Progetto C++

* 1. Introduzione

Il progetto C++ è un calendario annuale con un’agenda integrata per tenere traccia degli eventi inseriti.

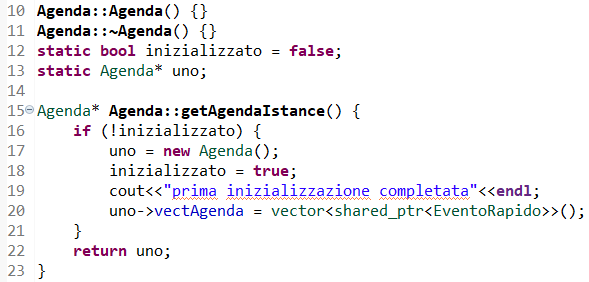
Si compone di 3 componenti indipendenti:

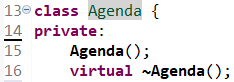
* Agenda, con gli eventi
* Calendario
* Visualizzazione, strumenti che fanno da UI per i due componenti sopra citati

Il calendario è implementato in maniera incrementale partendo dai giorni, che compongono i mesi e successivamente gli anni; ognuno di questi oggetti è implementato come classe. Non essendoci la stretta necessità di assegnare ogni giorno ad un mese, per ridurre l’utilizzo della memoria viene tenuta traccia solo del primo giorno del mese che, aggiunto ad un array dei numeri dei giorni di ogni mese, permette al programma di ricostruire il mese per intero alla necessità.

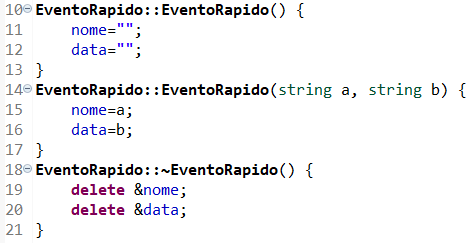
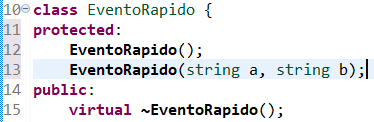
L’Agenda è implementata come singleton e tiene traccia degli eventi inseriti in un array dinamico. Gli eventi assegnabili all’agenda sono inclusi in questo componente, e sono implementati in maniera gerarchica

* 1. Costruttore e distruttore

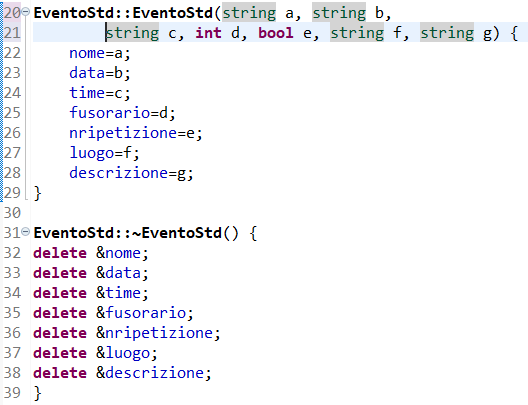
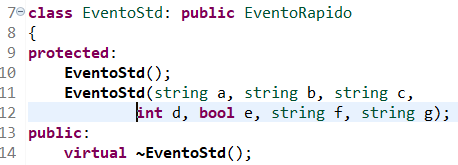




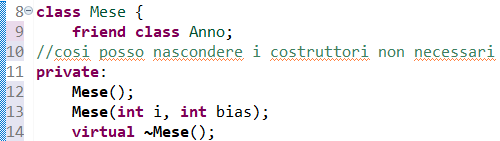
Header e implementazione del costruttore/distruzione di Agenda con la sua funzione singleton

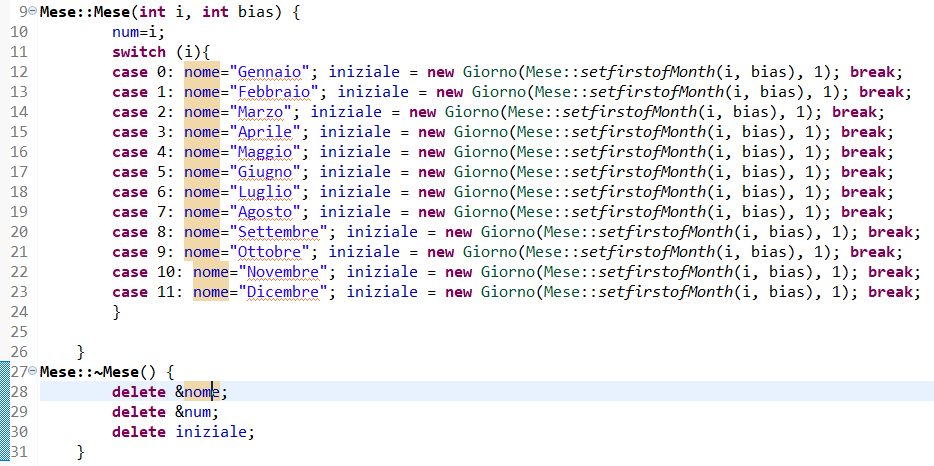


Header e implementazione del costruttore/distruttore di EventoRapido



Header e implementazione del costruttore/distruttore di EventoStd





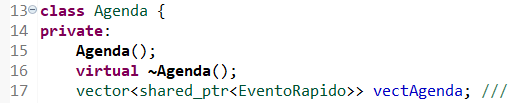
Header e implementazione del costruttore/distruttore di Mese

Gli header e le implementazioni di tutte le altre classi sono molto simili e non presentano aggiunte particolari rispetto a queste già viste.

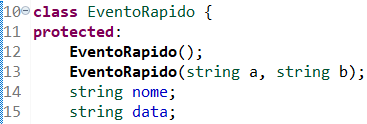
1.3 campi pubblici e privati

Esempio d’uso di:

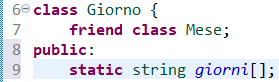
* campi private



* campi protected

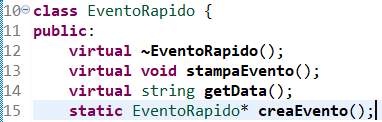
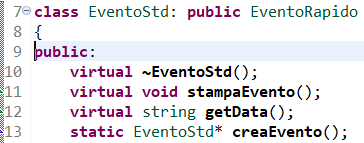


* campi public

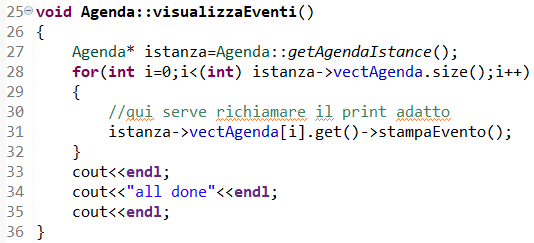


1.4 Membri virtual e non

Header degli overriding



Uso della funzione:

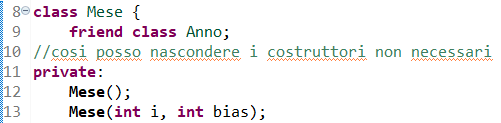


In particolare notiamo il membro stampaEvento(); reso virtual in modo da richiamare la funzione corretta per il tipo di oggetto e non per il contenitore. Non ci limitiamo a rendere virtual solo l’oggetto nella gerarchia più alta, o non saremmo in grado di utilizzare la funzione per gli oggetti figli.

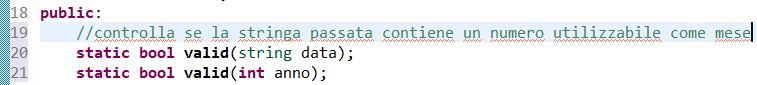
Invece lasciamo la funzione creaEvento() non virtuale operando quindi una ridefinizione, per via della diversa interazione che l’operazione ha con l’utente. È possibile ovviamente implementare la funzionalità partendo dalla classe base e facendo man mano override della funzione, ma viene preferito mostrare la ridefinizione di una funzione rendendo il codice un po’ più ripetitivo.

1.4.1 Classi Friend

Nel componente calendario, dichiariamo la friend class l’istanza superiore (Anno per la classe Mese, Mese per la classe Giorni), in modo da poter nascondere i costruttori con private al di fuori del componente.



1.5 Overloading

Nella classe mese abbiamo un esempio di overload

1.6 ereditarietà multipla

L’ereditarietà è così presentata:





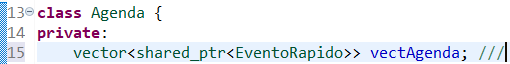


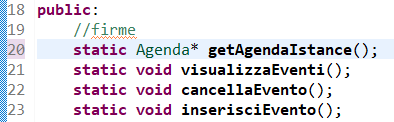


Anniversario e EventoStd ereditano in modalità virtual public per permettere Compleanno di ereditare correttamente tutti i membri delle classi padri.

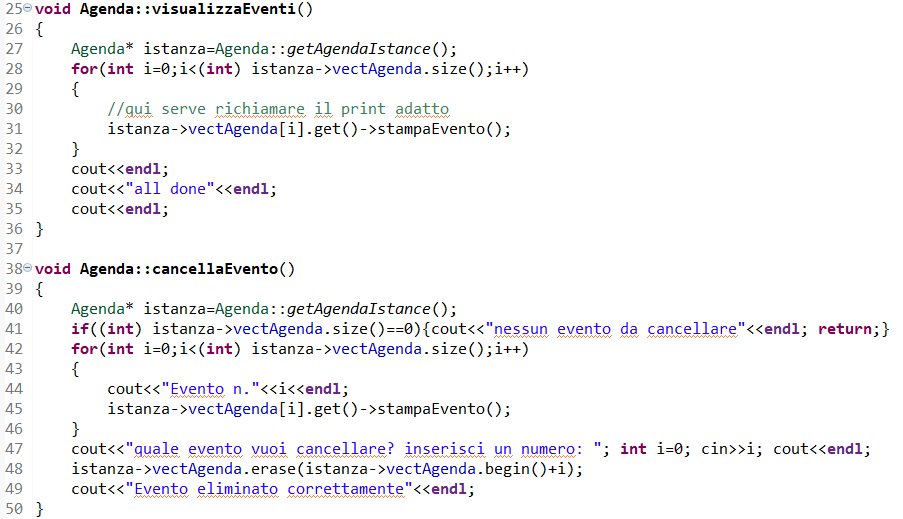
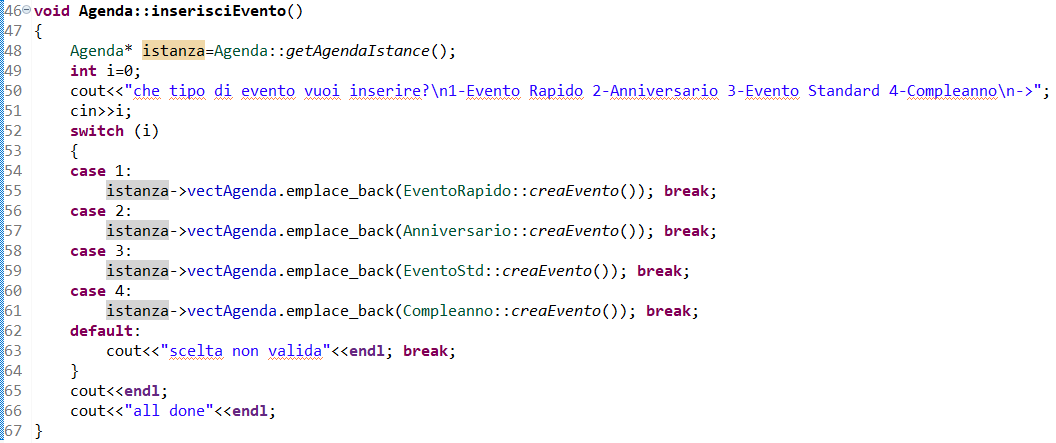
1.7 STL (con qualche struttura dati e algoritmo)

Utilizziamo un vector (array dinamico) con un puntatore intelligente shared\_pointer per tenere traccia degli eventi aggiunti





Di seguito l’implementazione



Altri dettagli:

Struttura di un anno

* Ci sono 12 mesi
  + Ognuno di essi ha una quantità diversa di giorni
  + Bisogna avere un calcolo per l’anno bisestile (febbraio a 29 gg)
  + Il nome dei giorni

Anno è un array di mesi (in particolare un array da 12)

Mesi è un insieme di giorni, non è necessario creare un array con tutti i giorni

In base al mese costruito assegno nome, num giorni e nome giorno di partenza

* + Ci saranno dei metodi che si occuperanno di scorrere i giorni e calcolare il nome del giorno

Il programma deve sapere il nome del giorno di partenza (ovvero ven 1/1/2021)

E come calcolare un anno bisestile

* Dovrei elaborare un modo per la visualizzazione del calendario, e degli eventi sul calendario
  + Visualizzazione per eventi, mensile, settimanale, giorno
* Tipi di eventi
  + Evento rapido
    - Nome
    - data
  + Evento standard
    - Nome evento
    - Quando (data/ora)
    - Tipo di promemoria
    - Fuso orario
    - ripetizione
    - Luogo
    - descrizione
  + Compleanno
    - Tipo di promemoria
  + Anniversario
    - Tipo di promemoria
  + Promemoria
    - Tipo di promemoria = anniversario
    - ripetizione

Progetto Scala

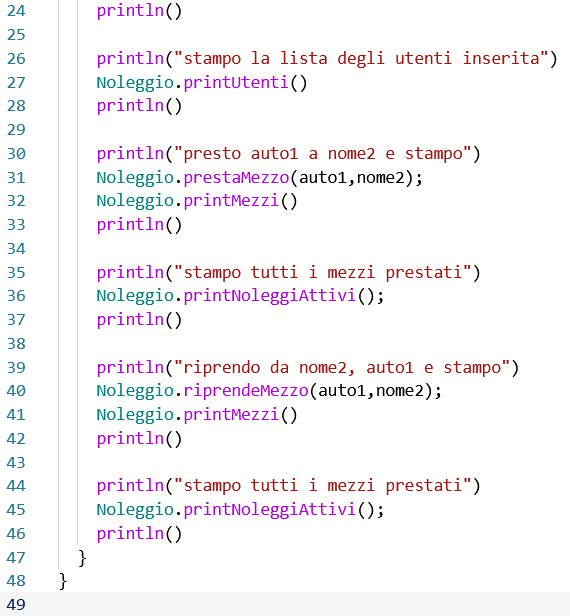
2.1 Introduzione

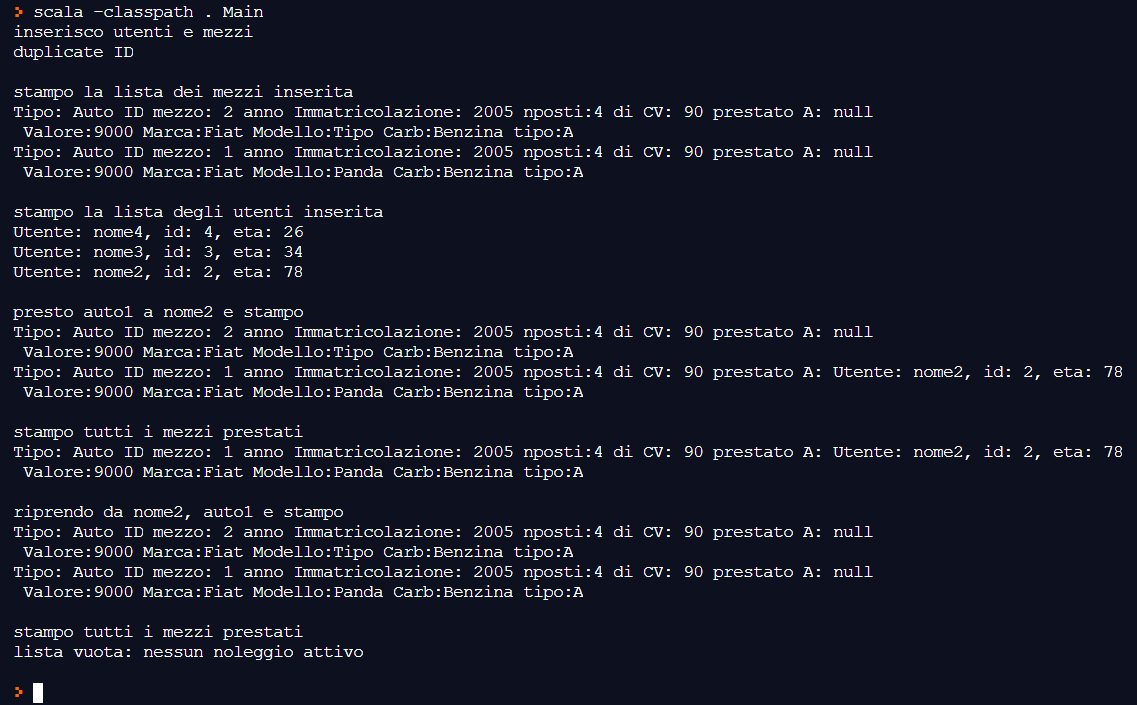
Viene presentato un sistema di un punto noleggio di autoveicoli per tenere traccia dei veicoli a disposizione e dei noleggi in corso.

Tutto il codice prodotto è disponibile pubblicamente su <https://replit.com/@atusghen/progettoINFO3>

Presentiamo prima di tutto il main con l’output di esecuzione della console

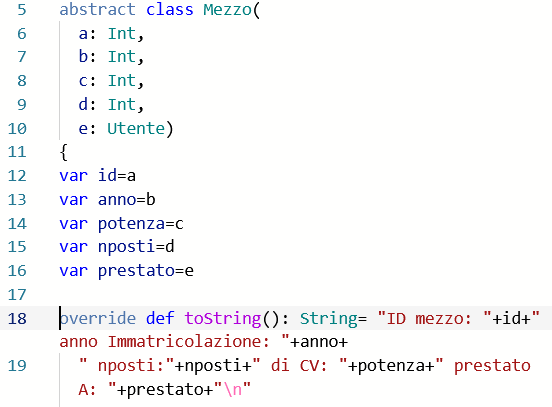






2.2 Costrutti Object Oriented

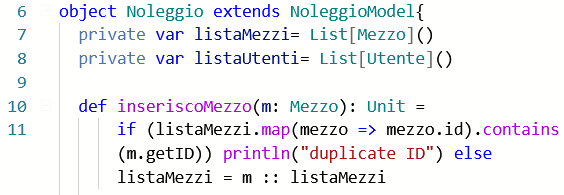
* I parametri del costruttore diventano membri pubblici, inoltre il costruttore primario è il corpo della classe

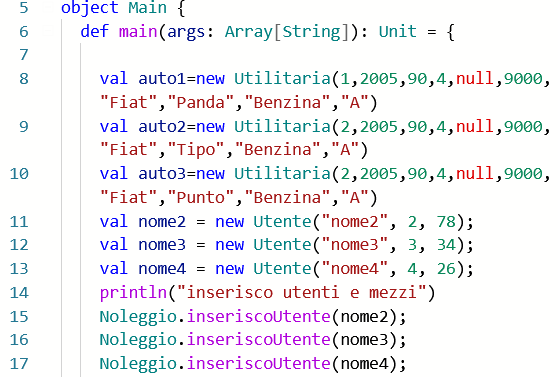


con var e val possiamo creare successivamente le variabili

* Un object singleton

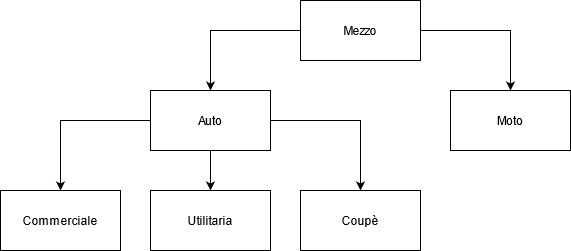
L’object singleton, e successivamente nel main notiamo l’uso del singleton nelle ultime righe



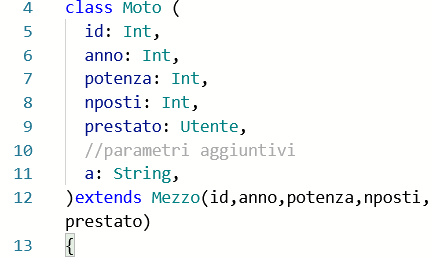
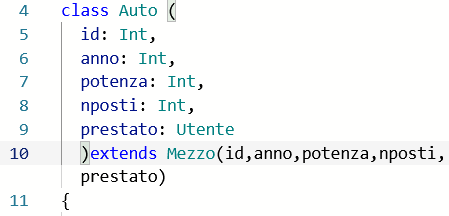


* 1. Estensione delle classi

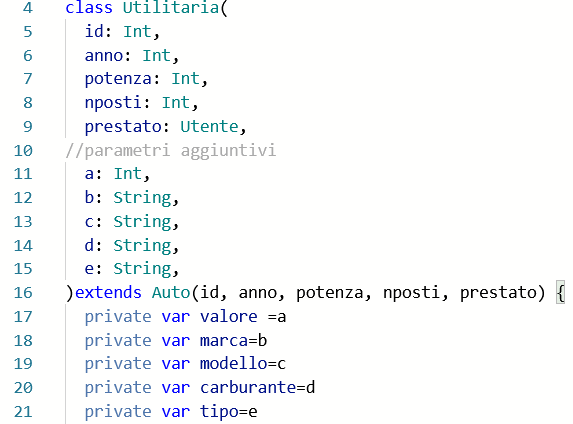
Di seguito la gerarchia delle classi seguita nel progetto



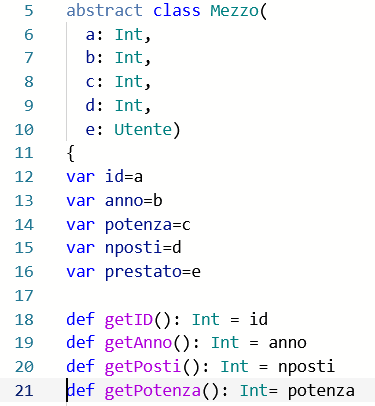
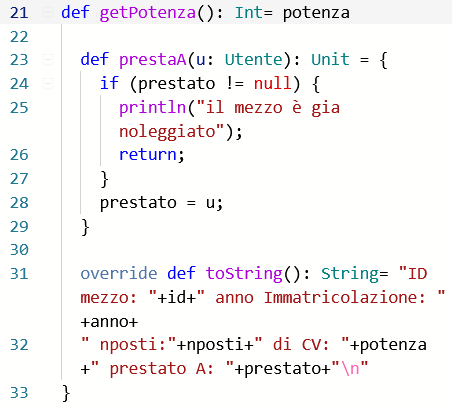
Esempi di implementazione:



qui possiamo notare anche l’uso di campi non public



* 1. Classi astratte

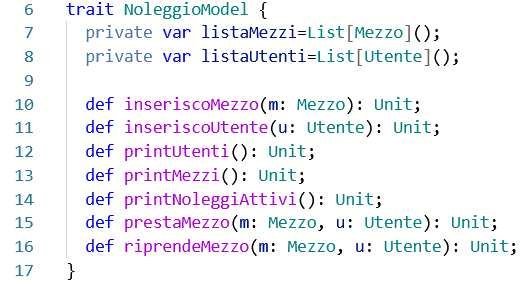


* Override

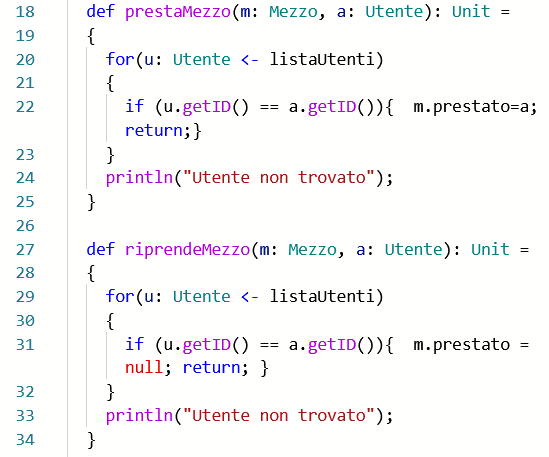
L’esempio precedente mostra già un esempio di override dalla superclasse Object, aggiungiamo per completezza un override della classe Mezzo già mostrata



* 1. Traits



* 1. For, foreach
* Uso di for-> nell’object singleton Noleggio in ../noleggiatore/Noleggio.scala



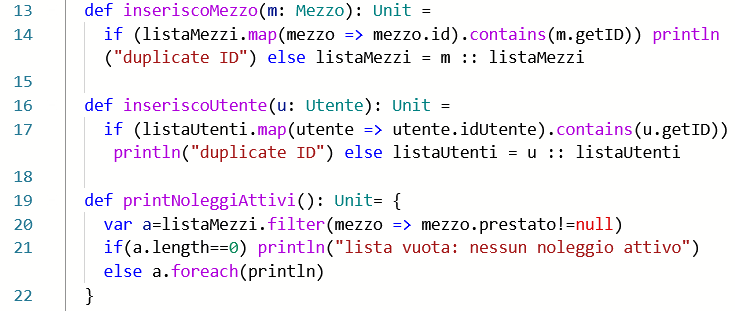
* Uso di foreach

Sempre nello stesso object



* 1. Filter, map

Siamo sempre nell’object Noleggiatore



Progetto ASM

* 1. Introduzione

Il gestore galleria si occupa di governare il semaforo posto all'ingresso della galleria sulla base del traffico rilevato all'interno di essa.

Ogni sensore può rilevare e inviare 3 differenti segnali: NOTRAFFIC, MODERATE (traffico < 20km/h), STANDING. Inoltre questi sensori sono posti ogni 100mt di galleria, ma in base alla capillarità del controllo si possono posizionare più vicini tra di loro ed in base alla capillarità del controllo richiesto questo intervallo potrà essere diminuito o aumentato

La logica di controllo è così implementata:

* condizione di verde: per garantire una circolazione ottimale della galleria, il semaforo rimarrà verde finchè ci sono non più della metà dei sensori che rilevano traffico al massimo moderato
* condizione di rosso: se più della metà dei sensori rilevano veicoli fermi oppure più della metà traffico moderato e il resto veicoli fermi
* condizione di giallo lampeggiante: in tutt gli altri casi. es: in caso di segnali non ottimali come più della metà dei segnali con traffico fermo ma tutto il resto con traffico moderato o anche nullo (può indicare un semplice rallentamento)
* override: consente di chiudere la galleria e comandare a piacere il semaforo a partire dal prossimo stato

Questo è un esempio per una piccola galleria da 300mt con soli 3 sensori: con più sensori la struttura è molto simile ma per un gran numero di sensori non consente il controllo del rosso ottimale: è necessario fare il count dei sensori

**signature**:

**enum** **domain** Colori={VERDE|GIALLO|GIALLOLAMP|ROSSO}

**enum** **domain** Segnali = {NOTRAFFIC|MODERATE|STANDING}

**dynamic** **controlled** semaforo: Colori

**dynamic** **monitored** semaforo\_Override : Colori

**monitored** sensore1: Segnali

**monitored** sensore2: Segnali

**monitored** sensore3: Segnali

**controlled** appling: **Boolean**

**monitored** override: **Boolean**

/\*

controlled nSegnali: Integer

controlled countNotraffic: Integer

controlled countModerate: Integer

controlled countStanding: Integer

\*

\*/

**definitions**:

**macro** **rule** r\_setBlinkYellow =

semaforo := GIALLOLAMP

**macro** **rule** r\_setRed =

**if**(semaforo = GIALLO **or** semaforo = GIALLOLAMP) **then**

**seq**

semaforo := ROSSO

appling := **false**

**endseq**

**else** **if**(semaforo = VERDE) **then**

**seq**

semaforo := GIALLO

appling := **true**

**endseq**

**endif**

**endif**

**macro** **rule** r\_setGreen =

semaforo := VERDE

**macro** **rule** r\_Override =

semaforo := semaforo\_Override

**macro** **rule** r\_ProcessingSignals =

**if**(appling) **then**

r\_setRed[]

**else**

**if**( (sensore1=NOTRAFFIC **and** sensore2=NOTRAFFIC **and** sensore3=NOTRAFFIC)

**or** (sensore1=NOTRAFFIC **and** sensore2=MODERATE **and** sensore3=NOTRAFFIC )

**or** (sensore2=NOTRAFFIC **and** sensore3=MODERATE **and** sensore1=NOTRAFFIC )

**or** (sensore3=NOTRAFFIC **and** sensore1=MODERATE **and** sensore2=NOTRAFFIC )

) **then**

r\_setGreen[]

**else**

**if**( (sensore1=STANDING **and** sensore2=STANDING **and** sensore3=STANDING)

**or** ( sensore1=STANDING **and** sensore2=STANDING **and** sensore3=MODERATE)

**or** ( sensore1=STANDING **and** sensore2=MODERATE **and** sensore3=STANDING)

**or** ( sensore1=MODERATE **and** sensore2=STANDING **and** sensore3=STANDING)

) **then**

r\_setRed[]

**else**

**if**( (sensore1=MODERATE **and** sensore2=MODERATE) **or** (sensore2=MODERATE **and** sensore3=MODERATE) **or** (sensore1=MODERATE **and** sensore3=MODERATE) ) **then**

r\_setBlinkYellow[]

**else** //include i casi dove tutti i segnali sono diversi (caso per 3 sensori)

r\_setBlinkYellow[]

**endif**

**endif**

**endif**

**endif**

**main** **rule** r\_Main =

**if**(override= **false**) **then**

r\_ProcessingSignals[]

**else**

r\_Override[]

**endif**

**default** **init** s0:

**function** override = **false**

**function** appling = **false**

**function** semaforo = VERDE

**function** sensore1= NOTRAFFIC

**function** sensore2= NOTRAFFIC

**function** sensore3= NOTRAFFIC

lo scenario utilizzato verifica il corretto funzionamento in caso di:

* nessun traffico rilevato (tutti i sensori non segnalano traffico)
* traffico moderato rilevato solo su un sensore (quindi tutti gli altri sensori non rilevano traffico) che porta all’accensione della luce verde del semaforo
* accensione della luce gialla lampeggiante nei seguenti casi:
  + traffico moderato rilevato su più della metà dei sensori (e gli altri sensori non rilevano traffico)
  + traffico misto con un gran numero di segnali diversi in arrivo
* accensione della luce rossa del semaforo con
  + traffico fermo (tutti i segnali in arrivo sono traffico fermo)
  + traffico fermo su più della meta dei sensori (e gli altri sensori rilevano traffico moderato)