# User

La classe *User* ha il ruolo di superclasse astratta per i figli *Admin* e *Customer*, fornisce quindi metodi per effettuare il login e cambiare il nome utente o la password.

## Attributi

* username: nome identificativo dell’utente
* password: codice di verifica per l’utente

# Admin

*Admin* è una sottoclasse di *User* riservata agli utenti del sistema che hanno il compito di gestire i voli e i gate, effettuare il check-in di passeggeri e aggiornare le informazioni dei bagagli.

# Customer

*Customer* è una sottoclassedi *User* che rappresenta gli utenti del sistema che possono prenotare voli per conto di passeggeri (salvati poi come *Passenger*).

## Relazioni

*Customer* è in una relazione uno a molti con la classe *Booking*, la visibilità è bidirezionale per permettere alle istanze di *Customer* di vedere i propri *Booking* (per la direzione inversa vedere *Booking)*.

Dal lato di *Customer*, la relazione è stata implementata con un ArrayList di *Booking.*

## Metodi

La classe *Customer* dispone di metodi per effettuare prenotazioni, cercare le prenotazioni già effettuate e verificarne lo stato e modificarle o disdirle. Può inoltre segnalare la perdita di un *Luggage* per conto di *Passenger* per cui ha effettuato la prenotazione.

# Booking

Booking è la classe le cui istanze rappresentano le prenotazioni che i *Customer* effettuano per un determinato voloappartenente alla classe *Flight*; pertanto, le suddette istanze sono in relazione con i passeggeri che le compongono.

## Attributi

Booking dispone dell’attributo status, rappresenta lo stato della prenotazione e ha il suo dominio nell’enumeration *BookingStatus* (booked, confirmed, pending, cancelled).

## Relazioni

*Booking* partecipa a relazioni con

* *Customer:* relazione di tipo molti a uno, in particolare non può esistere un’istanza della classe *Booking* che non è in relazione con un’istanza della classe *Customer;* la visibilità è bidirezionale, in quanto può essere utile poter risalire all’utente che effettua una determinata prenotazione (per il verso opposto vedere classe *Customer*).
* *Flight:* relazione di tipo molti a uno, in particolare non può esistere un’istanza della classe *Booking* che non è in relazione con un’istanza della classe *Flight;* la visibilità è bidirezionale, per permettere ai clienti di visualizzare i voli associati alle proprie prenotazioni (per il verso opposto vedere classe *Flight*).
* *Passenger:* relazione di aggregazione (i passeggeri sono parte della prenotazione) di tipo uno a molti, in particolare non può esistere un’istanza della classe *Booking* che non è in relazione con almeno un’istanza della classe *Passenger;* la visibilità è unidirezionale, in quanto i clienti devono essere in grado di visualizzare i passeggeri che fanno parte delle proprie prenotazioni.

# Passenger

Passenger è la classe le cui istanze rappresentano i passeggeri che prendono parte ai voli. Essi non accedono al database e non eseguono operazioni su di esso come gli utenti, in quanto esistono solo in relazione alla prenotazione e al volo di cui fanno parte, per queste operazioni vedere *Customer*.

## Attributi

* firstname, lastname e SSN rappresentano le informazioni personali del passeggero.
* ticket\_number, è il biglietto che identifica un passeggero.
* seat, rappresenta il posto a sedere del passeggero su un determinato volo.

## Relazioni

*Passenger* partecipa a relazioni con:

* *Booking*: relazione di aggregazione (i passeggeri sono parte della prenotazione) di tipo molti a uno, in particolare non può esistere un’istanza della classe *Passenger* che non è in relazione con un’istanza della classe *Booking;* la visibilità è unidirezionale, in quanto i passeggeri non sono in grado di accedere al database e di effettuare prenotazioni.
* *Flight:* relazione di tipo molti a uno, in particolare non può esistere un’istanza della classe *Passenger* che non è in relazione con un’istanza della classe *Flight;* la visibilità è unidirezionale, in quanto i passeggeri non sono in grado di accedere al database e di effettuare prenotazioni.
* *Luggage:* relazione di tipo uno a molti, la visibilità è bidirezionale, in quanto gli utenti tramite i passeggeri devono essere in grado di risalire ai loro bagagli (per il verso opposto vedere la classe *Luggage*).

# Luggage

*Luggage* è la classe le cui istanze rappresentano i bagagli posseduti dai passeggeri.

Attributi

* id, è l’identificativo necessario per il tracciamento del bagaglio.
* type, definisce il tipo di bagaglio e ha il suo dominio nell’enumeration *LuggageType* (carry\_on, checked).
* status, definisce lo stato del bagaglio e ha il suo dominio nell’enumeration *LuggageStatus* (booked, loaded, withdrawable, lost).

## Relazioni

* *Passenger:* relazione di tipo molti a uno, in particolare non può esistere un’istanza della classe *Luggage* che non è in relazione con un’istanza della classe *Passenger;* la visibilità è bidirezionale, in quanto è necessario poter risalire al proprietario di un determinato bagaglio (per il verso opposto vedere la classe *Passenger*).

# Flight

*Flight* ha il ruolo di superclasse astratta, con figli *Departing* e *Arriving*, che formano una specializzazione totale e disgiunta.

## Attributi

* id è l’identificatore univoco di un volo.
* status è lo stato di un volo, con dominio nell’enumeration *FlightStatus* (programmed, cancelled, departed, delayed, landed).
* max\_seats e free\_seats vengono utilizzati per gestire l’inserimento/rimozione di eventuali prenotazioni e relativi passeggeri.

## Relazioni

La classe *Flight* è in relazione con:

* *Booking*: relazione 1 a molti, la visibilità è bidirezionale, per motivi di gestione delle prenotazioni di uno specifico volo. La relazione è implementata in *Flight* con un ArrayList di *Booking*. (per l’altra direzione, vedere *Booking*).
* *Passenger*: relazione 1 a molti, la visibilità è unidirezionale (vedere *Passenger*). La relazione è implementata con un ArrayList di *Passenger*.

# Departing

*Departing* è una sottoclasse di *Flight*, rappresenta gli aerei in partenza da Napoli.

## Attributi

L’attributo departure\_delay è rappresentato come un intero di minuti (inizialmente posto a 0) per una facile gestione del codice, deve essere opportunamente convertito in ore e minuti in visualizzazione a schermo.

## Relazioni

È in relazione con la classe *Gate*, con una relazione 1 a molti, la visibilità è unidirezionale. La relazione è implementata mettendo in *Departing* l’id del gate come attributo (inizialmente con valore null, sarà istanziato quando l’aereo starà per partire).

# Arriving

*Arriving* è una sottoclasse di *Flight*, rappresenta gli aerei in arrivo a Napoli.

## Attributi

L’attributo arrival\_delay è rappresentato come un intero di minuti (inizialmente posto a 0) per una facile gestione del codice, deve essere opportunamente convertito in ore e minuti in visualizzazione a schermo.

## Gate

*Gate* rappresenta il varco da cui ogni aereo in partenza può partire, è rappresentato da un id univoco ed è in una relazione 1 a molti con la classe *Departing*, con visibilità unidirezionale (il gate non vede gli aerei che partono da esso).