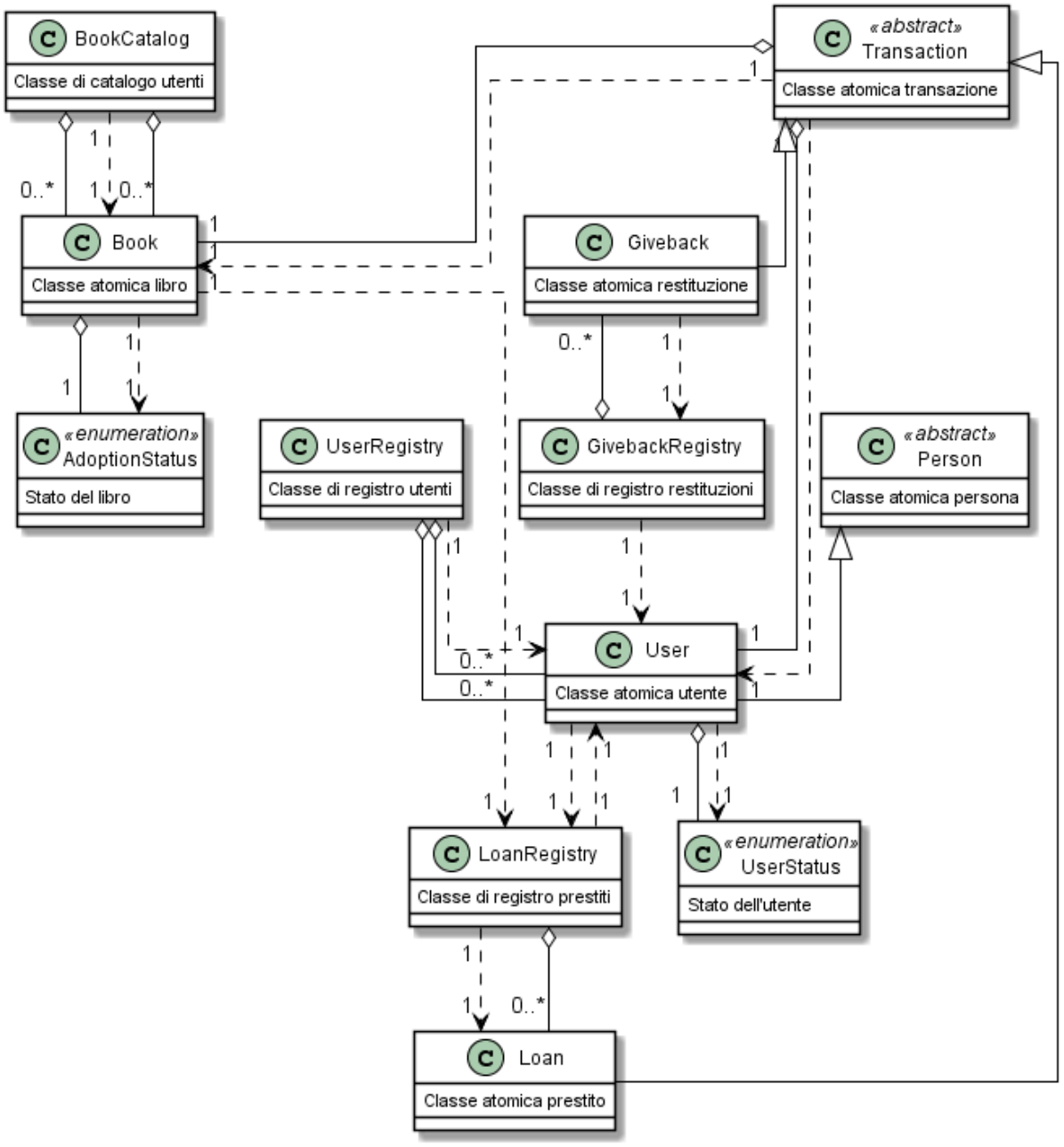
DIAGRAMMA DELLE CLASSI



Il modello del progetto si compone di classi che rappresentano gli elementi atomici di ogni collezione e di classi per la gestione delle collezioni. Questa suddivisione si riflette nella prima suddivisione dei package in:

- *it.campuslib.collections*
- *it.campuslib.domain.*

All'interno del package di domain, sono inseriti i package per la gestione delle classi atomiche:

- *it.campuslib.domain.users,* che contiene le classi per la gestione complessiva di Utente (*User*);
- *it.campuslib.domain.transactions,* che contiene le classi per la gestione complessiva di Prestito (*Loan*) e Restituzione (*Giveback*);
- *it.campuslib.domain.book,* che contiene le classi per la gestione complessiva di Libro (*Book).*

1. it.campuslib.domain.book

Il package si compone di una classe principale (Book), di classi per la gestione di eccezioni e di una classe per definire l'enumerazione che farà parte degli attributi dell'oggetto.
I metodi della classe principale sono volti alla gestione dello stato dell'oggetto, vi sono quindi i canonici getter e setter, a cui si aggiungono metodi più specifici per il controllo del formato per gli attributi e altri metodi generali.

La coesione complessiva risulta essere ***funzionale:*** l'intera classe è volta a rappresentare e gestire tutte le informazioni e i comportamenti dell'entità *Book.*

L'accoppiamento è per *Timbro*: la classe *Book* dipende da oggetti complessi, come *Loan* e *LoanRegistry,* ma non ha bisogno di conoscere la loro intera struttura (ad esempio, per verificare la disponibilità di un libro, ottiene la lista dei prestiti attivi per quel libro, tuttavia non utilizza tutte le informazioni relative ai prestiti, ma ricava soltanto il numero di essi).

2. it.campuslib.domain.user

Il package si compone di due classi principali, la classe astratta *Person* e la classe *User,* che che la estende.

Si è optato per l'implementazione della classe astratta *Person* al fine di rendere il programma estendibile e riutilizzabile in altri contesti: qualora si volessero ampliare le funzionalità del programma e gestire altri tipi di utenti, diversi dall'ambiente universitario, basterà estendere la superclasse.

Oltre alla relazione di ereditarietà che ha con la sua superclasse, la classe *User,* avendo tra i suoi attributi una lista riferita ai suoi prestiti attivi, si trova in una relazione di associazione con la classe *Transaction,* dovuta ai riferimenti a *Loan.*

La classe concreta *User* è rappresentativa degli Utenti, facenti parte del contesto universitario, che usufruiscono del servizio. La classe arricchisce gli attributi della sua superclasse con informazioni di servizio per la biblioteca. La gestione complessiva della

classe è speculare a quella della classe *Book*, ottenendo quindi un livello di coesione complessivo **funzionale.**

L'accoppiamento della classe User è per *Timbro:* come per la classe *Book*, la classe *User* dipende da oggetti complessi, come *Loan* e *LoanRegistry*, operando sull'intero oggetto invece di richiedere direttamente i dati necessari (ad esempio il conteggio dei prestiti).
La classe *Person* si limita ad un accoppiamento per *Dati*: gestisce solo dati primitivi o la base per la serializzazione. L'interazione è limitata al passaggio di dati semplici.

3. it.campuslib.domain.transactions

Il package contiene tre classi:

- la classe astratta *Transactions*
- la classe concreta *Giveback*
- la classe concreta *Loan*

Poiché le classi Loan e Giveback sono composte dagli stessi attributi, per evitare ripetitività del codice, si è inserita la classe astratta.

Le classi si trovano in relazione di associazione con le classi *Book e Person:* la classe astratta *Transaction* prevede tra i suoi attributi un riferimento per un elemento di entrambe le classi. Inoltre, trattandosi di estensioni di classe astratta, si trovano in relazione di ereditarietà con la superclasse transaction.

I metodi all'interno delle due classi sono prefissati alla gestione dei singoli oggetti, per cui la coesione risulta ***funzionale.***
Inoltre, la scelta di implementare la specializzazione della classe astratta facilita la manutenibilità: trattandosi di azioni diverse, qualora cambiassero regole per prestiti e restituzioni, basterà agire sulle singole classi.

Anche per questo package l'accoppiamento è per *Timbro:* la classe *Loan*, infatti, riceve e mantiene riferimenti a oggetti esterni come *Book* e *User,* non usufruendo di tutti gli attributi degli oggetti ricevuti.
Anche la classe astratta *Transaction,* che definisce la struttura condivisa per *Loan* e *Giveback,* include riferimenti a *Book* e *User.*

4. it.campuslib.collections.BookCatalog

La classe rappresenta l'archivio dei libri nella biblioteca. La collezione di oggetti viene gestita tramite due collezioni:

- una HashMap, per gestire l'inserimento di elementi uguali, utilizzando come chiave l'ISBN.
- una Lista Osservabile, per l'integrazione del modello con l'interfaccia utente.

La classe è in una doppia relazione di aggregazione con *Book,* dovuta alla presenza di due collezioni diverse, che devono però contenere gli stessi elementi.

I metodi implementati all'interno della classe sono specificatamente correlati alla gestione delle collezioni, risultando quindi in un livello di coesione funzionale.

L’accoppiamento è per *Timbro:* quasi tutti i metodi ricevono e restituiscono istanze di oggetti complessi.

5. it.campuslib.collections.UserRegistry

La classe rappresenta l’archivio degli utenti che accedono alla biblioteca. La collezione di oggetti viene gestita tramite una HashMap, per gestire l’inserimento di elementi uguali, utilizzando come chiave la matricola.
Per lo stesso fine di visibilità della classe *BookCatalog,* la classe si compone di un ulteriore collezione osservabile.

La presenza di due collezioni giustifica la doppia relazione di aggregazione con la classe *User:* le due collezioni aggregano gli stessi elementi.

Come per le rispettive classi atomiche, i metodi della collezione di Utenti seguono la stessa logica dei metodi della collezione dei libri, risultando in una coesione funzionale.

L’accoppiamento è per *Timbro:* quasi tutti i metodi ricevono e restituiscono istanze di oggetti complessi.

6. it.campuslib.collections.LoanRegistry

La classe funge da registro centrale unico per i soli prestiti attivi della biblioteca. La collezione viene gestita tramite un Set Osservabile, per garantire l’integrazione con l’interfaccia Utente.

La classe si trova in relazione di aggregazione con la rispettiva classe atomica *Loan* in quanto la collezione interna è aggregazione di questi elementi.

La classe ha la responsabilità unica di gestire la collezione di prestiti attivi. Tutti i metodi sono intrinsecamente legati a questa unica collezione.

Gestisce collezioni di oggetti complessi (*Loan* e *Transaction*). Le sue operazioni sono definite in termini della struttura di questi oggetti.

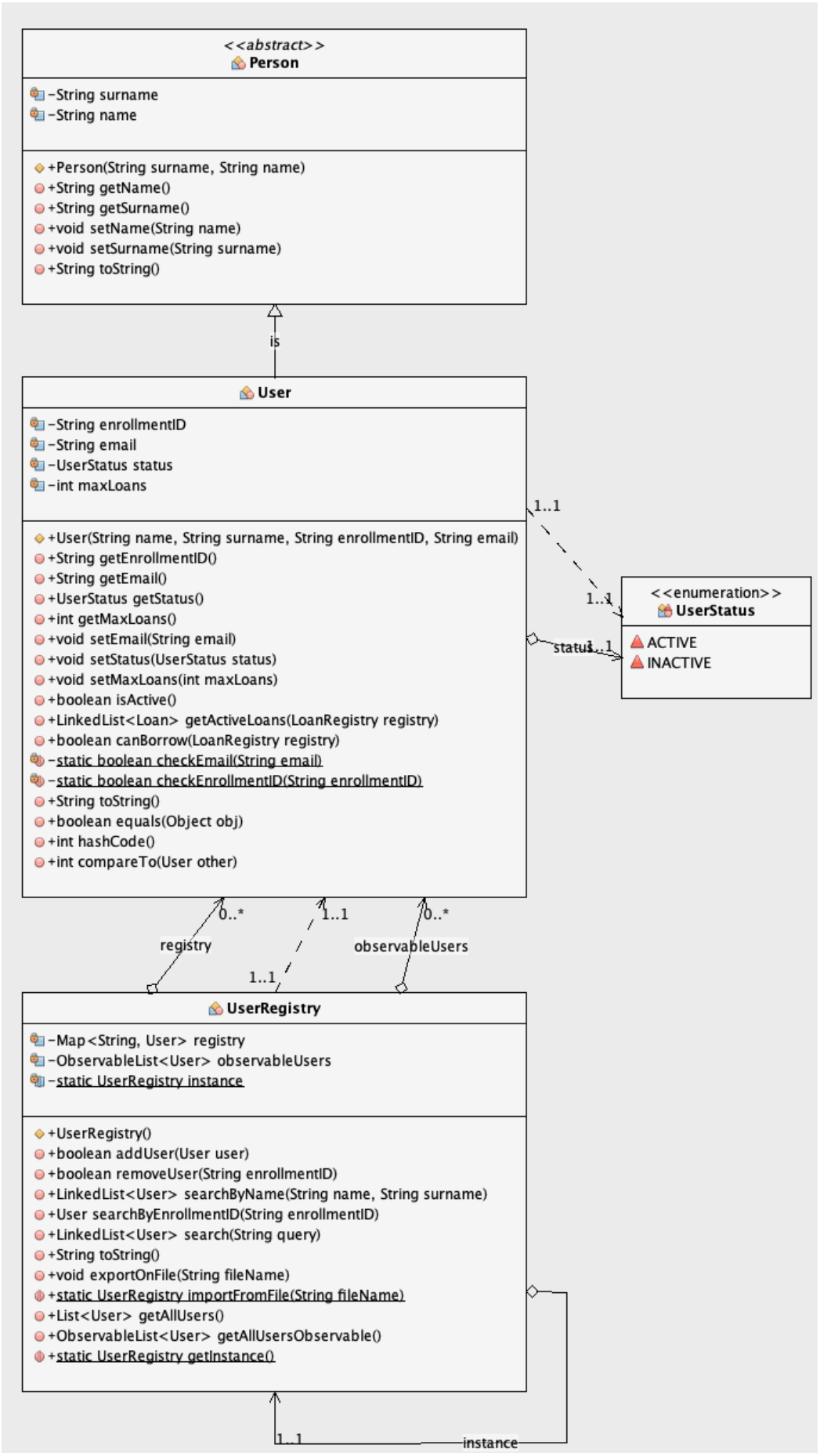
7. it.campuslib.collections.GivebackRegistry

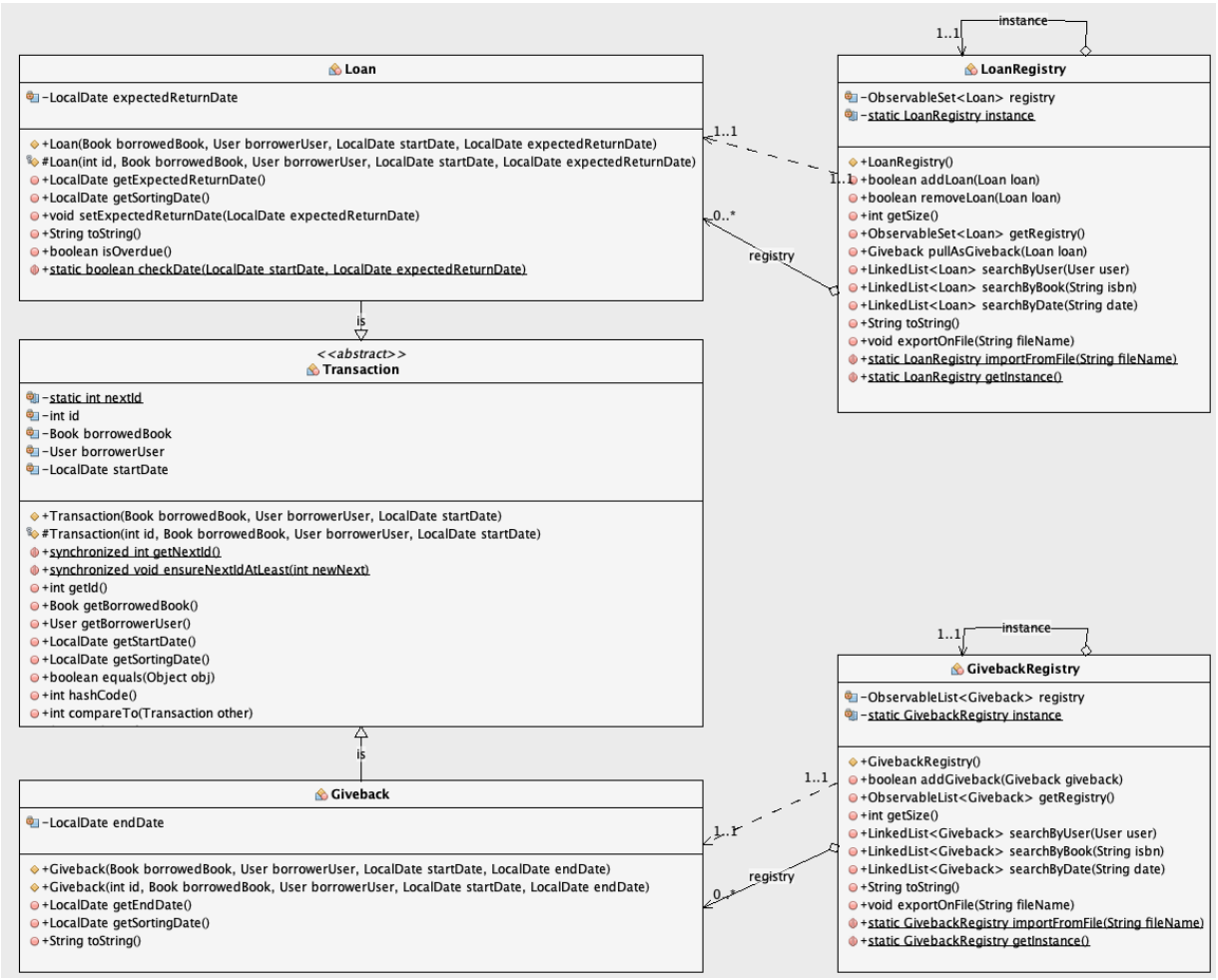
La classe GivebackRegistry funge da archivio storico delle restituzioni avvenute.La classe GivebackRegistry è speculare alla LoanRegistry e gestisce tutte le transazioni di restituzione completate.
Non prevedendo nessuna particolare azione se non l’ordine di inserimento, la collezione è implementata con una Lista Osservabile, per l’integrazione dell’interfaccia utente.

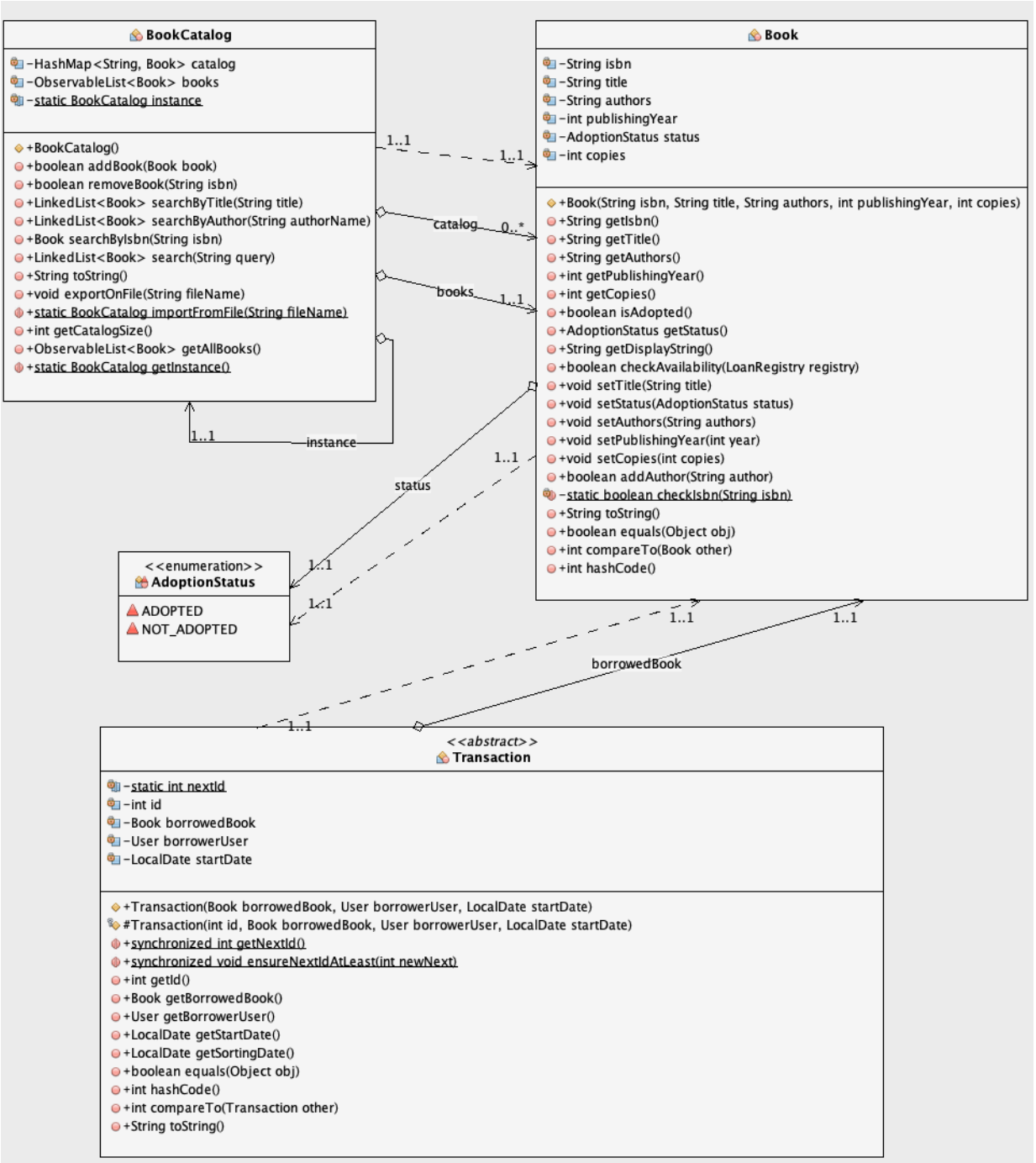
La classe si trova in relazione di aggregazione con la rispettiva classe atomica *Giveback* in quanto la collezione interna è aggregazione di questi elementi.

Permette solo l'aggiunta di nuove restituzioni, la dimensione e la consultazione. Non c'è un metodo di rimozione, il che è coerente con il ruolo di "registro storico".

L’accoppiamento è per *Timbro:* quasi tutti i metodi ricevono e restituiscono istanze di oggetti complessi.


<<abstract>>
Person
-String surname
-String name
+Person(String surname, String name)
+String getName()
+String getSurname()
+void setName(String name)
+void setSurname(String surname)
+String toString()
is
User
-String enrollmentID
-String email
-UserStatus status
-int maxLoans
+User(String name, String surname, String enrollmentID, String email)
+String getEnrollmentID()
+String getEmail()
+UserStatus getStatus()
+int getMaxLoans()
+void setEmail(String email)
+void setStatus(UserStatus status)
+void setMaxLoans(int maxLoans)
+boolean isActive()
+LinkedList<Loan> getActiveLoans(LoanRegistry registry)
+boolean canBorrow(LoanRegistry registry)
-static boolean checkEmail(String email)
-static boolean checkEnrollmentID(String enrollmentID)
+String toString()
+boolean equals(Object obj)
+int hashCode()
+int compareTo(User other)
1..1
1..1
status
1..1
<<enumeration>>
UserStatus
ACTIVE
INACTIVE
0..*
registry
1..1
1..1
0..*
observableUsers
UserRegistry
-Map<String, User> registry
-ObservableList<User> observableUsers
-static UserRegistry instance
+UserRegistry()
+boolean addUser(User user)
+boolean removeUser(String enrollmentID)
+LinkedList<User> searchByName(String name, String surname)
+User searchByEnrollmentID(String enrollmentID)
+LinkedList<User> search(String query)
+String toString()
+void exportOnFile(String fileName)
+static UserRegistry importFromFile(String fileName)
+List<User> getAllUsers()
+ObservableList<User> getAllUsersObservable()
+static UserRegistry getInstance()
1..1
instance


instance
1..1
Loan
-LocalDate expectedReturnDate
+Loan(Book borrowedBook, User borrowerUser, LocalDate startDate, LocalDate expectedReturnDate)
#Loan(int id, Book borrowedBook, User borrowerUser, LocalDate startDate, LocalDate expectedReturnDate)
+LocalDate getExpectedReturnDate()
+LocalDate getSortingDate()
+void setExpectedReturnDate(LocalDate expectedReturnDate)
+String toString()
+boolean isOverdue()
+static boolean checkDate(LocalDate startDate, LocalDate expectedReturnDate)
1..1
1..1
0..*
registry
LoanRegistry
-ObservableSet<Loan> registry
-static LoanRegistry instance
+LoanRegistry()
+boolean addLoan(Loan loan)
+boolean removeLoan(Loan loan)
+int getSize()
+ObservableSet<Loan> getRegistry()
+Giveback pullAsGiveback(Loan loan)
+LinkedList<Loan> searchByUser(User user)
+LinkedList<Loan> searchByBook(String isbn)
+LinkedList<Loan> searchByDate(String date)
+String toString()
+void exportOnFile(String fileName)
+static LoanRegistry importFromFile(String fileName)
+static LoanRegistry getInstance()
is
<<abstract>>
Transaction
-static int nextId
-int id
-Book borrowedBook
-User borrowerUser
-LocalDate startDate
+Transaction(Book borrowedBook, User borrowerUser, LocalDate startDate)
#Transaction(int id, Book borrowedBook, User borrowerUser, LocalDate startDate)
+synchronized int getNextId()
+synchronized void ensureNextIdAtLeast(int newNext)
+int getId()
+Book getBorrowedBook()
+User getBorrowerUser()
+LocalDate getStartDate()
+LocalDate getSortingDate()
+boolean equals(Object obj)
+int hashCode()
+int compareTo(Transaction other)
is
Giveback
-LocalDate endDate
+Giveback(Book borrowedBook, User borrowerUser, LocalDate startDate, LocalDate endDate)
+Giveback(int id, Book borrowedBook, User borrowerUser, LocalDate startDate, LocalDate endDate)
+LocalDate getEndDate()
+LocalDate getSortingDate()
+String toString()
1..1
1..1
0..*
registry
instance
1..1
GivebackRegistry
-ObservableList<Giveback> registry
-static GivebackRegistry instance
+GivebackRegistry()
+boolean addGiveback(Giveback giveback)
+ObservableList<Giveback> getRegistry()
+int getSize()
+LinkedList<Giveback> searchByUser(User user)
+LinkedList<Giveback> searchByBook(String isbn)
+LinkedList<Giveback> searchByDate(String date)
+String toString()
+void exportOnFile(String fileName)
+static GivebackRegistry importFromFile(String fileName)
+static GivebackRegistry getInstance()


BookCatalog
-HashMap<String, Book> catalog
-ObservableList<Book> books
-static BookCatalog instance
+BookCatalog()
+boolean addBook(Book book)
+boolean removeBook(String isbn)
+LinkedList<Book> searchByTitle(String title)
+LinkedList<Book> searchByAuthor(String authorName)
+Book searchByIsbn(String isbn)
+LinkedList<Book> search(String query)
+String toString()
+void exportOnFile(String fileName)
+static BookCatalog importFromFile(String fileName)
+int getCatalogSize()
+ObservableList<Book> getAllBooks()
+static BookCatalog getInstance()
Book
-String isbn
-String title
-String authors
-int publishingYear
-AdoptionStatus status
-int copies
+Book(String isbn, String title, String authors, int publishingYear, int copies)
+String getIsbn()
+String getTitle()
+String getAuthors()
+int getPublishingYear()
+int getCopies()
+boolean isAdopted()
+AdoptionStatus getStatus()
+String getDisplayString()
+boolean checkAvailability(LoanRegistry registry)
+void setTitle(String title)
+void setStatus(AdoptionStatus status)
+void setAuthors(String authors)
+void setPublishingYear(int year)
+void setCopies(int copies)
+boolean addAuthor(String author)
-static boolean checkIsbn(String isbn)
+String toString()
+boolean equals(Object obj)
+int compareTo(Book other)
+int hashCode()
1..1
1..1
catalog
0..*
books
1..1
1..1
instance
1..1
status
1..1
1..1
<<enumeration>>
AdoptionStatus
ADOPTED
NOT_ADOPTED
1..1
1..1
borrowedBook
1..1
<<abstract>>
Transaction
-static int nextId
-int id
-Book borrowedBook
-User borrowerUser
-LocalDate startDate
+Transaction(Book borrowedBook, User borrowerUser, LocalDate startDate)
#Transaction(int id, Book borrowedBook, User borrowerUser, LocalDate startDate)
+synchronized int getNextId()
+synchronized void ensureNextIdAtLeast(int newNext)
+int getId()
+Book getBorrowedBook()
+User getBorrowerUser()
+LocalDate getStartDate()
+LocalDate getSortingDate()
+boolean equals(Object obj)
+int hashCode()
+int compareTo(Transaction other)
+String toString()