



DIPARTIMENTO DI INGEGNERIA  
E SCIENZA DELL'INFORMAZIONE

# SleepCode

PROGETTO PER IL CORSO DI INGEGNERIA DEL SOFTWARE  
ANNO ACCADEMICO 2023-2024

---

## Documento di Architettura

---

*Descrizione:* descrizione dell'architettura di sistema sotto forma di diagrammi delle classi e codice OCL.

*Numero documento:* D3

*Versione documento:* 1.0

*Membri del gruppo:*

Raffaele CASTAGNA

Alberto ROVESTI

Zeno SALETTI

*Numero gruppo:* G17

*Ultima revisione:* 26 gennaio 2024

## Indice

<b>1</b>	<b>Definizione delle classi</b>	<b>3</b>
1.1	Utenti . . . . .	3
1.2	Gestione autenticazione . . . . .	6
1.3	Recupero della password . . . . .	7
1.4	Catalogo . . . . .	8
1.5	Problemi ed Esercitazione . . . . .	10
<b>2</b>	<b>Specifiche in codice OCL</b>	<b>12</b>
2.1	Autenticazione . . . . .	12
2.2	Recupero della password . . . . .	12
2.3	Utente autenticato . . . . .	12
2.4	Modifica della password . . . . .	13
2.5	Modifica del catalogo . . . . .	13
2.6	Test Cases . . . . .	15
2.7	Cronometro . . . . .	15
<b>3</b>	<b>Diagramma delle classi con codice OCL</b>	<b>16</b>

---

**Consigli utili per la consultazione del testo:** Se il lettore per file **.pdf** attualmente in uso lo consente, è possibile navigare con più semplicità e velocità all'interno di questo documento cliccando sugli elementi dell'indice.

## Scopo del documento

In questo documento viene riportata la definizione dell'architettura del progetto *SleepCode* impiegando diagrammi delle classi, realizzati secondo gli standard di Unified Modeling Language (UML), e codice scritto in Object Constraint Language (OCL). Nel documento precedente (D2, *Specifica dei Requisiti*) sono stati presentati il diagramma degli use case, quello di contesto e infine il Diagramma dei Componenti. Considerando tale progettazione, viene ora definita l'architettura del sistema specificando in modo più dettagliato le classi implementabili sotto forma di codice, insieme alla logica che regola il comportamento del software che si intende realizzare.

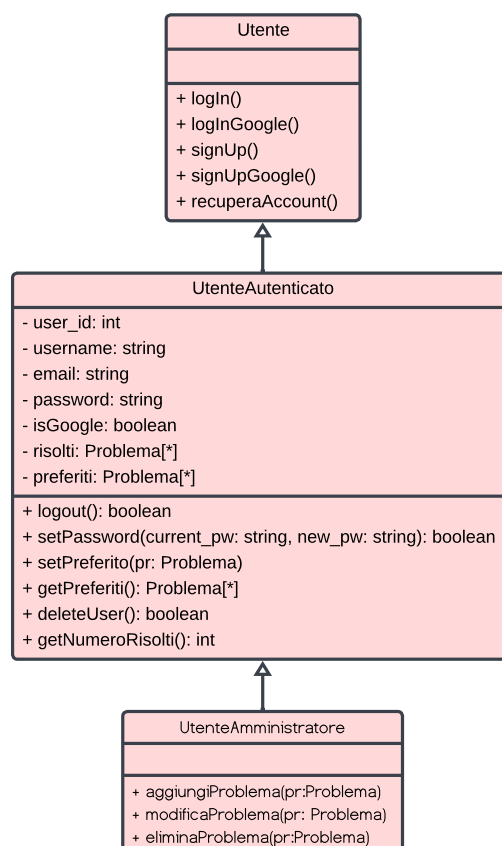
Il linguaggio UML, utilizzato per descrivere le classi, è supportato da codice OCL, impiegato invece per catturare gli aspetti logici citati sopra, che non sarebbero altrimenti esprimibili formalmente mediante i soli diagrammi delle classi.

## 1 Definizione delle classi

Nella presente sezione vengono illustrate in linguaggio UML le classi previste dal progetto *SleepCode*. Ogni componente del diagramma dei componenti, presente nel documento D2, viene qui rappresentato in forma di una o più classi. Le classi individuate sono costituite da un nome, un insieme di attributi che identificano i dati gestiti dalla classe stessa e una lista di metodi che definiscono le operazioni eseguibili da quella classe. Eventuali relazioni tra classi sono evidenziate da alcune associazioni.

### 1.1 Utenti

L'attore che nel diagramma di contesto usufruisce delle funzionalità offerte dal sistema è l'utente. Come descritto nei documenti di Analisi (D1) e Specifica dei Requisiti (D2), esistono tre categorie di utenti: anonimo, autenticato e amministratore. La Figura 1 illustra le rispettive classi.



**Figura 1:** Definizione della classe **Utente** e delle sue generalizzazioni

Dal momento che tutte queste categorie condividono alcune funzionalità, è stata creata una classe **Utente**, che logicamente corrisponde alla categoria degli utenti anonimi, ovvero utenti che non hanno effettuato il login. Di fatto, non è prevista la memorizzazione di alcuna informazione riguardo a questa categoria di utenti e le uniche operazioni disponibili sono:

- **login()**: richiesta di accesso con sistema di credenziali interne.
- **signUp()**: richiesta di registrazione con sistema di credenziali interne.
- **loginGoogle()**, **signUpGoogle()**: controparti dei metodi precedenti, che però si avvalgono dei servizi di autenticazione Google.
- **recuperaAccount()**: richiesta di avvio della procedura di recupero account.

Da questa prima classe viene definita, mediante generalizzazione, la classe **UtenteAutenticato**, che aggiunge metodi e attributi accessibili previo login:

- **user\_id**: questo attributo riveste il ruolo di identificatore univoco dell'account dell'utente registrato ed eventualmente autenticato.
- **username**, **email**, **password**: i dati richiesti per creare un account. In caso di autenticazione con Google, è richiesto che la classe memorizzi solo l'indirizzo email impiegato e l'username.
- **isGoogle**: parametro che distingue gli account collegati con Google da account registrati con il meccanismo di credenziali interne.
- **risolti**, **preferiti**: insiemi dei problemi, rispettivamente, risolti dall'utente e salvati come preferiti dall'utente.
- **logout()**: richiesta di logout.
- **setPassword(...)**: procedura atta a modificare la password dell'account. Oltre alla nuova password, viene richiesta, per ragioni di sicurezza, la password attualmente in uso. Viene restituita una risposta di tipo booleana per segnalare il successo o l'insuccesso dell'operazione.
- **setPreferito(...)**: questo metodo provvede ad aggiungere o eliminare dalla lista dei preferiti dell'utente il problema fornito: se presente, esso viene rimosso dalla lista, altrimenti viene aggiunto (viene inoltre incrementato il valore dell'attributo **preferito** della classe **Problema**, descritta nelle prossime sezioni).
- **getPreferiti()**: i preferiti dell'utente autenticato vengono messi a disposizione all'esterno della classe grazie a questo metodo. Come descritto nel diagramma dei componenti, in questo modo viene realizzata l'interfaccia che passa i preferiti dell'utente dal Gestore Utente verso il Catalogo, in modo da visualizzarli nella lista dei problemi.

- `deleteUser()`: il metodo per l'eliminazione dell'account restituisce un valore booleano per notificare l'esito dell'operazione.
- `getNumeroRisolti()`: il valore restituito rappresenta i progressi dell'utente, ovvero il numero di problemi risolti rispetto al totale presente nel catalogo. Questo valore viene calcolato in base al numero di problemi presenti in `risolti` e attualmente memorizzati nel database.

Infine, un'ulteriore generalizzazione specifica le funzionalità aggiuntive dell'utente amministratore, ovvero metodi utili alla gestione del catalogo e dei problemi: `aggiungiProblema`, `modificaProblema`, `eliminaProblema`.

Facendo riferimento al Diagramma dei Componenti del documento D2, queste classi sono state in parte estratte da tre componenti distinti che si occupano dell'interazione con l'utente esterno: dalla *Pagina di Autenticazione* sono stati raggruppate le interfacce che si trovano in `Utente`; in `UtenteAutenticato` sono state raccolte le interfacce presenti nel componente *Gestore Profilo*; in `UtenteAmministratore` sono state raggruppate le interfacce definite nel componente *Catalogo*.

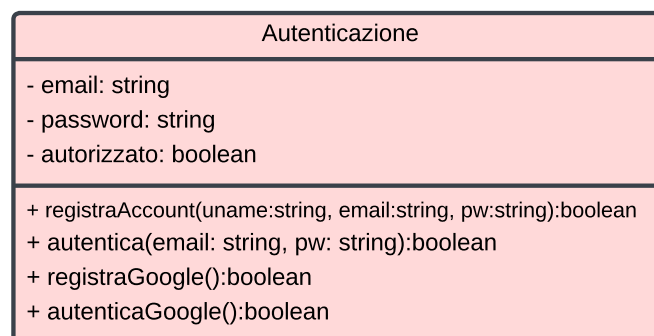
## 1.2 Gestione autenticazione

Nel diagramma di contesto, l'autenticazione si avvale di due sistemi subordinati: *Firebase*, che si occupa di memorizzare gli account registrati con sistema di autenticazione interno, e *Google SignIn*, con il quale è possibile effettuare la registrazione o l'autenticazione mediante un account Google.

Il componente *Gestore Autenticazione* del rispettivo diagramma si occupa di interagire con tali sistemi esterni in modo da convalidare le operazioni di registrazione e autenticazione. Per questo motivo, nel diagramma delle classi viene individuata la classe **Autenticazione** (Figura 2):

- **autorizzato**: questo attributo mantiene lo stato dell'autorizzazione dell'utente
- **registraAccount**: questo metodo esegue la registrazione comunicando con il servizio di database.
- **autentica**: l'autenticazione mediante credenziali interne avviene grazie a questo metodo.
- **registraGoogle**, **autenticaGoogle**: analoghi ai precedenti, questi metodi si occupano delle operazioni di registrazione ed autenticazione con Google.

Tutti i metodi comunicano all'esterno l'esito delle loro operazioni mediante un valore booleano di ritorno.

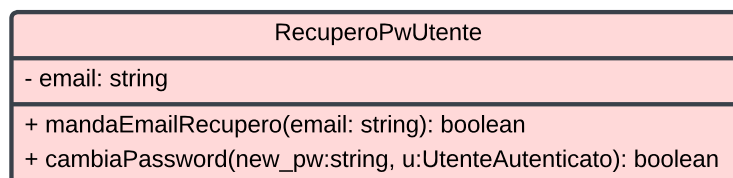


**Figura 2:** Definnizione della classe **Autenticazione**

### 1.3 Recupero della password

Nel Diagramma dei Componenti viene individuata la *Pagina di Recupero*, che offre supporto agli utenti registrati, con sistema di credenziali interne, che hanno dimenticato la password e che intendono recuperare l'account. Nel Diagramma di Contesto, il servizio di posta elettronica rientra nello scambio di informazioni necessario per la procedura di recupero. Per questi motivi viene definita la classe **RecuperoPwUtente** (Figura 3), che raccoglie attributi e metodi necessari per effettuare il recupero, interagendo tra l'altro con il servizio di posta elettronica.

- **email**: l'indirizzo email di recupero viene memorizzato per conservarlo durante tutta la procedura di recupero.
- **inviaEmailRecupero**: il messaggio di recupero viene inviato all'indirizzo specificato. Questo metodo rappresenta la richiesta di invio da parte del sistema nei confronti del servizio di posta.
- **cambiaPassword**: la nuova password inserita dall'utente viene aggiornata nel suo account. Come rappresentato dal Diagramma dei Componenti, questa è la realizzazione dell'interfaccia fornita al database. Questa operazione può essere effettuata solo su un account già registrato con l'email specificata e quindi esistente, come mostrato dal tipo del secondo parametro richiesto (**UtenteAutenticato**).

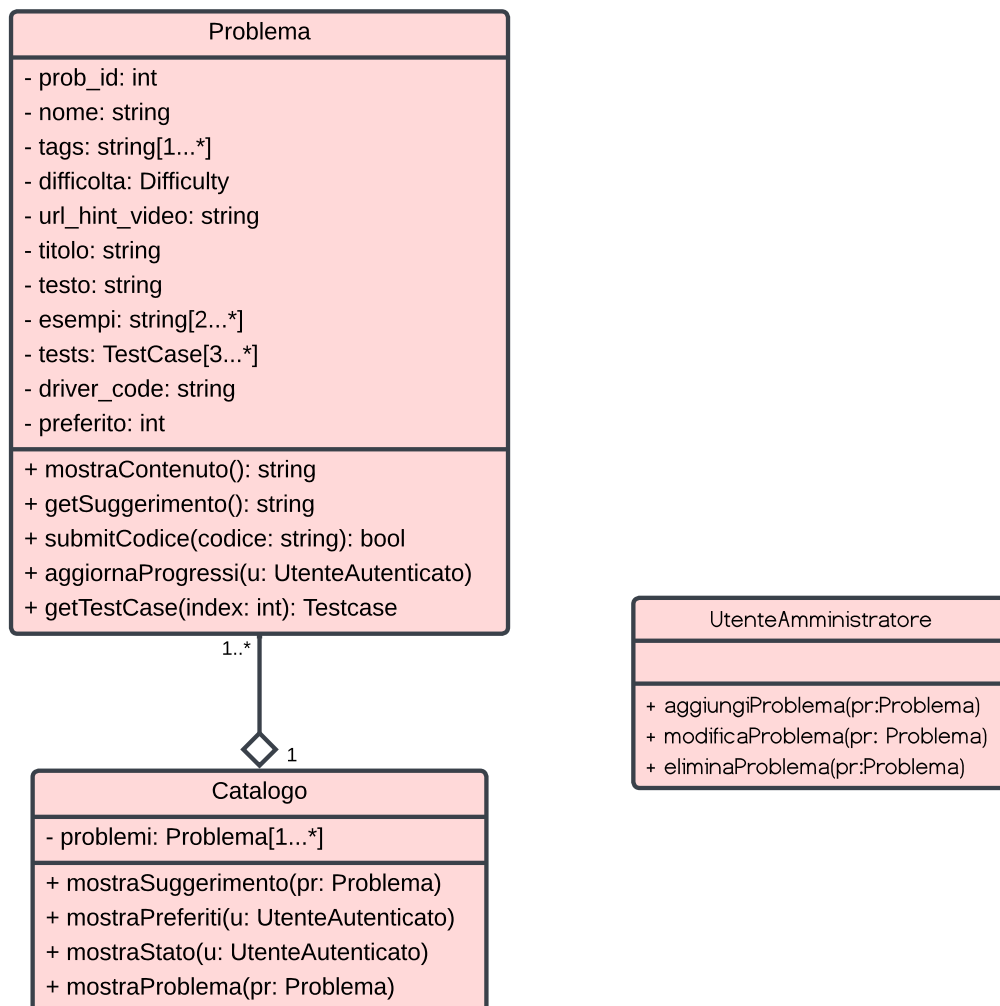


**Figura 3:** Specifica architettuale del componente *Pagina di Recupero* con la classe **RecuperoPwUtente**



## 1.4 Catalogo

Il *Catalogo* presente nel Diagramma dei Componenti è rappresentato in Figura 4 sotto forma di due classi principali: **UtenteAmministratore**, descritto insieme alle altre categorie di utenti, e **Catalogo**. Tale scelta architetturale è giustificata dall'esigenza di identificare chiaramente il ruolo dell'utente amministratore, mantenendo dunque una definizione dei livelli di accesso coerente con quella dei documenti precedenti, oltre alla volontà di ridurre il carico di informazione della classe **Catalogo** rendendo il rispettivo componente del diagramma più modulare e meno monolitico.



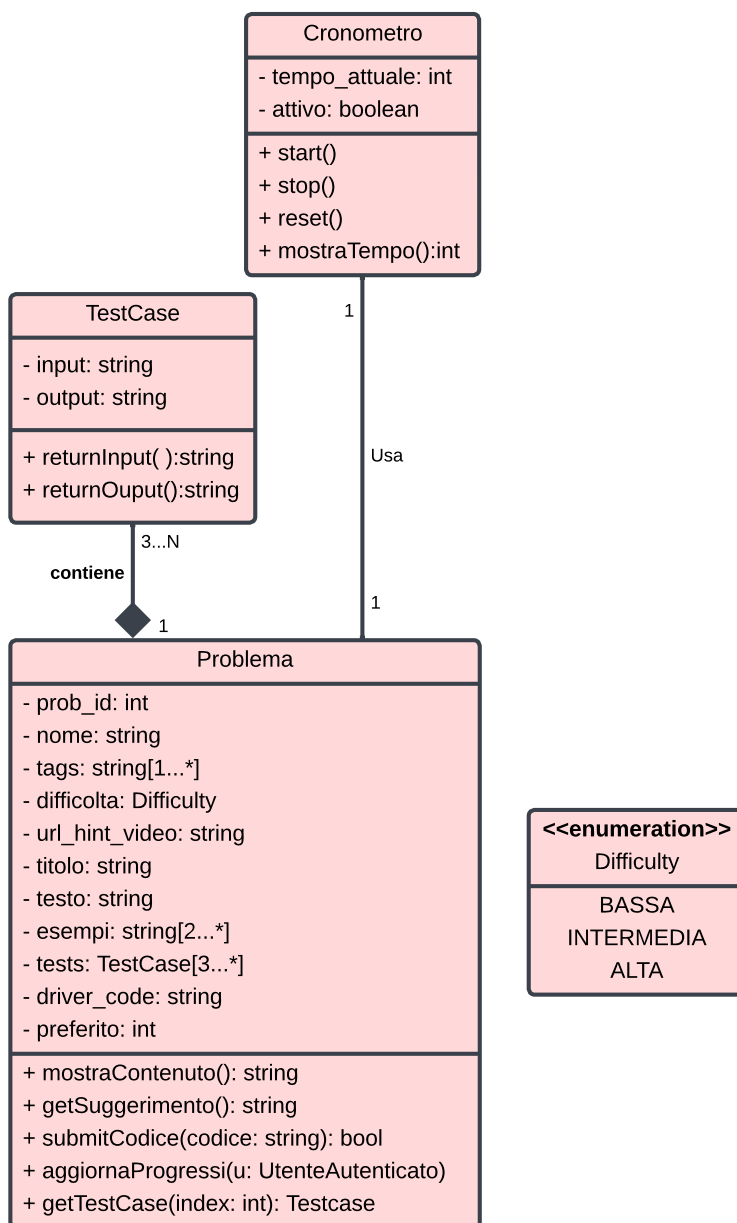
**Figura 4:** Specifica architetturale del componente *Catalogo* con la classe *Catalogo*

La Figura 4 mostra anche la classe **Problema**, descritta dettagliatamente nella prossima sezione. Il catalogo raccoglie i problemi disponibili sulla piattaforma, quindi **Catalogo** è legato a **Problema** da un'associazione di aggregazione.

- **problemi[1...\*]**: questo attributo indica che il catalogo consiste in una lista di almeno un problema.
- **mostraSuggerimento(...)**: sulla base del metodo selezionato, il metodo interagisce con YouTube per richiedere la riproduzione del video-suggerimento.
- **mostraPreferiti(...)**: sulla base dei dati dell'account dell'utente (autenticato), vengono contrassegnati i problemi preferiti all'interno del catalogo.
- **mostraStato(...)**: come per il metodo precedente, il **Catalogo** mostra lo stato dei problemi, evidenziando quelli già risolti dall'utente autenticato.
- **mostraProblema(...)**: questo metodo richiede l'apertura di un problema selezionato dall'utente, permettendo di consultare il contenuto e accedere alla rispettiva area di esercitazione.

## 1.5 Problemi ed Esercitazione

La Figura 5 illustra le classi che costituiscono il componente *Pagina di Esercitazione*.



**Figura 5:** Classi dedotte dal componente *Pagina di esercitazione*

La classe **Problema** è definita come segue:

- **nome**, **tags**, **difficolta**, **url\_hint\_video**: questi rappresentano i campi descrittivi del problema, come specificati nel documento D1. La difficoltà può assumere solamente uno dei valori definiti dal dominio del tipo enumerato **Difficulty**.
- **titolo**, **testo**, **esempi**: i dati strutturali del problema.
- **tests**: i test cases utilizzati per verificare il codice sottoposto dall'utente.
- **driver\_code**: il frammento di codice predefinito che viene mostrato nella zona di inserimento del codice, nell'area di esercitazione.
- **preferito**: contatore che registra la quantità di utenti registrati che hanno contrassegnato questo specifico problema.

Al fine di alleggerire la classe **Problema**, è stata accolta la scelta di dedicare una classe **TestCase**. Essa ha il compito di raccogliere le informazioni chiave necessarie al test del codice fornito dall'utente (il quale si esercita sul problema specifico): **input**, **output** e dei metodi che rendono questi attributi disponibili all'esterno. Si noti che la relazione che unisce **TestCase** e **Problema** è una **composizione** (indicato dall'estremità romboidale nera): i testcases sono infatti associati a specifici problemi, come espresso dalle cardinalità, ma non hanno alcun significato se il problema specifico al quale dovrebbero essere associati non esiste.

Altra parte integrante della Pagina di Esercitazione è la classe **Cronometro**, che prevede le semplici funzionalità per mostrare il tempo registrato in secondi, essere avviato, fermato e reimpostato a 0 (condizioni più dettagliate di questa classe sono mostrate nel codice OCL delle prossime sezioni).

## 2 Specifiche in codice OCL

Nella sezione che segue viene descritta formalmente la logica prevista nel comportamento di alcune classi, in relazione alle loro operazioni possibili. Il codice OCL impiegato consente di esprimere tale logica, non descrivibile con i soli diagrammi delle classi in UML.

### 2.1 Autenticazione

### 2.2 Recupero della password

Con riferimento alla classe `RecuperoPwUtente`, l'attributo `email` deve essere sempre presente affinché la classe possa essere impiegata.

```
context RecuperoPwUtente inv:  
email->notEmpty()
```

Inoltre, `cambiaPassword`, può essere utilizzato per il recupero di account non-Google e solo se viene fornita una password valida da associare ad un account esistente (da recuperare) con indirizzo email specificato.

```
context RecuperoPwUtente::cambiaPassword(...)  
pre: self.email == u.email and  
new_pw->notEmpty() and  
new_pw->isValid() and  
u.isGoogle == false  
post: u.password = new_pw
```

### 2.3 Utente autenticato

La classe `UtenteAutenticato` prevede che i campi `user_id`, `username` e `email` non siano mai vuoti, affinché un account (anche se creato con Google-SignIn) possa essere ufficialmente registrato sulla piattaforma.

```
context UtenteAutenticato inv
  user_id -> notEmpty() and
  username -> notEmpty() and
  email -> notEmpty()
```

## 2.4 Modifica della password

Gli utenti registrati con account Google non possono creare una nuova password, non avendo la necessità di utilizzare quel tipo di dato. La password dell'account può essere modificata qualora la password attuale venga inserita correttamente e solo se la nuova password è valida (**RNF 11** - Password sicura).

```
context UtenteAutenticato::setPasword(...)
pre: pw_attuale = password
    and nuova_pw.isValid()
    and self.IsGoogle = false
post: self.password = nuova_password
```

## 2.5 Modifica del catalogo

Nei documenti precedenti viene sottolineato che i campi descrittivi e strutturali di ogni problema del catalogo devono essere sempre presenti. Con riferimento alla classe **UtenteAmministratore**, oltre alle cardinalità specificate tra parentesi quadre per alcuni parametri in input dei metodi **aggiungiProblema** e **modificaProblema**, vengono definite queste ulteriori condizioni:

```
context UtenteAmministratore::aggiungiProblema(...)
pre: nome -> notEmpty() and dif -> notEmpty() and
    hint -> notEmpty() and
    titolo -> notEmpty() and
    testo -> notEmpty()
```

```
context UtenteAmministratore::modificaProblema(pr)
post: pr.nome -> notEmpty() and
    difficolta -> notEmpty() and
    hint -> notEmpty() and
    titolo -> notEmpty() and
    testo -> notEmpty()
```

## 2.6 Test Cases

La classe `TestCase` prevede che i suoi attributi non siano mai vuoti, affinché i test cases possano essere utilizzati per verificare la correttezza del codice sottoposto dall'utente.

```
context TestCase inv:
  input -> notEmpty()
  and output -> notEmpty()
```

## 2.7 Cronometro

Sono elencati di seguito i dettagli riguardanti il funzionamento della classe `Cronometro`:

- Il `tempo_attuale` registrato può assumere solamente valori positivi.

```
context Cronometro inv:
  tempo_attuale >= 0
```

- Il metodo `start()` esegue l'avvio della registrazione del tempo trascorso, a partire dal valore contenuto in `tempo_attuale`, nella situazione in cui il cronometro non è attivo. L'attributo `attivo` viene opportunamente aggiornato. L'operazione opposta è rappresentata da `stop()`, utilizzabile quando il cronometro è attivo. Il metodo `reset()` riporta il cronometro al suo stato iniziale, quindi inattivo e tempo registrato 0.

```
context Cronometro::start()
pre: attivo == false
post: attivo = true
```

```
context Cronometro::stop()
pre: attivo == true
post: attivo = false
```

```
context Cronometro::reset()
post: attivo = false
post: tempo_attuale = 0
```



### 3 Diagramma delle classi con codice OCL

In Figura 6 è riportato un diagramma che mostra nell'insieme sia le classi individuate che il relativo codice OCL, definiti nelle sezioni precedenti.

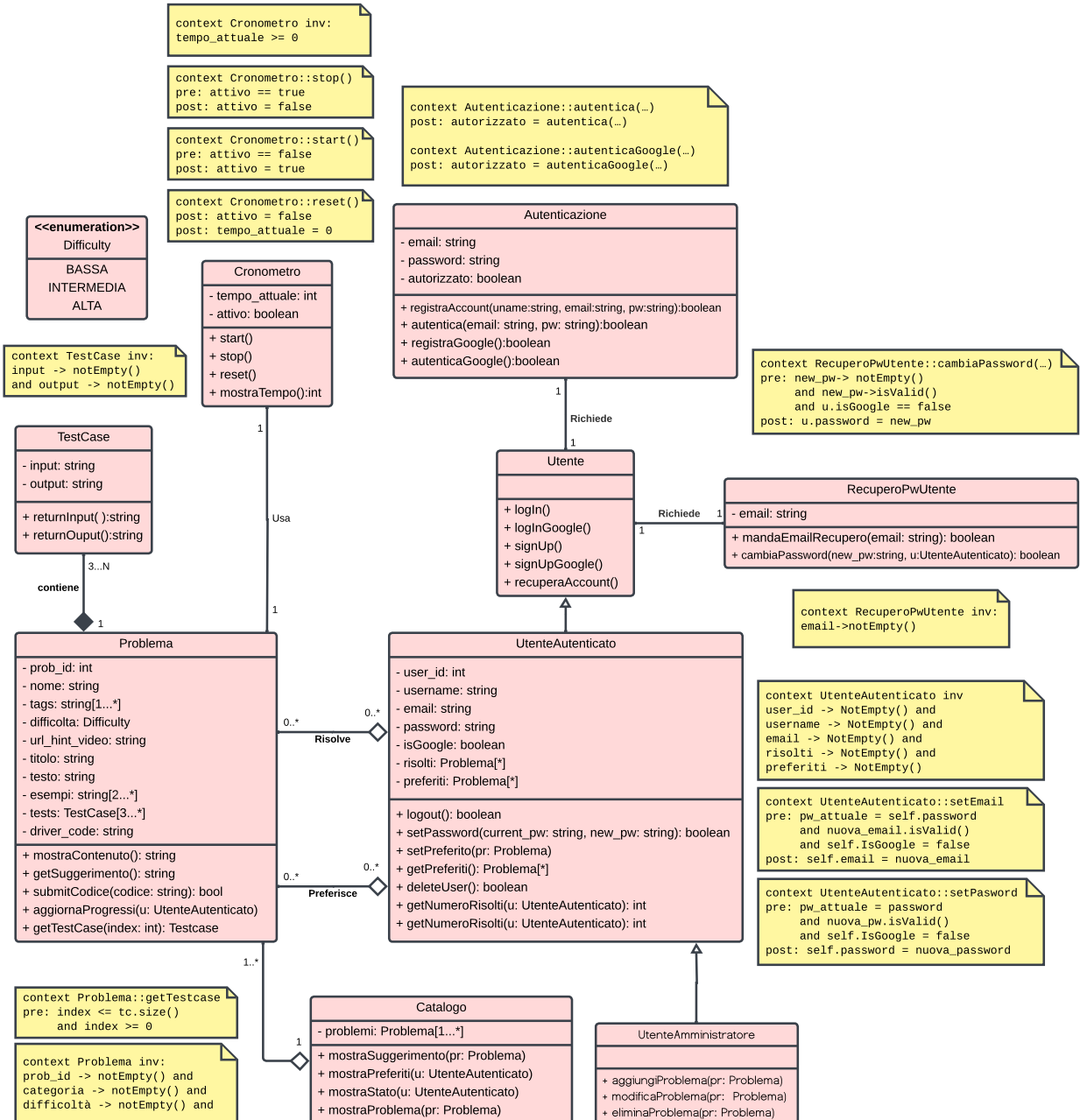


Figura 6: Diagramma delle classi insieme al codice OCL