Analisi e Progettazione del Software
Prova scritta del 21 gennaio 2019

Si vuole progettare una classe C++ per la gestione di una sessione di esami universitari.

Le operazioni principali per la sessione, di cui si memorizzano il nome e le date di inizio e fine, sono le seguenti:

$InserisciCorso(c : Corso)$
Il corso c viene inserito nella sessione d'esami.
Precondizioni: Il corso c non è già stato inserito.

$InserisciStudente(s : Studente)$
Lo studente s viene inserito nella sessione d'esami.
Precondizioni: Lo studente s non è già stato inserito.

$CreaEsame(c : Corso, d : data)$
Viene creato un esame per il corso c in data d. *Precondizioni:* Non esiste già un esame per il corso c in data d; la data d è compresa nel periodo nella sessione (estremi inclusi).

$PrenotaEsame(s : Studente, c : Corso, d : data)$
Lo studente s viene inserito tra i prenotati all'esame del corso c in data d. Se lo studente è già prenotato, l'operazione non ha effetto.
Precondizioni: Esistono sia lo studente s che un esame del corso c in data d.

$PosticipaEsame(c : Corso, d : data)$: $data$
L'esame del corso c in data d viene posticipato alla prima data successiva a d in cui non c'è alcun altro esame (di qualsiasi corso) a cui sia prenotato almeno uno studente e neanche alcun esame del corso c (indipendentemente se con prenotazioni o meno). La L'operazione restituisce la nuova data in cui è stato fissato l'esame, che può anche essere oltre la fine della sessione
Precondizioni: Esiste un esame del corso c in data d.

Esercizio 1 (punti 4) Si disegni il diagramma UML delle classi per l'applicazione.

Esercizio 2 (punti 5) Si scrivano le definizioni della classe `Sessione` e delle altre classi che compongono il diagramma UML, assumendo però già disponibili e *immodificabili* le classi `Corso` e `Studente` per la gestione di corsi e studenti, di cui non si conosce la rappresentazione (ad esempio, non si può assumere che sia presente l'operatore `==`).

Si consideri altresì disponibile la classe `Data` e si assuma che metta a disposizione tutti i costruttori, i metodi e gli operatori che si ritengono opportuni.

Esercizio 3 (punti 11) Si scrivano le definizioni dei metodi della classe `Sessione` che corrispondono alle operazioni sopra elencate e i selettori che si ritengono opportuni. Si gestiscano le precondizioni tramite il lancio dell'eccezione `invalid_argument`. Si definiscano i metodi (modificatori e selettori) delle altre classi del diagramma UML (escluse ovviamente `Corso` e `Studente`).

Esercizio 4 (punti 4) Si scriva l'operatore di output della classe `Sessione`, in modo che stampi anche tutti i dati ad essa collegati. Si assumano disponibili gli operatori di output delle classi `Corso`, `Studente` e `Data`.

Esercizio 5 (punti 6) Si scriva una funzione esterna (non *friend*) alla classe `Sessione` che riceva come parametro un oggetto della classe `Sessione` e restituisca il numero di conflitti della sessione. Rappresenta un conflitto la presenza di due esami nella stessa data con almeno uno studente in comune.