Dipartimento di Informatica Sistemistica e Comunicazione
Corso di Laurea in Informatica

# Programmazione 2

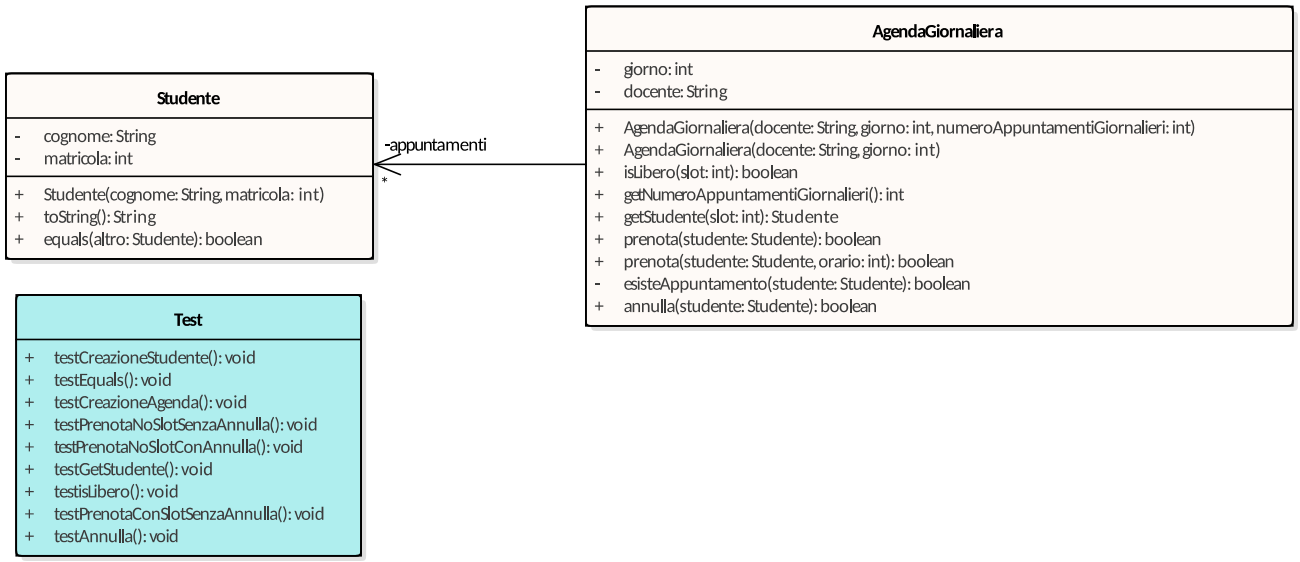
**20 Aprile 2017 – Primo Compitino**

Testo parte di pratica

Un docente universitario organizza gli appuntamenti con i propri studenti all'interno di una agenda giornaliera. Ogni agenda giornaliera dispone di un numero finito di slot di mezz'ora ciascuno da dedicare agli studenti che ne fanno esplicita prenotazione. Data un'agenda giornaliera relativa ad un docente, uno studente può richiedere un appuntamento specificando lo slot di interesse, oppure chiedendo il primo slot disponibile. Una volta effettuata una prenotazione, lo studente può anche decidere di annullarla. In uno stesso giorno, uno studente può richiedere un solo appuntamento.

Implementare le classi come rappresentate dal seguente diagramma UML. **I diagrammi delle classi Studente e AgendaGiornaliera NON includono gli eventuali metodi di incapsulamento che DEVONO essere individuati correttamente e codificati.**

Infine, la classe `Test` (già fornita) contiene un insieme di test che devono essere fatti girare di volta in volta in modo da verificare la corretta realizzazione del software. **Come requisito minimo per ottenere una valutazione positiva, lo studente deve garantire che la sua implementazione non presenti errori di compilazione e superi almeno 3 casi di test fra quelli dati.**




## Classe `Studente`:

- ✓ rappresenta uno studente. È caratterizzato da un cognome (`cognome`) e una matricola (`matricola`)
- ✓ tutti gli attributi sono immutabili, ma accessibili in lettura dall'esterno
- ✓ definisce un costruttore che inizializza gli attributi. Assumete che i valori passati in ingresso siano validi (stringhe diverse da `null` e matricola valida)
- ✓ due studenti sono uguali se hanno la stessa matricola
- ✓ il metodo `toString` restituisce una stringa con l'informazione relativa allo studente (matricola e cognome)

## Classe `AgendaGiornaliera`:

- ✓ rappresenta l'agenda degli appuntamenti con i propri studenti (associazione `appuntamenti`) di un docente (attributo `docente`). È relativa ad un giorno specifico (specificato dall'attributo `giorno` che può assumere un valore da 1 a 365). Il numero di appuntamenti giornalieri possibili è specificato in fase di costruzione. Sia giorno e sia docente sono immutabili, ma accessibili in lettura, appuntamenti non è né modificabile né accessibile in lettura dall'esterno. L'unica informazione che può essere acceduta in lettura è il numero totale

di appuntamenti giornalieri che il docente mette a disposizione (attraverso il metodo `getNumeroAppuntamentiGiornalieri`)

- ✓ definisce un costruttore che inizializza l'agenda con tutte le informazioni necessarie (docente, giorno, numero appuntamenti giornalieri). Effettua l'overloading del costruttore definendone uno ulteriore che pone il numero di appuntamenti giornalieri pari a 6. Si assume che tutti i valori passati in ingresso siano validi (docente diverso da `null`, giorno compreso da 1 e 365 e numero di appuntamenti giornalieri strettamente positivo)
- ✓ il metodo `getStudente` permette di ottenere lo studente che ha prenotato nello slot specificato, `null` nel caso in cui lo slot specificato non sia stato prenotato da nessuno studente. Non preoccupiamoci della privacy leak.
- ✓ il metodo `isLibero(slot)` permette di sapere se slot è stato prenotato (restituisce `false`), oppure è libero (restituisce `true`).
- ✓ il metodo privato `esisteAppuntamento(studente)` verifica se `studente` non ha una prenotazione già effettuata. Se si, restituisce `true`, `false` in caso contrario.
- ✓ il metodo `prenota(studente)` aggiunge `studente` nella prima posizione libera dell'agenda se studente è diverso da `null`, se non aveva già una prenotazione e se l'agenda ha ancora disponibilità. Il metodo ritorna `true` nel caso la prenotazione avvenga con successo, altrimenti ritorna `false`
- ✓ il metodo `prenota(studente, orario)` effettua l'overloading dell'omonimo metodo accettando però in ingresso anche lo slot in cui prenotarsi. Se la posizione specificata è libera, il metodo aggiunge `studente` nella posizione specificata se studente è diverso da `null` e se non aveva già una prenotazione. Il metodo ritorna `true` nel caso la prenotazione avvenga con successo, altrimenti ritorna `false`
- ✓ il metodo `annulla(studente)` permette, se studente è diverso `null`, di cancellare l'appuntamento prenotato dallo studente specificato. Il metodo restituisce `true` se cancella, `false` se studente era `null` oppure non aveva prenotato.