

Corso di
"PROGETTAZIONE DEL SOFTWARE I"
(Corso di Laurea in Ingegneria Informatica)
Prof. Giuseppe De Giacomo
Canali A-L & M-Z
A.A. 2006-07

Compito d'esame del 3 luglio 2007

SOLUZIONE

Requisiti (cont.)

Una icona-attiva è inizialmente nello stato *in-attesa*. Al click viene messa nello stato *in-animazione* e quando il sistema operativo lo segnala passa allo stato *in-esecuzione*. Infine, quando il sistema operativo segnala la terminazione della applicazione associata, si rimette in-attesa. L'icona-attiva può essere modificata (modificando l'applicazione attivata, le display-area occupate, la sequenza delle animazioni utilizzate) solo quando è in-attesa.

Requisiti

L'applicazione da progettare riguarda una parte dell'interfaccia ad icone di un telefono cellulare di nuova generazione. Ogni icona è caratterizzata da un codice (una stringa) e da una immagine (rappresentata anche essa da una stringa). Alcune icone sono *icone-attive* e sono caratterizzate da: *i.* un suono (rappresentato da una stringa) che viene prodotto al click su di esse, *ii.* dall'applicazione che viene attivata al click, *iii.* dalla sequenza (non vuota) di animazioni che vengono mostrate al click e *iv.* dalle *display-area* (una o più) che occupano, ciascuna con l'indicazione se essa è occupata interamente o meno. Le applicazioni sono caratterizzate dal loro nome e dal nome del file (una stringa) dove è memorizzato il codice eseguibile. Le animazioni sono caratterizzate da un *link* (una stringa) al codice di visualizzazione. Le *display-area* dalla posizione (un intero) e dall'immagine di *background* (una stringa). Un'animazione, a sua volta, può coinvolgere una o più *display-area* (anche non correlate con quelle occupate dall'icona-attiva che la mette in esecuzione).

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 2

Requisiti (cont.)

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

- data una icona-attiva *ia*, restituire una lista contenente l'inverso della sequenza delle animazioni che *ia* utilizza;
- data una animazione *a* restituire l'insieme delle icone-attive che mostrano *a*.

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 4

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:

- la classi **IconaAttiva** e tutte le associazioni in cui è coinvolta;
- il primo use case.

Fase di analisi

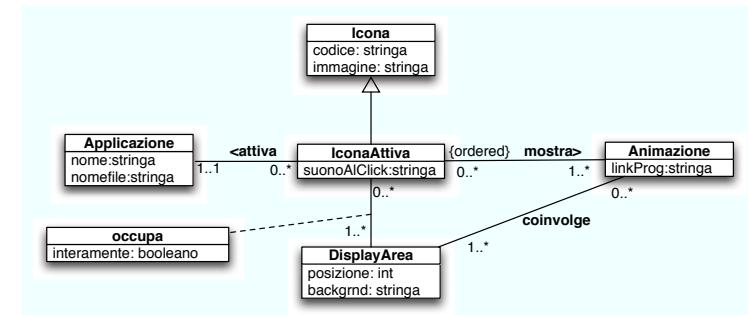
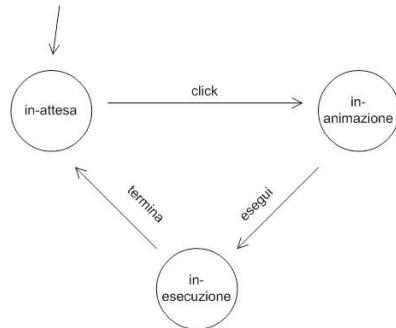
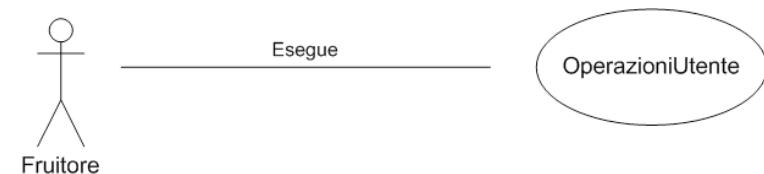


Diagramma degli stati e delle transizioni della classe Costruzione



U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 9

Diagramma degli use case



U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 10

Specifiche dello use case

InizioSpecificheUseCase OperazioniUtente

invertiAnimazioni (*ia: IIconaAttiva*): *Lista<Animazione>*

pre: true

post:

Definiamo la sequenza $S = s_1 \dots s_n$ dove, per ogni $i = 1 \dots n$, $s_i \in Animazione \wedge \langle ia, s_i \rangle \in mostra \wedge s_i < s_{i+1}$ ($i \neq n - 1$, secondo l'ordine definito su *mostra*, per *ia* fissata).

result = $s_n \dots s_1$

...

Specifiche dello use case (cont.)

chiMostra (*a: Animazione*): *Insieme<IIconaAttiva>*

pre: true

post: *result* = $\{ia \in IIconaAttiva \mid \langle ia, a \rangle \in mostra\}$

FineSpecifiche

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 11

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 12

Algoritmi per le operazioni dello use-case

Adottiamo i seguenti algoritmi:

- Per l'operazione **invertiAnimazioni**(ia: *IconaAttiva*): *Lista<Animazione>*

```
result = new List < Animazione >;
scandiamo, secondo l'ordine dato, l'insieme dei link di tipo mostra
in cui ia e' coinvolta e per ogni link l {
    result.aggiungiInTesta(l.Animazione); //cosi' otteniamo la lista invertita
}
return result
```

Fase di progetto

- Per l'operazione **chiMostra**(a: *Animazione*): *Insieme<IconaAttiva>*

```
result = new Insieme<IconaAttiva>;
per ogni link l di tipo mostra in cui a e' coinvolta{
    result.add(l.IconaAttiva);
}
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>attiva</i>	<i>IconaAttiva</i> <i>Applicazione</i>	SI ³ NO
<i>coinvolge</i>	<i>Animazione</i> <i>DisplayArea</i>	SI ³ NO
<i>mostra</i>	<i>IconaAttiva</i> <i>Animazione</i>	SI ^{2,3} SI ²
<i>occupa</i>	<i>IconaAttiva</i> <i>DisplayArea</i>	SI ³ NO

Strutture di dati

Abbiamo la necessità di rappresentare collezioni e liste omogenee di oggetti, eventualmente ordinate, a causa:

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

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

Corrispondenza fra tipi UML e Java

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

Tipo UML	Rappresentazione in Java
stringa	String
booleano	boolean
Insieme	HashSet
Lista	LinkedList

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
Icona	codice immagine
IconaAttiva	suonoAlClick
Applicazione	nome nomeFile
Animazione	linkProg
DisplayArea	posizione backgrnd

Classe UML	Proprietà	nota alla nascita	non nota alla nascita
IconaAttiva	—	—	applicazione

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.

Rappresentazione degli stati in Java

Per la classe UML *IconaAttiva*, 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.	int
nome var.	stato	
inAttesa	valore	1
inAnimazione	valore	2
inEsecuzione	valore	3

API delle classi Java progettate

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

Fase di realizzazione

Considerazioni iniziali

La traccia ci richiede di realizzare:

1. La classe IIconaAttiva.
2. l'associazione UML *mostra* con responsabilità doppia e con vincoli di molteplicità 1..* (molteplicità minima diversa da zero) e 0..*;
3. l'associazione UML *attiva* con responsabilità singola e con vincoli di molteplicità 1..1 (molteplicità massima e minima diverse da zero) e 0..*;
4. l'associazione UML *occupa* con responsabilità singola 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.

Struttura dei file e dei package

```
+---AppCellulare
|   |   TipoLinkMostra.java
|   |   AssociazioneMostra.java
|   |   TipoLinkCoinvolge.java
|   |   TipoLinkOccupa.java
|   |   OperazioniUtente.java
|   |   EccezioneMolteplicita.java
|   |   EccezionePrecondizioni.java
|
+---Icona
|   |   IIcona.java
|
+---IconaAttiva
|   |   IIconaAttiva.java
|
+---Applicazione
|   |   Applicazione.java
|
+---DisplayArea
|   |   DisplayArea.java
|
\---Animazione
    |   Animazione.java
```

[La classe Java Icona](#)

```
// File AppCellulare/Icona/Icona.java
package AppCellulare.Icona;
//import AppCellulare.*;
import java.util.*;

public class Icona {
    private String codice;
    private String immagine;

    public Icona(String codice, String immagine){
        this.codice = codice;
        this.immagine=immagine;
    }

    public String getCodice(){
        return codice;
    }

    public String getImmagine(){
        return immagine;
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 25

[La classe Java IconaAttiva](#)

```
// File AppCellulare/IconaAttiva/IconaAttiva.java
package AppCellulare.IconaAttiva;
import AppCellulare.*;
import AppCellulare.Icona.*;
import AppCellulare.Applicazione.*;
import java.util.*;

public final class IconaAttiva extends Icona{

    private final int IN_ATTESA=1, IN_ANIMAZIONE=2, IN_ESECUZIONE=3;
    private final int MOLT_MIN_MOSTRA=1,MOLT_MIN_OCCUPA=1;
    private String suonoAlClick;
    private Applicazione applicazione;
    private LinkedList<TipoLinkMostra> mostra;
    private HashSet<TipoLinkOccupata> occupa;
    private int stato;

    public IconaAttiva(String codice, String immagine, String suonoAlClick){
        super(codice, immagine);
        this.suonoAlClick = suonoAlClick;
        applicazione = null;
        mostra = new LinkedList<TipoLinkMostra>();
        occupa = new HashSet<TipoLinkOccupata>();
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 26

```
public String getSuonoAlClick(){
    return suonoAlClick;
}

public void inserisciApplicazione(Applicazione a) throws EccezionePrecondizioni{
    if (!estInAttesa()) //controllo stato!
        throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
    if (a != null)
        this.applicazione = a;
}

public Applicazione getApplicazione() throws EccezioneMolteplicita{
    if (applicazione == null) //controllo molteplicita'
        throw new EccezioneMolteplicita("Molteplicità min/max violata");
    return applicazione;
}

public void eliminaApplicazione() throws EccezionePrecondizioni{
    if (!estInAttesa()) //controllo stato!
        throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
    applicazione = null;
}

public boolean haApplicazioneAssociata() { //per verificare molteplicita'
```

```
        return applicazione !=null;
    }

    public void inserisciLinkMostra(AssociazioneMostra a){
        if (a != null && !mostra.contains(a.getLink())){ //non controlliamo lo stato qui ma
            mostra.add(a.getLink());
        }
    }

    public void eliminaLinkMostra(AssociazioneMostra a) {
        if (a != null) //non controlliamo lo stato qui ma in AssociazioneMostra.elimina()!
            mostra.remove(a.getLink());
    }

    public List<TipoLinkMostra> getLinkMostra() throws EccezioneMolteplicita{
        if (mostra.size() < MOLT_MIN_MOSTRA) //controllo molteplicita'
            throw new EccezioneMolteplicita("Molteplicità minima violata");
        return (LinkedList<TipoLinkMostra>) mostra.clone();
    }

    public int quantiLinkMostra() { //per verificare molteplicita'
        return mostra.size();
    }
}
```

```

public void inserisciLinkOcupa(TipoLinkOcupa a) throws EccezionePrecondizioni{
    if (!estInAttesa()) //controllo stato!
        throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
    if (a != null)
        occupa.add(a);
}

public void EliminaLinkOcupa(TipoLinkOcupa a) throws EccezionePrecondizioni{
    if (!estInAttesa()) //controllo stato!
        throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
    if (a != null)
        occupa.remove(a);
}

public Set<TipoLinkOcupa> getLinkOcupa() throws EccezioneMolteplicita{
    if (occupa.size() < MOLT_MIN_OCCUPA) //controllo molteplicità!
        throw new EccezioneMolteplicita("Molteplicità minima violata");
    return (HashSet<TipoLinkOcupa>) occupa.clone();
}

public int quantiLinkOcupa() { //per verificare molteplicità
    return occupa.size();
}

public void click(){
    if (stato == IN_ATTESA)
        stato = IN_ANIMAZIONE;
}

public void esegui(){
    if (stato == IN_ANIMAZIONE)
        stato = IN_ESECUZIONE;
}

public void termina(){
    if (stato == IN_ESECUZIONE)
        stato = IN_ATTESA;
}

public boolean estInAttesa(){
    return (stato == IN_ATTESA);
}
}

```

[La classe Java Applicazione](#)

```

// File AppCellulare/Applicazione/Applicazione.java
package AppCellulare.Applicazione;

import AppCellulare.*;
import java.util.*;

public class Applicazione {
    private String nome;
    private String nomeFile;

    public Applicazione(String nome, String nomeFile){
        this.nome = nome;
        this.nomeFile = nomeFile;
    }

    public String getNome(){
        return nome;
    }

    public String getNomeFile(){
        return nomeFile;
    }
}

```

[La classe Java DisplayArea](#)

```

// File AppCellulare/DisplayArea/DisplayArea.java
package AppCellulare.DisplayArea;
import AppCellulare.*;
import java.util.*;

public class DisplayArea {
    private int posizione;
    private String backgrnd;

    public DisplayArea(int posizione, String backgrnd){
        this.posizione = posizione;
        this.backgrnd = backgrnd;
    }

    public int getPosizione(){
        return posizione;
    }

    public String getBackgrnd(){
        return backgrnd;
    }
}

```

La classe Java Animazione

```
// File AppCellulare/Animazione/Animazione.java
package AppCellulare.Animazione;

import AppCellulare.*;
import java.util.*;

public class Animazione{
    private String linkProg;
    private HashSet<TipoLinkMostra> mostra;
    private HashSet<TipoLinkCoinvolge> coinvolge;
    private final int MOLT_MIN=1;

    public Animazione(String linkProg){
        this.linkProg = linkProg;
        mostra = new HashSet<TipoLinkMostra>();
        coinvolge = new HashSet<TipoLinkCoinvolge>();
    }

    public String getLinkProg(){
        return linkProg;
    }

    public void inserisciLinkMostra (AssociazioneMostra a){
        if (a != null)
            mostra.add(a.getLink());
    }

    public void eliminaLinkMostra(AssociazioneMostra a){
        if (a != null)
            mostra.remove(a.getLink());
    }

    public Set<TipoLinkMostra> getLinkMostra(){
        return (HashSet<TipoLinkMostra>) mostra.clone();
    }

    public void inserisciLinkCoinvolge(TipoLinkCoinvolge c){
        if (c != null)
            coinvolge.add(c);
    }

    public Set<TipoLinkCoinvolge> getLinkCoinvolge() throws EccezioneMolteplicita{
        if (coinvolge.size() < MOLT_MIN)
            throw new EccezioneMolteplicita("Cardinalità minima violata");
        return (HashSet<TipoLinkCoinvolge>) coinvolge.clone();
    }

    public void eliminaLinkCoinvolge(TipoLinkCoinvolge c){
        if (c != null)
            coinvolge.remove(c);
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 29

La classe Java TipoLinkMostra

```
// File AppCellulare/TipoLinkMostra.java
package AppCellulare;
import AppCellulare.IconaAttiva.*;
import AppCellulare.Animazione.*;
import java.util.*;

public class TipoLinkMostra{
    private final IconaAttiva laIconaAttiva;
    private final Animazione laAnimazione;

    public TipoLinkMostra(IconaAttiva ia, Animazione a)
        throws EccezionePrecondizioni {
        if (ia == null || a == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laIconaAttiva = ia;
        laAnimazione = a;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkMostra l = (TipoLinkMostra) o;
            return l.laIconaAttiva == laIconaAttiva &&
                l.laAnimazione == laAnimazione;
    }
}
```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 30

```

    }
    else return false;
}

public int hashCode() {
    return laIconaAttiva.hashCode() + laAnimazione.hashCode();
}

public IIconaAttiva getIconaAttiva(){
    return laIconaAttiva;
}

public Animazione getAnimazione(){
    return laAnimazione;
}

public String toString() {
    return "<" + laIconaAttiva + ", " + laAnimazione + ">";
}
}

```

```

public static void elimina(TipoLinkMostra y) throws EccezionePrecondizioni{
    if (y != null) {
        if (!y.getIconaAttiva().estInAttesa()) //controllo stato si fa qui!
            throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
        AssociazioneMostra k = new AssociazioneMostra(y);
        y.getIconaAttiva().eliminaLinkMostra(k);
        y.getAnimazione().eliminaLinkMostra(k);
    }
}

```

La classe Java AssociazioneMostra

```

// File AppCellulare/AssociazioneMostra.java
package AppCellulare;

public final class AssociazioneMostra{
    private TipoLinkMostra link;

    private AssociazioneMostra(TipoLinkMostra link){
        this.link = link;
    }

    public TipoLinkMostra getLink(){
        return link;
    }

    public static void inserisci(TipoLinkMostra y) throws EccezionePrecondizioni{
        if (y != null) {
            if (!y.getIconaAttiva().estInAttesa()) //controllo stato si fa qui!
                throw new EccezionePrecondizioni("Elemento modificabile solo se in attesa");
            AssociazioneMostra k = new AssociazioneMostra(y);
            y.getIconaAttiva().inserisciLinkMostra(k);
            y.getAnimazione().inserisciLinkMostra(k);
        }
    }
}

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 31

La classe Java TipoLinkCoinvolge

```

// File AppCellulare/TipoLinkCoinvolge.java
package AppCellulare;
import AppCellulare.Animazione.*;
import AppCellulare.DisplayArea.*;
import java.util.*;

public class TipoLinkCoinvolge{
    private final Animazione laAnimazione;
    private final DisplayArea laDisplayArea;

    public TipoLinkCoinvolge(Animazione a, DisplayArea da)
        throws EccezionePrecondizioni {
        if (a == null || da == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laAnimazione = a;
        laDisplayArea = da;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TipoLinkCoinvolge l = (TipoLinkCoinvolge) o;
            return l.laAnimazione == laAnimazione &&
                l.laDisplayArea == laDisplayArea;
        }
    }
}

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 32

```

    }
    else return false;
}

public int hashCode() {
    return laAnimazione.hashCode() + laDisplayArea.hashCode();
}

public Animazione getAnimazione(){
    return laAnimazione;
}

public DisplayArea getDisplayArea(){
    return laDisplayArea;
}

public String toString() {
    return "<" + laAnimazione + ", " + laDisplayArea + ">";
}
}

```

La classe Java TypoLinkOccup

```

// File AppCellulare/TypoLinkOccup.java
package AppCellulare;
import AppCellulare.IconaAttiva.*;
import AppCellulare.DisplayArea.*;
import java.util.*;

public class TypoLinkOccup{
    private final IconaAttiva laIconaAttiva;
    private final DisplayArea laDisplayArea;
    private final boolean interamente;

    public TypoLinkOccup(IconaAttiva a, DisplayArea da, boolean i)
        throws EccezionePrecondizioni {
        if (a == null || da == null) // CONTROLLO PRECONDIZIONI
            throw new EccezionePrecondizioni
                ("Gli oggetti devono essere inizializzati");
        laIconaAttiva = a;
        laDisplayArea = da;
        interamente = i;
    }

    public boolean equals(Object o) {
        if (o != null && getClass().equals(o.getClass())) {
            TypoLinkOccup l = (TypoLinkOccup) o;

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 33

```

        return laIconaAttiva == laIconaAttiva &&
            laDisplayArea == laDisplayArea; //Non verifico l'uguaglianza
        } //dell'attributo
        else return false;
    }

    public int hashCode() {
        return laIconaAttiva.hashCode() + laDisplayArea.hashCode();
    }

    public IconaAttiva getIconaAttiva(){
        return laIconaAttiva;
    }

    public DisplayArea getDisplayArea(){
        return laDisplayArea;
    }

    public boolean getInteramente(){
        return interamente;
    }

    public String toString() {
        return "<" + laIconaAttiva + ", " + laDisplayArea + ", " +
            interamente + ">";
    }
}

```

La classe Java OperazioniUtente

```

// File AppCellulare/OperazioniUtente.java
package AppCellulare;
import AppCellulare.*;
import AppCellulare.IconaAttiva.*;
import AppCellulare.Animazione.*;
import java.util.*;

public final class OperazioniUtente{
    public static List<Animazione> inverteAnimazioni(IconaAttiva ia)
        throws EccezioneMolteplicita {
        LinkedList<Animazione> result = new LinkedList<Animazione>();
        List<TypoLinkMostra> linkMostra = ia.getLinkMostra();
        Iterator<TypoLinkMostra> it = linkMostra.iterator();
        while(it.hasNext()){
            TypoLinkMostra l = it.next();
            linkAnimazioni.add(0,l.getAnimazione()); //Inserisce in testa alla lista
        }
        return result;
    }

    public static Set<IconaAttiva> chiMostra(Animazione a){
        HashSet<IconaAttiva> result = new HashSet<IconaAttiva>();
        Set<TypoLinkMostra> linksMostra = a.getLinkMostra();
        Iterator<TypoLinkMostra> it = linksMostra.iterator();

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 34

```

while(it.hasNext()){
    TipoLinkMostra linkMostra = it.next();
    result.add(linkMostra.getIconaAttiva());
}
return result;
}
}

```

Realizzazione in Java delle classi per eccezioni

```

// File AppCellulare/EccezioneMolteplicita.java
package AppCellulare;

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

// File AppCellulare/EccezionePrecondizioni.java
package AppCellulare;

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

```

U. "La Sapienza". Fac. Ingegneria. Progettazione del Software I. Soluzione compito 2007-07-03 35

```

public String toString() {
    return messaggio;
}
}
}

```