if (!found)
        return false;
} // while (itr.hasNext())
return true;
}

public static Set<Costruzione> costruzioniElett(TipoMattonciniElettrificati me){
    HashSet<Costruzione> result = new HashSet<Costruzione>();
    Iterator<TipoLinkRichiede> itr = me.getLinkRichiede().iterator();
    while (itr.hasNext())
        result.add(itr.next().getCostruzione());
    return result;
}
}
```

```
}

// File AppCostruzioni/EccezionePrecondizioni.java
package AppCostruzioni;

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