Progetto di Programmazione ad Oggetti

Giulia Coro’
matricola 1097666

Biblioteca Scolastica

<u>username e password per l'accesso alle interfacce</u> (case-insensitive):

- **Amministratore:**

  username**: admin**, password: **admin**
- **Utente Esterno:**

  username: **est**, password: **est**
- **Utente Studente:**

  username: **stud**, password: **stud**
- **Utente Contribuente:**

  username: **cont**, password: **cont**

Indice:

1) Scopo del progetto

Il progetto rappresenta il **software di una biblioteca scolastica**: una specie di database nel quale sono immagazzinate tutte le informazioni e lo stato delle varie tipologie di articoli posseduti dalla biblioteca, e le informazioni sugli utenti registrati (quest'ultima visualizzabili solamente dall'utente amministratore). Tutte queste informazioni possono essere gestite dall'utente amministratore.

Il software prevede infatti diverse funzionalità a seconda del tipo di utente che lo usa. I **tipi di utente** previsti sono 4: utente esterno, utente studente, utente contribuente e amministratore. Ogni utente ha un username che lo identifica univocamente, una password per accedere al sistema bibliotecario, nome, cognome e un tipo che lo caratterizza. Ogni diverso tipo di utente, effettuando la login, accede ad una diversa interfaccia.

Le **funzionalità per l'amministratore** includono la gestione degli articoli in biblioteca e degli utenti registrati. L'interfaccia a lui dedicata è strutturata in due *tab*: quello dedicato agli articoli della biblioteca, attraverso il quale, oltre a visualizzare la lista completa degli articoli e le loro informazioni, egli può aggiungere nuovi articoli alla biblioteca, cancellarne, oppure modificarne lo stato di prestito. L'altro è invece dedicato agli utenti registrati, che l'amministratore può creare, eliminare o modificarne il tipo.
Tutte le informazioni su utenti e articoli sono memorizzate in un **file XML**, e possono essere sovrascritte in un qualsiasi momento dall'amministratore, premendo sull'apposito pulsante di salvataggio.

Le **funzionalità rivolte agli altri utenti** (siano essi utenti esterni, studenti o contribuenti) prevedono la ricerca degli articoli nella biblioteca attraverso le loro caratteristiche (titolo, autore, ...). Ogni tipo di utente ha privilegi diversi per la ricerca degli articoli, un tipo di utente meno privilegiato potrà cercare solo determinati tipi di articoli e filtrarli solo tramite determinate caratteristiche, un tipo più privilegiato potrà invece cercare tutte le tipologie di articoli e filtrarli tramite più caratteristiche. I privilegi di ricerca per ogni utente vengono spiegati piu' in dettaglio nella sezione 3.1 della relazione, sotto la descrizione di ogni classe di utente.

Le **tipologie di articoli** presenti nella biblioteca scolastica sono: libri (di narrativa o testi scolastici), cd musicali, riviste. Ognuno di essi ha un ID che lo identifica univocamente e diverse altre caratteristiche a seconda del tipo. Queste vengono spiegate più in dettaglio nella sezione 3.1 nella descrizione di ogni sottoclasse di articolo.

2) Materiale consegnato

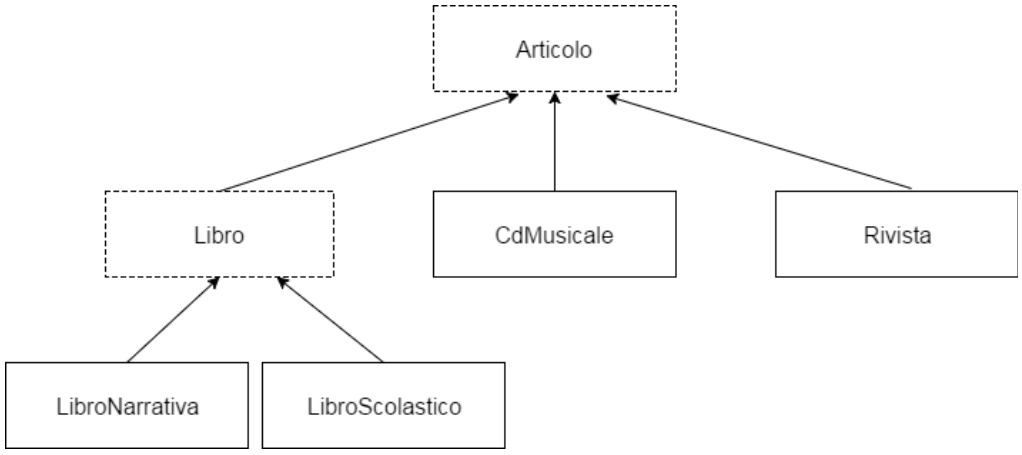
Il materiale consegnato è organizzato nel seguente modo:

- Cartella "bibliotecaScolastica" che contiene:
  - Tutti i file .h e .cpp delle classi necessari per la compilazione del progetto, e inoltre un file "bibliotecaScolastica.pro"
  - Cartella "gui", nella quale son presenti i file .h .cpp per la creazione delle interfacce
  - Cartella "xml" nella quale sono presenti i file "articolidb.xml" e "utentidb.xml" contenenti dei dati d'esempio per il riempimento, rispettivamente, del database degli articoli e degli utenti
- “relazione.pdf”

3) Descrizione delle classi

3.1) Descrizione delle gerarchie di tipi usate

- **Gerarchia degli articoli:**



- **Articolo** e’ la superclasse astratta che sta alla base della gerarchia, dalla quale derivano tutte le altre classi. Un oggetto di tipo Articolo rappresenta un qualunque tipo di articolo in possesso della biblioteca.

  Ogni articolo e’ identificato da un ID univoco, e possiede inoltre un titolo, anno di pubblicazione e un campo dati booleano disponibilita’, che indica se l’articolo e’ al momento disponibile nella biblioteca, oppure e’ fuori per un prestito.

  I metodi di Articolo sono:

  - getId(), getTitolo(), getAnnoPubblicazione(), getDisponibilita(), che ritornano rispettivamente ID, titolo, anno di pubblicazione e un valore booleano che indica la disponibilita’ (1 significa disponibile, 0 in prestito)
  - switchDisp() metodo che permette di modificare la disponibilita’ di un articolo, da disponibile a in prestito, e viceversa.
  - getDynamicType() metodo virtuale che ritorna il tipo dinamico di un oggetto polimorfo di tipo statico Articolo, grazie all’overriding del metodo stesso nelle classi derivate
  - clone() metodo virtuale che ritorna una copia esatta dell’oggetto di invocazione, anche questo metodo viene ovviamente implementato in tutte le classi derivate

- **Libro** e’ una classe astratta derivata da Articolo. Un oggetto di tipo Libro rappresenta un libro di testo generico. Da questa classe deriveranno poi LibroNarrativa e LibroScolastico.

  Oltre ai campi dati ereditati da Articolo, un Libro e’ caratterizzato anche da un autore, una casa editrice, e dal numero di pagine.

  I metodi di Libro sono quelli ereditati da Articolo, e inoltre:

  - getAutore(), getCasaEditrice(), getPagine() che ritornano rispettivamente l’autore, la casa editrice ed il numero di pagine
- **LibroNarrativa** e’ una classe derivata da Libro. Un LibroNarrativa rappresenta appunto un libro di narrativa e, oltre a quelli derivati, ha come campi dati linguaOriginale, che indica la lingua nella quale e’ stato scritto il libro in origine, genereNarrativo, che categorizza il genere del libro (es. romanzo,biografia…) ed etaTarget, che indica l’eta’ per la quale e’ indicata la lettura di quel libro (es. bambini, adulti..).

  Oltre a quelli ereditati dalle superclassi, i metodi propri di LibroNarrativa sono:

  - getLinguaOriginale(), getGenereNarrativo(), etaTarget() che ritornano la lingua, Il genere e l’eta consigliata per il libro
  - Inoltre vengono implementati i metodi ereditati dalle classi astratte: la clone() di un LibroNarrativa ritornera’ cosi’ un oggetto copia di tipo LibroNarrativa e getDynamicType() invocato su un LibroNarrativa ritornera’ una stringa “LibroNarrativa”.
- **LibroScolastico** e’ una classe derivata da Libro. Un LibroScolastico rappresenta un libro di testo ad uso scolastico e per l’insegnamento e, oltre a quelli derivati, ha come campi dati materia, che indica la materia trattata e numEdizione che indica quale edizione di quel libro e’.

  Oltre a quelli ereditati dalle superclassi, i metodi propri di LibroScolastico sono:

  - getMateria(), getEdizione() che ritornano la materia ed il numero di edizione
  - Inoltre vengono implementati i metodi ereditati dalle classi astratte: la clone() di un LibroScolastico ritornera’ cosi’ un oggetto copia di tipo LibroScolastico e getDynamicType() invocato su un oggetto di tipo LibroScolastico ritornera’ una stringa “LibroScolastico”.
- **CdMusicale** e’ una classe derivata da Articolo. Un oggetto CdMusicale rappresenta un CD con delle tracce audio musicali o parlate. Anch’esso in quanto articolo della biblioteca puo’ essere prestato.

  I campi di CdMusicale sono quelli ereditati da Articolo, e inoltre artista, genere, che specifica che genere di audio contiene il CD, e numTracce che ne indica il numero di tracce audio presenti.

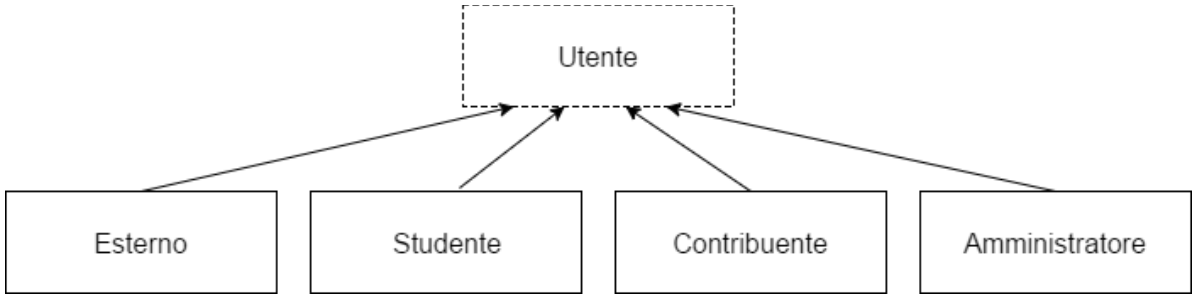
  Oltre a quelli ereditati da Articolo, i metodi di CdMusicale sono:

  - getArtista(), getGenere(), getNumeroTracce() che ritornano rispettivamente artista, genere e numero tracce
  - l'implementazione dei metodi ereditati dalle classi astratte: la clone() di un CdMusicale ritornera’ cosi’ un oggetto copia di tipo CdMusicale e getDynamicType() invocato su un oggetto di tipo CdMusicale ritornera’ una stringa “CdMusicale”.
- **Rivista** e' una classe derivata da Articolo. Un oggetto Rivista rappresenta un singolo numero di una rivista cartacea. In questo caso il campo 'titolo' di articolo indica ovviamente in nome della rivista. Oltre a quelli ereditati, gli altri campi dati di Rivista sono poi numero, che indica quale numero di quella rivista e', periodicita, che indica ogni quanto esce un nuovo numero di quella rivista (es. settimanale, mensile..), argomento, ossia l'argomento principale trattato dalla rivista e numPagine che indica quante pagine ha quella rivista.

  Oltre a quelli ereditati da Articolo, i metodi di Rivista sono:

  - getNumero(), getPeriodicita(), getArgomento(), getNumeroPagine() che ritornano numero, periodicita, argomento e numPagine
  - l'implementazione dei metodi ereditati dalle classi astratte: la clone() di un oggetto Rivista ritornera’ cosi’ un oggetto copia di tipo Rivista e getDynamicType() invocato su un oggetto di tipo Rivista ritornera’ una stringa “Rivista”.

- **Gerarchia degli utenti:**



- **Utente** e' la superclasse astratta che sta alla base della gerarchia degli utenti, dalla quale derivano le classi Esterno, Studente, Contribuente ed Amministratore. Un oggetto di tipo Utente rappresenta un utente generico registrato alla biblioteca, il quale puo', grazie ad un username univoco e ad una password, accedere a determinate funzionalita' del software bibliotecario, le quali variano a seconda del suo sottotipo.

  I campi dati di Utente sono: username (univoco), password, nome, cognome.

  I metodi di Utente sono :

  - getUsername(), getPassword(), getNome(), getCognome() che ritornano username, password, nome e cognome dell'utente.
  - getTipoUtente() metodo virtuale che viene poi implementato nelle classi derivate e permette di determinare a run-time qual e' il tipo dinamico dell'oggetto polimorfo Utente sul quale viene invocato, ritornando una diversa stringa per ogni sottotipo.
- **Esterno** e' una classe derivata da Utente. Un oggetto Esterno rappresenta un "utente esterno" ovvero un utente registrato non frequentante la scuola della quale la biblioteca scolastica fa parte. Un utente esterno avra' quindi meno privilegi rispetto agli altri tipi di utenti, egli potra' infatti ricercare nel database soltanto articoli di tipo LibroNarrativa o Rivista, e soltanto tramite alcune caratteristiche: titolo, autore e fascia d'eta' per quanto riguarda i libri di narrativa, e titolo, numero ed argomento per le riviste; potra' inoltre selezionare se cercare tra tutti gli articoli o soltanto tra quelli correntemente disponibili. Di ogni articolo trovato potra' poi visualizzarne tutte le caratteristiche.

  I campi dati e i metodi di Esterno sono gli stessi che vengono ereditati da Utente, inoltre il metodo getTipoUtente() viene implementato, e ritornera' quindi una stringa "Esterno" quando viene invocato su un oggetto con tipo dinamico Esterno.
- **Studente** e' una classe derivata da Utente. Un oggetto Studente rappresenta un utente registrato che e' uno studente, percio' un utente di questo tipo potra' ricercare libri di testo scolastici, libri di narrativa e cd musicali, tramite le seguenti caratteristiche: libri scolastici per titolo, autore, numero edizione, casa editrice, materia; libri di narrativa per titolo, autore, genere narrativo e lingua originale; cd musicali per titolo, autore e genere. Potra' inoltre selezionare se cercare tra tutti gli articoli o soltanto tra quelli correntemente disponibili. Di ogni articolo trovato potra' poi visualizzarne tutte le caratteristiche.

  I campi dati e i metodi di Studente sono quelli ereditati da Utente, inoltre il metodo getTipoUtente() viene implementato, e ritornera' quindi una stringa "Studente" quando viene invocato su un oggetto con tipo dinamico Studente.
- **Contribuente** e' una classe derivata da Utente. Un oggetto Contribuente rappresenta un utente registrato che ha effettuato donazioni, di articoli o monetarie, alla biblioteca. Questo e' il tipo di utente con maggiori privilegi (escludendo l'amministratore) e potra' percio' ricercare nel database tutti i tipi di articoli, tramite le loro principali caratteristiche (non tutte solo per una questione di sovraffollamento di informazioni nell'interfaccia, e di poca importanza di alcune caratteristiche es. numero pagine). I campi di ricerca quindi sono: per i libri di narrativa titolo, autore, genere narrativo, fascia d'eta', lingua originale; per i libri scolastici titolo, autore, numero edizione, casa editrice e materia; per i CD titolo, autore e genere; per le riviste nome rivista, numero e argomento. Potra' inoltre selezionare se cercare tra tutti gli articoli o soltanto tra quelli correntemente disponibili. Di ogni articolo

trovato potra' poi visualizzare tutte le caratteristiche.

I campi dati e i metodi di Contribuente sono quelli ereditati da Utente, inoltre il metodo getTipoUtente() viene implementato, e ritornera' quindi una stringa "Contribuente" quando viene invocato su un oggetto con tipo dinamico Contribuente.

- **Amministratore** e' una classe derivata da Utente. Un oggetto Amministratore rappresenta l'utente (o gli utenti) amministratore dei database della biblioteca. Le funzioni di un utente amministratore saranno percio' sostanzialmente diverse da quelle degli altri tipi di utente, in quanto la sua funzione principale non sara' ricercare articoli, ma gestire l'aggiunta, la modifica e l'eliminazione di utenti e articoli. L'utente amministratore e' inoltre l'unico che non puo' essere eliminato o modificato. Tutte le modifiche effettuate dall'amministratore potranno essere salvate sul file esterno, attraverso l'apposito pulsante dell'interfaccia. Queste funzioni vengono spiegate piu' in dettaglio nella sezione 5.2 della relazione, nel manuale utente della GUI.

  I campi dati e i metodi di Amministratore sono quelli ereditati da Utente, inoltre il metodo getTipoUtente() viene implementato, e ritornera' quindi una stringa "Amministratore" quando viene invocato su un oggetto con tipo dinamico Amministratore.

3.2) Descrizione delle classi contenitore

- **Biblioteca** e' la classe contenitore degli oggetti Articolo. Questa classe ha come campi dati una lista di Articolo* ed un intero totaleArt che tiene conto del numero di articoli presenti. Tutti gli articoli della biblioteca sono memorizzati in questa lista, ed ogni Articolo* e' un puntatore polimorfo ad un oggetto di una qualsiasi delle classi derivate da Articolo.

  I metodi di questa classe sono:

  - getNumeroArt() che ritorna il numero degli articoli
  - piuUno() che incrementa di uno il campo totaleArt, che tiene conto di quanti articoli ho inserito
  - checkAvaiableId() che esplora tutta la lista di articoli e controlla se un certo ID e' disponibile (cioe' non esiste gia' un articolo con quello stesso ID), e ritorna true se e' disponibile, false altrimenti
  - rimuoviArticolo(idArt) che rimuove, se esiste, l'articolo con ID idArt dalla lista di articoli, e decrementa quindi il numero di articoli di 1
  - cambiaDisponibilita(idArt) che trova nella lista l'articolo con ID idArt e ne modifica la disponibilita' (da disponibile a in prestito, oppure da in prestito a disponibile)
  - findArticolo(idArt) che trova nella lista l'articolo con ID idArt e lo ritorna
  - cercaLibroNarrativa(...), cercaLibroScolastico(...), cercaCdMusicale(...) e cercaRivista(...) che scorrono la lista di articoli e cercano tutti gli articoli che corrispondono alle caratteristiche che gli vengono passate come parametro, creano una copia di questi articoli e li inseriscono in un nuova lista che poi ritornano.
- **Registrazioni** e' la classe contenitore degli Utenti registrati. Questa classe infatti ha come campi dati una lista di Utente* ed un intero numUtenti che tiene conto del numero di utenti registrati.

  I metodi di questa classe sono:

  - getNumUtenti() che ritorna il numero degli utenti registrati
  - inserisciUtente(...) che crea un utente con le caratteristiche che gli vengono passate come parametri, e lo inserisce nella lista degli utenti registrati, poi incrementa il numero degli utenti di uno
  - rimuoviUtente(username) rimuove, se esiste, dalla lista di utenti l'utente con username username
  - checkAvaiableUsername(user) esplora tutta la lista di utenti e controlla se un certo username e' disponibile (cioe' non esiste gia' un utente con quello stesso username), e ritorna true se e' disponibile, false altrimenti
  - cambiaTipoUtente(u, t) cerca nella lista l'utente con username u, e modifica il suo tipo nel tipo t.

4) Uso di codice polimorfo

Il polimorfismo e' un elemento fondamentale che permette al progetto di funzionare. Infatti sono utilizzati ampiamente puntatori ad oggetti astratti (Articolo* o Utente*) che puntano in realta' ad oggetti delle loro sottoclassi. Questi oggetti hanno quindi, ad esempio, tipo statico Articolo e tipo dinamico Rivista. L'invocazione di metodi su questi oggetti puo' quindi richiamare i metodi astratti implementati nelle sottoclassi, secondo l'overriding migliore, trovato a run-time. Per invocare metodi propri soltanto delle sottoclassi, e' necessario invece "reinterpretare" l'oggetto come un effettivo sottoggetto, usando quindi un dynamic_cast.

I metodi che vengono invocati in modo polimorfo sono:

- nella classe Articolo, il metodo clone()
- sempre nella classe Articolo, il metodo getDynamicType()
- nella classe Utente, il metodo getTipoUtente()
- infine i distruttori di entrambe le classi Articolo e Utente, in modo da avere una pulizia profonda dei sottoggetti.

Il funzionamento di questi metodi e' gia' stato spiegato piu' nel dettaglio nella sezione 3.1, sotto la descrizione di ogni classe .

5) Manuale utente della GUI

Il progetto prevede un'identificazione dell'utente tramite il login. Le coppie [username, password] di prova per effettuare la login e poter cosi' accedere alle diverse interfacce sono state specificate anche nella prima pagina della relazione, e sono:

- per l'Amministratore: [admin, admin]
- per Utente Esterno: [est, est]
- per Utente Studente: [stud, stud]
- per Utente Contribuente: [cont, cont]

Tutte le interfacce sono state pensate per essere intuitive e di semplice utilizzo per l'utente, di seguito vengono spiegate brevemente le varie funzionalita':

5.1) Interfaccia utenti esterno / studente / contribuente

Le interfacce dedicate a questi 3 tipi di utenti permettono la ricerca di articoli all'interno del database bibliotecario. Diversi utenti presentano diversi privilegi, percio' un utente esterno potra' cercare soltanto libri di narrativa e riviste, un utente studente potra' cercare libri scolastici, di narrativa e cd, mentre un utente contribuente potra' cercare tutti i tipi di articoli.

Alcune caratteristiche dovranno venire inserite dall'utente in dei campi digitazione, in questo caso l'articolo verra' cercato anche come match parziale con il testo inserito (es. ad un input "sentiero" verra' trovato il libro "il sentiero dei nidi di ragno"), oppure se il campo verra' lasciato vuoto, verranno trovati gli articoli con un valore qualsiasi in quel campo.

Altre caratteristiche verranno inserite tramite delle tendine a scelta multipla, se non si vuole specificare nessuna caratteristica per quel campo, bastera' selezionare la voce "tutti/e".

Infine per ogni tipo di articolo si potra' spuntare una checkbox per specificare se si vogliono cercare soltanto articoli disponibili, se questa non e' selezionata si cerchera' invece tra tutti gli articoli (sia disponibili che in prestito).

5.2) Interfaccia amministratore

L'interfaccia amministratore e' organizzata in due schede, una per la gestione degli articoli ed una per la gestione degli utenti. Ognuna di queste schede e' composta da dei bottoni che permettono l'aggiunta, l'eliminazione e la modifica degli oggetti, oltre al salvataggio di queste modifiche su file, e da una tabella che in ogni momento visualizzera' la lista aggiornata degli oggetti registrati nel database. La scheda per la gestione degli articoli ha inoltre un pulsante per visualizzare le informazioni dettagliate di un certo articolo.

L'aggiunta di un oggetto aprira' una nuova interfaccia per l'inserimento delle caratteristiche dell'oggetto; invece l'eliminazione e la modifica richiedono che venga selezionato dalla tabella l'oggetto sul quale si vuole eseguire l'azione.

Il bottone "Salva modifiche" comporta la sovrascrittura su file dei precedenti dati con quelli nuovi. Se invece questo pulsante non verra' premuto, le modifiche apportate al database verranno perse alla chiusura dell'interfaccia.

6) Ore di lavoro

Il progetto ha impiegato 57 ore di lavoro, ripartite circa nel seguente modo:

- **Progettazione e codifica iniziale del modello** 10 ore
- **Codifica GUI e integrazione di essa con il modello** 32 ore
- **Debugging** 8 ore (ripartite durante la realizzazione di tutto il codice)
- **Riempimento del file d'esempio e testing finale** 2 ore
- **Stesura relazione** 5 ore

7) Sistema operativo di sviluppo e versioni di compilatore e libreria Qt

- **Sistema operativo:** Windows 7 Professional a 32 bit
- **Versione Qt:** Qt 5.6.2
- **Compilatore:** Mingw 4.9.2 a 32 Bit

Il progetto è comunque stato testato sui computer di laboratorio, che hanno le seguenti caratteristiche:

- **Sistema operativo:** Ubuntu 16.04.1 LTS a 64 bit
- **Versione Qt:** Qt 5.5.1
- **Compilatore:** GCC