

SAPIENZA Università di Roma
Facoltà di Ingegneria dell'Informazione

Esercitazioni di
PROGETTAZIONE DEL SOFTWARE
(Corso di Laurea in Ingegneria Informatica)
A.A. 2009-10

Estratto del compito d'esame del 20 aprile 2007

SOLUZIONE

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).

Requisiti (cont.)

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 (cont.)

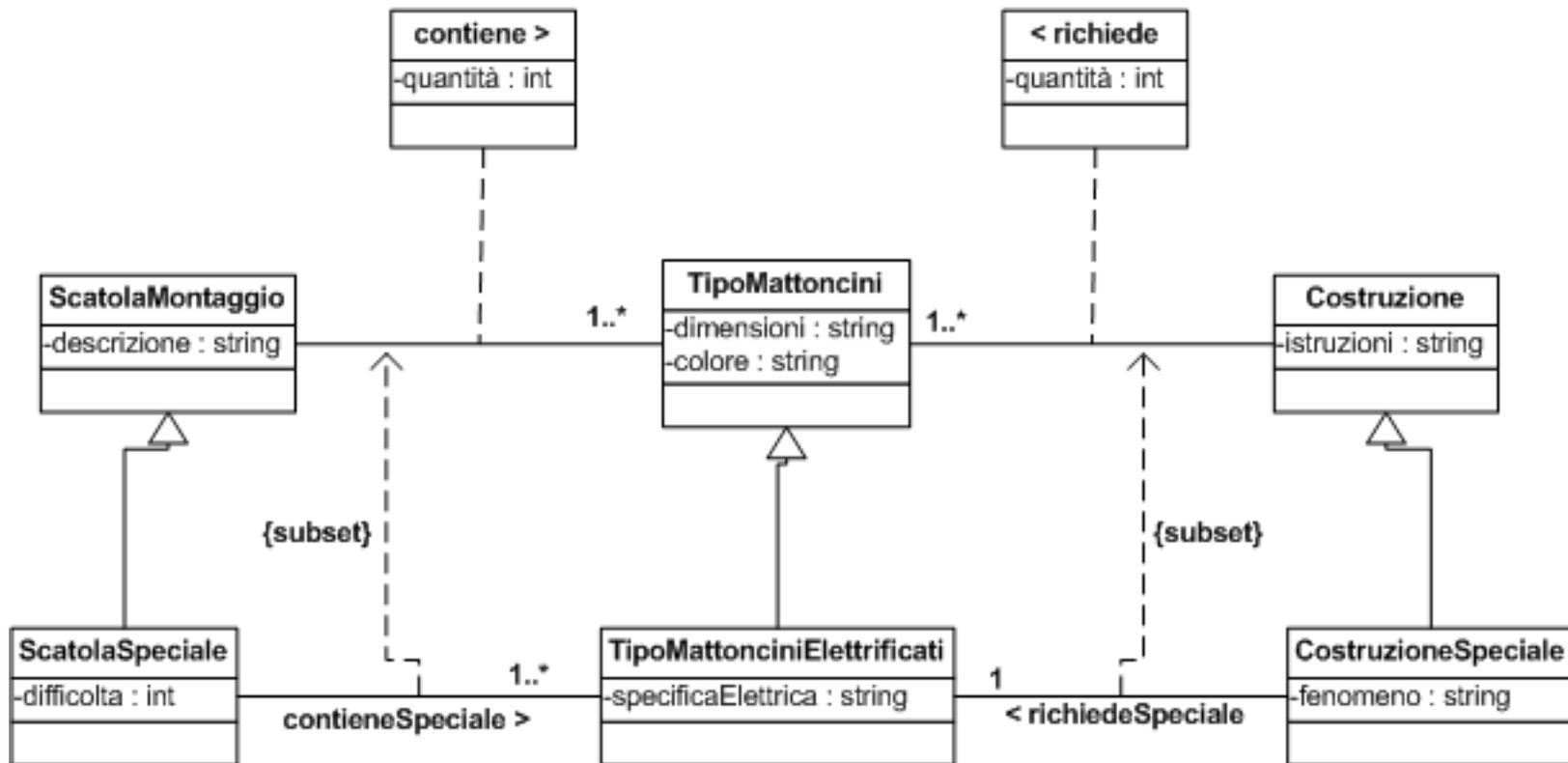
Domanda 1. Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in UML per l'applicazione e motivando, qualora ce ne fosse bisogno, le scelte effettuate.

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

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

Fase di analisi

Diagramma delle classi



Fase di progetto

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

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: Set, 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>ScatolaMontaggio</i>	<i>descrizione</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>

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.

Fase di realizzazione

Struttura dei file e dei package

```
+---AppCostruzioni
|   TipoLinkRichiede.java
|   ManagerRichiede.java
|   TipoLinkContiene.java
|   TipoLinkContieneSpeciale.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 String istruzioni;
    private HashSet<TipoLinkRichiede> richiede;

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

    public String getIstruzioni(){
        return istruzioni;
    }

    public void inserisciLinkRichiede (TipoLinkRichiede a){
        if (a != null && a.getCostruzione() == this)
            ManagerRichiede.inserisci(a);
    }
}
```

```
}

public void eliminaLinkRichiede(TipoLinkRichiede a){
    if (a != null && a.getCostruzione() == this)
        ManagerRichiede.elimina(a);
}

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

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

public void eliminaPerManagerRichiede(ManagerRichiede a) {
    if (a != null) richiede.remove(a.getLink());
}
}
```


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> richiede;

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

    public String getDimensioni(){
        return dimensioni;
    }

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

```
}

public void inserisciLinkRichiede (TipoLinkRichiede a){
    if (a != null && a.getTipoMattoncini() == this)
        ManagerRichiede.inserisci(a);
}

public void eliminaLinkRichiede(TipoLinkRichiede a){
    if (a != null && a.getTipoMattoncini() == this)
        ManagerRichiede.elimina(a);
}

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

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

public void eliminaPerManagerRichiede(ManagerRichiede a){
    if (a != null) richiede.remove(a.getLink());
}
}
```

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;
    }
}
```

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 && c.getScatolaMontaggio() == this)
```

```
        contiene.add(c);
    }

    public void eliminaLinkContiene(TipoLinkContiene c){
        if (c != null && c.getScatolaMontaggio() == this)
            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();
    }
}
```

La classe Java ScatolaSpeciale

```
// 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 final int MOLT_MIN=1;
    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;
    }
}
```

```

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

public void eliminaLinkContieneSpeciale(TipoLinkContieneSpeciale c){
    if (c != null && c.getScatolaSpeciale() == this)
        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())) {
```

```

        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 + ">";
}
}

```

La classe Java ManagerRichiede

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

public final class ManagerRichiede {

    private TipoLinkRichiede link;

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

    public TipoLinkRichiede getLink(){
        return link;
    }

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

    public static void elimina(TipoLinkRichiede y) {
```

```
if (y != null) {  
    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())) {
```

```

        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 + ">";
}
}

```

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 &&
```

```
        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 ClienteMattoncini

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

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

import java.util.*;

public final class ClienteMattoncini{
    public static boolean mattonciniSufficienti(ScatolaMontaggio s, Costruzione c)
        throws EccezioneMolteplicita {
        Iterator<TipoLinkRichiede> itr =
            c.getLinkRichiede().iterator(); // puo' lanciare EccezioneMolteplicita
        while(itr.hasNext()){
            TipoLinkRichiede richiede = itr.next();
            Iterator<TipoLinkContiene> itc =
                s.getLinkContiene().iterator(); // puo' lanciare EccezioneMolteplicita
            boolean found = false;
            while(itc.hasNext() && !found){
                TipoLinkContiene contiene = itc.next();
                if ((richiede.getTipoMattoncini() == contiene.getTipoMattoncini())&&
                    (richiede.getQuantita() <= contiene.getQuantita()))
```

```
        found = true;
    }//fine while(itc.hasNext())
    if (!found)
        return false;
}// fine while (itr.hasNext())
return true;
}
```

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

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;
    }
}
```

```
}  
  
// 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;  
    }  
}
```