UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea triennale in Ingegneria Informatica

Ingegneria del Software

Elaborato

Kevin Maggi

6128567

INTRODUZIONE

SCENARIO

Si vuole creare il sottosistema informatico della Federazione Italiana Pallacanestro che riguarda la gestione delle designazioni degli arbitri per le partite e successivamente il pagamento degli stessi.

L'Ufficio Gare all'inizio di ogni campionato definisce le partite che verranno disputate sotto la gestione della Federazione. Ad ogni partita devono essere associati 2 arbitri, 3 ufficiali di campo (da qui in avanti UDC) e un osservatore. L'Ufficio Designazioni ogni giorno fa le designazioni per le partite che si svolgeranno 15 giorni dopo.

Ai tesserati CIA (arbitri, UDC e osservatori) che dirigono una partita è riconosciuto il pagamento di un gettone di 25€, di un rimborso viaggio di 0.25€/km ed eventualmente alcuni extra: 10€ se il luogo della partita dista più di 100km dal luogo di residenza, 5€ se si tratta di una partita di playoff, 10€ se l'orario della partita è a ridosso di un pasto. Il tesserato CIA che intende ricevere il pagamento deve, prima della partita, inviare il modulo di rimborso con il totale richiesto e la descrizione, attraverso la propria interfaccia.

Dopo la partita uno degli arbitri dovrà inviare il referto di gara (una scansione di quello cartaceo compilato durante la gara) al sistema. Il giorno dopo il Giudice Sportivo analizza il referto e omologa il risultato.

Periodicamente l'Ufficio Amministrativo liquida i rimborsi richiesti dai tesserati relativi alle partite già omologate. Ogni tesserato può vedere lo stato dei rimborsi richiesti attraverso la propria interfaccia ed eventualmente decidere in che modo gli verrà inviata la notifica di effettuato pagamento tra mail ed SMS.

DESIGN PATTERNS

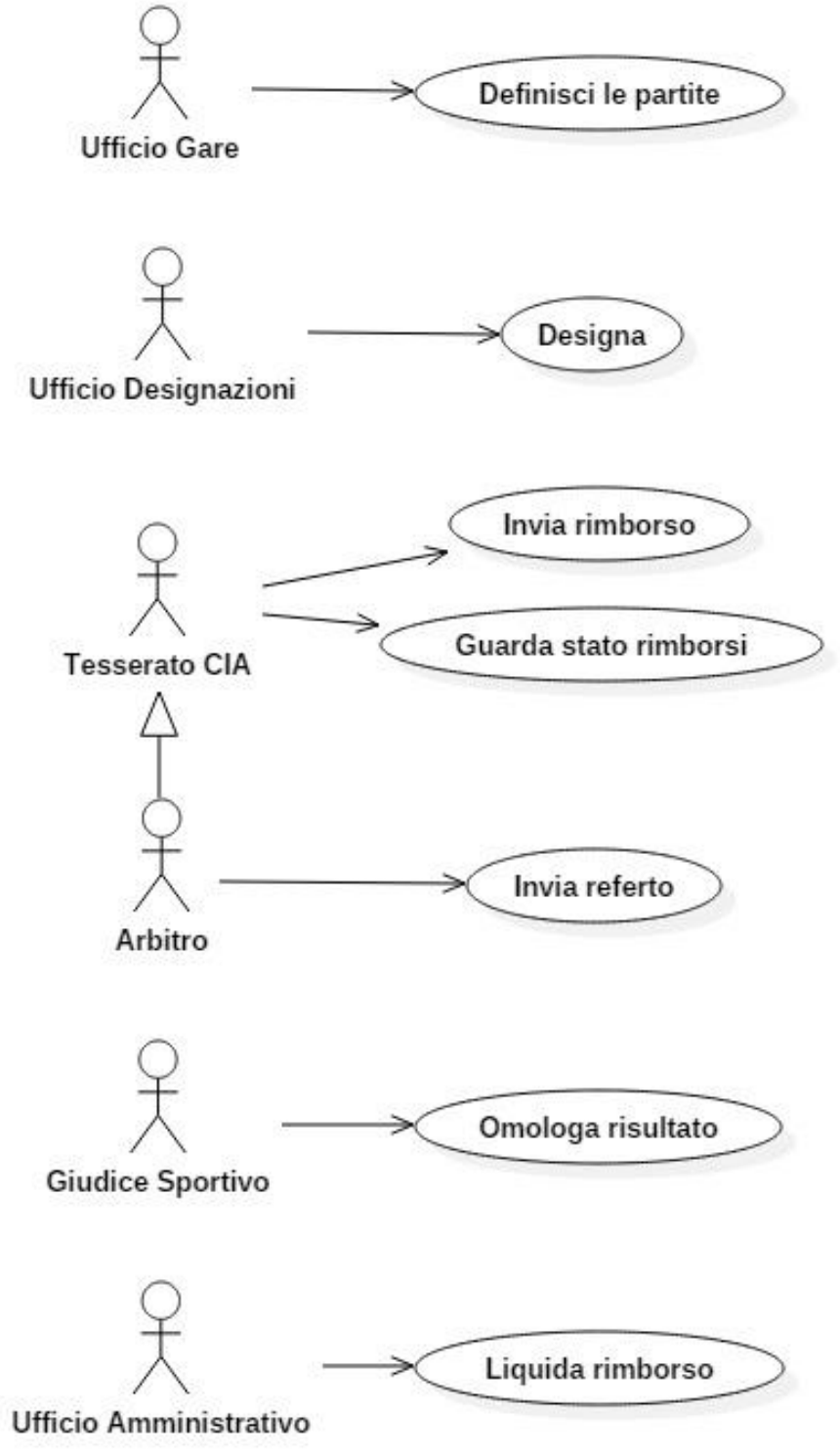
Per la gestione del modulo di rimborso, dalla creazione alla notifica di effettuato pagamento, risulterà utile ed efficace utilizzare nella progettazione del sistema i seguenti design patterns:

- **Decorator** per quanto riguarda la fase di creazione, con gli eventuali extra;
- **Observer** per tenere aggiornata l'interfaccia;
- **Strategy** per i vari metodi di notifica.

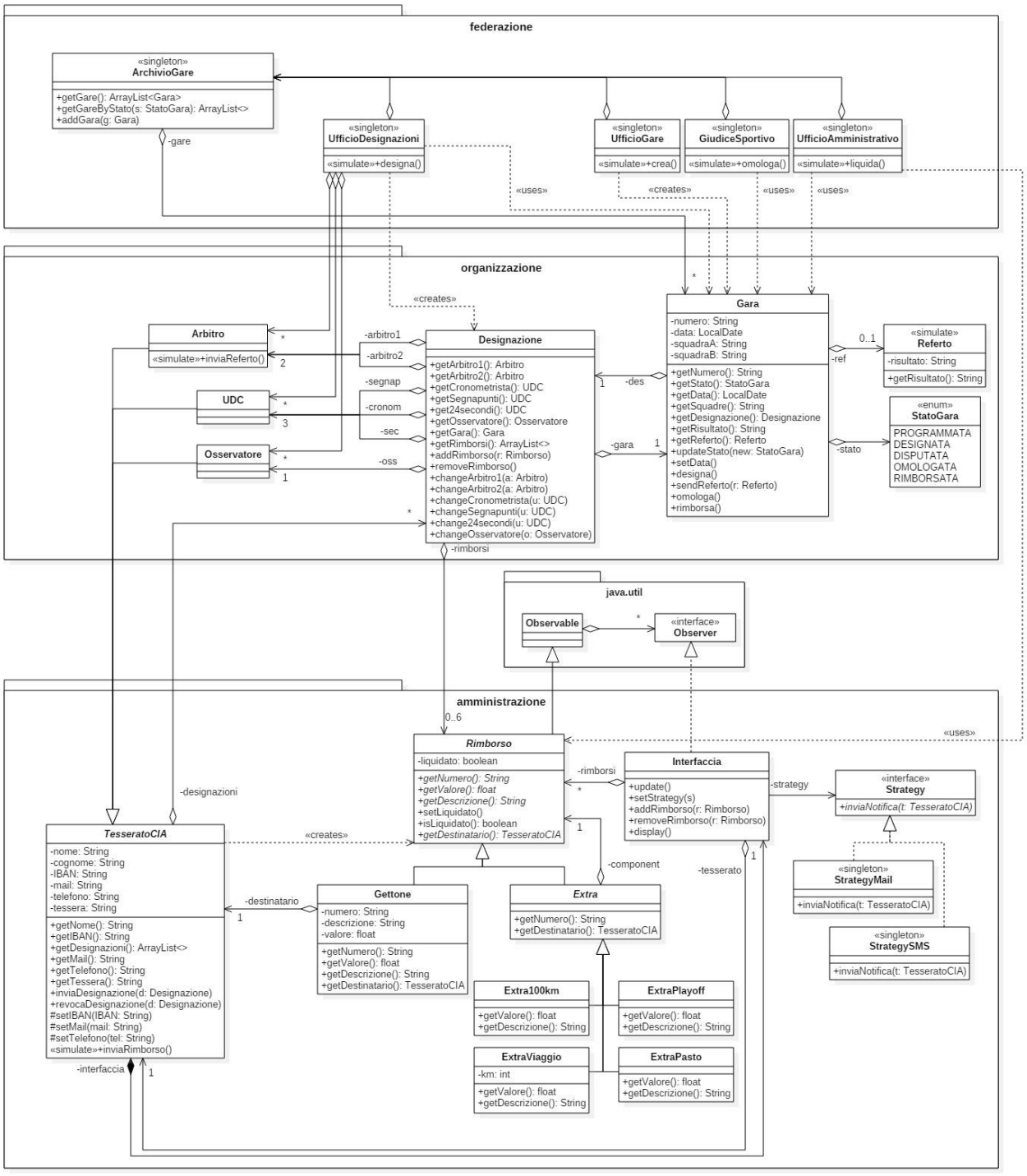
Approfondiremo successivamente questo aspetto.

PROGETTO

USE CASE DIAGRAM



CLASS DIAGRAM



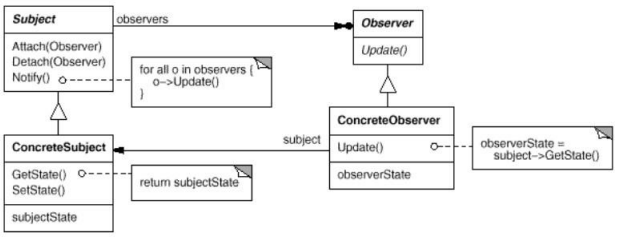
NOTA: Lo stereotipo <<simulate>> indica che l'implementazione del metodo o classe è ad hoc al fine della simulazione dell'azione umana e del test

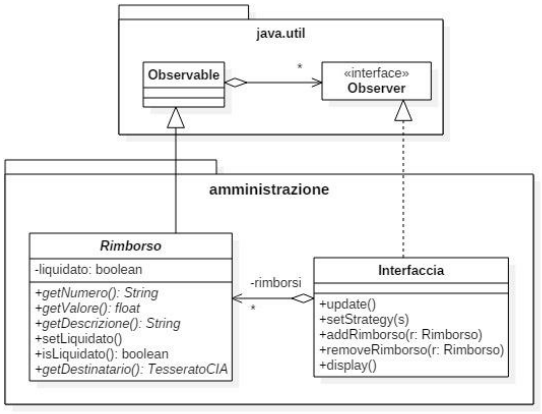
Partecipanti

- **ArchivioGare**: ha il compito di collezionare tutte le partite della federazione, espone il metodo *addGara()* per aggiungere una gara e i metodi *GetGare()* e *getGareByStato()* per selezionarle;
- **UfficioGare**: è implementato come singleton ed espone il metodo <<*simulate*>>*crea()*, per creare le gare;
- **UfficioDesignazioni**: è implementato come singleton ed espone il metodo <<*simulate*>>*designa()*, per designare le gare;
- **GiudiceSportivo**: è implementato come singleton ed espone il metodo <<*simulate*>>*omologa()*, per omologare le gare;
- **UfficioAmministrazione**: è implementato come singleton ed espone il metodo <<*simulate*>>*liquida()*, per liquidare i rimborsi delle gare;
- **Gara**: ha degli attributo contenenti le informazioni della gara, inoltre ha un referto e una designazione e i rispettivi metodi *sendReferto()* e *designa()* per settarli al momento opportuno da parte di diversi attori; questi ultimi, insieme a *omologa()* e *rimborsa()* provvedono a cambiare lo stato dalla gara;
- **Referto**: simula la scansione del referto cartaceo che il primo arbitro invia al sistema;
- **Designazione**: ha i riferimenti ai vari arbitri, UDC e osservatore che dirigeranno la partita, (esponendo anche una serie di metodi per cambiare in un secondo momento questi attributi) e una collezione di rimborsi impostabili dai tesserati col metodo *addRimborso()*;
- **TesseratoCIA**: ha i suoi dati anagrafici, insieme all'IBAN utilizzato dall'ufficio amministrativo per effettuare il bonifico e una collezione di designazioni. I metodi *inviaDesignazione()* e *revocaDesignazione()* sono utilizzati dall'ufficio designazioni per inviare/revocare le gare;
- **Arbitro/UDC/Osservatore**: sono sottoclassi concrete di TesseratoCIA per distinguere i vari ruoli;
- **Rimborso**: è una classe astratta, ha un attributo boolean che ne indica lo stato e i metodi per settarlo e verificarlo, deriva da Observable;
- **Gettone**: sottoclasse di Rimborso, ha degli attributi che identificano il rimborso concreto, tra cui anche il riferimento al destinatario del rimborso;
- **Extra**: sottoclasse (astratta) di Rimborso, fornisce un'interfaccia per i vari tipi di decorator;
- **Extra100km/ExtraViaggio/ExtraPlayoff/ExtraPasto**: decorator concreti;
- **Interfaccia**: implementa l'interfaccia observer e tiene una collezione di subject (rimborsi) oltre a una Strategy, il metodo *update()* provvede a chiamare il metodo di notifica della strategy (se impostata), il metodo *display()* mostra i dettagli dei rimborsi;
- **Strategy**: interfaccia per le strategy concrete, espone il metodo *inviaNotifica()*;
- **StrategyMail/StrategySMS**: strategy concrete.

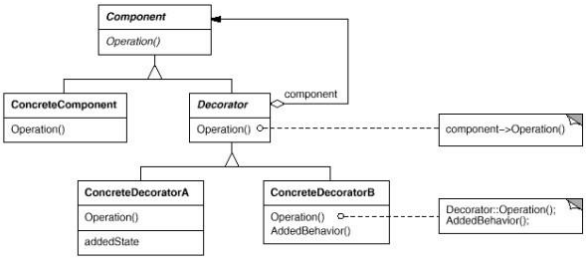
Design Patterns

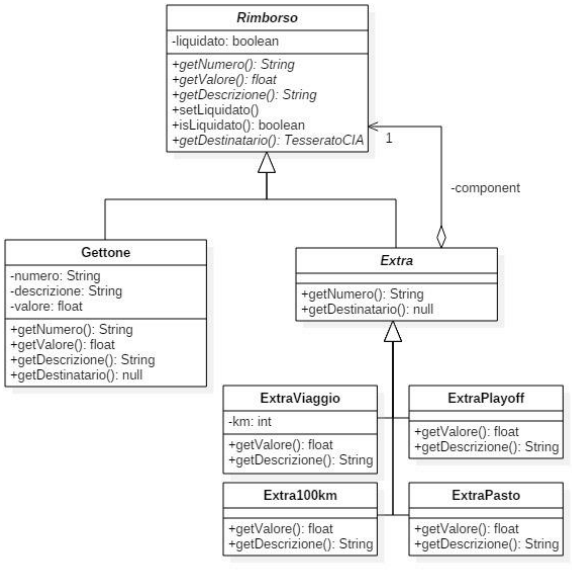
- **Observer**:
  il suo intento originale è quello di definire una dipendenza uno-a-molti così che quando lo stato del soggetto cambia, vengano notificati tutti i suoi observer per mantenere i loro stati allineati con quello del soggetto. In questa applicazione è stato parzialmente cambiata l'implementazione rispetto a quella standard in quanto l'intento differisce leggermente: un observer ha più soggetti e non ha uno stato interno. Vuole ricevere le notifiche quando il soggetto cambia per invocare delle azioni. È un'applicazione limite di questo pattern, ma risulta efficacie e, comunque, appropriata, soprattutto pensiamo se (con un approccio un po' upfront) che il sistema possa includere altri sottosistemi qua non presi in considerazione (ad esempio anche l'ufficio amministrativo potrebbe possedere una sua interfaccia che si sottoscrive ai rimborsi appena questi vengono inviati dei tesseratiCIA);



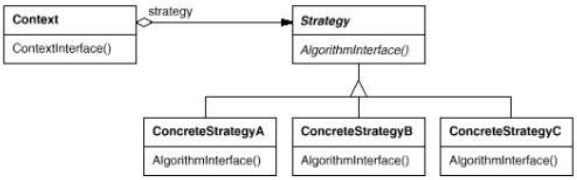


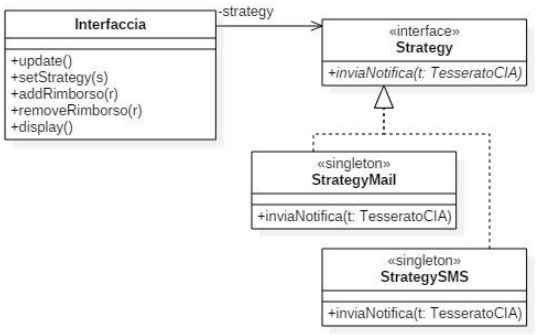
- **Decorator**:
  l'intento è quello di aggiungere dinamicamente responsabilità ad un oggetto; in questa implementazione ritroviamo a pieno questo intento: i decorator concreti aggiungono funzionalità al metodo getDescrizione e getValore(). È una soluzione molto versatile all'ereditarietà, perché il numero delle classi esploderebbe esponenzialmente col numero di possibili extra; anche la soluzione di un'unica classe rimborso che racchiudesse tutte le possibili opzioni non sarebbe una scelta particolarmente felice in quanto racchiudiamo in un'unica classe troppe responsabilità che è bene tenere divise;





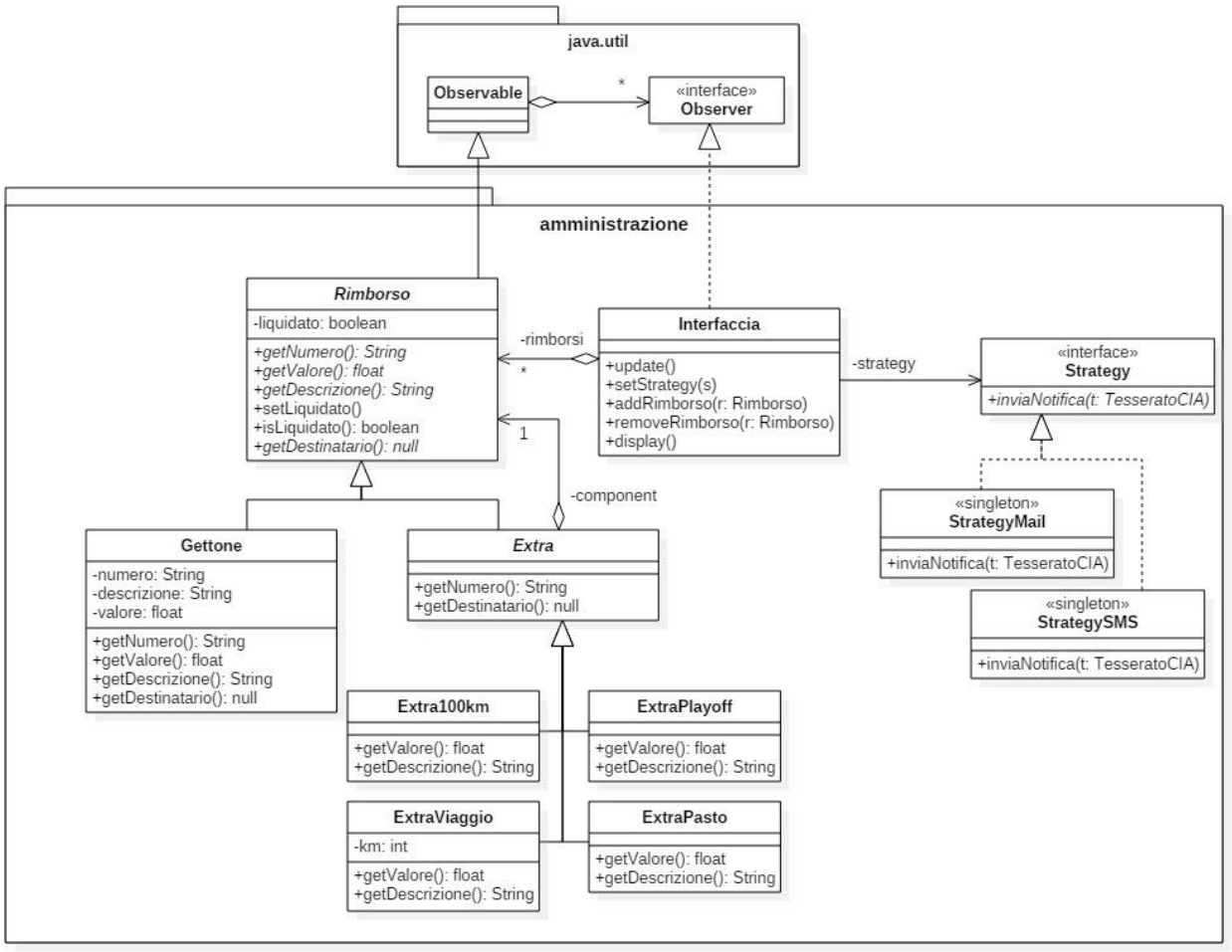
- **Strategy**: l'intento è quello di definire una famiglia di funzioni e renderle intercambiabili, in questo caso è l'invio della notifica che può essere fatto via mail o via SMS ed è possibile cambiare strategia a runtime.





Soluzione combinata

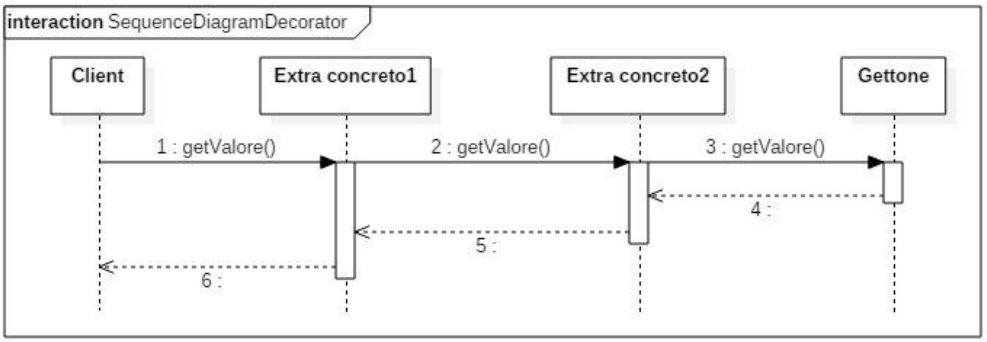
Questi 3 design pattern sono stati combinati così:



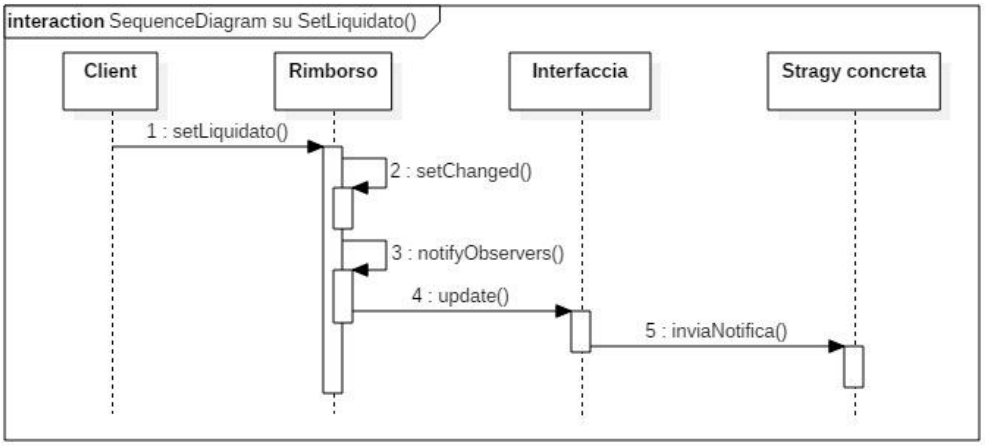
Rimborso è un component astratto per quanto riguarda il decorator e un subject per l'observer; Interfaccia è un observer per quanto riguarda il pattern observer e un context per quanto riguarda lo strategy.

Interfaccia ha una strategy e si sottoscrive ad una collezione di Rimborsi. Quando un Rimborso cambia stato (attraverso il metodo *setLiquidato()*), viene notificata Interfaccia, che a seconda dell'eventuale strategy settata invierà la notifica al tesserato in modi differenti. Rimborso non è un oggetto semplice: si tratta di un Gettone di base eventualmente decorato con comportamenti aggiuntivi, ma anche in un caso, con stato aggiuntivo.

Vediamo il sequence diagram sulla chiamata a *getValore()* (si tenga conto che funziona uguale anche per *getDescrizione()*, *getNumero()* e *getDestinatario()*, ma in questi ultimi due casi non c'è comportamento aggiuntivo, ma solo forward):



Vediamo anche quello certamente più interessante che riguarda il cambiamento di stato sulla chiamata *setLiquidato()*, dove entrano in gioco i pattern observer e strategy:



CODICE

ArchivioGare

```
package federazione;

import java.util.ArrayList;
import organizzazione.Gara;
import organizzazione.StatoGara;

public final class ArchivioGare {
	private final ArrayList<Gara> gare = new ArrayList<Gara>();
	private static ArchivioGare istance = null;

	private ArchivioGare() { };

	public final static ArchivioGare getIstance() {
		if (istance == null) {
		 istance = new ArchivioGare();
	    }
	    return istance;
	}

	public final ArrayList<Gara> getGare(){
		return new ArrayList<Gara>(gare);
	}
	public final void addGara(Gara g) {
		gare.add(g);
	}
	public final ArrayList<Gara> getGareByStato(StatoGara s){
		ArrayList<Gara> selected = new ArrayList<Gara>();
		for(Gara g:gare) {
			if(g.getStato()==s) {
				selected.add(g);
			}
		}
		return selected;
	}

}
```

UfficioGare

```
package federazione;

import java.time.LocalDate;
import organizzazione.Gara;

public final class UfficioGare {
	private static UfficioGare ufficio = null;
	private final ArchivioGare archivio = ArchivioGare.getIstance();

	private UfficioGare() { }

	public final static UfficioGare getIstance() {
		if (ufficio == null) {
	     ufficio = new UfficioGare();
	 }
	 return ufficio;
	}
```

```
	public void crea() {
		archivio.addGara(new Gara("1618", LocalDate.now().plusDays(3), "Pallacanestro
Vaiano", "Montemurlo Basket"));
		System.out.println("L'ufficio gare ha creato la partita n. 1618 tra Pallacanestro
Vaiano e Montemurlo Basket");
	}
}
```

UfficioDesignazioni

```
package federazione;

import java.util.ArrayList;
import organizzazione.*;

public final class UfficioDesignazioni {
	private static UfficioDesignazioni ufficio = null;
	private final ArchivioGare archivio = ArchivioGare.getIstance();
	private final ArrayList<Arbitro> arbitri = new ArrayList<Arbitro>();
	private final ArrayList<UDC> udc = new ArrayList<UDC>();
	private final ArrayList<Osservatore> osservatori = new ArrayList<Osservatore>();

	private UfficioDesignazioni() {	}

	public final static UfficioDesignazioni getIstance() {
		if (ufficio == null) {
		    ufficio = new UfficioDesignazioni();
		 }
		 return ufficio;
	}

	public final void addArbitro(Arbitro a) {
		arbitri.add(a);
	}
	public final void addUDC(UDC u) {
		udc.add(u);
	}
	public final void addOsservatore(Osservatore o) {
		osservatori.add(o);
	}

	public void designa() {
		ArrayList<Gara> selected = archivio.getGareByStato(StatoGara.PROGRAMMATA);
		for(Gara g: selected) {
			g.designa(arbitri.get(0), arbitri.get(1), udc.get(0), udc.get(1),
udc.get(2), osservatori.get(0));
		}
	}
}
```

GiudiceSportivo

```
package federazione;

import java.util.ArrayList;
import organizzazione.Gara;
import organizzazione.StatoGara;

public final class GiudiceSportivo {
	private static GiudiceSportivo giudice = null;
	private final ArchivioGare archivio = ArchivioGare.getIstance();

	private GiudiceSportivo() { }

	public final static GiudiceSportivo getIstance() {
		if (giudice == null) {
```

```
            giudice = new GiudiceSportivo();
         }
         return giudice;
        }

        public void omologa() {
               ArrayList<Gara> selected = archivio.getGareByStato(StatoGara.DISPUTATA);
               for(Gara g: selected) {
                      g.omologa();
               }
        }
}
```

UfficioAmministrativo

```
package federazione;

import java.util.ArrayList;
import amministrazione.Rimborso;
import organizzazione.Gara;
import organizzazione.StatoGara;

public final class UfficioAmministrativo {
        private static UfficioAmministrativo ufficio = null;
        private final ArchivioGare archivio = ArchivioGare.getIstance();

        private UfficioAmministrativo() { }

        public final static UfficioAmministrativo getIstance() {
               if (ufficio == null) {
              ufficio = new UfficioAmministrativo();
         }
         return ufficio;
        }

        public void liquida() {
               ArrayList<Gara> selected = archivio.getGareByStato(StatoGara.OMOLOGATA);
               for(Gara g:selected) {
                      g.rimborsa();
                      ArrayList<Rimborso> rimborsi = g.getDesignazione().getRimborsi();
                      for(Rimborso r:rimborsi) {
                             r.setLiquidato();
                      }
               }
        }
}
```

TesseratoCIA

```
package amministrazione;

import java.util.ArrayList;
import organizzazione.Designazione;

public abstract class TesseratoCIA {
        private final String nome;
        private final String cognome;
        private final String tessera;       //univoca
        private String IBAN;
        private String mail;
        private String telefono;
        private ArrayList<Designazione> designazioni;
        private final Interfaccia interfaccia;
```

```
	public TesseratoCIA(String nome, String cognome, String tessera, String IBAN, String mail,
String telefono) {
		this.nome = new String(nome);
		this.cognome = new String(cognome);
		this.tessera = new String(tessera);
		this.IBAN = new String(IBAN);
		this.mail = new String(mail);
		this.telefono = new String(telefono);
		designazioni = new ArrayList<Designazione>();
		interfaccia = new Interfaccia(this);
		interfaccia.setStrategia(StrategyMail.getIstance());
	}

	public final String getNome() {
		return new String(nome+" "+cognome);
	}
	public final String getTessera() {
		return new String(tessera);
	}
	public final String getIBAN() {
		return new String(IBAN);
	}
	public final String getMail() {
		return new String(mail);
	}
	public final String getTelefono() {
		return new String(telefono);
	}
	protected final void setIBAN(String IBAN) {
		this.IBAN = new String(IBAN);
	}
	protected final void setMail(String mail) {
		this.mail = new String(mail);
	}
	protected final void setTelefono(String telefono) {
		this.telefono = new String(telefono);
	}
	public final void inviaDesignazione(Designazione designazione) {
		designazioni.add(designazione);
		System.out.println("SMS("+getNome()+"): sei stato designato per la gara
"+designazione.getGara().getNumero());
	}
	public final void revocaDesignazione(Designazione designazione) {
		designazioni.remove(designazione);
	}
	public final ArrayList<Designazione> getDesignazioni(){
		return designazioni;
	}

	public void inviaRimborso() {
		Designazione des = designazioni.get(0);
		Rimborso rimb = new ExtraPlayoff(new ExtraViaggio(new
Gettone(des.getGara().getNumero(), this),15));
		des.addRimborso(rimb);
		interfaccia.addRimborso(rimb);
	}
	public final void visualizzaInterfaccia() {
		interfaccia.display();
	}
}
```

Arbitro

```
package organizzazione;

import amministrazione.TesseratoCIA;
```

```
public final class Arbitro extends TesseratoCIA {

	public Arbitro(String nome, String cognome, String tessera, String IBAN, String mail,
String telefono) {
		super(nome, cognome, tessera, IBAN, mail, telefono);
	}

	public void inviaReferto() {
		getDesignazioni().get(0).getGara().sendReferto(new Referto("89-72"));
	}
}
```

UDC

```
package organizzazione;

import amministrazione.TesseratoCIA;

public final class UDC extends TesseratoCIA {

	public UDC(String nome, String cognome, String tessera, String IBAN, String mail, String
telefono) {
		super(nome, cognome, tessera, IBAN, mail, telefono);
	}
}
```

Osservatore

```
package organizzazione;

import amministrazione.TesseratoCIA;

public final class Osservatore extends TesseratoCIA {

	public Osservatore(String nome, String cognome, String tessera, String IBAN, String mail,
String telefono) {
		super(nome, cognome, tessera, IBAN, mail, telefono);
	}
}
```

Gara

```
package organizzazione;

import java.time.LocalDate;

public final class Gara {
	private final String numero;
	private LocalDate data;
	private final String squadraA;
	private final String squadraB;
	private StatoGara stato;
	private Referto referto;
	private Designazione designazione;

	public Gara(String numero, LocalDate data, String squadraA, String squadraB) {
		this.numero = numero;
		this.data = data;
		this.squadraA = squadraA;
		this.squadraB = squadraB;
		stato = StatoGara.PROGRAMMATA;
		referto = null;
	}

	public final LocalDate getData() {
		return data;
```

```
	}
	public final void setData(LocalDate data) {
		this.data = data;
	}
	public final StatoGara getStato() {
		return stato;
	}
	public final void updateStato(StatoGara stato) {
		this.stato = stato;
	}
	public final Referto getReferto() throws Exception{
		if(referto == null)
			return new Referto(referto);
		else
			throw new Exception();
	}
	public final void sendReferto(Referto referto) {
		this.referto = referto;
		stato = StatoGara.DISPUTATA;
	}
	public final String getNumero() {
		return new String(numero);
	}
	public final String getSquadre() {
		return new String(squadraA+" VS "+squadraB);
	}

	public final String getRisultato() {
		if(referto != null)
			return referto.getRisultato();
		else
			return new String("Non disponibile");
	}

	public final void designa(Arbitro arbitro1, Arbitro arbitro2, UDC sec, UDC cronom, UDC
segnap, Osservatore oss) {
		if(designazione == null)
			designazione = new Designazione(this, arbitro1, arbitro2, sec, cronom,
segnap, oss);
		else {
			if(designazione.getArbitro1().getTessera() != arbitro1.getTessera())
				designazione.changeArbitro1(arbitro1);
			if(designazione.getArbitro2().getTessera() != arbitro2.getTessera())
				designazione.changeArbitro2(arbitro2);
			if(designazione.get24Secondi().getTessera() != sec.getTessera())
				designazione.change24Secondi(sec);
			if(designazione.getCronometrista().getTessera() != cronom.getTessera())
				designazione.changeCronometrista(cronom);
			if(designazione.getSegnapunti().getTessera() != segnap.getTessera())
				designazione.changeSegnapunti(segnap);
			if(designazione.getOsservatore().getTessera() != oss.getTessera())
				designazione.changeOsservatore(oss);
		}
		stato = StatoGara.DESIGNATA;
	}
	public final Designazione getDesignazione() {
		return designazione;
	}
	public final void omologa() {
		stato = StatoGara.OMOLOGATA;
	}
	public final void rimborsa() {
		stato = StatoGara.RIMBORSATA;
	}
}
```

StatoGara

```
package organizzazione;

public enum StatoGara {
	PROGRAMMATA,
	DESIGNATA,
	DISPUTATA,
	OMOLOGATA,
	RIMBORSATA
}
```

Referto

```
package organizzazione;

public class Referto {
	private final String risultato;

	public Referto(String risultato) {
		this.risultato = new String(risultato);
	}
	public Referto(Referto r) {
		risultato = r.getRisultato();
	}

	public String getRisultato() {
		return new String(risultato);
	}
}
```

Designazione

```
package organizzazione;

import java.util.ArrayList;
import amministrazione.Rimborso;

public final class Designazione {
	private final Gara gara;
	private Arbitro arbitro1;
	private Arbitro arbitro2;
	private UDC sec;
	private UDC cronom;
	private UDC segnap;
	private Osservatore oss;
	private ArrayList<Rimborso> rimborsi;

	public Designazione(Gara gara, Arbitro arbitro1, Arbitro arbitro2, UDC sec, UDC cronom,
UDC segnap,	Osservatore oss) {
		this.gara = gara;
		this.arbitro1 = arbitro1;
		this.arbitro2 = arbitro2;
		this.sec = sec;
		this.cronom = cronom;
		this.segnap = segnap;
		this.oss = oss;
		rimborsi = new ArrayList<Rimborso>();
		this.arbitro1.inviaDesignazione(this);
		this.arbitro2.inviaDesignazione(this);
		this.sec.inviaDesignazione(this);
		this.cronom.inviaDesignazione(this);
		this.segnap.inviaDesignazione(this);
		this.oss.inviaDesignazione(this);
	}

	public final Gara getGara() {
		return gara;
```

```
	}
	public final Arbitro getArbitro1() {
		return arbitro1;
	}
	public final void changeArbitro1(Arbitro arbitro) {
		arbitro1.revocaDesignazione(this);
		arbitro1 = arbitro;
		arbitro1.inviaDesignazione(this);
	}
	public final Arbitro getArbitro2() {
		return arbitro2;
	}
	public final void changeArbitro2(Arbitro arbitro) {
		arbitro2.revocaDesignazione(this);
		arbitro2 = arbitro;
		arbitro2.inviaDesignazione(this);
	}
	public final UDC get24Secondi() {
		return sec;
	}
	public final void change24Secondi(UDC udc) {
		sec.revocaDesignazione(this);
		sec = udc;
		sec.inviaDesignazione(this);
	}
	public final UDC getCronometrista() {
		return cronom;
	}
	public final void changeCronometrista(UDC udc) {
		cronom.revocaDesignazione(this);
		cronom = udc;
		cronom.inviaDesignazione(this);
	}
	public final UDC getSegnapunti() {
		return segnap;
	}
	public final void changeSegnapunti(UDC udc) {
		segnap.revocaDesignazione(this);
		segnap = udc;
		segnap.inviaDesignazione(this);
	}
	public final Osservatore getOsservatore() {
		return oss;
	}
	public final void changeOsservatore(Osservatore oss) {
		this.oss.revocaDesignazione(this);
		this.oss = oss;
		this.oss.inviaDesignazione(this);
	}

	public final void addRimborso(Rimborso rimborso) {
		rimborsi.add(rimborso);
	}
	public final void removeRimborso(Rimborso rimborso) {
		rimborsi.remove(rimborso);
	}
	public final ArrayList<Rimborso> getRimborsi(){
		return rimborsi;
	}
}
```

Rimborso

```
package amministrazione;

import java.util.Observable;
```

```
public abstract class Rimborso extends Observable {
	private boolean liquidato = false;

	public abstract String getNumero();
	public abstract float getValore();
	public abstract String getDescrizione();
	public abstract TesseratoCIA getDestinatario();

	public final void setLiquidato() {
		liquidato = true;
		setChanged();
		notifyObservers();
	}
	public final boolean isLiquidato() {
		return liquidato;
	}
}
```

Gettone

```
package amministrazione;

public final class Gettone extends Rimborso {
	private final String numero;
	private final String descrizione;
	private final float valore;
	private final TesseratoCIA destinatario;

	public Gettone(String numero, TesseratoCIA destinatario) {
		this.numero = new String(numero);
		this.descrizione = new String("gettone");
		valore = 25;
		this.destinatario = destinatario;
	}

	@Override
	public final String getNumero() {
		return new String(numero);
	}
	@Override
	public final float getValore() {
		return valore;
	}
	@Override
	public final String getDescrizione() {
		return new String(descrizione);
	}
	@Override
	public final TesseratoCIA getDestinatario() {
		return destinatario;
	}
}
```

Extra

```
package amministrazione;

public abstract class Extra extends Rimborso {
	final Rimborso component;

	public Extra(Rimborso component) {
		this.component = component;
	}

	@Override
	public final String getNumero() {
```

```
		return component.getNumero();
	}
	@Override
	public final TesseratoCIA getDestinatario() {
		return component.getDestinatario();
	}
}
```

Extra100km

```
package amministrazione;

public final class Extra100km extends Extra {

	public Extra100km(Rimborso component) {
		super(component);
	}

	@Override
	public final float getValore() {
		return super.component.getValore()+10;
	}
	@Override
	public final String getDescrizione() {
		return super.component.getDescrizione()+" + extra >100km";
	}
}
```

ExtraPlayoff

```
package amministrazione;

public final class ExtraPlayoff extends Extra {

	public ExtraPlayoff(Rimborso component) {
		super(component);
	}

	@Override
	public final float getValore() {
		return super.component.getValore()+5;
	}
	@Override
	public final String getDescrizione() {
		return super.component.getDescrizione()+" + extra playoff";
	}
}
```

ExtraPasto

```
package amministrazione;

public final class ExtraPasto extends Extra {

	public ExtraPasto(Rimborso component) {
		super(component);
	}

	@Override
	public final float getValore() {
		return super.component.getValore()+10;
	}
	@Override
	public final String getDescrizione() {
		return super.component.getDescrizione()+" + extra pasto";
	}
```

```
}
```

ExtraViaggio

```
package amministrazione;

public final class ExtraViaggio extends Extra {
	private final int km;

	public ExtraViaggio(Rimborso component, int km) {
		super(component);
		this.km = (km>0)?km:0;
	}

	@Override
	public final float getValore() {
		return super.component.getValore()+(km*0.25f);
	}
	@Override
	public final String getDescrizione() {
		return super.component.getDescrizione()+" + extra "+km+" km";
	}
}
```

Interfaccia

```
package amministrazione;

import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

public final class Interfaccia implements Observer {
	private final TesseratoCIA tesserato;
	private ArrayList<Rimborso> rimborsi;
	private Strategy strategia;

	public Interfaccia(TesseratoCIA t) {
		tesserato = t;
		rimborsi = new ArrayList<Rimborso>();
		strategia = null;
	}

	@Override
	public final void update(Observable arg0, Object arg1) {
		strategia.inviaNotifica(tesserato);
	}

	public final void addRimborso(Rimborso r) {
		rimborsi.add(r);
		r.addObserver(this);
	}
	public final void removeRimborso(Rimborso r) {
		rimborsi.remove(r);
		r.deleteObserver(this);
	}
	public final void setStrategia(Strategy s) {
		strategia = s;
	}

	public final void display() {
		System.out.println("Interfaccia di: "+tesserato.getNome());
		for(Rimborso r:rimborsi) {
			System.out.println(r.getNumero()+": "+r.getValore()+"€
("+r.getDescrizione()+") -> "+(r.isLiquidato()?"Liquidato":"non liquidato"));
		}
```

```
	}
}
```

Strategy

```
package amministrazione;

public interface Strategy {

	public void inviaNotifica(TesseratoCIA t);
}
```

StrategyMail

```
package amministrazione;

public final class StrategyMail implements Strategy {
	private static StrategyMail istance = null;

	private StrategyMail() { }

	public final static StrategyMail getIstance() {
		if (istance == null) {
	     istance = new StrategyMail();
	 }
	 return istance;
	}

	@Override
	public final void inviaNotifica(TesseratoCIA t) {
		System.out.println("Mail("+t.getNome()+"): è stato liquato un rimborso. Accedi alla
tua interfaccia per scoprire di più");
	}
}
```

StrategySMS

```
package amministrazione;

public final class StrategySMS implements Strategy {
	private static StrategySMS istance = null;

	private StrategySMS() { }

	public final static StrategySMS getIstance() {
		if (istance == null) {
	     istance = new StrategySMS();
	 }
	 return istance;
	}

	@Override
	public final void inviaNotifica(TesseratoCIA t) {
		System.out.println("SMS("+t.getNome()+"): è stato liquidato un rimborso. Accedi
alla tua interfaccia per scoprire di più");
	}
}
```

TEST

MAIN

```
import organizzazione.*;
import federazione.*;

public class Main {

	public static void main(String[] args) {
		//Istanziamo i vari uffici
		UfficioGare uffGare = UfficioGare.getIstance();
		UfficioDesignazioni uffDes = UfficioDesignazioni.getIstance();
		GiudiceSportivo giudice = GiudiceSportivo.getIstance();
		UfficioAmministrativo uffAmm = UfficioAmministrativo.getIstance();

		//Istanziamo alcuni tesserati CIA e diamone il riferimento all'ufficio
designazioni
		Arbitro a1 = new Arbitro("Kevin", "Maggi", "59366", "IT01...",
"ma.kev97@gmail.com", "3463803150");
		Arbitro a2 = new Arbitro("Mattia", "Parigi", "59358", "IT01...",
"par***@***", "347*******");
		UDC udc1 = new UDC("Francesco", "Rivituso", "61256", "IT00...",
"riv***@***", "348*******");
		UDC udc2 = new UDC("Sofia", "Pierallini", "61261", "IT00...",
"sof***@***", "380*******");
		UDC udc3 = new UDC("Lorenzo", "Bezzi", "39657", "IT00...", "lor***@***",
"331*******");
		Osservatore oss = new Osservatore("Antonio", "Massafra", "38567",
"IT02...", "mas***@***", "333*******");

		uffDes.addArbitro(a1);
		uffDes.addArbitro(a2);
		uffDes.addUDC(udc1);
		uffDes.addUDC(udc2);
		uffDes.addUDC(udc3);
		uffDes.addOsservatore(oss);

		//L'ufficio gare crea una gara
		System.out.println("     ADESSO L'UFFICIO GARE CREERA' UNA GARA");
		uffGare.crea();

		//L'ufficio designazioni designa la gara
		System.out.println("\n     ADESSO L'UFFICIO DESIGNAZIONI INVIERA' LA
DESIGNAZIONI PER LA GARA");
		uffDes.designa();

		//I tesserati prima della partita inviano il rimborso
		a1.inviaRimborso();
		a2.inviaRimborso();
		udc1.inviaRimborso();
		udc2.inviaRimborso();
		udc3.inviaRimborso();
		oss.inviaRimborso();
```

```
            //L'interfaccia del primo arbitro mostra questo
            System.out.println("\n     DOPO AVER INVIATO IL RIMBORSO, L'INTERFACCIA
DEL PRIMO ARBITRO APPARIRA' COSI'");
            a1.visualizzaInterfaccia();

            //Dopo la partita il primo arbitro invia il referto
            a1.inviaReferto();

            //Il giudice sportivo omologa la gara
            giudice.omologa();

            //L'ufficio amministrativo liquida i rimborsi passati
            System.out.println("\n     ADESSO L'UFFICIO AMMINISTRATIVO LIQUIDERA' I
RIMBORSI");
            uffAmm.liquida();

            //L'interfaccia del primo arbitro adesso mostra questo
            System.out.println("\n     DOPO LA LIQUIDAZIONE L'INTERFACCIA DEL PRIMO
ARBITRO APPARIRA' COSI'");
            a1.visualizzaInterfaccia();
      }

}
```

OUTPUT

```
     ADESSO L'UFFICIO GARE CREERA' UNA GARA
L'ufficio gare ha creato la partita n. 1618 tra Pallacanestro Vaiano e Montemurlo
Basket

     ADESSO L'UFFICIO DESIGNAZIONI INVIERA' LA DESIGNAZIONI PER LA GARA
SMS(Kevin Maggi): sei stato designato per la gara 1618
SMS(Mattia Parigi): sei stato designato per la gara 1618
SMS(Francesco Rivituso): sei stato designato per la gara 1618
SMS(Sofia Pierallini): sei stato designato per la gara 1618
SMS(Lorenzo Bezzi): sei stato designato per la gara 1618
SMS(Antonio Massafra): sei stato designato per la gara 1618

     DOPO AVER INVIATO IL RIMBORSO, L'INTERFACCIA DEL PRIMO ARBITRO APPARIRA' COSI'
Interfaccia di: Kevin Maggi
1618: 33.75€ (gettone + extra 15 km + extra playoff) -> non liquidato

     ADESSO L'UFFICIO AMMINISTRATIVO LIQUIDERA' I RIMBORSI
Mail(Kevin Maggi): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più
Mail(Mattia Parigi): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più
Mail(Francesco Rivituso): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più
Mail(Sofia Pierallini): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più
Mail(Lorenzo Bezzi): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più
Mail(Antonio Massafra): è stato liquato un rimborso. Accedi alla tua interfaccia per
scoprire di più

     DOPO LA LIQUIDAZIONE L'INTERFACCIA DEL PRIMO ARBITRO APPARIRA' COSI'
Interfaccia di: Kevin Maggi
1618: 33.75€ (gettone + extra 15 km + extra playoff) -> Liquidato
```

COMMENTI

Il focus di questo approfondimento è sulla composizione del 3 design pattern per arrivare a un costrutto dal comportamento più articolato (notare che sia observer che strategy sono design patterns comportamentali).
Tutto il resto del programma ha il solo fine di contestualizzazione, per questo non sono stati presi in considerazione alcuni accorgimenti di buona programmazione in quanto avrebbero complicato e appesantito inutilmente il codice.
Vediamo alcuni casi (item 23):

- Non si è verificato nella classe Designazione (sia nel costruttore che nei setter) la validità dei parametri: in particolare deve risultare che primo e secondo arbitro non siano lo stesso (verificabile tramite la tessera) e analogamente i 3 UDC;
- Non si è verificato in TesseratoCIA la validità dei parametri passati al costruttore e ai setter, in particolare le stringhe devono essere non nulle e non vuote; discorso analogo per la classe Gara.

Sugli oggetti di tipo Gara, Designazione e Rimborso non è stato possibile applicare gli item 13 e 24 riguardo immutabilità e copie difensive, in quanto sia la loro mutabilità che la loro condivisione sono alla base del funzionamento del programma. Ci sono comunque esempi di questi item negli attributi di Gara e TesseratoCia e nella classe Referto.

È stata minimizzata l'accessibilità degli attributi e dei metodi, con particolare attenzione alla classe TesseratoCIA, dove i metodi setter sono stati resi protected.

È stato preferito (dato il fine semplicemente illustrativo del contesto) impedire l'ereditarietà (item 15) anziché documentarla, anche quando sarebbe stato più produttivo documentarla, se pensiamo ad un contesto reale, dove questo programma risulterebbe incompleto.

Abbiamo degli esempi di item 2 riguardo il singleton in varie classe (implementato con creazione differita) e di item 16 sulla preferibilità delle interfacce sulle classi astratte: è il caso di Strategy, che essendo un functioide, non aveva motivo di essere implementato come classe astratta.