

## **Programmazione a oggetti e Ingegneria del Software**

Progetto per la sessione estiva 2018/2019

*----- Gestore di schede per strutture sportive -----*

*Sviluppatore*

*Luca Panariello*

*Matricola 289182*

## Indice

<b>I – Specifica del problema</b>	3
<b>II – Specifica dei requisiti</b>	4
II.1 – Diagramma dei casi d’uso	4
II.2 – Tabelle dei casi d’uso	5
<b>III – Analisi e Progettazione</b>	10
III.1 – Linguaggio di programmazione	10
III.2 – Ambiente di sviluppo	10
III.3 – Design Pattern	10
III.4 – Strutture di memorizzazione	10
III.5 – Organizzazione del progetto	11
III.6 – Descrizione e Analisi delle classi	12
III.6.1 – Classi astratte (base):	12
III.6.2 – Classi derivate:	12
III.6.3 – Classi base:	14
III.6.4 – Classi base (Singleton):	15
III.6.5 – Classi statiche:	18
III.6.6 – Tipi strutturati:	19
III.6.7 – Interfacce:	20
III.7 – Diagramma UML delle classi	21
<b>IV – Implementazione dell’algoritmo</b>	22
<b>V – Testing del programma</b>	51
V.1 – Test durante lo sviluppo	51
V.2 – Test White-Box	52
V.2.1 – Branch Coverage Test	52
V.2.1 – Condition Coverage Test	53
V.3 – Test Black-box	54
V.3.1 – Funzioni errate o mancanti	54
V.3.2 – Errori durante l’accesso al database	54
V.3.3 – Errori nelle prestazioni	54
V.3.4 – Errori di inizializzazione o terminazione	54
V.3.4 – Valutazione dei casi d’uso	55
<b>VI – Compilazione ed esecuzione</b>	60

## I – Specifica del problema

Si vuole progettare e implementare un programma per la gestione delle schede di una struttura sportiva, costituita da una palestra e una piscina. L'applicazione sarà a riga di comando, data la natura e la semplicità delle informazioni che gestirà.

La scheda è un documento (digitale o cartaceo) su cui vengono annotati gli esercizi che l'atleta dovrà compiere durante una sessione di allenamento. La versione digitale viene creata e gestita dagli istruttori.

In caso di una scheda per la palestra, gli esercizi sono costituiti dal numero di serie e ripetizioni per ogni gruppo muscolare (Deltoidi, Dorsali, Bicipiti, Tricipiti, Addominali e Gambe).

In caso di una scheda per la piscina, gli esercizi sono costituiti dal numero di serie e vasche degli stili acquatici (Crawl, Dorso, Rana e Delfino).

La ripetizione è il numero di volte che un esercizio va ripetuto (ad esempio 10). La serie indica un blocco di ripetizioni consecutive, al termine del quale ci sarà una pausa di recupero. Quindi 3 x 10 indica 10 ripetizioni, pausa, poi 10, pausa e infine altre 10 ripetizioni. Idem per le vasche in piscina.

Ovviamente una struttura si fatta permetterà anche un abbonamento doppio; quindi un singolo atleta potrà avere 2 diverse schede: una per la palestra e l'altra per la piscina.

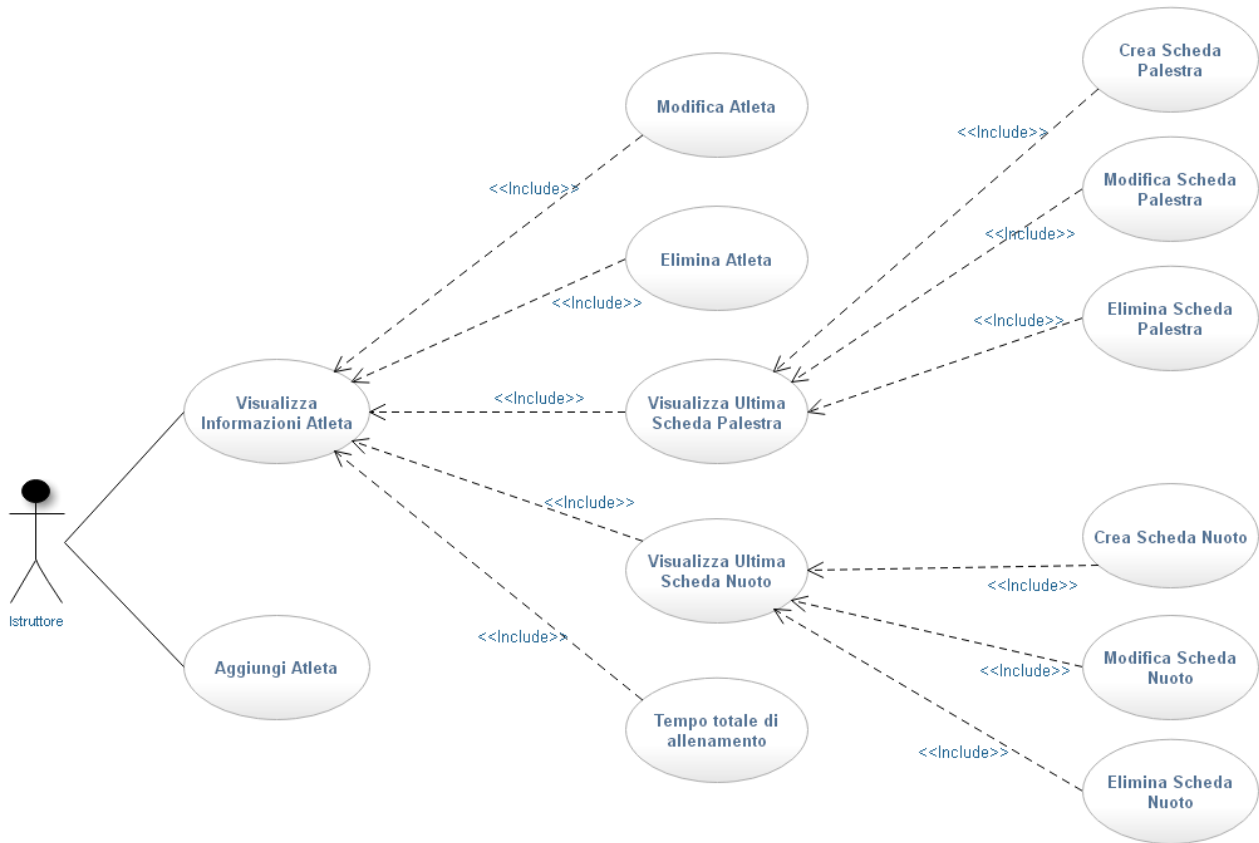
L'utente finale sarà un generico istruttore.

Il programma dovrà compiere le seguenti elaborazioni:

1. Acquisire i dati dei nuovi atleti e memorizzarli.
2. Acquisire i dati relativi agli esercizi della scheda della palestra/piscina e memorizzarli.
3. Permettere la visualizzazione/inserimento/modifica/eliminazione dei dati degli atleti e delle loro schede.
4. Permettere il calcolo del tempo totale di allenamento nel caso si ne vogliano eseguire 2 nello stesso giorno.

## II – Specifica dei requisiti

### II.1 – Diagramma dei casi d'uso



## II.2 – Tabelle dei casi d'uso

<b>Nome</b>	Visualizza Informazioni Atleta
<b>Id</b>	#1
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Lancia l'app, che mostra il menu generale
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 1 e preme Invio;</li> <li>2. Il programma mostra un elenco degli atleti e chiede di inserire l'id</li> <li>3. Inserisce l'id dell'atleta desiderato e preme Invio</li> </ol>
<b>Postcondizioni</b>	L'app mostra le informazioni dell'atleta con il relativo menu
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri

---

<b>Nome</b>	Aggiungi Atleta
<b>Id</b>	#2
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Lancia l'app, che mostra il menu generale
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>2. Il programma mostra la scritta "Inserire il Nome";</li> <li>3. Inserisce il Nome dell'atleta e preme Invio;</li> <li>4. Il programma mostra la scritta "Inserire il Cognome";</li> <li>5. Inserisce il Cognome dell'atleta e preme Invio;</li> <li>6. Il programma mostra la scritta "Inserire l'Età";</li> <li>7. Inserisce l'età dell'atleta in numeri e preme Invio;</li> </ol>
<b>Postcondizioni</b>	L'app inserisce i dati nelle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Errore: esiste già un atleta con gli stessi dati

<b>Nome</b>	Modifica Atleta
<b>Id</b>	#3
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #1
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Mostra un menu con le opzioni relative all'atleta</li> <li>2. Inserisce da tastiera il numero dell'opzione 1 e preme Invio;</li> <li>3. Il programma mostra la scritta "Inserire il Nome";</li> <li>4. Inserisce il Nome dell'atleta e preme Invio;</li> <li>5. Il programma mostra la scritta "Inserire il Cognome";</li> <li>6. Inserisce il Cognome dell'atleta e preme Invio;</li> <li>7. Il programma mostra la scritta "Inserire l'Età";</li> <li>8. Inserisce l'età dell'atleta in numeri e preme Invio;</li> </ol>
<b>Postcondizioni</b>	L'app aggiorna i dati nelle strutture e mostra un messaggio di conferma
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Errore: esiste già un atleta con gli stessi dati

---

<b>Nome</b>	Elimina Atleta
<b>Id</b>	#4
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #1
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Mostra un menu con le opzioni relative all'atleta</li> <li>2. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>3. L'app chiede conferma dell'eliminazione;</li> <li>4. Inserisce "S";</li> </ol>
<b>Postcondizioni</b>	L'app elimina i dati dalle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Opzione: inserisce "N" e l'app torna al menu principale

<b>Nome</b>	Visualizza Ultima Scheda Palestra
<b>Id</b>	#5
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #1
<b>Caso d'azione di base</b>	1. L'app mostra l'ultima scheda della palestra in ordine cronologico 2. L'app mostra un menu con le opzioni relative alla scheda;
<b>Postcondizioni</b>	L'app attende in input un numero, relativo all'opzione desiderata
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri

-----

<b>Nome</b>	Visualizza Ultima Scheda Nuoto
<b>Id</b>	#6
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #1
<b>Caso d'azione di base</b>	1. L'app mostra l'ultima scheda della piscina in ordine cronologico 2. L'app mostra un menu con le opzioni relative alla scheda;
<b>Postcondizioni</b>	L'app attende in input un numero, relativo all'opzione desiderata
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri

-----

<b>Nome</b>	Visualizza Tempo Totale di Allenamento
<b>Id</b>	#7
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #1
<b>Caso d'azione di base</b>	1. L'app mostra la somma delle durate delle 2 schede in input
<b>Postcondizioni</b>	L'app attende un input per tornare al menu principale
<b>Percorso alternativo</b>	-

<b>Nome</b>	Crea Scheda Palestra
<b>Id</b>	#8
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #5
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 1 e preme Invio;</li> <li>2. L'app mostra la scritta "Inserire nr. di serie di &lt;muscolo&gt;";</li> <li>3. Inserisce un numero e preme Invio;</li> <li>4. L'app mostra la scritta "Inserire nr. di ripetizioni di &lt;muscolo&gt;";</li> <li>5. Inserisce un numero e preme Invio;</li> <li>6. Ricomincia dal punto 2 fino all'inserimento dell'ultimo esercizio.</li> </ol>
<b>Postcondizioni</b>	L'app inserisce i dati nelle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri

---

<b>Nome</b>	Modifica Scheda Palestra
<b>Id</b>	#9
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #5
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>2. L'app mostra la scritta "Inserire nr. di serie di &lt;muscolo&gt;";</li> <li>3. Inserisce un numero e preme Invio;</li> <li>4. L'app mostra la scritta "Inserire nr. di ripetizioni di &lt;muscolo&gt;";</li> <li>5. Inserisce un numero e preme Invio;</li> <li>6. Ricomincia dal punto 2 fino all'inserimento dell'ultimo esercizio.</li> </ol>
<b>Postcondizioni</b>	L'app aggiorna i dati nelle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Errore: esiste già un atleta con gli stessi dati

---

<b>Nome</b>	Elimina Scheda Palestra
<b>Id</b>	#10
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #5
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>2. L'app chiede conferma dell'eliminazione;</li> <li>3. Inserisce "S";</li> </ol>
<b>Postcondizioni</b>	L'app elimina i dati dalle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Opzione: inserisce "N" e l'app torna al menu principale



<b>Nome</b>	Crea Scheda Nuoto
<b>Id</b>	#11
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #6
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>1. Inserisce da tastiera il numero dell'opzione 1 e preme Invio;</li> <li>2. L'app mostra la scritta "Inserire nr. di serie &lt;stile&gt;";</li> <li>3. Inserisce un numero e preme Invio;</li> <li>4. L'app mostra la scritta "Inserire nr. di vasche &lt;stile&gt;";</li> <li>5. Inserisce un numero e preme Invio;</li> <li>6. Ricomincia dal punto 2 fino all'inserimento dell'ultimo esercizio.</li> </ol>
<b>Postcondizioni</b>	L'app inserisce i dati nelle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri

---

<b>Nome</b>	Modifica Scheda Nuoto
<b>Id</b>	#12
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #6
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>7. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>8. L'app mostra la scritta "Inserire nr. di serie &lt;stile&gt;";</li> <li>9. Inserisce un numero e preme Invio;</li> <li>10. L'app mostra la scritta "Inserire nr. di ripetizioni &lt;stile&gt;";</li> <li>11. Inserisce un numero e preme Invio;</li> <li>12. Ricomincia dal punto 2 fino all'inserimento dell'ultimo esercizio.</li> </ol>
<b>Postcondizioni</b>	L'app aggiorna i dati nelle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Errore: esiste già un atleta con gli stessi dati

---

<b>Nome</b>	Elimina Scheda Nuoto
<b>Id</b>	#13
<b>Attore</b>	Istruttore
<b>Precondizioni</b>	Postcondizione #6
<b>Caso d'azione di base</b>	<ol style="list-style-type: none"> <li>4. Inserisce da tastiera il numero dell'opzione 2 e preme Invio;</li> <li>5. L'app chiede conferma dell'eliminazione;</li> <li>6. Inserisce "S";</li> </ol>
<b>Postcondizioni</b>	L'app elimina i dati dalle strutture e mostra il menu generale
<b>Percorso alternativo</b>	Errore: l'input da tastiera non rispetta i criteri Opzione: inserisce "N" e l'app torna al menu principale

## III – Analisi e Progettazione

### III.1 – Linguaggio di programmazione

C# è un linguaggio orientato agli oggetti, che permette di rappresentare le entità del mondo reale come oggetti.

### III.2 – Ambiente di sviluppo

Visual Studio 2017 Community, distribuito gratuitamente, con .NET Framework 4.6.1

### III.3 – Design Pattern

Singleton: design pattern creazionale.

- permette 1 sola istanza di una classe.
- Usando singleton, invece che dichiarare la classe come statica, è possibile utilizzare le interfacce, rendendo la progettazione più comprensibile.
- Effettua “l’inizializzazione pigra”: l’oggetto istanziato non viene realmente creato fino al primo accesso. In questo progetto è utile perché i metodi di palestra e nuoto appartengono a classi diverse e, se in una sessione non vi si accede, non viene allocato spazio in memoria.

### III.4 – Strutture di memorizzazione

- Database SQL (memorizzazione persistente): è stato in particolare SQLite, per la sua semplicità d’uso e portabilità. Infatti, il file che contiene la base di dati risiede all’interno della directory del programma. Il database contiene 3 tabelle, una per ogni entità (atleti, scheda nuoto e scheda palestra).
- Lista (memorizzazione temporanea): è una struttura dati dinamica ad accesso seriale, utile soprattutto quando non si conosce a priori la quantità di elementi che conterrà. Si è scelto di utilizzare una struttura di memorizzazione temporanea durante l’esecuzione del programma per semplificare e velocizzare l’accesso ai dati. Il database risiede su hard disk, mentre la lista (o meglio i riferimenti ai suoi elementi) risiede sulla RAM, decisamente più veloce. I dati vengono copiati dal database alla lista all’avvio del programma. Esiste una lista di oggetti per ogni entità (atleti, scheda palestra e scheda nuoto), ognuna gestita da una diversa classe.

I motivi per cui è stata scelta questa struttura dati sono i seguenti:

- Dinamica: la lista memorizza solo il riferimento all’oggetto (oltre ai riferimenti per scorrerla). Invece ad esempio l’array è statico e memorizza l’intera grandezza degli elementi (nel nostro caso gli oggetti). La modifica dinamica della grandezza di un array e di una lista hanno un costo computazionale di  $O(n)$  ma, nel caso dell’array vengono spostati gli oggetti, mentre nel caso della lista solo i riferimenti, quindi è meno dispendiosa dal punto di vista della memoria.
- Accesso: l’accesso diretto dell’array non potrebbe essere usato in modo corretto perché, in caso di rimozione di record dal database, gli id dei record e gli indici non corrisponderebbero più, a meno che non si lascino dei posti vuoti; ma sarebbe uno spreco di spazio. Non rimarrebbe che l’accesso seriale anche per l’array. Quindi si è optato per le liste.

### III.5 – Organizzazione del progetto

Il programma è costituito da:

- Classi base e derivate: rappresentano le entità coinvolte (atleti, scheda della palestra e scheda della piscina);
- Classi statiche: raggruppano i metodi relativi a funzionalità omogenee (ad esempio la classe *Menu* o *GestioneDB*). In questo caso le classi sono un insieme di utility.
- Classi base con Singleton: raggruppano i metodi che gestiscono una determinata struttura dati (le classi il cui nome inizia per *Interakt*). Si occupano di far interagire, per mezzo di messaggi testuali, l'utilizzatore del programma con le entità digitali (classi) e le strutture dati in cui sono memorizzate le informazioni (Liste e database).
- Tipi strutturati: sono dei tipi contenenti molteplici costanti omogenee (enum) o più variabili omogenee o eterogenee (struct). Si è scelto di rappresentare l'entità *Esercizio* con una variabile strutturata invece che con una classe perché l'entità che andrà a rappresentare è molto semplice e, in questo caso, struct è più performante per 2 ragioni:
  1. Il tipo struct "Esercizio" è costituito da 2 variabili intere (4 bytes cadauna) per un totale di 8 bytes senza riferimenti. Una classe avrebbe 8 bytes + 4 del puntatore, per un totale di 12 bytes. Durante l'esecuzione del programma non c'è passaggio per valore di variabili definite struct, ma solo delle chiamate a procedura del metodo interno a struct. Perciò il consumo di memoria è decisamente ridotto di 1/3 rispetto alla ipotetica classe equivalente.
  2. Le variabili definite come struct vengono allocate sullo stack, mentre i riferimenti alle classi sull'heap. Lo stack è più veloce, ergo l'esecuzione sarà più rapida.

Altre scelte di progetto:

- Si è scelto di racchiudere il programma in una classe statica denominata *MainApp* per dividere meglio il codice e quindi renderlo più comprensibile. In questa classe i messaggi a schermo sono ridotti al minimo: 2 soli in fase di congedo prima della terminazione dell'esecuzione.
- Si è scelto di dichiarare la classe base *Scheda* come astratta perché non ha metodi tali da giustificare l'utilizzo come classe standard, al contrario delle sue classi derivate.
- La classe *InteraktSchede* contiene un metodo statico che prende in input 2 oggetti *Scheda*, ne somma le rispettive durate e mostra a video il risultato. In questo caso è stato effettuato un upcasting: dichiarando i parametri con l'oggetto base, il metodo accetta entrambe le classi derivate. In questo modo si rispetta anche una norma della qualità del software: estensibilità. Infatti, se in futuro dovessero essere aggiunte altre schede di altri corsi della struttura, potranno essere accettate da questo metodo senza alcuna modifica.

Legenda:



attributo privato: per motivi di sicurezza. Sono accessibili solo da membri della stessa classe



proprietà: accedono agli attributi in modo sicuro e semplice



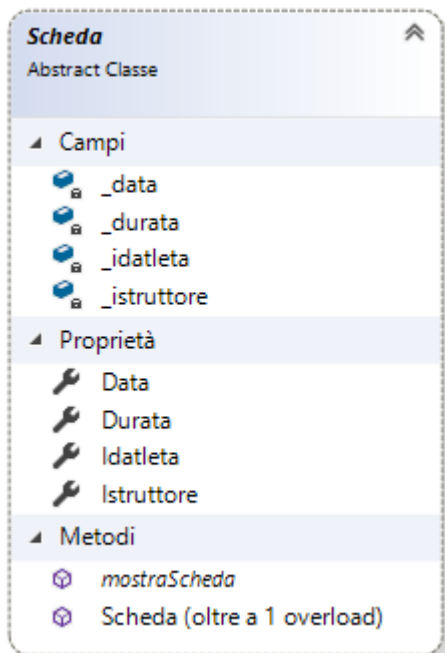
metodo



metodo protetto

## III.6 – Descrizione e Analisi delle classi

## III.6.1 – Classi astratte (base):

**Scheda**DiagrammaMetodi

*mostraScheda()*: è un metodo astratto che verrà richiamato dalle classi collegate con override

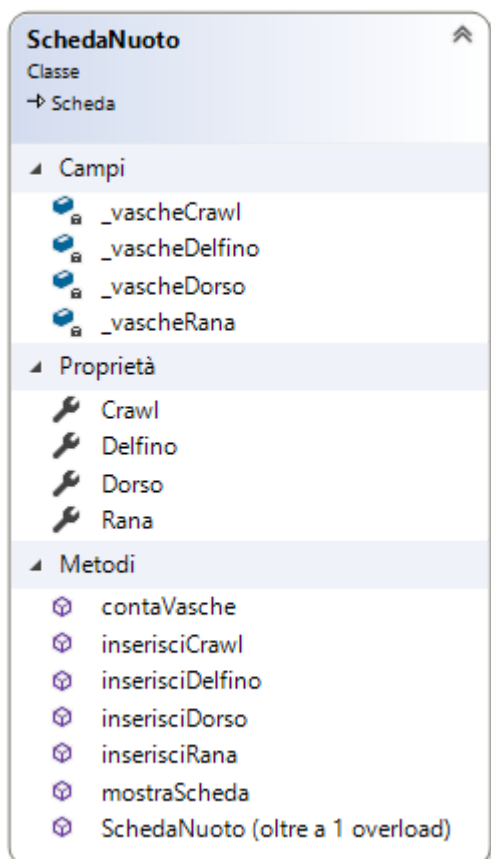
*Scheda()*: costruttore

*Scheda(<parametri>)*: È un overload del costruttore. Contiene dei parametri per impostare alcuni attributi all'atto della creazione dell'istanza (delle classi derivate ovviamente).

Descrizione

Rappresenta la scheda di allenamento. Si è scelto di renderla astratta perché dà un contributo quasi totalmente logico e permette l'aggiunta di altri tipi di schede.

## III.6.2 – Classi derivate:

**SchedaNuoto**DiagrammaMetodi

*mostraScheda()*: è un metodo ereditato dalla classe astratta "Scheda".

Mostra a video le informazioni della scheda.

In questa occasione è stato utilizzato il polimorfismo con override: cioè le sottoclassi ereditano lo stesso metodo ma il contenuto è specifico di ogni classe derivata.

*SchedaNuoto()*: costruttore (con e senza parametri)

*Inserisci...()*: inserisce le serie e ripetizioni in un'unica chiamata. Usati nel metodo popolaLista() della classe InteraktSchedaNuoto

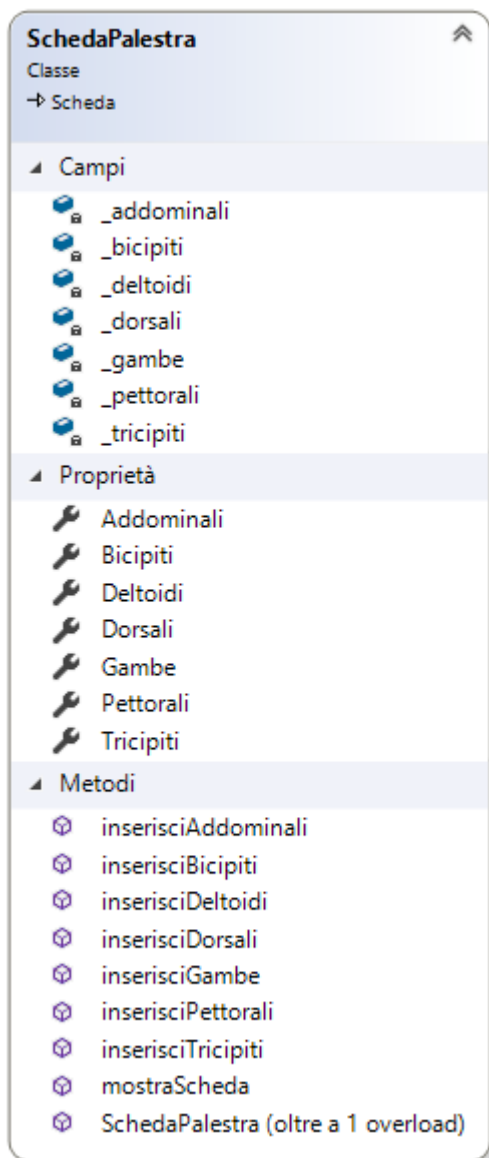
Descrizione

Rappresenta la scheda di allenamento in piscina con gli esercizi

---

**SchedaPalestra**

---

DiagrammaMetodi

mostraScheda(): è un metodo ereditato dalla classe astratta "Scheda".

Mostra a video le informazioni della scheda.

In questa occasione è stato utilizzato il polimorfismo con override: cioè lo stesso metodo è ereditato da più classi ma l'output è specifico di ogni sottoclasse.

SchedaPalestra(): costruttore (con e senza parametri)

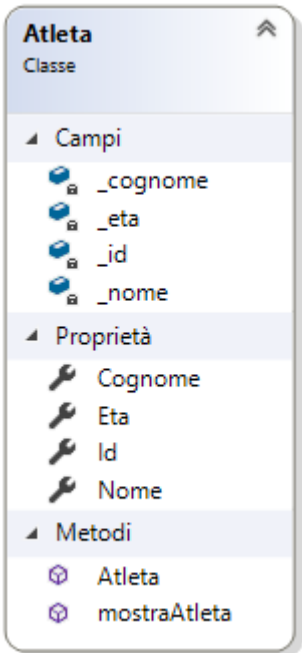
Inserisci...(): inserisce le serie e ripetizioni in un'unica chiamata. Usati nel metodo popolaLista() della classe InteraktSchedaPalestra

---

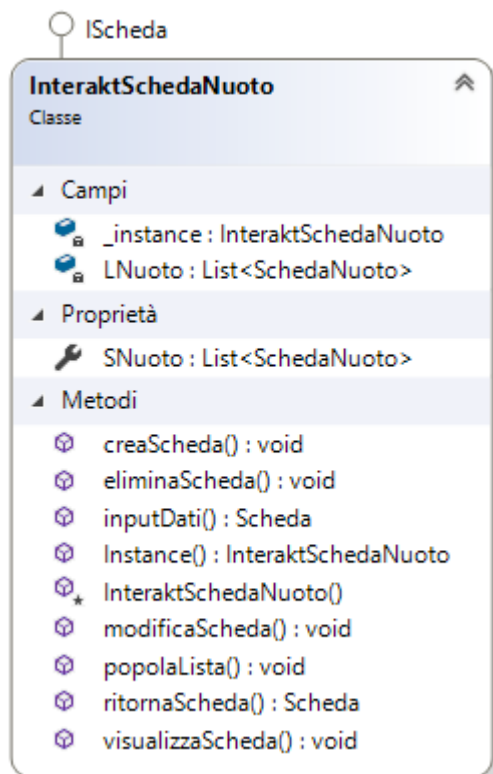
Descrizione

Rappresenta la scheda di allenamento in palestra con gli esercizi

III.6.3 – Classi base:

Atleta	
Diagramma	Metodi
 <p>The diagram shows the structure of the <b>Atleta</b> class. It includes a header with the class name and type, followed by three sections: <b>Campi</b> (private attributes: <code>_cognome</code>, <code>_eta</code>, <code>_id</code>, <code>_nome</code>), <b>Proprietà</b> (public properties: <code>Cognome</code>, <code>Eta</code>, <code>Id</code>, <code>Nome</code>), and <b>Metodi</b> (public methods: <code>Atleta</code>, <code>mostraAtleta</code>).</p>	<p>Atleta(): costruttore della classe mostraAtleta():Mostra a video le informazioni dell’atleta.</p>
	Descrizione
	<p>Rappresenta l’entità atleta con i suoi dati personali</p>

## III.6.4 – Classi base (Singleton):

**InteraktSchedaNuoto**DiagrammaAttributi

**LNuoto:** è la lista che contiene tutte le schede nuoto

Metodi

**creaScheda():** permette all'utente di inserire i dati di una nuova scheda e li inserisce in un oggetto **SchedaNuoto** e crea un nuovo record nel database. L'oggetto viene poi aggiunto alla lista **LNuoto** in modo da renderne l'accesso più agevole.

**eliminaScheda():** elimina la scheda dalla lista e dal database, sulla base dell'id dell'atleta fornito in input.

**inputDati():** comunica con l'utente, memorizza in un oggetto temporaneo i dati inseriti dall'utente da tastiera. Ritorna in output l'oggetto **Scheda**.

**Instance():** crea un'istanza della classe in modo che ce ne sia una e una soltanto. (Singleton)

**InteraktSchedaNuoto():** costruttore protetto

**modificaScheda():** modifica la scheda nella lista e nel database, sulla base dell'id dell'atleta fornito in input.

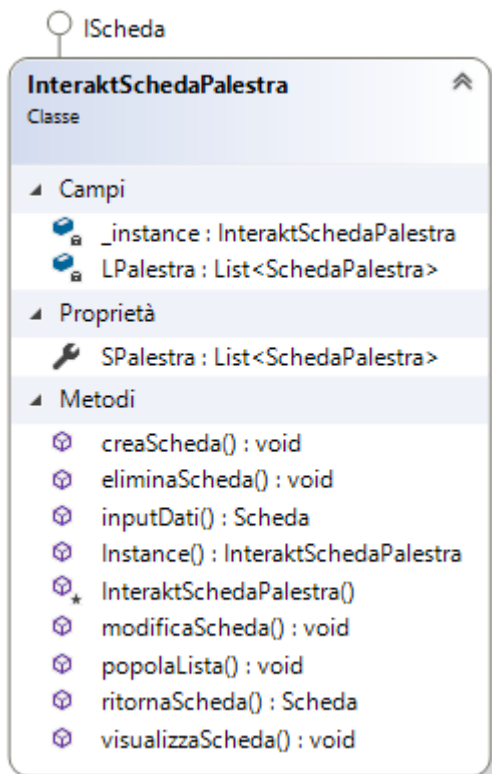
**popolaLista():** crea un oggetto per ogni record scheda nel database e lo inserisce nella lista.

**ritornaScheda():** ritorna in output l'oggetto scheda richiesto tra quelli della lista.

**visualizzaScheda():** visualizza la scheda di un determinato atleta

## InteraktSchedaPalestra

### Diagramma



### Attributi

**LPalestra:** è la lista che contiene tutte le schede nuoto

### Metodi

**creaScheda():** permette all'utente di inserire i dati di una nuova scheda e li inserisce in un oggetto `SchedaPalestra` e crea un nuovo record nel database. L'oggetto viene poi aggiunto alla lista `LPalestra` in modo da renderne l'accesso più agevole.

**eliminaScheda():** elimina la scheda dalla lista e dal database, sulla base dell'id dell'atleta fornito in input.

**inputDati():** comunica con l'utente, memorizza in un oggetto temporaneo i dati inseriti dall'utente da tastiera. Ritorna in output l'oggetto `Scheda`.

**Instance():** crea un'istanza della classe in modo che ce ne sia una e una soltanto. (Singleton)

**InteraktSchedaPalestra():** costruttore protetto

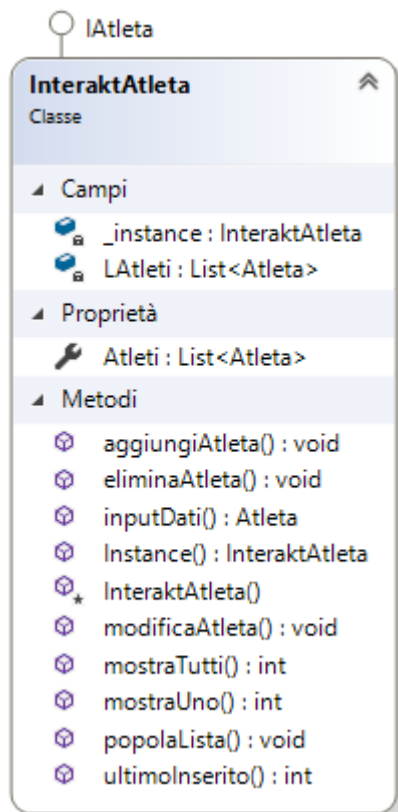
**modificaScheda():** modifica la scheda nella lista e nel database, sulla base dell'id dell'atleta fornito in input.

**popolaLista():** crea un oggetto per ogni record scheda nel database e lo inserisce nella lista.

**ritornaScheda():** ritorna in output l'oggetto scheda richiesto tra quelli della lista.

**visualizzaScheda():** visualizza la scheda di un determinato atleta



**InteraktAtleta**DiagrammaAttributi

LAtleti: è la lista che contiene tutti gli oggetti Atleta

Metodi

`aggiungiAtleta()`: permette all'utente di inserire i dati di un nuovo atleta, li inserisce in un oggetto Atleta e crea un nuovo record nel database. L'oggetto viene poi aggiunto alla lista in modo da renderne l'accesso più agevole.

`eliminaAtleta()`: comunica con l'utente elimina le info dalla lista e dal database, sulla base dell'id dell'atleta fornito in input.

`inputDati()`: comunica con l'utente, memorizza in un oggetto temporaneo i dati inseriti dall'utente da tastiera. Ritorna in output l'oggetto Atleta.

`Instance()`: crea un'istanza della classe in modo che ce ne sia una e una soltanto. (Singleton)

`InteraktAtleta()`: costruttore protetto

`modificaAtleta()`: comunica con l'utente e modifica le info nella lista e nel database, sulla base dell'id dell'atleta fornito in input.

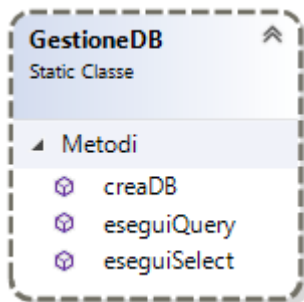
`mostraTutti()`: mostra un elenco di tutti gli atleti

`mostraUno()`: comunica con l'utente e mostra le info di un singolo atleta sulla base dell'id fornito.

`popolaLista()`: crea un oggetto per ogni record scheda nel database e lo inserisce nella lista.

`ultimInserito()`: ritorna l'id dell'ultimo record inserito nella tabella Atleti del database.

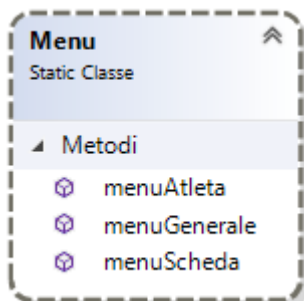
## III.6.5 – Classi statiche:

**GestioneDB**DiagrammaMetodi

**creaDB():** gestisce la connessione al database e crea le tabelle in caso non esistano

**eseguiQuery():** effettua una generica query (Insert, Update o Delete) che non prevede valori di ritorno né output.

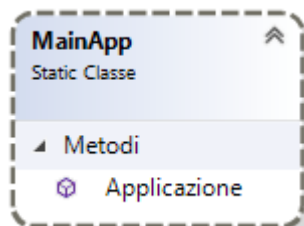
**eseguiSelect():** effettua una select al database. I record risultanti vengono memorizzati in una lista che a sua volta è l'output del metodo

**Menu**DiagrammaMetodi

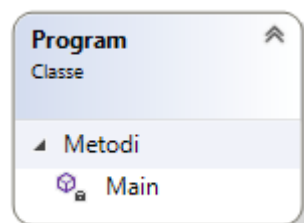
**MenuGenerale():** mostra a video una serie di opzioni e richiede all'utente di sceglierne una.

**menuScheda():** mostra a video una serie di opzioni relative all'entità scheda (palestra o nuoto) e richiede all'utente di scegliere una delle opzioni.

**menuUtente():** mostra a video una serie di opzioni relative all'entità utente (palestra o nuoto) e richiede all'utente di scegliere una delle opzioni

**MainApp**DiagrammaMetodi

**Applicazione():** è il metodo dell'app da cui vengono richiamate tutte le classi. Rappresenta il cuore dell'applicazione.

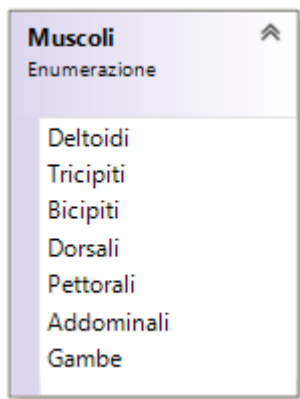
**Program**DiagrammaMetodi

**Main():** è il metodo principale

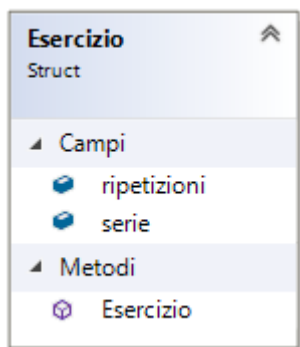
## III.6.6 – Tipi strutturati:

**Stili (enum)**DiagrammaDescrizione

Ogni valore rappresenta un diverso stile di nuoto

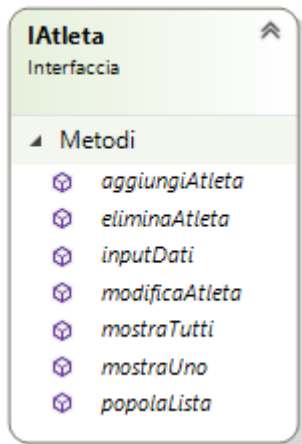
**Muscoli (enum)**DiagrammaDescrizione

Ogni valore rappresenta un diverso gruppo muscolare a cui è collegato un esercizio

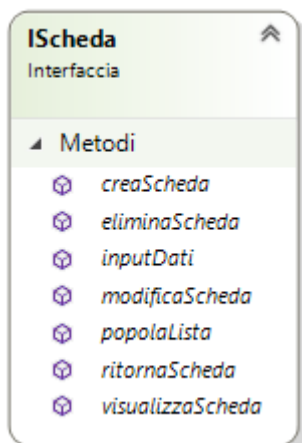
**Esercizio (struct)**DiagrammaMetodi

Il metodo Esercizio permette di assegnare un valore alle variabili

## III.6.7 – Interfacce:

**IAtleta**DiagrammaDescrizione

Metodi astratti che rappresentano quelli concreti della classe InteraktAtleta

**IScheda**DiagrammaDescrizione

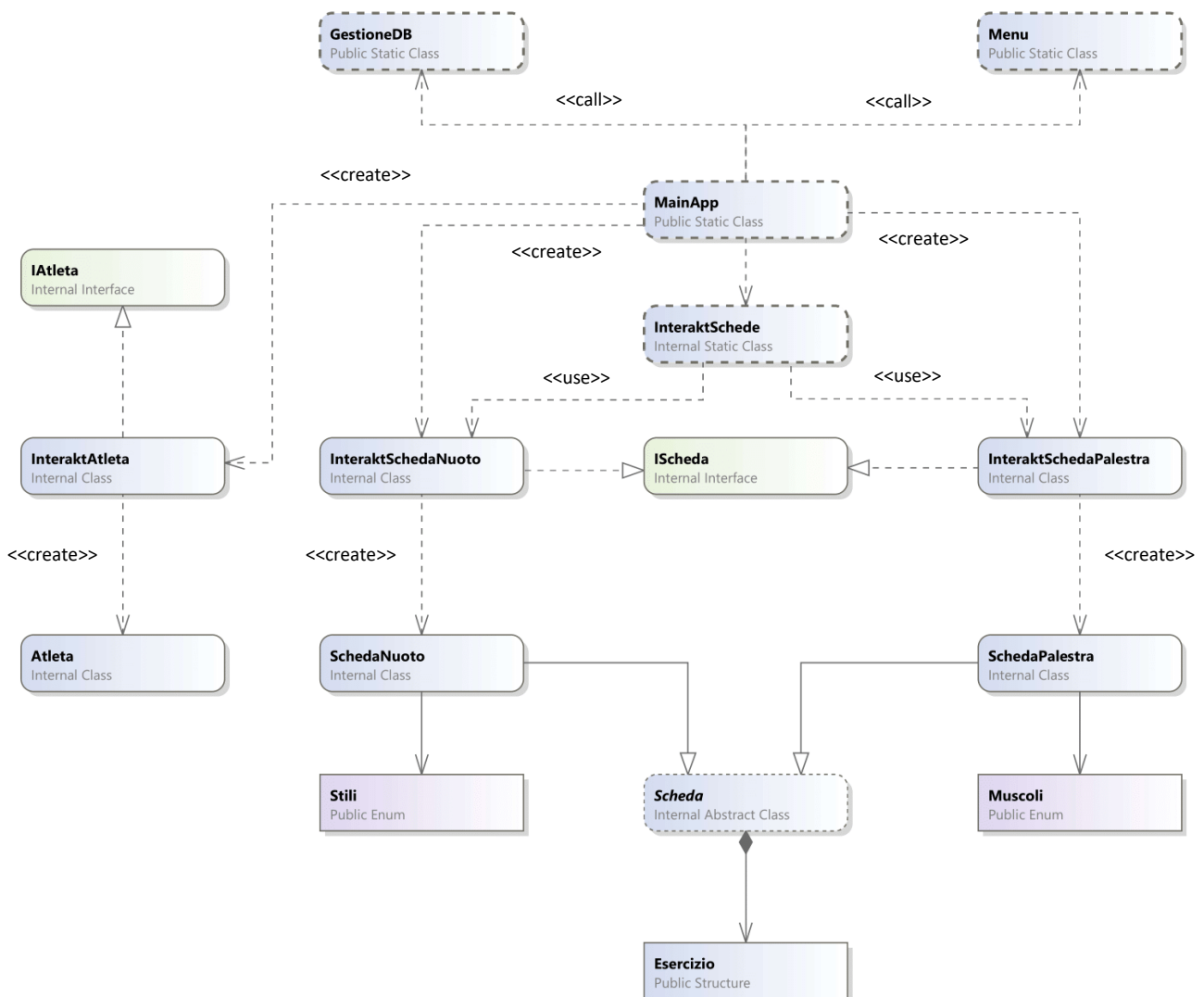
Metodi astratti che rappresentano quelli concreti della classe InteraktSchedaPalestra e InteraktSchedaNuoto

### III.7 – Diagramma UML delle classi

Il diagramma UML mostra le relazioni tra le classi, in termini di dipendenza, associazione, composizione e realizzazione. Si è scelto di inserire solo le intestazioni dei diversi oggetti, sia perché i membri sono stati già ampiamente analizzati nel paragrafo precedente, sia per rendere il diagramma più essenziale e quindi più leggibile.

Le relazioni più importanti sono:

- la generalizzazione tra l'oggetto Scheda e le sue classi derivate SchedaNuoto e SchedaPalestra;
- la composizione 1 a molti (\*) tra la classe astratta Scheda e il tipo strutturato Esercizio. Per ogni scheda ci sono molteplici Esercizi(o) e l'esistenza delle strutture Esercizio usate dalla classe dipende dall'esistenza della classe stessa (e quindi delle sue sottoclassi);
- la realizzazione tra le classi Interakt<...> e le loro rispettive interfacce.



## IV – Implementazione dell'algoritmo

### Atleta

```
using System;

namespace GymManager
{
    class Atleta
    {
        // attributi
        private int _id = 0;
        private string _nome;
        private string _cognome;
        private int _eta;

        // costruttore
        public Atleta()
        {
            // niente
        }

        // proprietà
        public int Id
        {
            get { return _id; }
            set { _id = value; }
        }

        public string Nome
        {
            get { return _nome; }
            set { _nome = value; }
        }

        public string Cognome
        {
            get { return _cognome; }
            set { _cognome = value; }
        }

        public int Eta
        {
            get { return _eta; }
            set { _eta = value; }
        }

        // metodi
        public string mostraAtleta()
        {
            string result;

            result = Id + ".\t" +
                Nome + "\t" +
                Cognome + "\t" +
                Eta + Environment.NewLine;

            return result;
        }
    }
}
```

**Esercizio**

```
namespace GymManager
{
    public struct Esercizio
    {
        public int serie,
            ripetizioni;

        public Esercizio(int s,
            int r)
        {
            serie = s;
            ripetizioni = r;
        }
    }
}
```

**Gestione DB**

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;

namespace GymManager
{
    public static class GestioneDB
    {
        // Metodo che crea il database
        public static void creaDB()
        {
            SQLiteConnection gymdbConnection = null;

            try
            {
                if (!System.IO.File.Exists("gymdb.sqlite"))
                    SQLiteConnection.CreateFile("gymdb.sqlite");

                gymdbConnection = new SQLiteConnection("Data Source=gymdb.sqlite;Version=3;");

                // Apre la connessione al database
                gymdbConnection.Open();

                string TabellaUtenti = "CREATE TABLE IF NOT EXISTS Atleti( " +
                    "ID          INTEGER          PRIMARY KEY AUTOINCREMENT, " +
                    "Nome        VARCHAR(50)      NOT NULL, " +
                    "Cognome     VARCHAR(50)      NOT NULL, " +
                    "Eta         TINYINT          NOT NULL DEFAULT 0, " +
                    "Data        DATETIME         DEFAULT CURRENT_TIMESTAMP)";

                string TabellaPalestra = "CREATE TABLE IF NOT EXISTS Palestra( " +
                    "ID          INTEGER          PRIMARY KEY AUTOINCREMENT, " +
                    "Durata       TINYINT          NOT NULL DEFAULT 1, " +
                    "Istruttore   VARCHAR(50)      NOT NULL, " +
                    "SerieDeltoidi  TINYINT          NOT NULL, " +
                    "SerieTricipiti TINYINT          NOT NULL, " +
                    "SerieBicipiti  TINYINT          NOT NULL, " +
                    "SerieDorsali   TINYINT          NOT NULL, " +
                    "SeriePettorali TINYINT          NOT NULL, " +
                    "SerieAddominali TINYINT          NOT NULL, " +
                    "SerieGambe     TINYINT          NOT NULL, " +
                    "RipDeltoidi    TINYINT          NOT NULL, " +
                    "RipTricipiti   TINYINT          NOT NULL, " +
                    "RipBicipiti    TINYINT          NOT NULL, " +
                    "RipDorsali    TINYINT          NOT NULL, " +
                    "RipPettorali   TINYINT          NOT NULL, " +
                    "RipAddominali TINYINT          NOT NULL, " +
                    "RipGambe      TINYINT          NOT NULL, " +
                    "IdAtleta      INTEGER          NOT NULL, " +
                    "DataP         DATETIME         DEFAULT CURRENT_TIMESTAMP, " +
                    "FOREIGN KEY (IdAtleta) REFERENCES Atleti(ID) ON DELETE CASCADE)";
            }
        }
    }
}
```

```

        string TabellaNuoto = "CREATE TABLE IF NOT EXISTS Nuoto( " +
                                "ID            INTEGER      PRIMARY KEY AUTOINCREMENT, " +
                                "Durata         TINYINT      NOT NULL DEFAULT 1, " +
                                "Istruttore     VARCHAR(50)  NOT NULL, " +
                                "SerieCrawl     TINYINT      NOT NULL, " +
                                "SerieDorso     TINYINT      NOT NULL, " +
                                "SerieRana      TINYINT      NOT NULL, " +
                                "SerieDelfino   TINYINT      NOT NULL, " +
                                "RipCrawl      TINYINT      NOT NULL, " +
                                "RipDorso      TINYINT      NOT NULL, " +
                                "RipRana       TINYINT      NOT NULL, " +
                                "RipDelfino    TINYINT      NOT NULL, " +
                                "IdAtleta      INTEGER      NOT NULL, " +
                                "DataN         DATETIME      DEFAULT CURRENT_TIMESTAMP, " +
                                "FOREIGN KEY (IdAtleta) REFERENCES Atleti(ID) ON DELETE CASCADE)";

        // Creazione delle tabelle
        SQLiteCommand creaTabellaUtenti = new SQLiteCommand(TabellaUtenti, gymdbConnection);
        creaTabellaUtenti.ExecuteNonQuery();

        SQLiteCommand creaTabellaPalestra = new SQLiteCommand(TabellaPalestra, gymdbConnection);
        creaTabellaPalestra.ExecuteNonQuery();

        SQLiteCommand creaTabellaNuoto = new SQLiteCommand(TabellaNuoto, gymdbConnection);
        creaTabellaNuoto.ExecuteNonQuery();
    }
    catch (Exception exdb)
    {
        // Stampa il testo dell'eccezione a schermo
        Console.WriteLine("Errore di accesso al db: " + exdb);
    }
    finally
    {
        if (gymdbConnection != null)
        {
            try
            {
                // Chiusura del database
                gymdbConnection.Close();
            }
            catch (Exception exdb)
            {
                // Stampa il testo dell'eccezione a schermo
                Console.WriteLine("Errore di chiusura del db: " + exdb);
            }
            finally
            {
                // Chiude la connessione al db e dealloca le risorse
                gymdbConnection.Dispose();
            }
        }
    }
}

public static bool eseguiQuery(string query) // input: testo della query
{
    SQLiteConnection gymdbConnection = null;
    bool executed = false;

    try
    {
        gymdbConnection = new SQLiteConnection("Data Source=gymdb.sqlite;Version=3;");

        gymdbConnection.Open();

        SQLiteCommand querySemplice = new SQLiteCommand(query, gymdbConnection);
        querySemplice.ExecuteNonQuery();

        executed = true;
    }
    catch (Exception exdb)
    {
        // Stampa l'eccezione a video
        Console.WriteLine("Errore di accesso al db: " + exdb);
    }
    finally

```



```

    {
        if (gymdbConnection != null)
        {
            try
            {
                // Chiusura del database
                gymdbConnection.Close();
            }
            catch (Exception exdb)
            {
                // Stampa il testo dell'eccezione/errore a schermo
                Console.WriteLine("Errore di chiusura del db: " + exdb);
            }
            finally
            {
                // Chiude la connessione al db e dealloca le risorse
                gymdbConnection.Dispose();
            }
        }
    }

    return executed;
}

public static List<string[]> eseguiSelect(string query) // input: testo della query
{
    List<string[]> risultati = new List<string[]>(); // output: dati estratti dal db
    SQLiteConnection gymdbConnection = null;       // lavoro: oggetto connessione
    SQLiteDataReader reader = null;                 // lavoro: oggetto lettore dei dati

    try
    {
        gymdbConnection = new SQLiteConnection("Data Source=gymdb.sqlite;Version=3;");

        gymdbConnection.Open();

        SQLiteCommand querySelect = new SQLiteCommand(query, gymdbConnection);
        reader = querySelect.ExecuteReader();

        while (reader.Read())
        {
            int i = 0;

            // creo un array temporaneo per il record
            string[] record = new string[reader.GetValues().Count];

            // scorro i risultati
            foreach (string campo in reader.GetValues())
            {
                record[i] = reader[campo].ToString(); // aggiungo i campi al record

                i++; // incremento il contatore
            }

            risultati.Add(record);
        }
    }
    catch (Exception exdb)
    {
        // Stampa l'eccezione a video
        Console.WriteLine("Errore di accesso al db: " + exdb);
    }
    finally
    {
        if (gymdbConnection != null)
        {
            try
            {
                // Chiusura del database
                gymdbConnection.Close();
            }
            catch (Exception exdb)
            {
                // Stampa il testo dell'eccezione/errore a schermo
                Console.WriteLine("Errore di chiusura del db: " + exdb);
            }
        }
    }
}

```

```

        finally
        {
            // Chiude la connessione al db e dealloca le risorse
            gymdbConnection.Dispose();
        }
    }
}

return risultati;
}
}
}

```

## InteraktAtleta

```

using System;
using System.Collections.Generic;

namespace GymManager
{
    class InteraktAtleta : IAtleta
    {
        private static InteraktAtleta _instance = null; // tiene traccia dell'istanza

        // costruttore protetto
        protected InteraktAtleta()
        {
            // niente
        }

        // metodo per la creazione dell'istanza
        public static InteraktAtleta Instance()
        {
            if (_instance == null)
                _instance = new InteraktAtleta();

            return _instance;
        }

        private List<Atleta> LATleti = new List<Atleta>(); // attributo contenente gli atleti

        // proprietà
        public List<Atleta> Atleti
        {
            get { return LATleti; }
        }

        // metodi
        // --- CREA UN ATLETA ---
        public void aggiungiAtleta()
        {
            bool errore = false; // lavoro: flag in caso di eccezione
            Atleta atleta = new Atleta(); // lavoro: oggetto contenente le info dell'atleta

            Console.WriteLine("*** Inserimento Nuovo Atleta ***"
                               + Environment.NewLine
                               + Environment.NewLine);

            do
            {
                // prende i dati in input da tastiera
                atleta = inputDati();

                // inserisce i dati nel database
                if (GestioneDB.eseguiQuery("INSERT INTO Atleti (Nome,Cognome,Eta) " +
                                           "VALUES(' " + atleta.Nome +
                                           "', ' " + atleta.Cognome +
                                           "', " + atleta.Eta + ")"))
                {
                    // memorizza l'id dell'ultimo record inserito
                    atleta.Id = ultimoInserito(atleta);

                    // aggiunge l'oggetto alla lista
                    LATleti.Add(atleta);
                    Console.WriteLine("Inserimento effettuato" + Environment.NewLine);
                }
            }
        }
    }
}

```

```

        else
        {
            errore = true;
            Console.WriteLine("Errore di inserimento" + Environment.NewLine);
        }

        Console.WriteLine("Premere un tasto per continuare...");
        Console.ReadKey();
    } while (errore);
}

// --- MODIFICA UN ATLETA ---
public void modificaAtleta(int id) // input: id atleta
{
    bool trovato = false,           // output: flag in caso di atleta presente
        errore = false;           // lavoro: flag in caso di eccezione
    Atleta atleta = new Atleta(); // lavoro: oggetto d'appoggio

    Console.WriteLine("*** Modifica Atleta ***" + Environment.NewLine + Environment.NewLine);

    foreach (Atleta a in LAtleti)
    {
        if(a.Id == id)
        {
            trovato = true;
            do
            {
                errore = false;

                // prende i dati da tastiera
                atleta = inputDati();

                // aggiorna dati dell'atleta nel database
                if (GestioneDB.eseguiQuery("UPDATE Atleti SET" +
                    " Nome = '" + atleta.Nome +
                    "', Cognome = '" + atleta.Cognome +
                    "', Eta = " + atleta.Eta +
                    " WHERE ID = " + a.Id))
                {
                    a.Nome = atleta.Nome;
                    a.Cognome = atleta.Cognome;
                    a.Eta = atleta.Eta;
                    Console.WriteLine("Aggiornamento effettuato" + Environment.NewLine);
                }

                else
                {
                    errore = true;
                    Console.WriteLine("Errore di inserimento" + Environment.NewLine);
                }
            } while (errore);
        }
    }

    if (!trovato)
        Console.WriteLine("Non esiste un atleta con questo nome");

    Console.WriteLine("Premere un tasto per continuare...");
    Console.ReadKey();
}

// --- ELIMINA UN ATLETA ---
public void eliminaAtleta(int id) // input: id atleta
{
    bool errore;           // lavoro: flag per errore
    string elimina = "N"; // lavoro: conferma eliminazione
    int indice = -1;       // lavoro: posizione dell'oggetto in lista

    Console.WriteLine("*** Elimina Atleta ***" + Environment.NewLine);

    foreach (Atleta a in LAtleti)
    {
        if (a.Id == id)
        {
            do
            {

```

```

        errore = false;

        try
        {
            Console.WriteLine("Eliminare l'atleta? S/N");
            elimina = Console.ReadLine();

            if (!(elimina is string))
                throw new Exception("Inserire una lettera, 'S' o 'N'");

            if (elimina == "S" || elimina == "s")
            {
                // elimina l'atleta dal database
                if (GestioneDB.eseguiQuery("DELETE FROM Atleti WHERE ID = " + id))
                {
                    // salva la posizione dell'oggetto in lista
                    indice = LATleti.IndexOf(a);
                    Console.WriteLine("Atleta eliminato con successo" + Environment.NewLine);
                }
            }
        }
        catch (Exception e) // eccezione
        {
            errore = true;
            Console.WriteLine("Errore di input: " + e);
        }
    } while (errore);
}

if (indice > -1) // rimuove l'oggetto dalla lista
    LATleti.RemoveAt(indice);
else if (elimina == "S" || elimina == "s")
    Console.WriteLine("Atleta non presente" + Environment.NewLine);

Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
Console.ReadKey();
Console.Clear();
}

// --- MOSTRA UN ELENCO DI TUTTI GLI ATLETI ---
public int mostraTutti()
{
    int numero_atleti = LATleti.Count; // output: numero atleti

    Console.WriteLine("*** Elenco Atleti ***" + Environment.NewLine);

    if (numero_atleti > 0)
    {
        Console.WriteLine("Id\t" +
                           "Nome\t" +
                           "Cognome \t" +
                           "Eta" + Environment.NewLine);

        foreach (Atleta a in LATleti)
            Console.WriteLine(a.mostraAtleta());
    }

    return numero_atleti;
}

// --- VISUALIZZA UN SINGOLO ATLETA ---
public int mostraUno(int id) // input: id atleta
{
    int idA = 0, // output: id atleta
        numero_atleti; // lavoro: totale atleti in lista
    bool trovato = false, // lavoro: flag in caso di atleta presente
        errore; // lavoro: flag in caso di eccezione

    numero_atleti = LATleti.Count;

    if (numero_atleti > 0)
    {
        do
        {
            errore = false;

```

```

try
{
    if (id == 0)
    {
        Console.WriteLine("Inserire l'id dell'atleta che si desidera visualizzare");
        idA = int.Parse(Console.ReadLine());
    }
    else
        idA = id;

    if (idA < 0)
        throw new Exception("Inserire un numero intero positivo");

    foreach (Atleta a in LATleti)
    {
        if (a.Id == idA)
        {
            trovato = true;
            Console.Clear();
            Console.WriteLine(Environment.NewLine + a.mostraAtleta());
        }
    }

    if (!trovato)
    {
        idA = 0;
        Console.WriteLine("Atleta non presente" + Environment.NewLine);
        Console.ReadKey();
    }
}
catch (FormatException e)
{
    errore = true;
    Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
}
catch (Exception e)
{
    errore = true;
    Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
}

if (errore)
{
    Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
    Console.ReadKey();
}

} while (errore || !trovato);
}
else
    Console.WriteLine(Environment.NewLine + "Non sono presenti atleti" + Environment.NewLine);

if (numero_atleti == 0 || !trovato)
{
    Console.WriteLine(Environment.NewLine + "Premi un tasto per continuare...");
    Console.ReadKey();
}

return idA;
}

// --- ID ULTIMO UTENTE INSERITO ---
public int ultimoInserito(Atleta a)
{
    int id = 0;
    List<string[]> risultati;

    risultati = GestioneDB.eseguiSelect("SELECT ID FROM Atleti" +
        " WHERE Nome = '" + a.Nome +
        "' AND Cognome = '" + a.Cognome +
        "' ORDER BY data DESC " +
        " LIMIT 1");

    if (risultati.Count > 0)
        foreach (string[] record in risultati)

```

```

        id = int.Parse(record[0]);

    return id;
}

// --- PRENDE DATI IN INPUT ---
public Atleta inputDati()
{
    Atleta atleta = new Atleta(); // lavoro: oggetto atleta
    bool errore;                 // lavoro: flag in caso di eccezione

    do {
        errore = false;

        try
        {
            // prende in input i valori da inserire nell'oggetto
            Console.WriteLine("Inserire il Nome: ");
            atleta.Nome = Console.ReadLine();

            Console.WriteLine(Environment.NewLine + "Inserire il Cognome: ");
            atleta.Cognome = Console.ReadLine();

            Console.WriteLine(Environment.NewLine + "Inserire l'eta': ");
            atleta.Eta = int.Parse(Console.ReadLine());

            foreach (Atleta a2 in LAtleti)
            {
                if ((a2.Nome == atleta.Nome)
                    && (a2.Cognome == atleta.Cognome)
                    && (a2.Eta == atleta.Eta))
                {
                    throw new Exception("Atleta gia' presente in memoria");
                }
            }
        }
        catch (FormatException e) // eccezione in caso di formato errato
        {
            errore = true;
            Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
        }
        catch (Exception e) // eccezione
        {
            errore = true;
            Console.WriteLine(Environment.NewLine + "Errore: " + e);
        }
    } while (errore);

    if (errore)
    {
        Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
        Console.ReadLine();
    }

    return atleta;
}

// --- TRASFERISCE I DATI DAL DATABASE ALLA LISTA ---
public void popolaLista()
{
    Atleta atleta = null; // lavoro: oggetto atleta
    List<string[]> risultati; // lavoro: memorizza i record del database

    risultati = GestioneDB.eseguiSelect("SELECT * FROM Atleti");

    if (risultati.Count > 0)
    {
        foreach (string[] record in risultati)
        {
            atleta = new Atleta();

            atleta.Id = int.Parse(record[0]);
            atleta.Nome = record[1];
            atleta.Cognome = record[2];
            atleta.Eta = int.Parse(record[3]);

            // inserisce l'oggetto nella lista
            LAtleti.Add(atleta);
        }
    }
}

```

```

    }
}
}

```

## InteraktSchedaNuoto

```

using System;
using System.Collections.Generic;

namespace GymManager
{
    public enum Stili
    {
        Crawl,
        Dorso,
        Rana,
        Delfino
    }

    class InteraktSchedaNuoto : IScheda
    {
        private static InteraktSchedaNuoto _instance = null;

        // costruttore protetto
        protected InteraktSchedaNuoto()
        {
            // niente
        }

        // crea una singola istanza della classe
        public static InteraktSchedaNuoto Instance()
        {
            if (_instance == null)
                _instance = new InteraktSchedaNuoto();

            return _instance;
        }

        private List<SchedaNuoto> LNuoto = new List<SchedaNuoto>();

        public List<SchedaNuoto> SNuoto
        {
            get { return LNuoto; }
        }

        // --- CREA UNA SCHEDA ---
        public void creaScheda(int id) // input: id atleta
        {
            bool errore = false; // lavoro: flag in caso di eccezione
            SchedaNuoto nuoto = new SchedaNuoto(); // lavoro: oggetto contenente le info della scheda

            Console.WriteLine("*** Crea una nuova scheda nuoto ***"
                + Environment.NewLine
                + Environment.NewLine);

            do
            {
                // downcasting dell'oggetto Scheda
                nuoto = (SchedaNuoto)inputDati();
                nuoto.Idatleta = id;

                // inserisce i dati nel database
                if (GestioneDB.eseguiQuery("INSERT INTO Nuoto (Durata," +
                    "Istruttore," +
                    "SerieCrawl," +
                    "SerieDorso," +
                    "SerieRana," +
                    "SerieDelfino," +
                    "RipCrawl," +
                    "RipDorso," +
                    "RipRana," +
                    "RipDelfino," +
                    "IdAtleta) VALUES (" +
                    nuoto.Durata + ", " +

```

```

        nuoto.Istruttore + ", " +
        nuoto.Crawl.serie + ", " +
        nuoto.Dorso.serie + ", " +
        nuoto.Rana.serie + ", " +
        nuoto.Delfino.serie + ", " +
        nuoto.Crawl.ripetizioni + ", " +
        nuoto.Dorso.ripetizioni + ", " +
        nuoto.Rana.ripetizioni + ", " +
        nuoto.Delfino.ripetizioni + ", " +
        id + ")"))
    {
        // aggiunge l'oggetto alla lista
        LNuoto.Add(nuoto);
        popolaLista();
        Console.WriteLine("Inserimento effettuato" + Environment.NewLine);
    }
    else
    {
        errore = true;
        Console.WriteLine(Environment.NewLine + "Errore di inserimento" + Environment.NewLine);
    }

    Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
    Console.ReadKey();
} while (errore);
}

// --- MODIFICA L'ULTIMA SCHEDA ---
public void modificaScheda(int id) // input: id atleta
{
    bool errore, // lavoro: flag in caso di eccezione
        trovato = false; // lavoro: flag per oggetto trovato
    SchedaNuoto nuoto = new SchedaNuoto(); // lavoro: oggetto contenente le info della scheda

    Console.WriteLine("*** Aggiorna la scheda nuoto ***" + Environment.NewLine +
        Environment.NewLine);

    foreach (SchedaNuoto n in LNuoto)
    {
        if (n.Idatleta == id)
        {
            trovato = true;

            do
            {
                errore = false;

                // downcasting dell'oggetto
                nuoto = (SchedaNuoto)inputDati();

                // aggiorna id dati nel db
                if (GestioneDB.eseguiQuery("UPDATE Nuoto SET " +
                    "SerieCrawl = " + n.Crawl.serie +
                    ",SerieDorso = " + n.Dorso.serie +
                    ",SerieRana = " + n.Rana.serie +
                    ",SerieDelfino = " + n.Delfino.serie +
                    ",RipCrawl = " + n.Crawl.ripetizioni +
                    ",RipDorso = " + n.Dorso.ripetizioni +
                    ",RipRana = " + n.Rana.ripetizioni +
                    ",RipDelfino = " + n.Delfino.ripetizioni +
                    " WHERE IdAtleta = " + n.Idatleta +
                    " AND DataN = (SELECT DataN" +
                    " FROM Nuoto" +
                    " WHERE IdAtleta = " + n.Idatleta +
                    " ORDER BY DataN DESC" +
                    " LIMIT 1)"))
                {
                    n.Crawl = nuoto.Crawl;
                    n.Dorso = nuoto.Dorso;
                    n.Rana = nuoto.Rana;
                    n.Delfino = nuoto.Delfino;
                    Console.WriteLine("Aggiornamento effettuato" + Environment.NewLine);
                }
            }
            else
            {
                errore = true;
            }
        }
    }
}

```



```

        Console.WriteLine(Environment.NewLine + "Errore di aggiornamento" +
Environment.NewLine);
    }
    } while (errore);
}

if (!trovato)
    Console.WriteLine(Environment.NewLine + "Scheda non presente" + Environment.NewLine);

Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
Console.ReadKey();
}

// --- CANCELLA L'ULTIMA SCHEDA ---
public void eliminaScheda(int id) // input: id atleta
{
    bool errore;           // lavoro: flag per errore
    string elimina = "N"; // lavoro: conferma eliminazione
    int indice = -1;       // lavoro: flag per scheda presente

    foreach (SchedaNuoto n in LNuoto)
    {
        if (n.Idatleta == id)
        {
            do
            {
                errore = false;
                try
                {
                    Console.WriteLine("Eliminare la scheda? S/N");
                    elimina = Console.ReadLine();

                    if (!(elimina is string))
                        throw new Exception("Inserire una lettera, 'S' o 'N'");

                    if (elimina == "S" || elimina == "s")
                    {
                        // elimina il record dal database
                        if (GestioneDB.eseguiQuery("DELETE FROM Nuoto WHERE" +
                            " IdAtleta = " + n.Idatleta +
                            " AND DataN = (SELECT DataN" +
                                " FROM Nuoto" +
                                " WHERE IdAtleta = " + n.Idatleta +
                                " ORDER BY DataN DESC" +
                                " LIMIT 1)"))
                        {
                            // salva l'indice dell'elemento da rimuovere
                            indice = LNuoto.IndexOf(n);
                            Console.WriteLine(Environment.NewLine + "Scheda eliminata con successo" +
Environment.NewLine);
                        }
                    }
                    else
                    {
                        errore = true;
                        Console.WriteLine(Environment.NewLine + "Errore in fase di eliminazione"
+ Environment.NewLine);
                    }
                }
            }
            } while (errore);
        }
    }
    catch (Exception e) // eccezione
    {
        errore = true;
        Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
    }
}

if (indice > -1)
    LNuoto.RemoveAt(indice); // elimina la scheda dalla lista
else if (indice < 0 || (elimina == "S" || elimina == "s"))
    Console.WriteLine("Scheda non presente" + Environment.NewLine);

Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
Console.ReadKey();

```

```
// --- MOSTRA L'ULTIMA SCHEDA ---
public void visualizzaScheda(int id)    // input: id atleta
{
    bool trovato = false; // lavoro: flag per in caso di scheda presente

    foreach (SchedaNuoto n in LNuoto)
    {
        if (n.Idatleta == id && !trovato)
        {
            // mostra a video la scheda
            Console.WriteLine(n.mostraScheda());
            trovato = true;
        }
    }

    if (!trovato)
    {
        Console.Clear();
        Console.WriteLine("Non esiste una scheda nuoto per questo atleta" + Environment.NewLine);
    }
}

// --- RESTITUISCE L'ULTIMA SCHEDA ---
public Scheda ritornaScheda(int id)    // input: id atleta
{
    bool trovato = false; // lavoro: flag per in caso di scheda presente
    SchedaNuoto ultima_scheda = null; // lavoro: oggetto contenente le info della scheda

    foreach (SchedaNuoto n in LNuoto)
    {
        if (n.Idatleta == id && !trovato)
        {
            ultima_scheda = n;
            trovato = true;
        }
    }

    return ultima_scheda;
}

// --- PRENDE DATI IN INPUT ---
public Scheda inputDati()
{
    // Dichiarazione variabili
    int i = 0; // lavoro: contatore
    bool errore = false; // lavoro: flag in caso di eccezione
    SchedaNuoto nuoto = new SchedaNuoto(); // lavoro: oggetto contenente le info della scheda nuoto
    Esercizio ex = new Esercizio(); // lavoro: variabile strutturata Esercizio

    do
    {
        try
        {
            // prende in input i valori da inserire nell'oggetto
            Console.WriteLine("Inserire il tempo di esecuzione della scheda in minuti: ");
            nuoto.Durata = int.Parse(Console.ReadLine());

            Console.WriteLine(Environment.NewLine + "Inserire nome dell'istruttore: ");
            nuoto.Istruttore = Console.ReadLine();

            foreach (string stile in Enum.GetNames(typeof(Stili)))
            {
                // Prende in input serie e vasche di ogni stile
                Console.WriteLine(Environment.NewLine + "Inserire nr. di serie " + stile + ":");
                ex.serie = int.Parse(Console.ReadLine());

                Console.WriteLine(Environment.NewLine + "Inserire nr. di vasche " + stile + ":");
                ex.ripetizioni = int.Parse(Console.ReadLine());

                if (ex.serie < 0 || ex.ripetizioni < 0)
                    throw new Exception("Il numero delle serie e delle vasche deve essere positivo");
            }
            // segue l'ordine degli elementi del tipo enum
            switch (i)
            {
                case 0:

```

```

        nuoto.Crawl = ex;
        break;
    case 1:
        nuoto.Dorso = ex;
        break;
    case 2:
        nuoto.Rana = ex;
        break;
    case 3:
        nuoto.Delfino = ex;
        break;
    }

    i++;
}
}

} catch (FormatException e) // formato errato
{
    Console.WriteLine(Environment.NewLine + "Errore di input: " + e + Environment.NewLine);
    errore = true;
}
catch (Exception e) // input fuori dal dominio accettato
{
    Console.WriteLine(Environment.NewLine + "Errore di input: " + e + Environment.NewLine);
    errore = true;
}

if (errore)
{
    Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
    Console.ReadKey();
}
} while (errore);

return nuoto;
}

// --- TRASFERISCE I DATI DAL DATABASE ALLA LISTA ---
public void popolaLista()
{
    SchedaNuoto nuoto = null; // lavoro: oggetto scheda nuoto
    List<string[]> risultati = null; // lavoro: memorizza i record del database

    risultati = GestioneDB.eseguiSelect("SELECT * " +
                                        "FROM Atleti as A LEFT JOIN Nuoto as N " +
                                        "WHERE A.ID = N.IdAtleta " +
                                        "ORDER BY N.DataN DESC");

    if (risultati.Count > 0)
    {
        LNuoto.Clear(); // inizializza la scheda nuoto

        foreach (string[] record in risultati)
        {
            int Id = int.Parse(record[0]), // lavoro: id atleta
                Durata = int.Parse(record[5]); // lavoro: durata della scheda
            string Istruttore = record[6], // lavoro: nome dell'istruttore
                Data = record[16]; // lavoro: data della scheda

            nuoto = new SchedaNuoto(Id, Durata, Istruttore, Data);

            // inserisce i dati nell'oggetto nuoto
            nuoto.inserisciCrawl(int.Parse(record[7]), int.Parse(record[11]));
            nuoto.inserisciDorso(int.Parse(record[8]), int.Parse(record[12]));
            nuoto.inserisciRana(int.Parse(record[9]), int.Parse(record[13]));
            nuoto.inserisciDelfino(int.Parse(record[10]), int.Parse(record[14]));

            // inserisce l'oggetto nella lista schede
            LNuoto.Add(nuoto);
        }
    }
}
}

```

## InteraktSchedaPalestra

```

using System;
using System.Collections.Generic;

namespace GymManager
{
    // contiene i nomi dei gruppi muscolari
    public enum Muscoli
    {
        Deltoidi,
        Tricipiti,
        Bicipiti,
        Dorsali,
        Pettorali,
        Addominali,
        Gambe
    }

    class InteraktSchedaPalestra : IScheda
    {
        private static InteraktSchedaPalestra _instance = null;

        // costruttore protetto
        protected InteraktSchedaPalestra()
        {
            // niente
        }

        // crea una singola istanza
        public static InteraktSchedaPalestra Instance()
        {
            if (_instance == null)
                _instance = new InteraktSchedaPalestra();

            return _instance;
        }

        // attributo contenente le schede
        private List<SchedaPalestra> LPalestra = new List<SchedaPalestra>();

        public List<SchedaPalestra> SPalestra
        {
            get { return LPalestra; }
        }

        // --- CREA UNA NUOVA SCHEDA ---
        public void creaScheda(int id) // input: id atleta
        {
            // dichiarazione variabili
            bool errore; // lavoro: flag in caso di eccezione
            SchedaPalestra palestra = new SchedaPalestra(); // lavoro: oggetto contenente le info della scheda

            Console.WriteLine("*** Crea una nuova scheda palestra ***"
                + Environment.NewLine
                + Environment.NewLine);

            do
            {
                errore = false;

                // downcasting
                palestra = (SchedaPalestra)inputDati();
                palestra.Idatleta = id;

                // inserisce i dati nel database
                if (GestioneDB.eseguiQuery("INSERT INTO Palestra (Durata," +
                    "Istruttore," +
                    "SerieDeltoidi," +
                    "SerieTricipiti," +
                    "SerieBicipiti," +
                    "SerieDorsali," +
                    "SeriePettorali," +
                    "SerieAddominali," +
                    "SerieGambe," +
                    "RipDeltoidi," +

```

```

        "RipTricipiti," +
        "RipBicipiti," +
        "RipDorsali," +
        "RipPettorali," +
        "RipAddominali," +
        "RipGambe," +
        "IdAtleta) VALUES (" +
        palestra.Durata + "," +
        palestra.Istruttore + "," +
        palestra.Deltoidi.serie + "," +
        palestra.Tricipiti.serie + "," +
        palestra.Bicipiti.serie + "," +
        palestra.Dorsali.serie + "," +
        palestra.Pettorali.serie + "," +
        palestra.Addominali.serie + "," +
        palestra.Gambe.serie + "," +
        palestra.Deltoidi.ripetizioni + "," +
        palestra.Tricipiti.ripetizioni + "," +
        palestra.Bicipiti.ripetizioni + "," +
        palestra.Dorsali.ripetizioni + "," +
        palestra.Pettorali.ripetizioni + "," +
        palestra.Addominali.ripetizioni + "," +
        palestra.Gambe.ripetizioni + "," +
        id + ")"))

    {
        // aggiunge l'oggetto alla lista
        LPalestra.Add(palestra);
        popolaLista();
        Console.WriteLine("Inserimento effettuato" + Environment.NewLine);
    }
    else
    {
        errore = true;
        Console.WriteLine("Errore di inserimento" + Environment.NewLine);
    }

    Console.WriteLine("Premere un tasto per continuare...");
    Console.ReadKey();
} while (errore);
}

// --- MODIFICA UNA SCHEDA ---
public void modificaScheda(int id) // input: id atleta
{
    bool errore, // lavoro: flag in caso di eccezione
        trovato = false; // lavoro: flag in caso di record presente
    SchedaPalestra palestra = new SchedaPalestra(); // lavoro: oggetto scheda

    foreach (SchedaPalestra p in LPalestra)
    {
        if (p.Idatleta == id)
        {
            trovato = true;

            Console.WriteLine("*** Aggiorna la scheda palestra ***" + Environment.NewLine +
Environment.NewLine);
            do
            {
                errore = false;

                // downcasting
                palestra = (SchedaPalestra)inputDati();

                // aggiorna id dati nel db
                if (GestioneDB.eseguiQuery("UPDATE Palestra SET " +
                    "Istruttore = " + p.Istruttore +
                    ",SerieDeltoidi = " + p.Deltoidi.serie +
                    ",SerieTricipiti = " + p.Tricipiti.serie +
                    ",SerieBicipiti = " + p.Bicipiti.serie +
                    ",SerieDorsali = " + p.Dorsali.serie +
                    ",SeriePettorali = " + p.Pettorali.serie +
                    ",SerieAddominali = " + p.Addominali.serie +
                    ",SerieGambe = " + p.Gambe.serie +
                    ",RipDeltoidi = " + p.Deltoidi.ripetizioni +
                    ",RipTricipiti = " + p.Tricipiti.ripetizioni +
                    ",RipBicipiti = " + p.Bicipiti.ripetizioni +

```

```

        ",RipDorsali = " + p.Dorsali.ripetizioni +
        ",RipPettorali = " + p.Pettorali.ripetizioni +
        ",RipAddominali = " + p.Addominali.ripetizioni +
        ",RipGambe = " + p.Gambe.ripetizioni +
        " WHERE IdAtleta = " + p.Idatleta +
        " AND DataP = (SELECT DataP" +
        " FROM Palestra" +
        " WHERE IdAtleta = " + p.Idatleta +
        " ORDER BY DataP DESC" +
        " LIMIT 1)"))
    {
        // aggiorna l'oggetto
        p.Idatleta = id;
        p.Istruttore = palestra.Istruttore;
        p.Durata = palestra.Durata;
        p.Istruttore = palestra.Istruttore;
        p.Deltoidi = palestra.Deltoidi;
        p.Tricipiti = palestra.Tricipiti;
        p.Bicipiti = palestra.Bicipiti;
        p.Dorsali = palestra.Dorsali;
        p.Pettorali = palestra.Pettorali;
        p.Addominali = palestra.Addominali;
        p.Gambe = palestra.Gambe;

        Console.WriteLine("Aggiornamento effettuato" + Environment.NewLine);
    }
    else
    {
        errore = true;
        Console.WriteLine("Errore di aggiornamento" + Environment.NewLine);
    }
} while (errore);
}

if (!trovato)
    Console.WriteLine("Scheda non presente" + Environment.NewLine);

Console.WriteLine("Premere un tasto per continuare...");
Console.ReadKey();
}

// --- CANCELLA L'ULTIMA SCHEDA ---
public void eliminaScheda(int id) // input: id dell'atleta
{
    int indice = -1; // lavoro: flag per scheda presente
    bool errore; // lavoro: flag per errore
    string elimina = "N"; // lavoro: conferma eliminazione

    foreach (SchedaPalestra p in LPalestra)
    {
        if (p.Idatleta == id)
        {
            do
            {
                errore = false;
                try
                {
                    Console.WriteLine("Eliminare la scheda? S/N");
                    elimina = Console.ReadLine();

                    if (!(elimina is string))
                        throw new Exception("Inserire una lettera, 'S' o 'N'");

                    if (elimina == "S" || elimina == "s")
                    {
                        // elimina il record dal database
                        if (GestioneDB.eseguiQuery("DELETE FROM Palestra WHERE" +
                            " IdAtleta = " + p.Idatleta +
                            " AND DataP = (SELECT DataP" +
                            " FROM Palestra" +
                            " WHERE IdAtleta = " + p.Idatleta +
                            " ORDER BY DataP DESC" +
                            " LIMIT 1)"))
                        {
                            // salva l'indice dell'elemento da rimuovere

```

```

        indice = LPalestra.IndexOf(p);
        Console.WriteLine(Environment.NewLine + "Scheda eliminata con successo" +
Environment.NewLine);
    }
    else
    {
        errore = true;
        Console.WriteLine("Errore in fase di eliminazione" +
Environment.NewLine);
    }
}
}
catch (Exception e)
{
    errore = true;
    Console.WriteLine("Errore di input: " + e + Environment.NewLine);
}
} while (errore);
}
}

if (indice > -1)
    LPalestra.RemoveAt(indice); // elimina la scheda dalla lista
else if (indice < 0 || (elimina == "S" || elimina == "s"))
    Console.WriteLine("Scheda non presente" + Environment.NewLine);

Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare...");
Console.ReadKey();
}
// --- MOSTRA L'ULTIMA SCHEDA DELL'ATLETA RICHIESTO
public void visualizzaScheda(int id) // input: id atleta
{
    bool trovato = false; // lavoro: flag per in caso di scheda presente

    foreach (SchedaPalestra p in LPalestra)
    {
        if (p.Idatleta == id && !trovato)
        {
            // mostra a video la scheda
            Console.WriteLine(p.mostraScheda());
            trovato = true;
        }
    }

    if (!trovato)
    {
        Console.Clear();
        Console.WriteLine("Non esiste una scheda palestra per questo atleta" + Environment.NewLine);
    }
}

// --- RESTITUISCE L'ULTIMA SCHEDA ---
public Scheda ritornaScheda(int id) // input: id atleta
{
    bool trovato = false; // lavoro: flag per in caso di scheda presente
    SchedaPalestra ultima_scheda = null; // lavoro: oggetto contenente le info della scheda

    foreach (SchedaPalestra p in LPalestra)
    {
        if (p.Idatleta == id && !trovato)
        {
            ultima_scheda = p;
            trovato = true;
        }
    }

    return ultima_scheda;
}

// --- PRENDE DATI IN INPUT ---
public Scheda inputDati()
{
    // Dichiarazione variabili
    int i = 0; // lavoro: contatore
    bool errore = false; // lavoro: flag in caso di eccezione

```

```

        SchedaPalestra palestra = new SchedaPalestra(); // lavoro: oggetto contenente le info della
scheda
        Esercizio ex = new Esercizio(); // lavoro: struttura di appoggio

        do
        {
            try
            {
                // prende in input i valori da inserire nell'oggetto
                Console.WriteLine("Inserire il tempo di esecuzione della scheda in minuti: ");
                palestra.Durata = int.Parse(Console.ReadLine());

                Console.WriteLine(Environment.NewLine + "Inserire nome dell'istruttore: ");
                palestra.Istruttore = Console.ReadLine();

                foreach (string muscolo in Enum.GetNames(typeof(Muscoli)))
                {
                    // Prende in input serie e vasche di ogni stile
                    Console.WriteLine(Environment.NewLine + "Inserire nr. di serie di " + muscolo + ":");
                    ex.serie = int.Parse(Console.ReadLine());

                    Console.WriteLine(Environment.NewLine + "Inserire nr. di ripetizioni di " + muscolo +
":");
                    ex.ripetizioni = int.Parse(Console.ReadLine());

                    if (ex.serie < 0 || ex.ripetizioni < 0)
                        throw new Exception("Il numero delle serie e delle ripetizioni deve essere > 0");
                    else
                    {
                        // segue l'ordine degli elementi del tipo enum
                        switch (i)
                        {
                            case 0:
                                palestra.Deltoidi = ex;
                                break;
                            case 1:
                                palestra.Tricipiti = ex;
                                break;
                            case 2:
                                palestra.Bicipiti = ex;
                                break;
                            case 3:
                                palestra.Dorsali = ex;
                                break;
                            case 4:
                                palestra.Pettorali = ex;
                                break;
                            case 5:
                                palestra.Addominali = ex;
                                break;
                            case 6:
                                palestra.Gambe = ex;
                                break;
                        }

                        i++;
                    }
                }
            }
            catch (FormatException e) // formato errato
            {
                Console.WriteLine("Errore di input: " + e + Environment.NewLine);
                errore = true;
            }
            catch (Exception e) // input fuori dal dominio accettato
            {
                Console.WriteLine("Errore di input: " + e + Environment.NewLine);
                errore = true;
            }

            if (errore)
            {
                Console.WriteLine("Premere un tasto per continuare...");
                Console.ReadKey();
                Console.Clear();
            }
        } while (errore);

```



```

        return palestra;
    }

    // --- TRASFERISCE I DATI DAL DATABASE ALLA LISTA
    public void popolaLista()
    {
        SchedaPalestra palestra;          // lavoro: oggetto scheda palestra
        List<string[]> risultati = null;    // lavoro: memorizza i record del database

        risultati = GestioneDB.eseguiSelect("SELECT * " +
            "FROM Atleti as A LEFT JOIN Palestra as P " +
            "WHERE A.ID = P.IdAtleta " +
            "ORDER BY P.DataP DESC");

        if(risultati.Count > 0)
        {
            LPalestra.Clear(); // inizializza la lista

            foreach (string[] record in risultati)
            {
                int Id = int.Parse(record[0]),      // lavoro: id atleta
                    Durata = int.Parse(record[5]); // lavoro: durata della scheda
                string Istruttore = record[6],      // lavoro: nome dell'istruttore
                    Data = record[22];              // lavoro: data della scheda

                palestra = new SchedaPalestra(Id, Durata, Istruttore, Data);

                // inserisce i dati nell'oggetto palestra
                palestra.inserisciDeltoidi(int.Parse(record[7]), int.Parse(record[14]));
                palestra.inserisciTricipiti (int.Parse(record[8]), int.Parse(record[15]));
                palestra.inserisciBicipiti(int.Parse(record[9]), int.Parse(record[16]));
                palestra.inserisciDorsali(int.Parse(record[10]), int.Parse(record[17]));
                palestra.inserisciPettorali(int.Parse(record[11]), int.Parse(record[18]));
                palestra.inserisciAddominali (int.Parse(record[12]), int.Parse(record[19]));
                palestra.inserisciGambe(int.Parse(record[13]), int.Parse(record[20]));

                // inserisce l'oggetto nella lista schede
                LPalestra.Add(palestra);
            }
        }
    }
}

```

## InteraktSchede

```

using System;

namespace GymManager
{
    static class InteraktSchede
    {
        // --- TEMPO TOTALE DI ESECUZIONE DI ENTRAMBE LE SCHEDE ---
        public static void tempoEsecuzione(Scheda s1, // input: oggetto Scheda
            Scheda s2) // input: oggetto Scheda
        {
            int tempo_totale, // lavoro: tempo totale di allenamento
                tempoS1 = 0,  // lavoro: tempo di allenamento scheda 1
                tempoS2 = 0;  // lavoro: tempo di allenamento scheda 2

            try
            {
                tempoS1 = s1.Durata;
            }
            catch (NullReferenceException) // oggetto inesistente
            {
                tempoS1 = 0;
            }

            try
            {
                tempoS2 = s2.Durata;
            }
            catch (NullReferenceException) // oggetto inesistente
            {
            }
        }
    }
}

```

```

        {
            tempoS2 = 0;
        }

        tempo_totale = tempoS1 + tempoS2;

        Console.WriteLine(Environment.NewLine + "Il tempo totale di esecuzione di entrambe le schede e':
" +
                                tempo_totale + " minuti" + Environment.NewLine);

        Console.WriteLine("Premere un tasto per tornare al menu principale...");
        Console.ReadKey();
    }
}

```

## Interfacce

```

namespace GymManager
{
    interface IScheda
    {
        void creaScheda(int id);
        void modificaScheda(int id);
        void eliminaScheda(int id);
        void visualizzaScheda(int id);
        Scheda ritornaScheda(int id);
        Scheda inputDati();
        void popolaLista();
    }

    interface IAtleta
    {
        void aggiungiAtleta();
        void modificaAtleta(int id);
        void eliminaAtleta(int id);
        int mostraTutti();
        int mostraUno(int id);
        Atleta inputDati();
        int ultimoInserito(Atleta a);
        void popolaLista();
    }
}

```

## MainApp

```

using System;

namespace GymManager
{
    public static class MainApp
    {
        public static void Applicazione()
        {
            // Inizializza le variabili
            int scelta_generale = -1, // lavoro: opzione del menu generale
                scelta_atleta = -1, // lavoro: opzione del menu atleta
                scelta_scheda = -1, // lavoro: opzione del menu scheda
                id = 0; // lavoro: id dell'atleta

            // istanzia le classi Interakt con il metodo Singleton
            InteraktAtleta interaktAtleta = InteraktAtleta.Instance();
            InteraktSchedaPalestra interaktSchedaPalestra = InteraktSchedaPalestra.Instance();
            InteraktSchedaNuoto interaktSchedaNuoto = InteraktSchedaNuoto.Instance();

            // Crea il database se non esiste e apre la connessione
            GestioneDB.creaDB();

            // Trasferisce i dati dal db alle strutture dati
            interaktAtleta.popolaLista();
            interaktSchedaPalestra.popolaLista();
            interaktSchedaNuoto.popolaLista();

            do
            {

```

```

do
{
    Console.WriteLine("--- GESTORE SCHEDE ---" + Environment.NewLine);

    // Mostra il menu generale
    scelta_generale = Menu.menuGenerale();
} while (scelta_generale < 1 || scelta_generale > 3);

switch (scelta_generale)
{
    case 1:
        // Mostra l'elenco di tutti gli atleti iscritti
        interagAtleta.mostraTutti();

        // Sceglie e visualizza l'atleta in base all'id prescelto
        id = interagAtleta.mostraUno(0);

        // Controllo del valore id
        if (id > 0)
        {
            do
            {
                interagAtleta.mostraUno(id);
                // Mostra le opzioni relative all'atleta prescelto
                scelta_atleta = Menu.menuAtleta();

            } while (scelta_atleta < 0 || scelta_atleta > 6);

            switch (scelta_atleta)
            {
                case 1: // Modifica le info dell'atleta prescelto
                    interagAtleta.modificaAtleta(id);
                    break;
                case 2: // Elimina l'atleta prescelto
                    interagAtleta.eliminaAtleta(id);
                    break;
                case 3: // Mostra la scheda dell'atleta prescelto
                    do
                    {
                        interagSchedaPalestra.visualizzaScheda(id);

                        // Mostra le opzioni relative alla scheda palestra dell'atleta
                        scelta_scheda = Menu.menuScheda();

                    } while (scelta_scheda < 0 || scelta_scheda > 4);

                    switch (scelta_scheda)
                    {
                        case 1: // Crea una nuova scheda palestra
                            interagSchedaPalestra.creaScheda(id);
                            break;
                        case 2: // Modifica la scheda palestra
                            interagSchedaPalestra.modificaScheda(id);
                            break;
                        case 3: // Elimina la scheda palestra
                            interagSchedaPalestra.eliminaScheda(id);
                            break;
                        default: // Torna al menu principale
                            scelta_generale = 0;
                            break;
                    }
                    break;
                case 4: // Mostra la scheda nuoto dell'atleta
                    interagSchedaNuoto.visualizzaScheda(id);

                    // Mostra le opzioni relative alla scheda nuoto dell'atleta prescelto
                    scelta_scheda = Menu.menuScheda();
                    switch (scelta_scheda)
                    {
                        case 1: // Crea una nuova scheda palestra
                            interagSchedaNuoto.creaScheda(id);
                            break;
                        case 2: // Modifica la scheda palestra
                            interagSchedaNuoto.modificaScheda(id);
                            break;
                        case 3: // Elimina la scheda palestra

```

```

        interaktSchedaNuoto.eliminaScheda(id);
        break;
    default: // Torna al menu principale
        scelta_generale = 0;
        break;
    }
    break;
case 5: // Calcola la durata di entrambe le schede
    InteraktSchede.tempoEsecuzione(interaktSchedaNuoto.ritornaScheda(id),
                                    interaktSchedaPalestra.ritornaScheda(id));
    break;
default: // Torna al menu principale
    scelta_generale = 0;
    break;
    }
}
else // Torna al menu principale
    scelta_generale = 0;
break;
case 2: // Aggiunge un nuovo atleta
    interaktAtleta.aggiungiAtleta();
    scelta_generale = 0;
    break;
default: // Termina l'esecuzione
    Console.WriteLine("Buona giornata..." + Environment.NewLine);
    Console.WriteLine("Premere un tasto per terminare...");
    Console.ReadKey();
    break;
}

// Pulisce lo schermo
Console.Clear();

} while (scelta_generale == 0 || scelta_generale != 3);
}
}
}

```

## Menu

```

using System;

namespace GymManager
{
    public static class Menu
    {
        public static int menuGenerale()
        {
            int scelta = 0; // output: input dell'utente

            try
            {
                Console.WriteLine("-----" + Environment.NewLine +
                                   "1. Visualizza Info Atleta" + Environment.NewLine +
                                   "2. Aggiungi Atleta" + Environment.NewLine +
                                   "3. Termina app" + Environment.NewLine + Environment.NewLine +
                                   "Inserire il numero dell'opzione desiderata:");

                scelta = int.Parse(Console.ReadLine());

                if (scelta < 1 || scelta > 3)
                    throw (new Exception("Il valore inserito, " + scelta +
                                           " deve essere un numero intero compreso tra 1 e 3" +
                                           Environment.NewLine));
            }
            catch (FormatException e) // formato errato
            {
                Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
            }
            catch (Exception e) // eccezione
            {
                Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
            }

            if (scelta < 1 || scelta > 3)

```

```

    {
        Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare... ");
        Console.ReadKey();
    }

    Console.Clear();

    return scelta;
}

public static int menuAtleta()
{
    int scelta = 0; // output: input dell'utente

    try
    {
        scelta = -1;

        Console.WriteLine("-----" + Environment.NewLine +
            "1. Modifica atleta" + Environment.NewLine +
            "2. Elimina atleta" + Environment.NewLine +
            "3. Visualizza ultima scheda Palestra" + Environment.NewLine +
            "4. Visualizza ultima Scheda Nuoto" + Environment.NewLine +
            "5. Durata totale delle schede" + Environment.NewLine +
            "6. Torna al menu principale" + Environment.NewLine + Environment.NewLine +
            "Inserire il numero dell'opzione desiderata:");

        scelta = int.Parse(Console.ReadLine());

        if (scelta < 1 || scelta > 6)
            throw (new Exception("Il valore inserito, " + scelta +
                " deve essere un numero intero compreso tra 1 e 6" +
                Environment.NewLine));
    }
    catch (FormatException e) // formato errato
    {
        Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
    }
    catch (Exception e) // eccezione
    {
        Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
    }

    if (scelta < 0 || scelta > 6)
    {
        Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare... ");
        Console.ReadKey();
    }

    Console.Clear();

    return scelta;
}

public static int menuScheda()
{
    int scelta = -1; // output: input dell'utente

    try
    {
        Console.WriteLine("-----" + Environment.NewLine +
            "1. Aggiungi scheda" + Environment.NewLine +
            "2. Modifica scheda" + Environment.NewLine +
            "3. Elimina scheda" + Environment.NewLine +
            "4. Torna al menu principale" + Environment.NewLine +
Environment.NewLine +
            "Inserire il numero dell'opzione desiderata:");

        scelta = int.Parse(Console.ReadLine());

        if (scelta < 1 || scelta > 4)
            throw (new Exception("Il valore inserito, " + scelta +
                " deve essere un numero intero compreso tra 1 e 4" +
                Environment.NewLine));
    }
    catch (FormatException e) // formato errato

```

```

        {
            Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
        }
        catch (Exception e) // formato errato
        {
            Console.WriteLine(Environment.NewLine + "Errore di input: " + e);
        }

        if (scelta < 0 || scelta > 4)
        {
            Console.WriteLine(Environment.NewLine + "Premere un tasto per continuare... ");
            Console.ReadKey();
        }

        Console.Clear();

        return scelta;
    }
}

```

## Program

```

namespace GymManager
{
    class Program
    {
        static void Main(string[] args)
        {
            MainApp.Applicazione();
        }
    }
}

```

## Scheda

```

namespace GymManager
{
    abstract class Scheda
    {
        // attributi
        private int _idatleta,
            _durata;
        private string _istruttore,
            _data;

        // costruttore
        public Scheda()
        {
            // niente
        }

        // costruttore
        public Scheda(int idatleta,
            int durata,
            string istruttore,
            string data)
        {
            _idatleta = idatleta;
            _durata = durata;
            _istruttore = istruttore;
            _data = data;
        }

        // proprietà
        public int Idatleta
        {
            get { return _idatleta; }
            set { _idatleta = value; }
        }

        public int Durata
        {

```

```

        get { return _durata; }
        set { _durata = value; }
    }

    public string Istruttore
    {
        get { return _istruttore; }
        set { _istruttore = value; }
    }

    public string Data
    {
        get { return _data; }
        set { _data = value; }
    }

    // metodo astratto
    public abstract string mostraScheda();
}

```

## SchedaNuoto

```

using System;

namespace GymManager
{
    class SchedaNuoto : Scheda
    {
        // attributi
        private Esercizio _vascheCrawl,
                        _vascheDorso,
                        _vascheRana,
                        _vascheDelfino;

        // costruttore
        public SchedaNuoto()
        {
            // niente
        }

        // costruttore
        public SchedaNuoto(int id,
                          int durata,
                          string istruttore,
                          string data)
            : base(id, durata, istruttore, data)
        {
            // niente
        }

        // Proprietà
        public Esercizio Crawl
        {
            get { return _vascheCrawl; }
            set { _vascheCrawl = value; }
        }

        public Esercizio Dorso
        {
            get { return _vascheDorso; }
            set { _vascheDorso = value; }
        }

        public Esercizio Rana
        {
            get { return _vascheRana; }
            set { _vascheRana = value; }
        }

        public Esercizio Delfino
        {
            get { return _vascheDelfino; }
            set { _vascheDelfino = value; }
        }
    }
}

```

```

// metodi
public void inserisciCrawl(int s,
                          int r)
{
    _vascheCrawl.serie = s;
    _vascheCrawl.ripetizioni = r;
}

public void inserisciDorso(int s,
                          int r)
{
    _vascheDorso.serie = s;
    _vascheDorso.ripetizioni = r;
}

public void inserisciRana(int s,
                        int r)
{
    _vascheRana.serie = s;
    _vascheRana.ripetizioni = r;
}

public void inserisciDelfino(int s,
                          int r)
{
    _vascheDelfino.serie = s;
    _vascheDelfino.ripetizioni = r;
}

public int contaVasche()
{
    int totale = 0; // output: numero totale delle vasche

    totale = Crawl.serie * Crawl.ripetizioni +
            Dorso.serie * Dorso.ripetizioni +
            Rana.serie * Rana.ripetizioni +
            Delfino.serie * Delfino.ripetizioni;

    return totale;
}

// metodo ereditato da Scheda
public override string mostraScheda()
{
    string result = null; // output: contiene la versione testuale della scheda

    result = "Scheda Nuoto" + Environment.NewLine + Environment.NewLine +
            "Istruttore: " + Istruttore + "\tData: " + Data + "\tDurata: " + Durata + " min" +
Environment.NewLine + Environment.NewLine +
            "Nr. vasche a Stile Libero: " + Crawl.serie + " x " + Crawl.ripetizioni +
Environment.NewLine +
            "Nr. vasche a Dorso: " + Dorso.serie + " x " + Dorso.ripetizioni + Environment.NewLine +
            "Nr. vasche a Rana: " + Rana.serie + " x " + Rana.ripetizioni + Environment.NewLine +
            "Nr. vasche a Delfino: " + Delfino.serie + " x " + Delfino.ripetizioni +
Environment.NewLine +
            "Totale vasche: " + contaVasche() + Environment.NewLine +
            "-----";

    return result;
}
}
}

```

## SchedaPalestra

```

using System;

namespace GymManager
{
    class SchedaPalestra : Scheda
    {
        // attributi
        private Esercizio _deltoidi,
            _tricipiti,

```



```
        _bicipiti,
        _dorsali,
        _pettorali,
        _addominali,
        _gambe;

// costruttore
public SchedaPalestra()
{
    // niente
}

// costruttore
public SchedaPalestra(int id,
                      int durata,
                      string istruttore,
                      string data)
    : base(id, durata, istruttore, data)
{
    // niente
}

// Proprietà
public Esercizio Deltoidi
{
    get { return _deltoidi; }
    set { _deltoidi = value; }
}

public Esercizio Tricipiti
{
    get { return _tricipiti; }
    set { _tricipiti = value; }
}

public Esercizio Bicipiti
{
    get { return _bicipiti; }
    set { _bicipiti = value; }
}

public Esercizio Dorsali
{
    get { return _dorsali; }
    set { _dorsali = value; }
}

public Esercizio Pettorali
{
    get { return _pettorali; }
    set { _pettorali = value; }
}

public Esercizio Addominali
{
    get { return _addominali; }
    set { _addominali = value; }
}

public Esercizio Gambe
{
    get { return _gambe; }
    set { _gambe = value; }
}

// Metodi
public void inserisciDeltoidi(int s,
                             int r)
{
    _deltoidi.serie = s;
    _deltoidi.ripetizioni = r;
}

public void inserisciTricipiti(int s,
                              int r)
{

```

```

        _tricipiti.serie = s;
        _tricipiti.ripetizioni = r;
    }

    public void inserisciBicipiti(int s,
                                  int r)
    {
        _bicipiti.serie = s;
        _bicipiti.ripetizioni = r;
    }

    public void inserisciDorsali(int s,
                                  int r)
    {
        _dorsali.serie = s;
        _dorsali.ripetizioni = r;
    }

    public void inserisciPettorali(int s,
                                    int r)
    {
        _pettorali.serie = s;
        _pettorali.ripetizioni = r;
    }

    public void inserisciAddominali(int s,
                                      int r)
    {
        _addominali.serie = s;
        _addominali.ripetizioni = r;
    }

    public void inserisciGambe(int s,
                                int r)
    {
        _gambe.serie = s;
        _gambe.ripetizioni = r;
    }

    // metodo ereditato da Scheda
    public override string mostraScheda()
    {
        string result = null; // output: contiene la versione testuale della scheda

        result = "Scheda Palestra" + Environment.NewLine + Environment.NewLine +
            "Istruttore: " + Istruttore + "\tData: " + Data + "\tDurata: " + Durata + " min" +
Environment.NewLine + Environment.NewLine +
            "Deltoidi: " + Deltoidi.serie + " x " + Deltoidi.ripetizioni + Environment.NewLine +
            "Tricipiti: " + Tricipiti.serie + " x " + Tricipiti.ripetizioni + Environment.NewLine +
            "Bicipiti: " + Bicipiti.serie + " x " + Bicipiti.ripetizioni + Environment.NewLine +
            "Dorsali: " + Dorsali.serie + " x " + Dorsali.ripetizioni + Environment.NewLine +
            "Pettorali: " + Pettorali.serie + " x " + Pettorali.ripetizioni + Environment.NewLine +
            "Addominali: " + Addominali.serie + " x " + Addominali.ripetizioni + Environment.NewLine
+
            "Gambe: " + Gambe.serie + " x " + Gambe.ripetizioni + Environment.NewLine +
            "-----";

        return result;
    }
}

```

## V – Testing del programma

### V.1 – Test durante lo sviluppo

Si è preferito testare le funzionalità del programma durante lo sviluppo, resolvendo i problemi sintattici e semantici di volta in volta. Si è deciso di procedere in tal senso perché l'applicazione è costituita da diverse classi interconnesse e un errore in una avrebbe avuto ripercussioni sulle altre, rendendo il testing finale più complesso e dispendioso in termini di tempo.

## V.2 – Test White-Box

Il test del white-box rappresenta una classe di test che considera il punto di vista dello sviluppatore, quindi richiede la conoscenza del codice interno al programma e la creazione del diagramma di flusso relativo a tale codice o alla porzione che si vuole testare. Esistono diversi tipi di white-box, ognuno con un diverso percorso logico, in base a quale parte del diagramma si desidera analizzare.

In questa sede testeremo il comportamento dei metodi della classe statica Menu e della classe InteraktSchedaNuoto

### V.2.1 – Branch Coverage Test

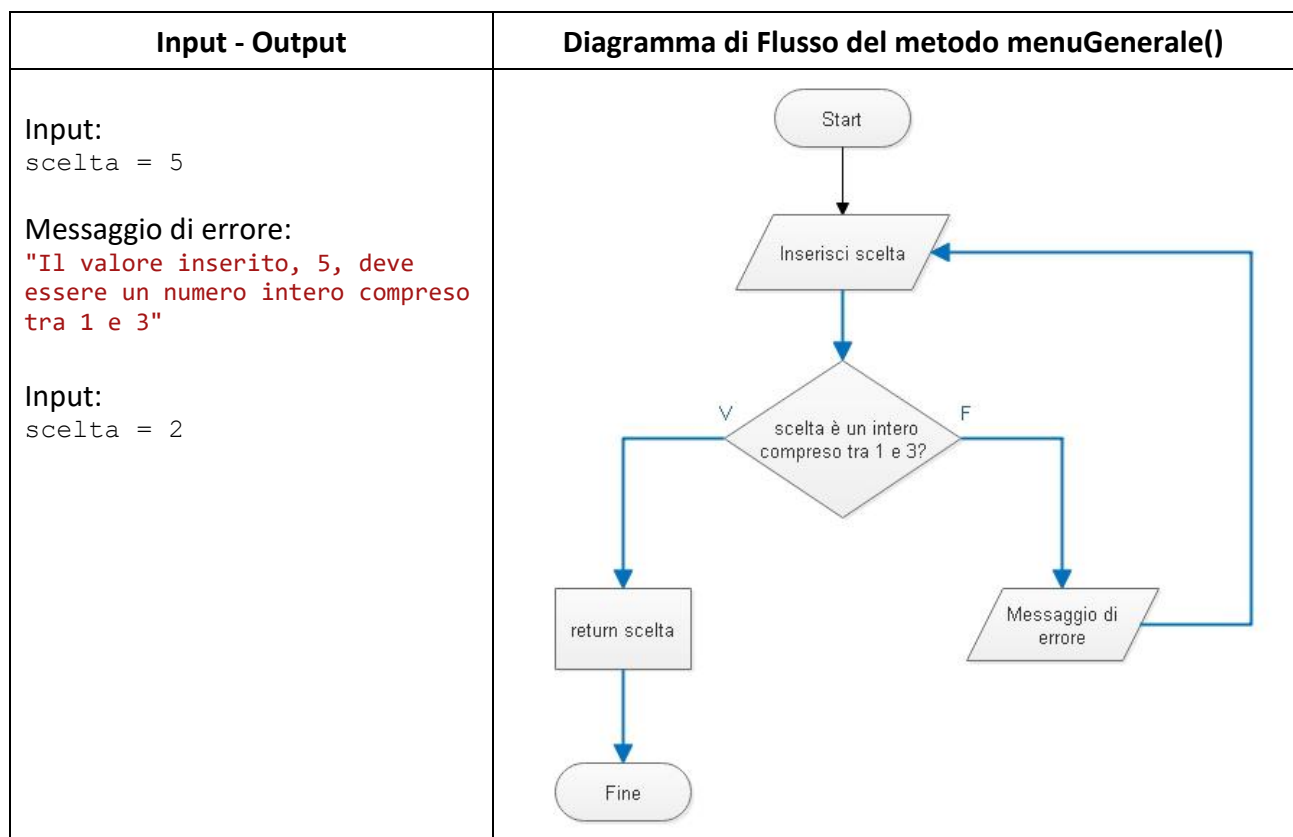
Questa tipologia di test verifica che tutti i rami del diagramma di flusso della funzione vengano attraversati almeno una volta.

Prendiamo il metodo menuGenerale(), all'interno della classe statica Menu. Il metodo mostra a video una serie di scelte testuali e attende in input un valore intero positivo compreso tra 1 e 3. Se l'input risponde ai criteri, la funzione ritorna il suddetto valore in output, altrimenti mostra un messaggio di errore e torna all'inserimento della scelta con un ciclo do-while.

Consideriamo quindi 2 input diversi:

1. l'input non risponde ai criteri
2. l'input risponde ai criteri

Come possiamo notare dal diagramma, siamo riusciti a coprire tutti i rami almeno una volta



## V.2.1 – Condition Coverage Test

Questa tipologia di test verifica che ogni elemento di un blocco condizionale composto sia vero e falso almeno una volta.

Prendiamo il metodo `ritornaScheda()`, all'interno della classe `InteraktSchedaNuoto`. Il metodo scorre tutti gli elementi della lista `schede` con un ciclo `foreach`. All'interno del ciclo c'è una condizione `if` doppia: cioè è vera quando l'id dato in input è uguale a quello dell'atleta nell'oggetto `scheda` della lista e quando il flag `trovato` è impostato su `false`. Se entrambe le condizioni sono verificate, esegue il blocco d'istruzioni all'interno. Terminati gli elementi della lista, esce dal ciclo e ritorna l'oggetto `scheda`, se trovato.

Immaginiamo che la lista sia composta da N elementi e che quello cercato si trovi fortunatamente in prima posizione.

Consideriamo quindi 2 casi diversi:

1. la scheda è presente nella lista
2. la scheda non è presente nella lista

Ogni elemento del blocco condizionale composto è vero e falso almeno una volta, così come l'elemento del blocco condizionale del ciclo `foreach`.

Input - Output	Diagramma di Flusso del metodo <code>ritornaScheda()</code>
<p><u>1° giro:</u></p> <p>Input: id</p> <p>Condizioni:  <code>n.idatleta == id</code> <b>TRUE</b>  <code>trovato == FALSE</code> (!trovato = TRUE)</p> <p>Valori uguali: entra nel ciclo</p> <p>Blocco d'istruzioni:  <code>trovato = TRUE</code>  <code>ultima_scheda = n</code></p> <p>Riprende il ciclo</p>	<pre> graph TD     Start([Start]) --&gt; Cond1{La lista è terminata?}     Cond1 -- V --&gt; Return[return ultima_scelta]     Cond1 -- F --&gt; Cond2{n.idatleta == id &amp;&amp; !trovato}     Cond2 -- V --&gt; Blocco[Blocco di istruzioni]     Blocco --&gt; Return     Cond2 -- F --&gt; Start     Return --&gt; Fine([Fine])   </pre>
<p><u>2° fino a N-mo giro:</u></p> <p>Input: id</p> <p>Condizioni:  <code>n.idatleta == id</code> <b>FALSE</b>  <code>trovato == TRUE</code></p> <p>Valori diversi: non entra nel ciclo</p> <p>Output: ultima_scheda</p>	

### V.3 – Test Black-box

Al contrario del precedente, questo è un test svolto dal punto di vista dell'utilizzatore, cioè di colui che non conosce il codice dietro l'applicazione. Da qui il nome: black-box.

#### V.3.1 – Funzioni errate o mancanti

Non sono stati riscontrati errori

#### V.3.2 – Errori durante l'accesso al database

Non sono stati riscontrati errori

#### V.3.3 – Errori nelle prestazioni

Ho testato la scrittura/lettura di 1000 record nel database.

Non sono stati riscontrati errori o cali di prestazioni degni di nota dal punto di vista dell'utente.

È bene sottolineare che non si è proceduto ad un calcolo dei millisecondi; sarebbe stato sicuramente più oggettivo e quindi attendibile, ma anche in contrasto con la natura stessa del test BB.

#### V.3.4 – Errori di inizializzazione o terminazione

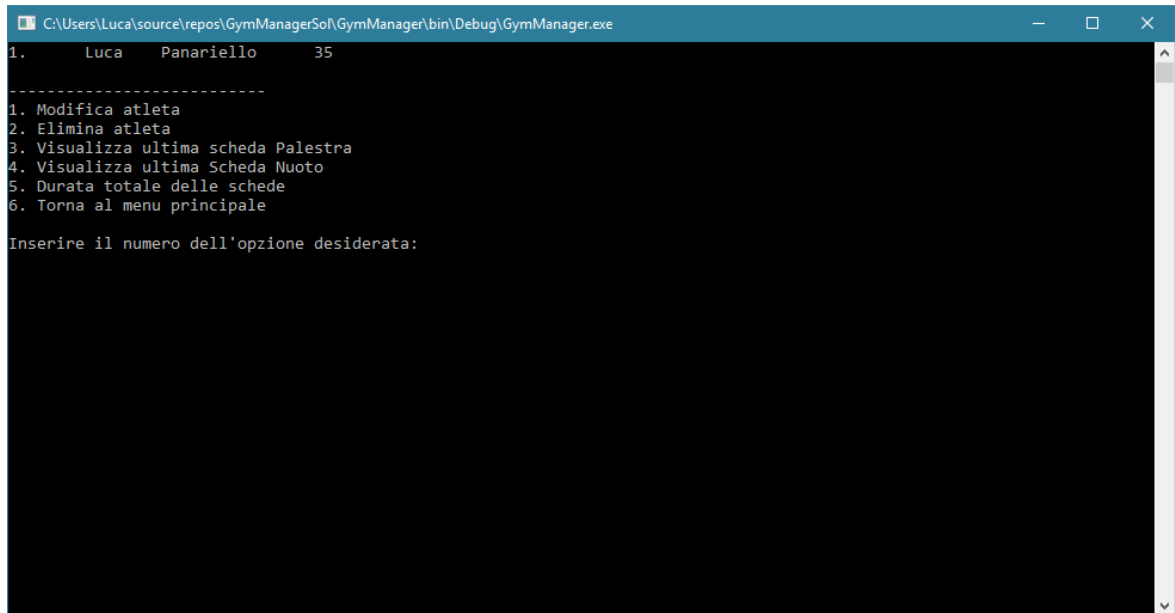
Non sono stati riscontrati errori

## V.3.4 – Valutazione dei casi d'uso

## Test caso d'uso con ID #3

Precondizione: postcondizione del caso d'uso con ID #1

1. Visualizza Info Atleta: mostra la pagina personale dell'atleta

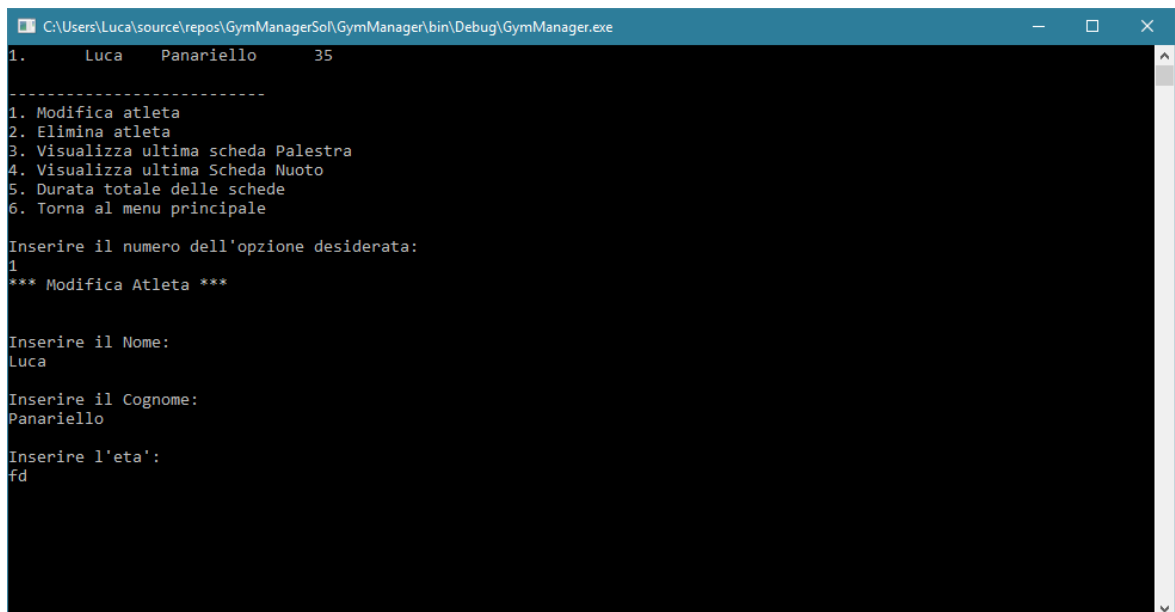


```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
1.      Luca      Panariello      35
-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

Inserire il numero dell'opzione desiderata:
```

2. Inseriamo 1, premiamo Invio e inseriamo l'input come richiesto volta per volta.

Provochiamo un errore in modo da verificare uno dei percorsi alternativi



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
1.      Luca      Panariello      35
-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

Inserire il numero dell'opzione desiderata:
1
*** Modifica Atleta ***

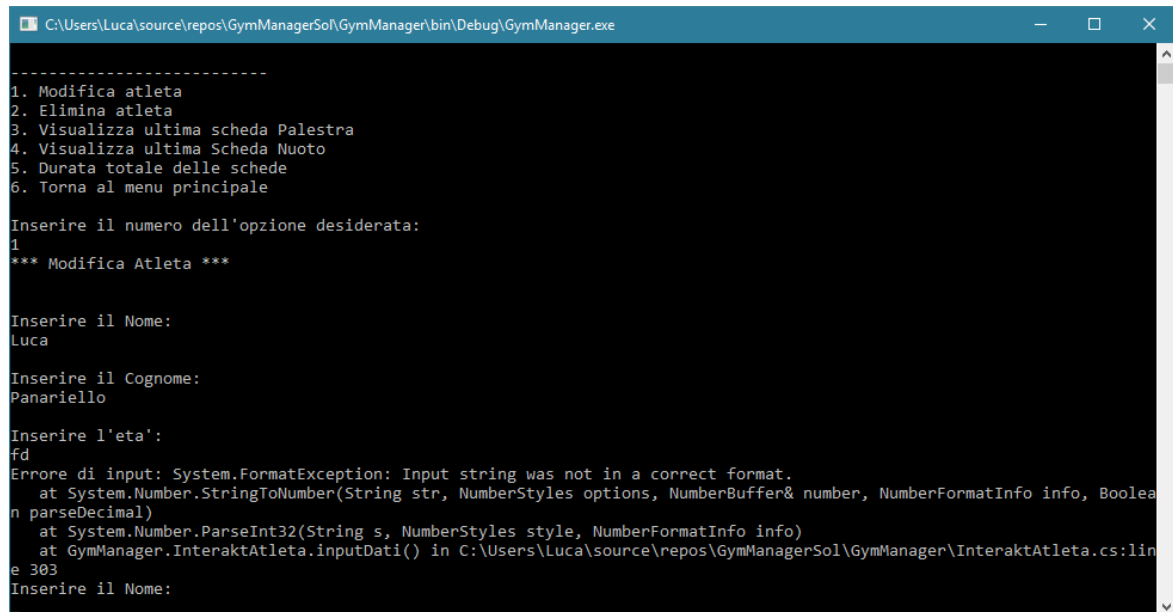
Inserire il Nome:
Luca

Inserire il Cognome:
Panariello

Inserire l'eta':
fd
```

*Continua...*

3. Abbiamo inserito delle lettere invece che dei numeri e infatti l'applicazione ha risposto come ci aspettavamo: mostrando un errore a schermo.



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe

-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

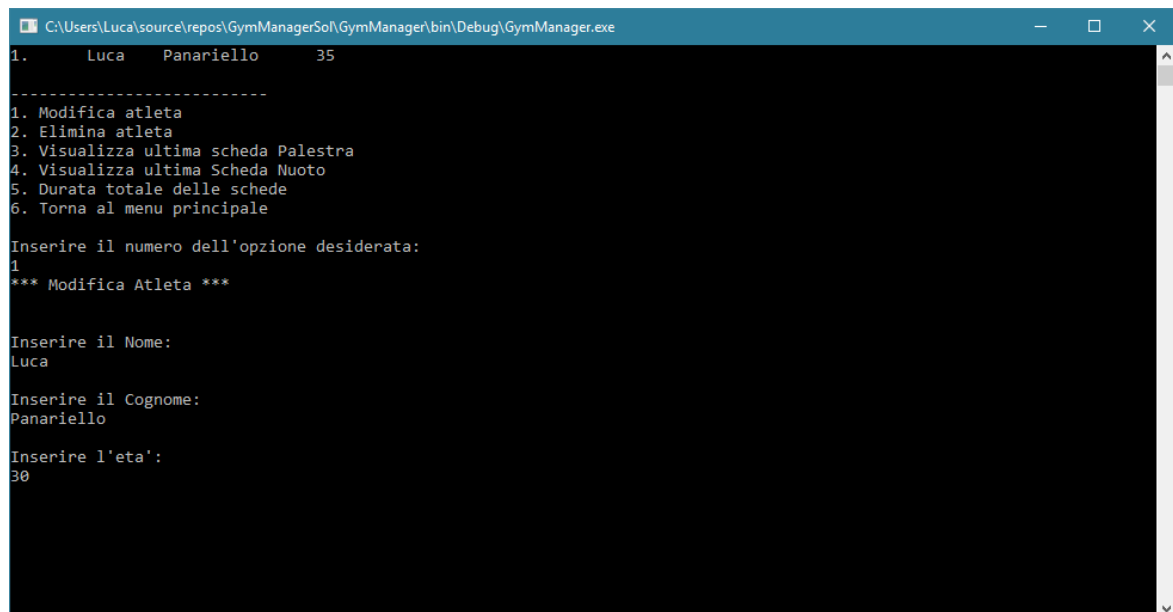
Inserire il numero dell'opzione desiderata:
1
*** Modifica Atleta ***

Inserire il Nome:
Luca

Inserire il Cognome:
Panariello

Inserire l'eta':
fd
Errore di input: System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean
n parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at GymManager.InteraktAtleta.inputDati() in C:\Users\Luca\source\repos\GymManagerSol\GymManager\InteraktAtleta.cs:lin
e 303
Inserire il Nome:
```

4. Ora inseriamo un valore accettabile: intero positivo.  
Scriviamo quindi 30 e diamo invio



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe

1.      Luca      Panariello      35

-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

Inserire il numero dell'opzione desiderata:
1
*** Modifica Atleta ***

Inserire il Nome:
Luca

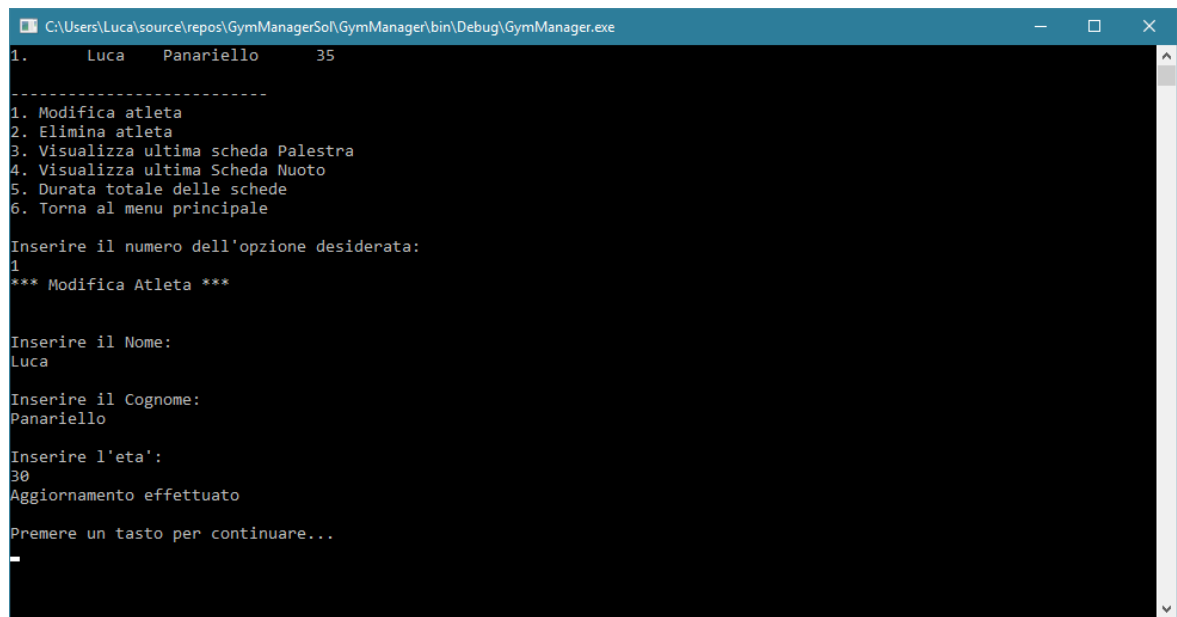
Inserire il Cognome:
Panariello

Inserire l'eta':
30
```

*Continua...*



5. Appare un messaggio che conferma l'avvenuta operazione



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
1.      Luca      Panariello      35
-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

Inserire il numero dell'opzione desiderata:
1
*** Modifica Atleta ***

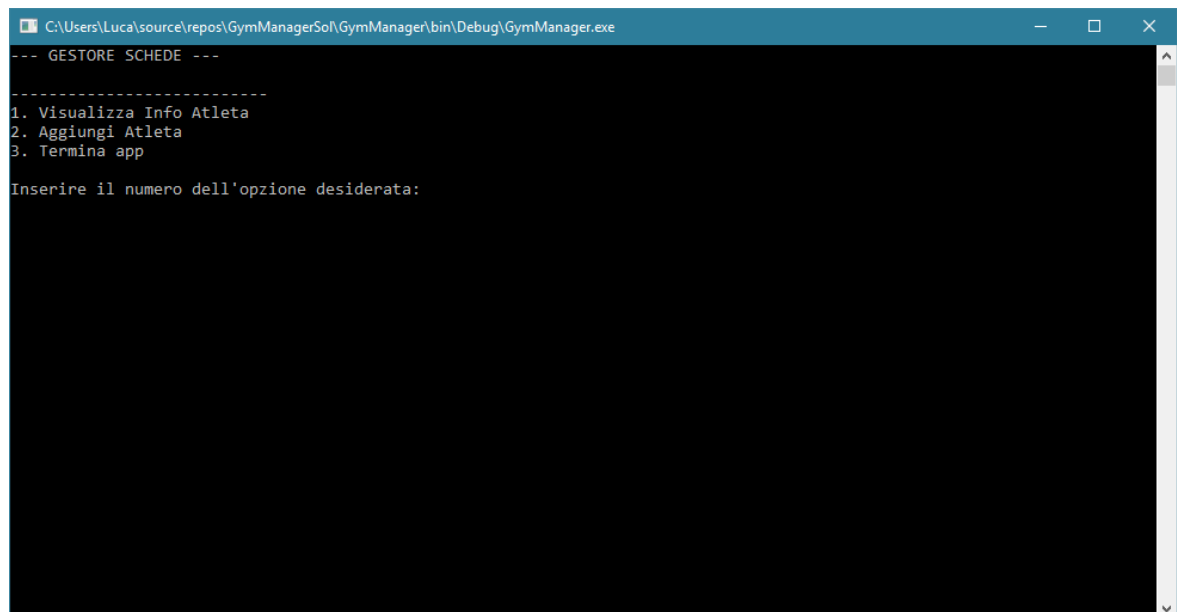
Inserire il Nome:
Luca

Inserire il Cognome:
Panariello

Inserire l'eta':
30
Aggiornamento effettuato

Premere un tasto per continuare...
_
```

6. Premendo ulteriormente Invio torna al menu principale, rispettando così la postcondizione



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
--- GESTORE SCHEDE ---
-----
1. Visualizza Info Atleta
2. Aggiungi Atleta
3. Termina app

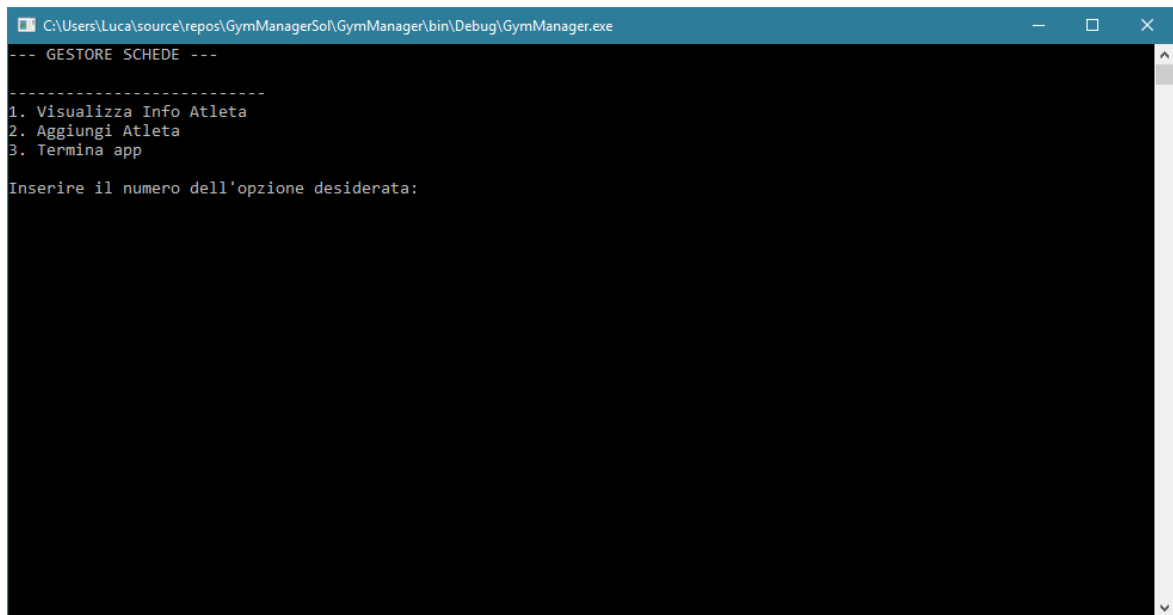
Inserire il numero dell'opzione desiderata:
```

Verifichiamo che la modifica sia avvenuta correttamente e approfittiamo per fare un ulteriore test black box

## Test caso d'uso con ID #1

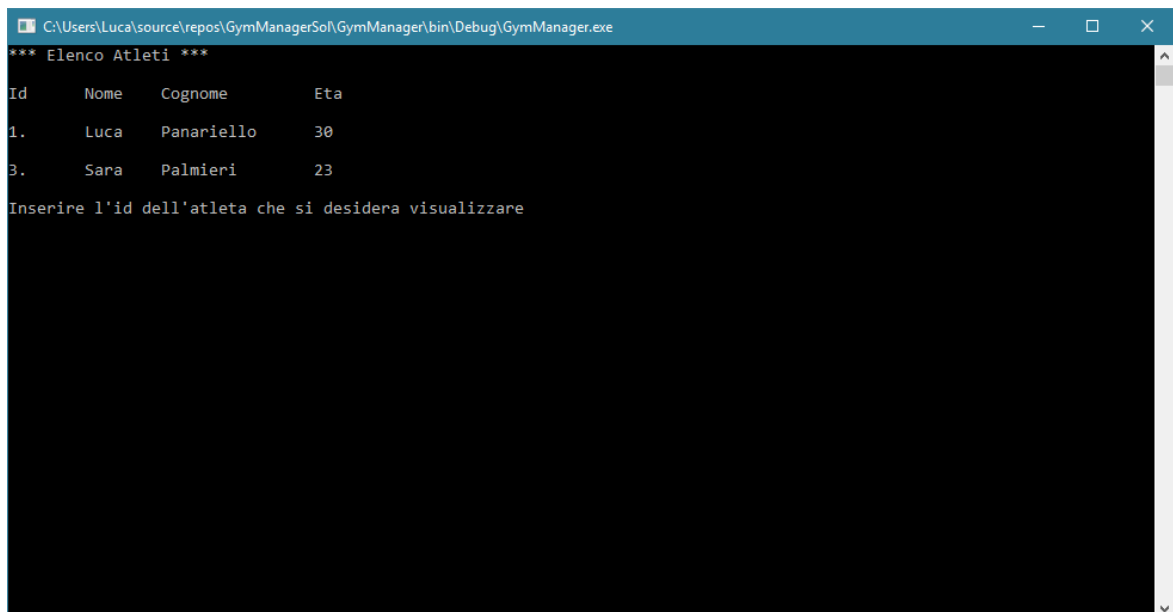
Precondizione: avviare l'app

1. Inseriamo il numero 1 e premiamo Invio



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
--- GESTORE SCHEDE ---
-----
1. Visualizza Info Atleta
2. Aggiungi Atleta
3. Termina app
Inserire il numero dell'opzione desiderata:
```

2. Appare una videata con l'elenco degli atleti e, come possiamo notare, in seguito alla modifica effettuata nel test black box nr. 1, l'età dell'atleta con Id 1 è ora 30.

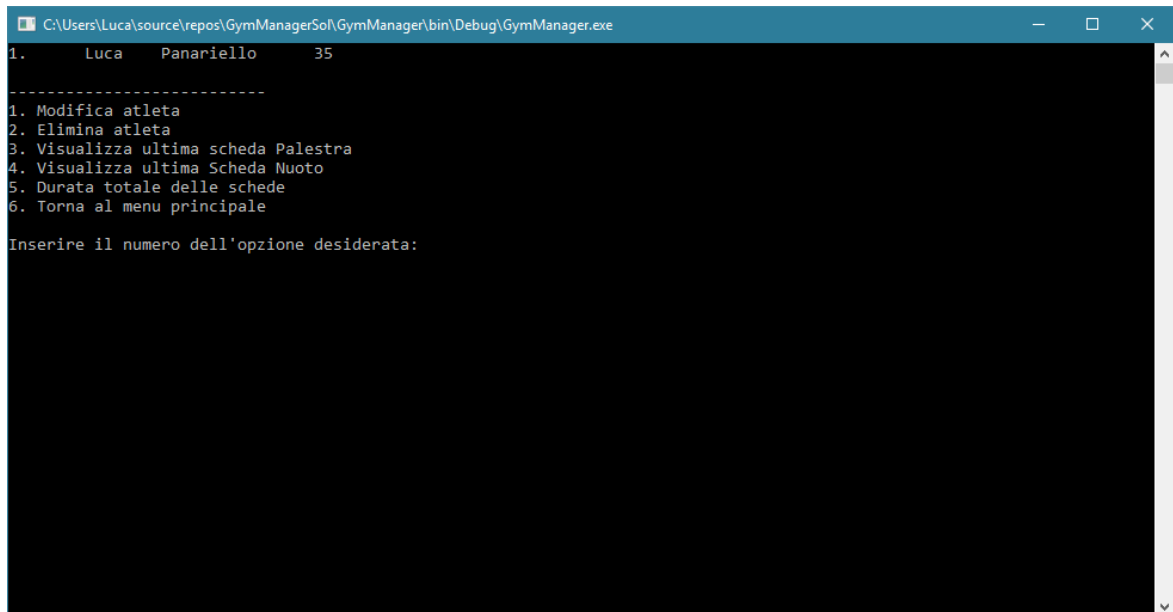


```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
*** Elenco Atleti ***
Id      Nome   Cognome   Eta
1.      Luca   Panariello 30
3.      Sara   Palmieri  23
Inserire l'id dell'atleta che si desidera visualizzare
```

## Test caso d'uso con ID #7

Precondizione: postcondizione del caso d'uso con ID #1

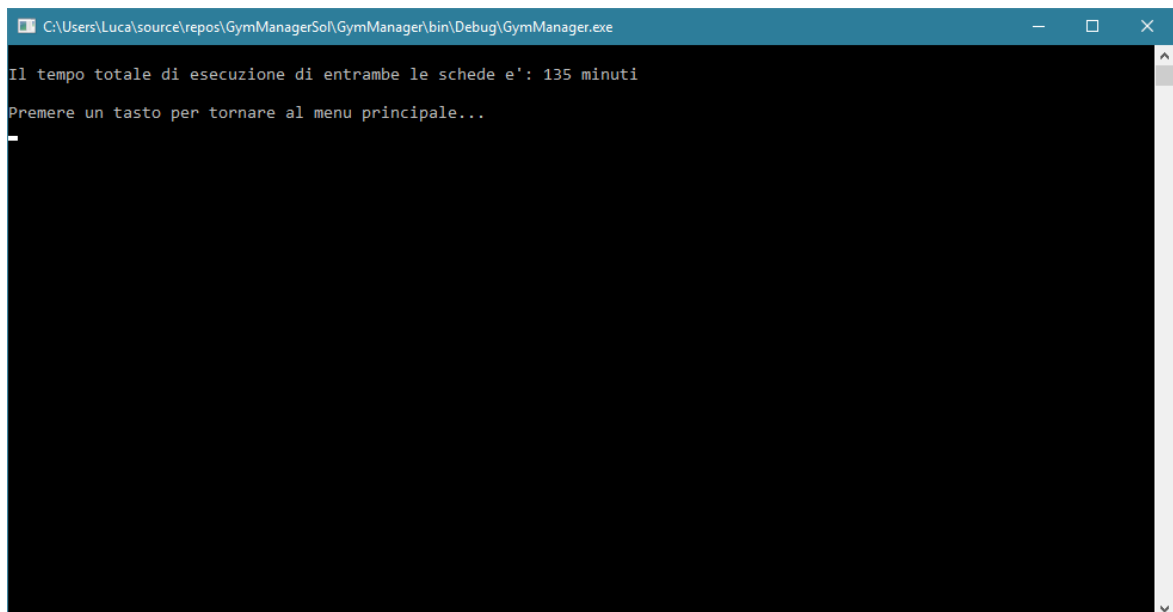
1. Inserisco il numero 5 e premo Invio.



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
1.      Luca      Panariello      35
-----
1. Modifica atleta
2. Elimina atleta
3. Visualizza ultima scheda Palestra
4. Visualizza ultima Scheda Nuoto
5. Durata totale delle schede
6. Torna al menu principale

Inserire il numero dell'opzione desiderata:
```

2. Appare una videata in cui è mostrata la somma delle durate delle 2 schede



```
C:\Users\Luca\source\repos\GymManagerSol\GymManager\bin\Debug\GymManager.exe
Il tempo totale di esecuzione di entrambe le schede e': 135 minuti
Premere un tasto per tornare al menu principale...
_
```

Commenti: possiamo affermare che i test black-box sono stati superati

## VI – Compilazione ed esecuzione

Ambiente di sviluppo utilizzato: Visual Studio 2017 Community

Framework: .NET Framework 4.6.1

Compilazione:

1. Estrarre l'archivio GymManager.zip
2. Avviare il file GymManagerSol.sln
3. Cliccare sul menu Compila->Compila Soluzione

Esecuzione:

1. Andare nella directory "GymManager/bin/Debug"
2. Avviare l'eseguibile appena creato "GymManager.exe"
3. Appare la videata principale dell'app a riga di comando che attende l'input

L'applicazione è eseguibile su sistemi dotati di .NET Framework 4.6+. È stato testato sulle seguenti macchine con .NET Framework 4.6.1 e non ha riportato errori, cali delle performance o crash.

Sistema Operativo	CPU	RAM	Errori / Problemi
Windows 10 32/64-bit	2 Core	8 GB	–
Ubuntu 18.04 64-bit	2 Core	8 GB	–