SAPIENZA Università di Roma

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Esercitazioni di

# PROGETTAZIONE DEL SOFTWARE

(Corso di Laurea in Ingegneria Informatica ed Automatica

Corso di Laurea in Ingegneria dei Sistemi Informatici

A.A. 2010/11

## Soluzione dell'esercitazione basata sul compito d'esame del 26 febbraio 2010

# Requisiti

L'applicazione da progettare riguarda la gestione di squadre e giocatori di una lega professionistica. I giocatori sono caratterizzati da un nome (una stringa) ed un anno di nascita (un intero). Le squadre sono caratterizzate da un nome (una stringa). In una squadra giocano almeno 15 giocatori. Tra i giocatori che giocano in una squadra, uno gioca nel ruolo di capitano. Delle squadre alcune sono neopromosse e in esse giocano esattamente 15 giocatori (vincolo non esprimibile). Le squadre si incontrano tra di loro in partite, di cui interessa conoscere quale squadra gioca in casa e quale in trasferta, ed il risultato (due interi: uno per la squadra di casa, ed uno per quella in trasferta). Non possono esserci due partite in cui le stesse due squadre giocano con gli stessi ruoli in entrambe. Data una squadra è d'interesse conoscere le squadre con cui si è incontrata sia in casa che in trasferta.

# Requisiti (cont.)

Siamo interessati a progettare la seguente attività: ripetutamente fino a quando l'utente lo richiede, l'utente tramite un'attività di I/O indica una partita restituendo il link corrispondente. Quindi concorrentemente si procede con le seguenti sottoattività: (i) si chiede tramite una operazione di I/O se operare sulla squadra in trasferta o sulla squadra di casa e si calcola la media delle età dei giocatori della squadra indicata, dopodichè si chiede all'utente se vuole operare con l'altra squadra e, in caso affermativo, si calcola la media delle età dei giocatori anche per questa; (ii) si calcola il numero delle partite vinte (escluse quella indicata) dalla squadra in trasferta e il numero delle partire vinte (esclusa quella indicata) dalla squadra in casa. Una volta completate entrambe queste sottoattività, si produce una pagina html con il report che mostra le informazioni calcolate.

# Requisiti (cont.)

**Domanda 1.** Basandosi sui requisiti riportati sopra, effettuare la fase di analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi, diagramma delle attività, specifica delle attività atomiche che operano sul diagramma delle classi (i task), motivando, qualora ce ne fosse bisogno, le scelte effettuate. La specifica delle attività di I/O non è richiesta.

**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 richiesto di definire solo le responsabilità su tutte le associazioni del diagramma delle classi ed il progetto dell'algoritmo dell'attività atomica (task) di calcolo della media delle età dei giocatori di una squadra.

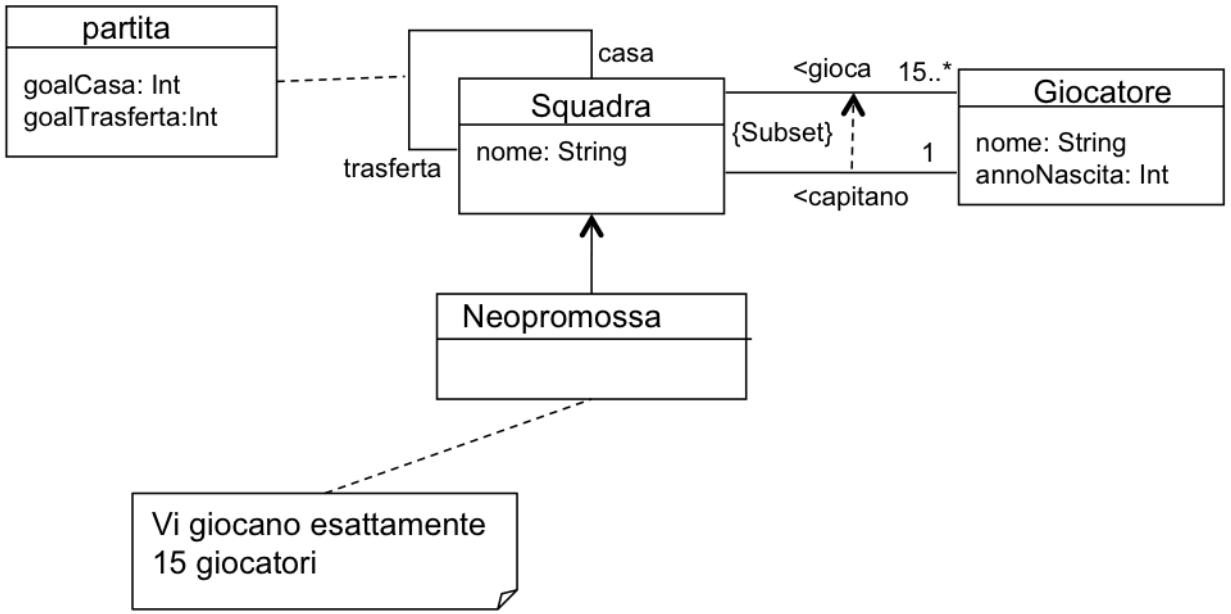
**Domanda 3.** Effettuare la fase di realizzazione, producendo un programma Java e motivando, qualora ce ne fosse bisogno, le scelte effettuate. È richiesto di realizzare in Java solo i seguenti aspetti dello schema concettuale:

- La classe Squadra e tutte le associazioni a cui partecipa (ignorando i vincoli di subset sulle associazioni a cui partecipa).

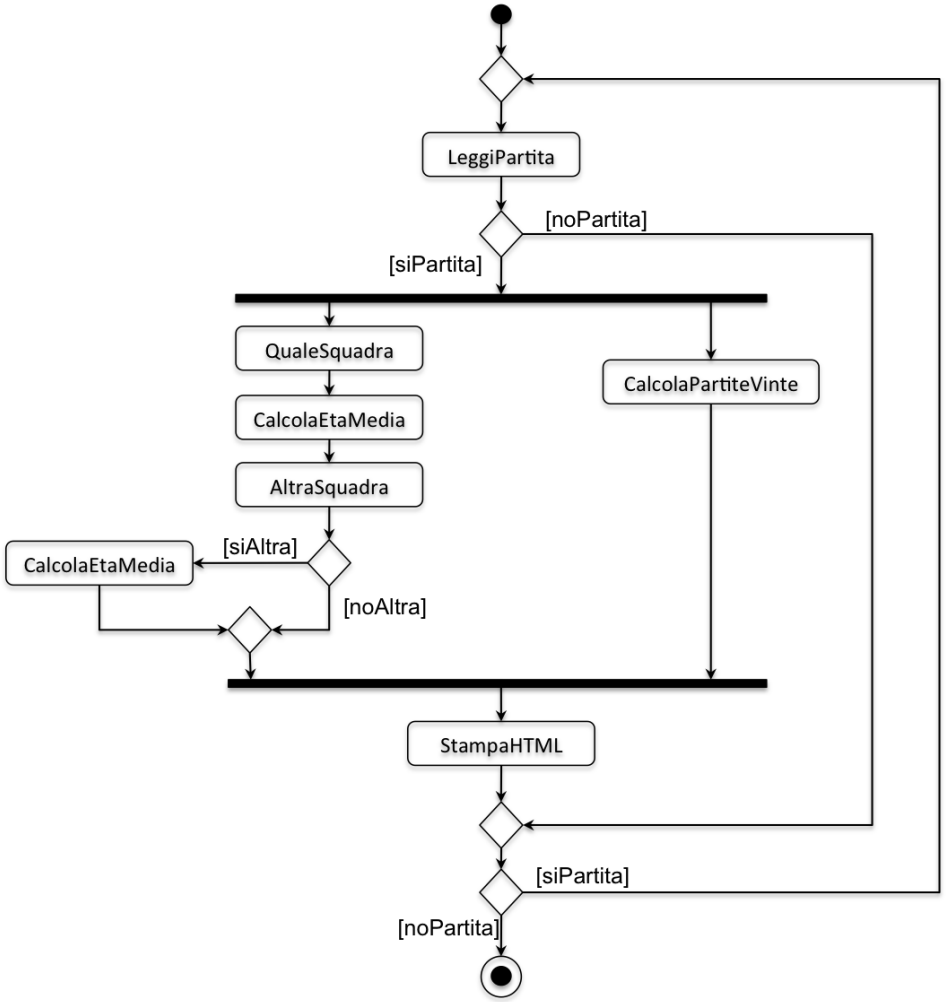
- L'attività principale, le eventuali sottoattività non atomiche, l'attività atomica (task) di calcolo della media delle età dei giocatori di una squadra –si assuma che l'anno corrente sia 2010. Le altre sottoattività non vanno realizzate.

# Fase di analisi

# Diagramma delle classi



# Diagramma delle attività



# Specifica delle classi

Non sono presenti operazioni di classe.

# Specifica delle attività

## Attività di I/O

InizioSpecificaAttivitàAtomica LeggiPartita
    LeggiPartita () : (Partita)
        pre: –
        post: mostra la lista di tutte le partite,
            se l'utente seleziona una partita, `result` è la partita selezionata
            se l'utente non seleziona alcuna partita, `result=null`
FineSpecifica

InizioSpecificaAttivitàAtomica QualeSquadra
    QualeSquadra (p:partita) : (Squadra)
        pre: –
        post: mostra all'utente le opzioni *casa* e *trasferta*
            se l'utente sceglie *casa*, `result` è la squadra con ruolo *casa* in p
            se l'utente sceglie *trasferta*, `result` è la squadra con ruolo *trasferta* in p
FineSpecifica

# Specifica delle attività (cont.)

InizioSpecificaAttivitàAtomica AltraSquadra
    AltraSquadra () : (Bool)
        pre: –
        post: mostra all'utente le opzioni *sì* e *no*
            se l'utente sceglie *sì*, `result=true`
            se l'utente sceglie *no*, `result=false`
FineSpecifica

InizioSpecificaAttivitàAtomica StampaHTML
    StampaHTML (vinteCasa : Int, vinteTrasferta : Int,
                    mediaEtaCasa: Real, mediaEtaTrasferta: Real)
        pre: mediaEtaCasa $> 0 \vee$ mediaEtaTrasferta $> 0$
        post: mostra le informazioni in HTML
            se mediaEtaCasa$<= 0$ non mostra questa informazione
            se mediaEtaTrasferta$<= 0$ non mostra questa informazione
FineSpecifica

# Specifica delle attività (cont.)

## Attività atomiche (task)

InizioSpecificaAttivitàAtomica CalcolaEtaMedia

CalcolaEtaMedia (s:Squadra) : (Real)

pre: –

post: sia $G = \{g \in Giocatore \mid \langle g, s\rangle \in gioca\}$

$$\texttt{result} = \frac{\sum_{g \in G}(2010 - g.annoNascita)}{|G|}$$

FineSpecifica

# Specifica delle attività (cont.)

InizioSpecificaAttivitàAtomica CalcolaPartiteVinte

CalcolaPartiteVinte (p:partita) : (vinteCasa : Int, vinteTrasferta : Int)

pre: –

post:

sia $c$ la squadra che partecipa al link p con ruolo *casa*, e siano:

$P_c = \{q \in partita \setminus \{p\} \mid q.casa = c \vee q.trasferta = c\}$ (partite diverse da p giocate da $c$)

$V_c^c = \{q \in P_c \mid q.casa = c \wedge q.goalCasa > q.goalTrasferta\}$ (partite in $P_c$ vinte da $c$, in casa)

$V_c^t = \{q \in P_c \mid q.trasferta = c \wedge q.goalTrasferta > q.goalCasa\}$

(partite in $P_c$ vinte da $c$, in trasferta)

inoltre, sia $t$ la squadra che partecipa al link p con ruolo *trasferta*, e siano:

$P_t = \{q \in partita \setminus \{p\} \mid q.casa = t \vee q.trasferta = t\}$ (partite diverse da p giocate da $t$)

$V_t^c = \{q \in P_t \mid q.casa = t \wedge q.goalCasa > q.goalTrasferta\}$ (partite in $P_t$ vinte da $t$, in casa)

$V_t^t = \{q \in P_t \mid q.trasferta = t \wedge q.goalTrasferta > q.goalCasa\}$

(partite in $P_t$ vinte da $t$, in trasferta)

allora:

vinteCasa= $|V_c^c \cup V_c^t|$

vinteTrasferta= $|V_t^c \cup V_t^t|$

FineSpecifica

# Specifica delle attività (cont.)

## Attività complesse

InizioSpecificaAttività sottoramoSx

```
    sottoramoSx (p:partita) : (mediaEtaCasa: Real, mediaEtaTrasferta: Real)

    Variabili Processo
        squadraSelezionata : Squadra
        altra : Bool

    InizioProcesso
        mediaEtaCasa = -1;
        mediaEtaTrasferta = -1;
        QualeSquadra(p) : (squadraSelezionata);
        if (p.casa == squadraSelezionata)
            CalcolaEtaMedia(squadraSelezionata) : (mediaEtaCasa);
        else
            CalcolaEtaMedia(squadraSelezionata) : (mediaEtaTrasferta);
        AltraSquadra() : (altra);
        if (altra){
            if (p.casa == squadraSelezionata)
                CalcolaEtaMedia(p.fuori) : mediaEtaTrasferta;
            else
                CalcolaEtaMedia(p.casa) : mediaEtaCasa;
        }

    FineProcesso
FineSpecifica
```

# Specifica delle attività (cont.)

InizioSpecificaAttività sottoramoDx

sottoramoDx (p:partita) : (vinteCasa: Int, vinteTrasferta: Int)

Variabili Processo
–

InizioProcesso
CalcolaPartiteVinte(p) : (vinteCasa,vinteTrasferta);
FineProcesso

FineSpecifica

# Specifica delle attività (cont.)

```
InizioSpecificaAttività attivitaPrincipale
    attivitaPrincipale () : ()

    Variabili Processo
        partitaSelezionata : Partita
        mediaEtaCasa : Real
        mediaEtaTrasferta : Real
        vinteCasa : Int
        vinteTrasferta : Int

    InizioProcesso
        do{
            LeggiPartita():(partitaSelezionata);
            if(partitaSelezionata != null){
                fork {
                    thread t1 : sottoramoSx(partitaSelezionata):(mediaEtaCasa,mediaEtaTrasferta);
                    thread t2 : sottoramoDx(partitaSelezionata):(vinteCasa,vinteTrasferta);
                }
                join t1, t2;
                StampaHTML (vinteCasa, vinteTrasferta, mediaEtaCasa, mediaEtaTrasferta);
            }
        } while(partitaSelezionata != null)
    FineProcesso
FineSpecifica
```

# Fase di progetto

# Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

- 1. requisiti
- 2. operazioni di classe ed attività
- 3. vincoli di molteplicità nel diagramma delle classi

| Associazione | Classe (o ruolo) | ha resp. |
|---|---|---|
| *partita* | *casa* | Sì[1,2] |
| | *trasferta* | Sì[1,2] |
| *gioca* | *Squadra* | Sì[3] |
| | *Giocatore* | NO |
| *capitano* | *Squadra* | Sì[3] |
| | *Giocatore* | NO |

# Strutture di dati

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

- dei vincoli di molteplicità 15..* delle associazioni,
- delle variabili necessarie per vari algoritmi.

Per fare ciò, utilizzeremo le classi del Java Collection Framework : `Set`, `HashSet`.

# Corrispondenza fra tipi UML e Java

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

| Tipo UML | Rappresentazione in Java |
|---|---|
| Int | `int` |
| Real | `float` |
| String | `String` |
| boolean | `boolean` |
| 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** |
|---|---|
| *Squadra* | *nome* |
| *Giocatore* | *nome* |
| | *annoNascita* |

| **Classe** UML | **Proprietà** | |
|---|---|---|
| | **nota alla nascita** | **non nota alla nascita** |
| *Squadra* | – | *capitano* |

# Altre considerazioni

**Molteplicità:** Dobbiamo assicurare che le istanze di Neopromossa contengano esattamente 15 giocatori.

**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

# Considerazioni iniziali

La traccia ci richiede di realizzare:

1. La classe *Squadra*.

2. le associazioni *partita*, *gioca*, *capitano*.

3. le attività *AttivitaPrincipale* e *CalcolaEtaMedia* (assumendo che l'anno corrente sia il 2010).

Nel seguito verranno realizzate tutte le classi e le attività individuate in fase di analisi.

# Struttura dei file e dei package

```
AppLega
  +---lega
  |      | TipoLinkPartita.java
  |      | ManagerPartita.java
  |      | EccezioneMolteplicita.java
  |      | EccezioneSubset.java
  |      | EccezionePrecondizioni.java
  |      |
  |      +---giocatore
  |      |       Giocatore.java
  |      |
  |      +---squadra
  |      |       Squadra.java
  |      |
  |      \---neopromossa
  |                Neopromossa.java
  |
  \---attivita
          +---atomiche
          |      CalcolaEtaMedia.java
          |      CalcolaPartiteVinte.java
          |
          \---complesse
                  AttivitaPrincipale.java
                  SottoramoDx.java
                  SottoramoSx.java
```

# La classe Java Giocatore

```
package lega.giocatore;

public class Giocatore {
        private final String nome;
        private final int annoNascita;

        public Giocatore(String nome, int annoNascita){
                this.nome = nome;
                this.annoNascita = annoNascita;
        }

        public String getNome(){
                return nome;
        }

        public int getAnnoNascita(){
                return annoNascita;
        }
}
```

# La classe Java Squadra

```
package lega.squadra;

import java.util.*;

import lega.EccezioneMolteplicita;
import lega.EccezioneSubset;
import lega.ManagerPartita;
import lega.TipoLinkPartita;
import lega.giocatore.Giocatore;

public class Squadra {
        private final int MOLT_MIN_GIOCA = 15;

        private final String nome;
        private HashSet<Giocatore> gioca;
        private Giocatore capitano;
        private HashSet<TipoLinkPartita> casa;
        private HashSet<TipoLinkPartita> trasferta;

        public Squadra(String nome){
                casa = new HashSet<TipoLinkPartita>();
                trasferta = new HashSet<TipoLinkPartita>();
                gioca = new HashSet<Giocatore>();
                this.nome = nome;
        }

        public String getNome(){
                return nome;
        }
```

```
public Set<Giocatore> getLinkGioca() throws EccezioneMolteplicita{
        if (gioca.size() < MOLT_MIN_GIOCA){
                throw new EccezioneMolteplicita("Squadra " + nome + ": Molteplicita' minima violata!")
        }
        return (Set)gioca.clone();
}

public void insericiLinkGioca(Giocatore g){
        gioca.add(g);
}

public void eliminaLinkGioca(Giocatore g){
        gioca.remove(g);
}

public void setCapitano(Giocatore capitano){
        this.capitano = capitano;
}

public Giocatore getCapitano() throws EccezioneMolteplicita, EccezioneSubset {
        if (capitano == null){
                throw new EccezioneMolteplicita("Molteplicita' min/max violata!");
        }
        if (!gioca.contains(capitano)){
                throw new EccezioneSubset("Vincolo di Subset violato!");
        }
        return capitano;
}

public Set<TipoLinkPartita> getLinkCasa(){
```

```
        return (Set<TipoLinkPartita>) casa.clone();
}

public Set<TipoLinkPartita> getLinkTrasferta(){
        return (Set<TipoLinkPartita>) trasferta.clone();
}

public void inserisciLinkPartita(TipoLinkPartita l){
        if (l != null && l.getCasa() == this || l.getTrasferta() == this){
                ManagerPartita.inserisci(l);
        }
}

public void eliminaLinkPartita(TipoLinkPartita l){
        if (l != null && l.getCasa() == this || l.getTrasferta() == this){
                ManagerPartita.elimina(l);
        }
}

public void inserisciCasaPerManagerPartita(ManagerPartita m){
        if(m != null){
                casa.add(m.getLink());
        }
}

public void inserisciTrasfertaPerManagerPartita(ManagerPartita m){
        if(m != null){
                trasferta.add(m.getLink());
        }
}
```

```
        public void eliminaCasaPerManagerPartita(ManagerPartita m){
                if(m != null){
                        casa.remove(m.getLink());
                }
        }

        public void eliminaTrasfertaPerManagerPartita(ManagerPartita m){
                if(m != null){
                        trasferta.remove(m.getLink());
                }
        }
}
```

# La classe Java Neopromossa

```
package lega.neopromossa;

import java.util.*;

import lega.EccezioneMolteplicita;
import lega.giocatore.Giocatore;
import lega.squadra.Squadra;

public class Neopromossa extends Squadra {
        private final int MOLT_MIN_MAX = 15;

        public Neopromossa(String nome){
                super(nome);
        }

        public Set<Giocatore> getLinkGioca() throws EccezioneMolteplicita{
                Set<Giocatore> gioca = super.getLinkGioca();
                if(gioca.size() != MOLT_MIN_MAX){
                        throw new EccezioneMolteplicita("Molteplicita' min/max violata!");
                }
                return gioca;
        }
}
```

# La classe Java TipoLinkPartita

```
package lega;

import lega.squadra.Squadra;

public class TipoLinkPartita {
        private final int goalCasa;
        private final int goalTrasferta;
        private final Squadra casa;
        private final Squadra trasferta;

        public TipoLinkPartita(Squadra casa, int goalCasa, Squadra trasferta,int goalTrasferta)
                                                                throws EccezionePrecondizioni{
                if (casa == null || trasferta == null){
                        throw new EccezionePrecondizioni("Oggetti non inizializzati!");
                }
                this.casa = casa;
                this.trasferta = trasferta;
                this.goalCasa = goalCasa;
                this.goalTrasferta = goalTrasferta;
        }

        // Astrazione di tipo: ridefiniamo equals
        public boolean equals(Object o){
                if (o != null && getClass().equals(o.getClass())){
                        TipoLinkPartita l = (TipoLinkPartita) o;
                        return (casa == l.casa && trasferta == l.trasferta);
                }
                return false;
        }
```

```
        // equals() ridefinito -> dobbiamo ridefinire hashCode()
        public int hashCode(){
                return casa.hashCode() + trasferta.hashCode();
        }

        public Squadra getCasa(){
                return casa;
        }

        public Squadra getTrasferta(){
                return trasferta;
        }

        public int getGoalCasa(){
                return goalCasa;
        }

        public int getGoalTrasferta(){
                return goalTrasferta;
        }

}
```

# La classe Java ManagerPartita

```
package lega;

public class ManagerPartita {
        private TipoLinkPartita link;

        private ManagerPartita(TipoLinkPartita link){
                this.link = link;
        }

        public TipoLinkPartita getLink(){
                return link;
        }

        public static void inserisci(TipoLinkPartita l){
                if (l != null){
                        ManagerPartita m = new ManagerPartita(l);
                        l.getCasa().inserisciCasaPerManagerPartita(m);
                        l.getTrasferta().inserisciTrasfertaPerManagerPartita(m);
                }
        }

        public static void elimina(TipoLinkPartita l){
                if (l != null){
                        ManagerPartita m = new ManagerPartita(l);
                        l.getCasa().eliminaCasaPerManagerPartita(m);
                        l.getTrasferta().eliminaTrasfertaPerManagerPartita(m);
                }
        }
}
```

# La classe Java CalcolaEtaMedia

```
package attivita.atomiche;

import java.util.*;
import _framework.*;
import lega.EccezioneMolteplicita;
import lega.squadra.*;
import lega.giocatore.*;

public class CalcolaEtaMedia implements Task {
        // Task per il calcolo dell'eta' media dei giocatori di una squadra
        private final int ANNO_CORRENTE = 2010;
        private Squadra squadra;
        private boolean eseguita = false;
        private float risultato = 0;

        public CalcolaEtaMedia(Squadra squadra){
                if(squadra == null){
                        throw (new RuntimeException("Il parametro non puo' essere null!"));
                }
                this.squadra = squadra;
        }

        public synchronized void esegui(Executor e){
                if (e == null || eseguita)
                        return;
                eseguita = true;
                try{
                        Set<Giocatore> giocatori = squadra.getLinkGioca();
                        Iterator<Giocatore> it = giocatori.iterator();
```

```
                        while(it.hasNext()){
                                risultato += (ANNO_CORRENTE - it.next().getAnnoNascita());
                        }
                        risultato = risultato/giocatori.size();
                }
                catch (EccezioneMolteplicita ecc){
                        ecc.printStackTrace();
                }
        }

        public synchronized boolean estEseguita(){
                return eseguita;
        }

        public synchronized float getRisultato(){
                if(!eseguita)
                        throw new RuntimeException("Risultato non disponibile!");
                return risultato;
        }
}
```

# La classe Java CalcolaPartiteVinte

```
package attivita.atomiche;

import java.util.*;
import _framework.*;
import lega.*;
import lega.squadra.*;

public class CalcolaPartiteVinte implements Task {
        private int vinteCasa = 0;
        private int vinteTrasferta = 0;
        private TipoLinkPartita partita;
        private boolean eseguita = false;

        public CalcolaPartiteVinte(TipoLinkPartita partita){
                if(partita == null){
                        throw (new RuntimeException("Il parametro non puo' essere null!"));
                }
                this.partita = partita;
        }

        public synchronized void esegui(Executor e){
                if (e == null || eseguita)
                        return;
                eseguita = true;

                // Crea l'insieme delle partite giocate dalla squadra di casa e
                // vi aggiunge le partite che essa ha giocato in casa e fuori.

                // insieme vuoto
```

```
        Set<TipoLinkPartita> giocateDaSquadraCasa = new HashSet<TipoLinkPartita>();
         // unione con le partite giocate in casa
        giocateDaSquadraCasa.addAll(partita.getCasa().getLinkCasa());
         // unione con le partite giocate in trasferta
        giocateDaSquadraCasa.addAll(partita.getCasa().getLinkTrasferta());
         // Rimuove la partita fornita in input all'attivita'
        giocateDaSquadraCasa.remove(partita);

        // Calcola le partite vinte dalla squadra in casa
        vinteCasa = partiteVinte(partita.getCasa(),giocateDaSquadraCasa);

        // (Come sopra, per la squadra in trasferta)
        Set<TipoLinkPartita> giocateDaSquadraTrasferta = new HashSet<TipoLinkPartita>();
        giocateDaSquadraTrasferta.addAll(partita.getTrasferta().getLinkCasa());
        giocateDaSquadraTrasferta.addAll(partita.getTrasferta().getLinkTrasferta());
        giocateDaSquadraTrasferta.remove(partita);

        // Calcola le partite vinte dalla squadra in trasferta
        vinteTrasferta = partiteVinte(partita.getTrasferta(),giocateDaSquadraTrasferta);
}

private int partiteVinte(Squadra s, Set<TipoLinkPartita> partite ){
        /*Data una squadra s ed un insieme di partite, calcola quante delle ultime sono
         * state vinte da s
         * ATTENZIONE! Metodo privato (ausiliario): puo' essere eseguito solo dal Task,
         * che lo fara' all'interno di un blocco/metodo synchronized!
        */
        int risultato = 0;
        Iterator<TipoLinkPartita> itp = partite.iterator();
        while(itp.hasNext()){
                TipoLinkPartita partitaCorrente = itp.next();
```

```
                if(((//la squadra s ha giocato in casa e la squadra in casa ha vinto
                                partitaCorrente.getCasa() == s &&
                                (partitaCorrente.getGoalCasa() > partitaCorrente.getGoalTrasferta())
                        )
                        ||
                        (//la squadra s ha giocato in trasferta e la squadra in trasferta ha vinto
                                partitaCorrente.getTrasferta() == s &&
                                (partitaCorrente.getGoalTrasferta() > partitaCorrente.getGoalCasa())
                        )
                ){
                        risultato++;
                }
        }

        return risultato;
}

public synchronized boolean estEseguita(){
        return eseguita;
}

public synchronized int getVinteCasa(){
        if(!eseguita)
                throw new RuntimeException("Risultato non disponibile!");
        return vinteCasa;
}

public synchronized int getVinteTrasferta(){
        if(!eseguita)
                throw new RuntimeException("Risultato non disponibile!");
        return vinteTrasferta;
```

```
        }
}
```

# La classe Java AttivitaPrincipale

```
package attivita.complesse;

import lega.*;
import gui.*;

public class AttivitaPrincipale implements Runnable{
        boolean eseguita = false;

        public synchronized void run(){
                if (eseguita)
                        return;
                eseguita = true;

                TipoLinkPartita partitaSelezionata = null;

                do{
                        partitaSelezionata = AttivitaIO.LeggiPartita();

                        if (partitaSelezionata != null){
                                SottoramoSx sottoramoSx = new SottoramoSx(partitaSelezionata);
                                SottoramoDx sottoramoDx = new SottoramoDx(partitaSelezionata);
                                Thread t1 = new Thread(sottoramoSx);
                                Thread t2 = new Thread(sottoramoDx);
                                t1.start();
                                t2.start();
                                try{
                                        t1.join();
                                        t2.join();
                                }
```

```
                                catch (InterruptedException e){
                                        e.printStackTrace();
                                }
                                AttivitaIO.stampaHTML(sottoramoDx.getVinteCasa(),
                                                sottoramoDx.getVinteTrasferta(),
                                                sottoramoSx.getMediaEtaCasa(),
                                                sottoramoSx.getMediaEtaTrasferta());
                        }
                }
                while(partitaSelezionata != null);
        }

        synchronized boolean estEseguita(){
                return eseguita;
        }

}
```

# La classe Java SottoramoDx

```
package attivita.complesse;

import lega.*;
import attivita.atomiche.*;
import _framework.*;

public class SottoramoDx implements Runnable{
        private boolean eseguita = false;
        private int vinteCasa = 0;
        private int vinteTrasferta = 0;
        private TipoLinkPartita partita;

        public SottoramoDx(TipoLinkPartita partita){
                this.partita = partita;
        }

        public synchronized void run(){
                if (eseguita)
                        return;
                eseguita = true;

                CalcolaPartiteVinte cpv = new CalcolaPartiteVinte(partita);
                Executor.perform(cpv);
                vinteCasa = cpv.getVinteCasa();
                vinteTrasferta = cpv.getVinteTrasferta();
        }

        public synchronized int getVinteCasa(){
                if (!eseguita)
```

```
                throw new RuntimeException("Il risultato non e' ancora pronto!");
        return vinteCasa;
    }

    public synchronized int getVinteTrasferta(){
        if (!eseguita)
                throw new RuntimeException("Il risultato non e' ancora pronto!");
        return vinteTrasferta;
    }

    public synchronized boolean estEseguita(){
        return eseguita;
    }
}
```

# La classe Java SottoramoSx

```
package attivita.complesse;

import lega.squadra.*;
import lega.*;
import gui.*;
import attivita.atomiche.*;
import _framework.*;

public class SottoramoSx implements Runnable{
        private boolean eseguita = false;
        private float mediaEtaCasa = -1;
        private float mediaEtaTrasferta = -1;
        private TipoLinkPartita partita;

        public SottoramoSx(TipoLinkPartita partita){
                this.partita = partita;
        }

        public synchronized void run(){
                if (eseguita)
                        return;
                eseguita = true;

                Squadra squadraSelezionata = null;
                boolean altra = false;

                squadraSelezionata = AttivitaIO.qualeSquadra(partita);
                CalcolaEtaMedia cem = new CalcolaEtaMedia(squadraSelezionata);
                Executor.perform(cem);
```

```
        if (squadraSelezionata == partita.getCasa()){// E' stata selezionata la squadra in casa
                mediaEtaCasa = cem.getRisultato();
        }
        else{// E' stata selezionata la squadra in trasferta
                mediaEtaTrasferta = cem.getRisultato();
        }

        altra = AttivitaIO.altraSquadra();
        if (altra){
                if (squadraSelezionata == partita.getCasa()){// era stata selezionata la squadra di ca
                        // dobbiamo calcolare l'eta' media della squadra in trasferta
                        CalcolaEtaMedia cem2 = new CalcolaEtaMedia(partita.getTrasferta());
                        Executor.perform(cem2);
                        mediaEtaTrasferta = cem2.getRisultato();
                }
                else{// era stata selezionata la squadra in trasferta
                        // dobbiamo calcolare l'eta' media della squadra in casa
                        CalcolaEtaMedia cem2 = new CalcolaEtaMedia(partita.getCasa());
                        Executor.perform(cem2);
                        mediaEtaCasa = cem2.getRisultato();
                }
        }

}

public synchronized float getMediaEtaCasa(){
        if (!eseguita)
                throw new RuntimeException("Il risultato non e' ancora pronto!");
        return mediaEtaCasa;
}
```

```
        public synchronized float getMediaEtaTrasferta(){
                if (!eseguita)
                        throw new RuntimeException("Il risultato non e' ancora pronto!");
                return mediaEtaTrasferta;
        }

        public synchronized boolean estEseguita(){
                return eseguita;
        }
}
```

# Realizzazione in Java delle classi per eccezioni

```
package lega;

public class EccezioneMolteplicita extends Exception {
        private String messaggio;

        public EccezioneMolteplicita(String messaggio) {
                this.messaggio = messaggio;
        }
        public String toString() {
                return messaggio;
        }
}

package lega;

public class EccezioneSubset extends Exception{
        private String messaggio;

        public EccezioneSubset(String messaggio){
                this.messaggio=messaggio;
        }

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

package lega;
```

```
public class EccezionePrecondizioni extends Exception{
        private String messaggio;

        public EccezionePrecondizioni(String messaggio){
                this.messaggio=messaggio;
        }

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