Ingegneria del Software A.A. 2017/2018
Esame 2018-06-15

Esercizio 1 (6 punti)

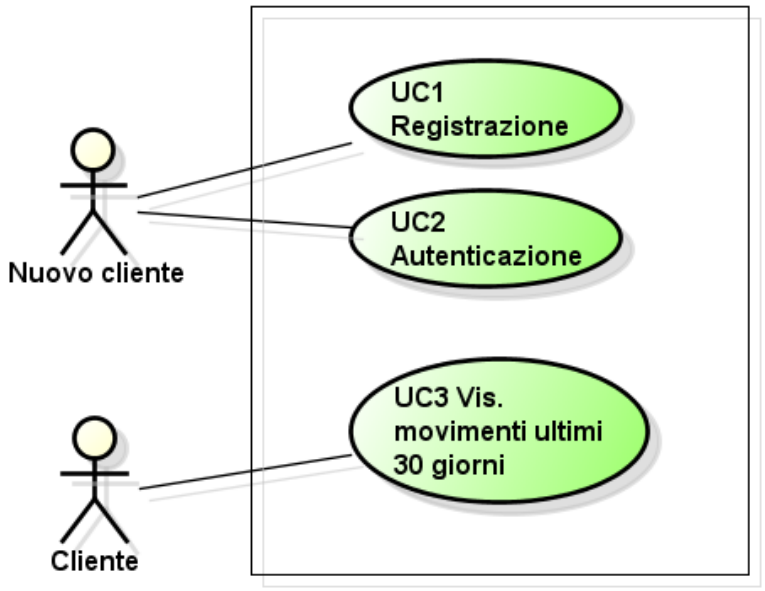
Descrizione

Un sempre maggior numero di banche decidono di fornire ai propri clienti la possibilità di aprire una nuova posizione (conto corrente) senza doversi recare in filiale. Tramite l'utilizzo di foto e videochiamate, è possibile eseguire un processo di riconoscimento sicuro. In particolare, un nuovo cliente, nella fase di registrazione, deve inserire d'apprima un email valida e il suo numero di cellulare. Successivamente, il sistema invia due codici distinti, uno all'email e uno al numero di cellulare, che l'utente deve inserire per confermare la propria identità. Successivamente, il sistema richiede di effettuare una foto al proprio tesserino sanitario e alla propria carta di identità. Da questi documenti, vengono desunte le generalità del nuovo cliente, tra le quali nome, cognome, codice fiscale e numero di documento di identità. Queste vengono visualizzate in una pagine riassuntiva. Una volta inserite le suddette informazioni, viene iniziata una video chiamata con un operatore, durante la quale l'operatore stesso effettua una verifica più approfondita dei dati inseriti in precedenza. Una volta che la video chiamata è terminata con successo, è possibile autenticarsi all'applicazione web delle banca, utilizzando le credenziali appena fornite dall'operatore. Essendo la posizione aperta una posizione bancaria valida a tutti gli effetti, il sito della banca permette di vedere i propri movimenti degli ultimi 30 giorni. Per ogni movimento di uscita nella lista, vengono visualizzati la data della valuta, il beneficiario e l'importo.

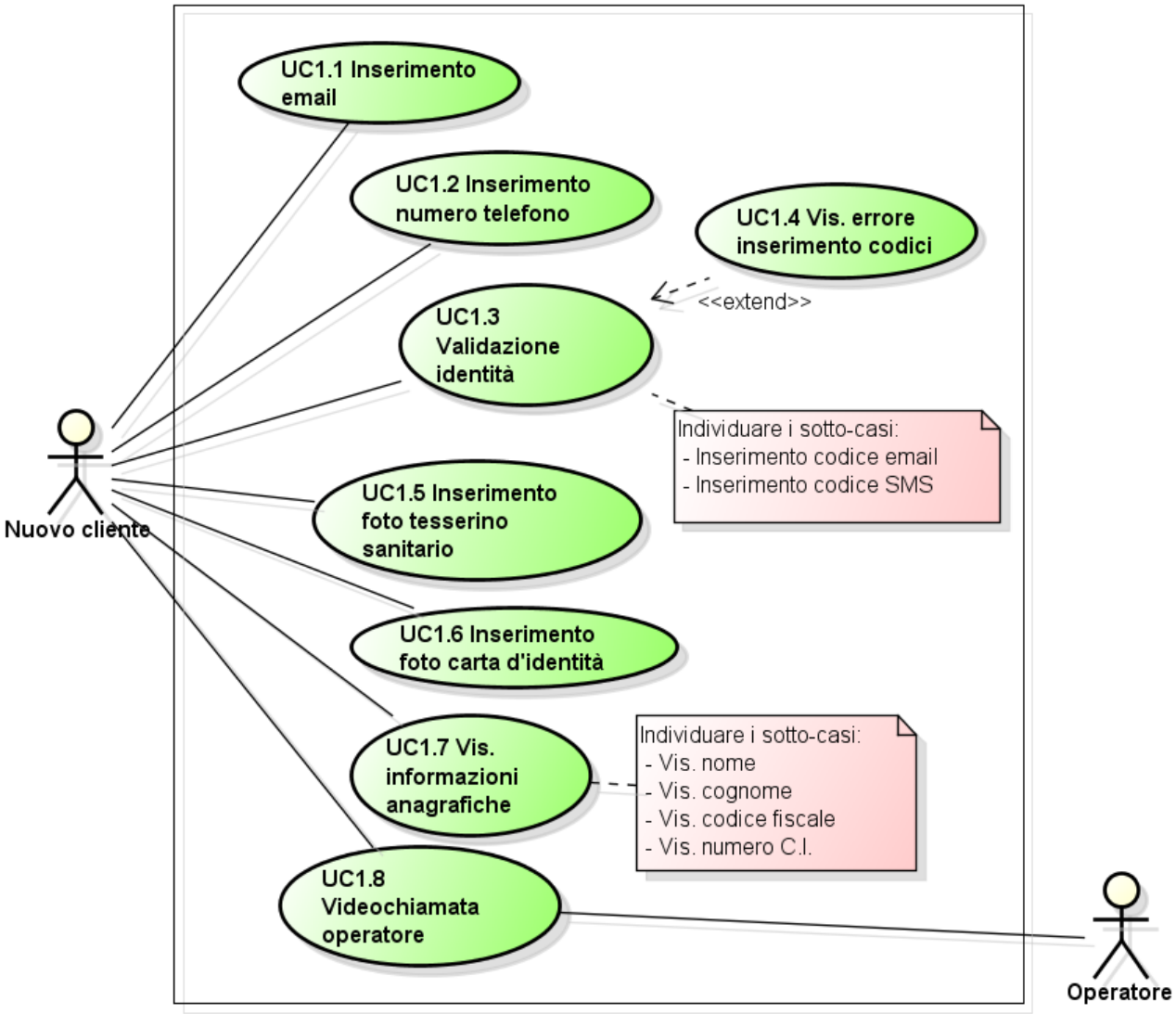
Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

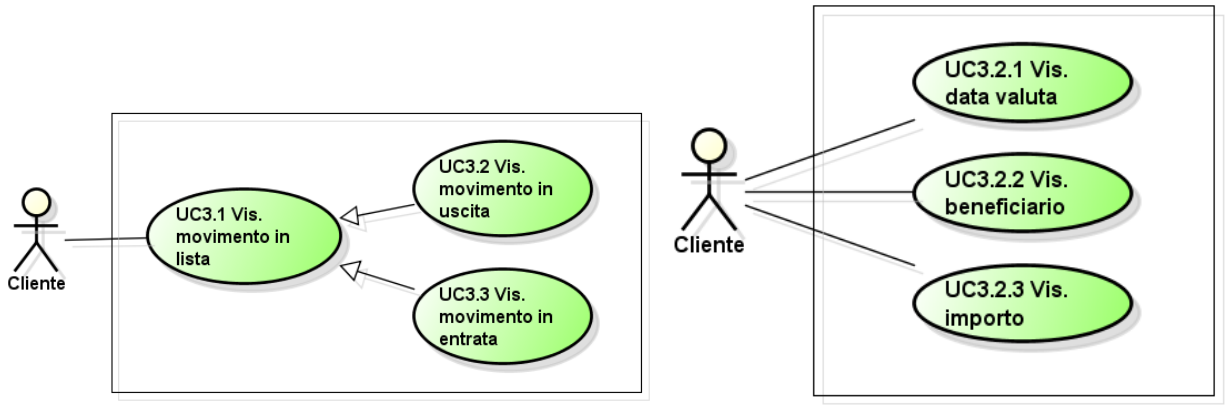
La soluzione parte con la differenziazione dei casi d'uso dei nuovi clienti e dei clienti già riconosciuti.



La registrazione può essere dettagliata come segue.



La visualizzazione dei movimenti in uscita si può modellare come segue.



Esercizio 2 (7 punti)

Descrizione

Spotify è una nota applicazione per l'ascolto di musica in streaming ed *offline*. I server di Spotify forniscono lo streaming musicale utilizzando una struttura remota a pacchetti (*chunk*). Una canzone è composta da più *chunk*. Ogniqualvolta un nuovo *chunk* è disponibile durante uno streaming audio, l'applicazione client connessa ne viene opportunamente informata. Un brano può essere riprodotto con qualità standard, pari a 192Kbps, oppore con qualità elevata, pari a 320Kbps. I chunk vengono man mano decodificati in da un algoritmo opportunamente scelto in istruzioni per la scheda audio. Spotify, però, fornisce anche la modalità di ascolto *offline*, durante la quale i brani vengono letti da filesystem. Seppur la libreria utilizzata per la lettura del file, utilizzi un semplice array di byte per rappresentare il file audio, l'applicazione adatta tale informazione alla modalità di lettura a *chunk.*

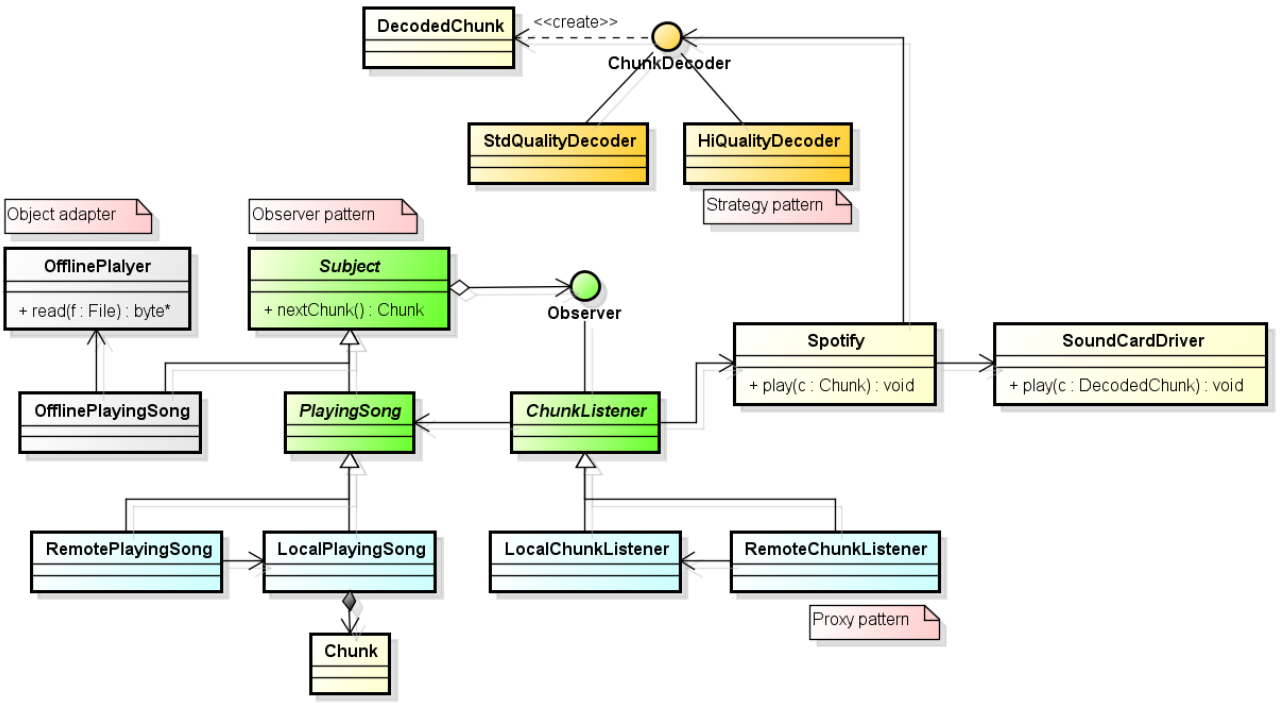
Si modelli tale sistema mediante un diagramma delle classi e i *design pattern* a esso pertinenti. Utilizzando un diagramma di sequenza, si descriva poi l'arrivo di un nuovo *chunk* remoto e il suo processamento da parte del sistema in istruzioni per la scheda audio.

Soluzione

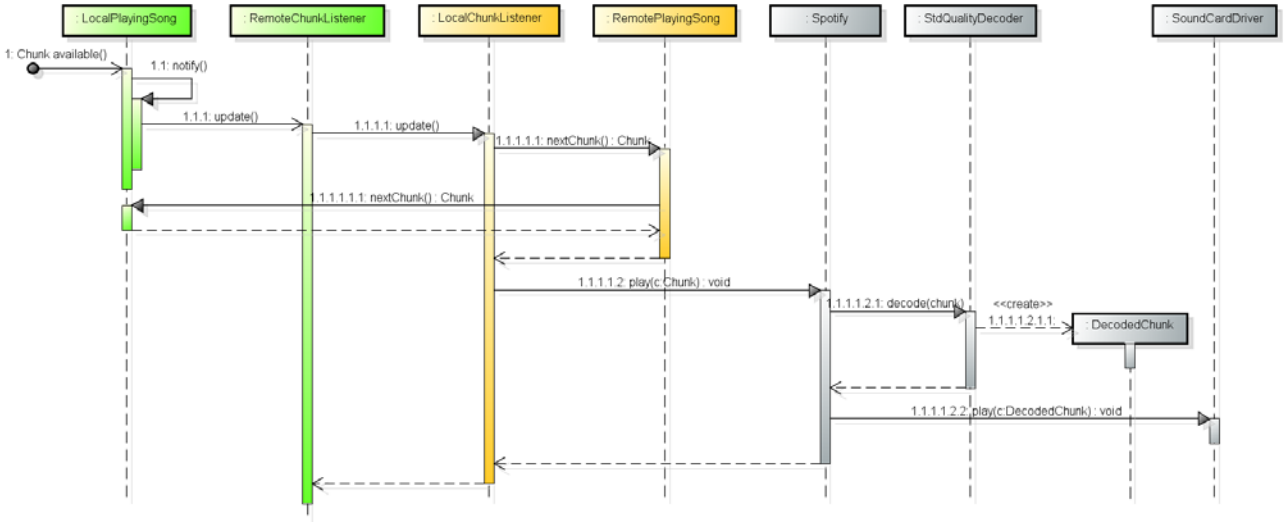
I design pattern da utilizzare sono i seguenti:

- Observer pattern
- Proxy pattern
- Object adapter pattern
- Strategy pattern

Il diagramma delle classi corrispondente è il seguente.



Il diagramma di sequenza è invece il seguente.



Esercizio 3 (3 punti)

Descrizione

Sia dato il seguente frammento di codice sorgente in Java.

```
public class Animal {
    private String animalType;
    public static void letsTalk(Animal[] animals) {
        for (Animal a: animals) {
            switch (a.getAnimalType) {
                case “Dog”:
                    ((Dog) a).bark();
                    break;
                case “Cat”:
                    ((Cat) a).mew();
                    break;
            }
        }
    }
    // Other methods…
}
public class Dog {
    // Code that properly initialize super animalType
```

```
    public void bark() { System.out.println(“Bark bark”); }
}
public class Cat {
    // Code that properly initialize super animalType
    public void miaow() { System.out.println(“Mew mew”); }
}
```

Si modifichi il suddetto codice al fine di fare aderire il metodo `letsTalk` al principio Open-Closed.

Soluzione

È necessario uniformare i metodi nelle classi concrete in una unica interfaccia `talk`, dichiarata in `Animal`. In questo modo, è possibile estendere il sistema aggiungendo nuovi animali, senza tuttavia modificare il codice esistente.

```
public abstract class Animal {
    public abstract void talk();
    public static void letsTalk(Animal[] animals) {
        for (Animal a: animals) {
            a.talk();
        }
    }
}
public class Dog extends Animal {
    @Override
    public void talk() { System.out.println(“Bark bark”); }
}
public class Cat extends Animal {
    @Override
    public void talk() { System.out.println(“Mew mew”); }
}
```