UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

Animati

Titolo del documento:

Architettura

Gruppo:

T51

Indice

Scopo del documento

[...]

Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto Animati. Vengono riportate di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti.

Classi enumerative di supporto

Unità

> *La classe Unità è una classe di supporto utilizzata nella classe Info, che sta ad indicare con i suoi attributi la durata di un'attività.*

Ruolo

> *La classe Ruolo è una classe di supporto utilizzata nella classe Utente, che sta ad indicare con i suoi attributi il ruolo assunto da uno specifico utente.*

Formato

> *La classe Formato è una classe di supporto utilizzata ogniqualvolta si deve indicare il formato di un file da esportare. In questo caso nella classe ListaAttività, quando viene esportata una lista col metodo esporta(formato : Formato).*

TipoDado

> *La classe TipoDado è una classe di supporto utilizzata nella classe Dado, che sta ad indicare con i suoi attributi il tipo di faccia utilizzata dallo strumento dado.*

MetodoDivisione

> *La classe MetodoDivisione è una classe di supporto utilizzata nella classe CreazioneSquadre, che sta ad indicare con i suoi attributi la metodologia di divisione scelta dall'utente per l'estrazione delle squadre.*

Codice in Object Constraint Language

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

Cronometro

stato : Enum

Invarianti:

- stato assume i valori "reset", "run", "pause"

```
context Cronometro inv :
(stato = "reset") OR (stato = "run") OR (stato = "pause")
```

Metodo	Precondizioni	Postcondizioni
start()	**stato** deve essere in "reset" o in "pause"	**stato** assume valore "run"
pause()	**stato** deve essere in "run"	**stato** assume valore "pause"
stop()	**stato** deve essere in "pause"	• **stato** assume valore "reset" • **parziali** è una lista vuota
parziale()	**stato** deve essere in "run"	• **stato** rimane al valore "run" • aggiunto tempo a lista **parziali**

```
context Cronometro::start()
pre: (self.stato = "reset") OR (self.stato = "pause")
post: self.stato = "run"
```

```
context Cronometro::pause()
pre: self.stato = "run"
post: self.stato = "pause"
```

```
context Cronometro::stop()
pre: self.stato = "pause"
post: (self.stato = "reset") AND (self.parziali -> isEmpty())
```

```
context Cronometro::parziale()
pre: self.stato = "run"
post: (self.stato = "run") AND (self.parziali -> size() = self.parziali@pre -> size()+1) AND
(self.parziali -> includes (self.tempo))
```

Segna-Punti

contatori : Map<String,int>

Invarianti:

- contatori contiente alpiù 99 elementi

```
context Segna-Punti inv :
contatori -> size() <= 99
```

Metodo	Precondizioni	Postcondizioni
aggiungiContatore(nome : String)	• il nome non può eccedere i 99 caratteri • il numero di squadre può essere al massimo 98	il valore del contatore col nome scelto è pari a 0
incrementa(nome : String)	il contatore può assumere valore minore di 500	il contatore col nome scelto viene incrementato di 1
decrementa(nome : String)	il contatore può assumere valore maggiore di -500	il contatore col nome scleto viene decrementato di 1

```
context Segna-Punti::aggiungiContatore(nome : String)
pre: (nome -> size() <= 99) AND (self.contatori -> size() <= 98)
post: self.contatori[nome] = 0
```

```
context Segna-Punti::incrementa(nome : String)
pre: self.contatori[nome] < 500
post: self.contatori[nome] = self.contatori[nome]@pre + 1
```

```
context Segna-Punti::decrementa(nome : String)
pre: self.contatori[nome] > -500
post: self.contatori[nome] = self.contatori[nome]@pre - 1
```

Timer

stato : Enum

Invarianti

- stato assume i valori "reset", "run", "pause"

```
context Timer inv :
(stato = "reset") OR (stato = "run") OR (stato = "pause")
```

Metodo	Precondizioni	Postcondizioni
start()	**stato** deve essere in "reset"	**stato** assume valore "run"
riprendi()	**stato** deve essere in "pause"	**stato** assume valore "run"
stop()	**stato** deve essere in "run"	**stato** assume valore "pause"
annulla()	**stato** deve essere in "pause"	**stato** assume valore "reset"
imposta(tempo : Time)	il tempo fonito deve essere positivo	**tempo** assume valore del tempo fornito
scegliSuono(suono : URL)		la sorgente del suono è quella scelta

```
context Timer::start()
pre: self.stato = "reset"
post: self.stato = "run"
```

```
context Timer::riprendi()
pre: self.stato = "pause"
post: self.stato = "run"
```

```
context Timer::stop()
pre: self.stato = "run"
post: (self.stato = "pause")
```

```
context Timer::annulla()
pre: self.stato = "pause"
post: self.stato = "reset"
```

```
context Timer::imposta(tempo : Time)
pre: tempo > 0
post: self.tempo = tempo
```

```
context Timer::scegliSuono(sorgente : URL)
post: self.suono.sorgente = sorgente
```

Dado

tipo : Enum

Invarianti:

- tipo assume i valori "Numeri", "Colori", "Immagini"

Metodo	Precondizioni	Postcondizioni

estrai() : Object	• se riestrazione assume il valore "False" l'utente non può estrarre un numero maggiore degli elementi selezionati • deve venire selezionato il tipo di oggetto estratto	vengono estratti gli elementi del tipo selezionato
cambiaModalità(riestrazione : bool)		l'attributo riestrazione viene impostato al valore selezionato
scegliTipo(tipo : Enum)		l'attributo tipo viene impostato al valore selezionato
aggiungi(oggetto : Object)		viene aggiunto un oggetto alla lista di quelli selezionabili

---

// TODO: @teopan21

Suono

Metodo	Precondizioni	Postcondizioni
start()	premuto deve assumere il valore "True"	il suono è in riproduzione
stop()	premuto deve assumere il valore "False"	il suono non è in riproduzione
scegliSuono()		lla sorgente del suono è quella scelta

```
context Fischietto::start()
pre: NOT self.premuto
post: self.suono.inRiproduzione = true
```

```
context Fischietto::stop()
pre: self.premuto
post: self.suono.inRiproduzione = false
```

```
context Fischietto::scegliSuono(sorgente : URL)
post: self.suono.sorgente = sorgente
```

Creazione Squadre

metodo : Enum

Invarianti:

- metodo assume i valori "Round robin", "Random", "Fill first" e "Balanced"

```
context Creazione Squadre inv :
(stato = "Round robin") OR (stato = "Random") OR (stato = "Fill first") OR (stato = "Balanced")
numeroSquadre <= 99
numeroComponeti <= 99
numeroPartecipanti <= 9801
```

Metodo	Precondizioni	Postcondizioni
inserisciNumSquadre(numero : int)	numeroSquadre deve essere minore di 99	informazioni viene incrementato di 1
inserisciNumComponenti(numero : int)	numeroComponenti deve essere minore di 99	informazioni viene incrementato di 1
inserisciNumPartecipanti(numero : int)	numeroPartecipanti deve essere minore di 9801	informazioni viene incrementato di 1
scegliMetodo(metodo : Enum)		l'attributo metodo viene impostato con quello scelto
inserisciNome(nome : String)	la lunghezza dei nomi non deve eccedere i 99 caratteri	l'attributo nomi viene impostato con i nomi scelti
estrai()	• informazioni deve essere uguale a 2 o 3 • nel caso i tre valori: numeroSquadre, numeroPartecipanti e numeroComponenti non siano compatibili, verranno solamente considerati numeroPartecipanti e numeroSquadre.	• se metodo assume il valore "Random" l'ordine delle squadre assegnate sarà completamente casuale • se metodo assume il valore "Round robin" l'assegnamento delle squadre sarà sequenziale • se il metodo assume il valore "Fill first" l'assegnamento avverrà per completamento delle squadre, ovvero riempiendo i posti di ogni squadra prima di procedere con l'assegnamento per la prossima • se il metodo assume il valore "Balanced" tutte le squadre dovranno avere lo stesso numero di partecipanti prima di procedere con gli assegnamenti

		• qualsiasi sia il metodo scelto, ogni volta che un partecipante viene assegnato ad una squadra il contatore di quella squadra viene incrementato di 1

Utente

ruolo : Enum

Invarianti:

- ruolo assume i valori "Amministratore" e "Base"

```
context Utente inv :
(ruolo = "Amministratore") OR (ruolo = "Base")
```

Metodo	Precondizioni	Postcondizioni
cambiaRuolo(utente, promotore, nuovoRuolo)	• un amministratore può essere declassato a utente comune unicamente dall'amministratore che lo ha promosso • se l'attributo ruolo dell'utente ha il valore "Amministratore", il suo ruolo può essere cambiato solo se promotore è diverso dall'attributo promossoDa dell'utente • il promotore deve avere ruolo "Amministratore"	l'attributo ruolo dell'utente assume il valore di nuovoRuolo

---

Autenticazione

Metodo	Precondizioni	Postcondizioni
verificaOAuth()		
generaToken()		
ottieniUtente() : Utente		
logout()		

// TODO: @teopan21

Catalogo

Metodo	Precondizioni	Postcondizioni

aggiornaCatalogo() : Attività[0...N]		l'attributo ultimoAggiornamento assume il valore della data corrente
filtra(cerca : String, etichette : Etichette[0...N]) : Attività[0...N]	• nella barra di ricerca del titolo non si possono inserire più di 20 caratteri • i due valori della durata media sono compresi tra 0 e 999, sono interi e il primo è minore del secondo • il numero di partecipanti non può superare 99	il catalogo viene filtrato secondo le etichette previste
creaAttività(attività : Attività)	• la descrizione non può superare i 2000 caratteri • i due valori della durata media sono compresi tra 0 e 999, sono interi e il primo è minore del secondo • il numero di partecipanti non può superare 99 • il titolo non può superare i 20 caratteri di lunghezza	viene aggiunta una nuova attività al catalogo

---

```
context Catalogo::aggiornaCatalogo()
post: self.ultimoAggiornamento = Data.now()
```

// TODO: @teopan21

Lista di Attività

Metodo	Precondizioni	Postcondizioni
creaLista() : listaAttivita	• il nome della lista di attività non può superare i 20 caratteri di lunghezza • un utente non può creare più di 99 liste di attività • il nome della lista di attività non può essere uguale al nome di un'altra lista già presente • il nome della lista di attività non può essere nessuno	viene creata una nuova lista di attività
aggiungiAttività(attività : Attività)	il numero di attività in una lista non può superare 9999	l'attività scelta viene aggiunta alla lista
esporta(formato : String) :		la lista viene esportata in

File		formato pdf o json
eliminaAttività(indice : int)		l'attività con l'indice scelto viene rimossa dalla lista

// TODO: @teopan21

Attività

// TODO: @teopan21 Invarianti

Metodo	Precondizioni	Postcondizioni
modifica(attivitàModificata : Attività)	• la descrizione non può superare i 2000 caratteri • i due valori della durata media sono compresi tra 0 e 999, sono interi e il primo è minore del secondo • il numero di partecipanti non può superare 99 • il titolo non può superare i 20 caratteri di lunghezza	• tutti gli attributi assumono il valore dell'attività modificata • l'attributo ultimaModifica assume il valore della data corrente

Segnalazione

// TODO: @teopan21 invarianti

Metodo	Precondizioni	Postcondizioni
inviaSegnala(autore : Utente, attività : Attività, voto : String)	l'attributo messaggio non può superare i 500 caratteri di lunghezza	viene aggiunta una segnalazione per l'attività scelta

Valutazione

// TODO: @teopan21 invarianti

Metodo	Precondizioni	Postcondizioni
inviaValutazione(autore : Utente, attività : Attività, voto : String)	il voto inserito deve essere un numero decimale compreso tra 0 e 5, con scarto di 0.5	• viene aggiunta la valutazione all'attività scelta • cambia la media di voti dell'attività scelta

Feedback

Metodo	Precondizioni	Postcondizioni

GestioneDatiOffline

Metodo	Precondizioni	Postcondizioni

MongoDB

Metodo	Precondizioni	Postcondizioni

Info

Metodo	Precondizioni	Postcondizioni

Filtro

Metodo	Precondizioni	Postcondizioni

TipoDado

Metodo	Precondizioni	Postcondizioni

Colore

Metodo	Precondizioni	Postcondizioni

URL

Metodo	Precondizioni	Postcondizioni

Data

Metodo	Precondizioni	Postcondizioni

Time

Metodo	Precondizioni	Postcondizioni
