Ingegneria del Software A.A. 2016/2017 Esame 2017-06-27

Esercizio 1 (6 punti)

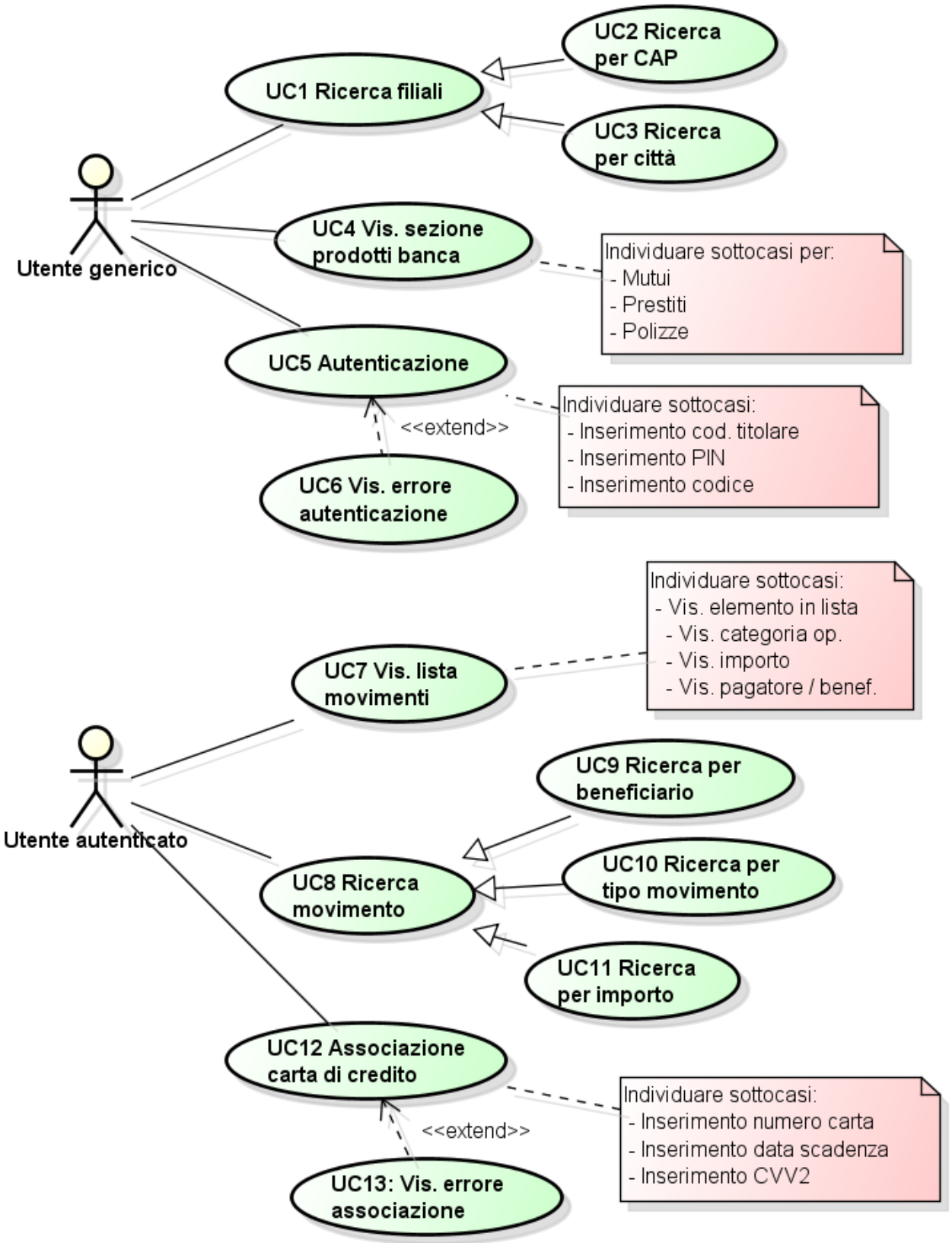
Descrizione

Il sito online di una banca fa parte dei progetti chiamati di “multicanalità”. Attraverso il sito è possibile sia recuperare delle informazioni generiche, sia effettuare delle operazioni dispositive. In particolare, per essere riconosciuto dalla banca, un utente deve inserire il proprio codice titolare, un PIN ed un codice random, solitamente generato *ad hoc* da un token fisico. All’interno della pagina a lui dedicata, un utente può quindi vedere la lista di movimenti effettuati sul suo conto negli ultimi 30 giorni. Per ogni movimento, nella lista sono riportati la categoria di operazione (pagamento stipendio / pensione, operazione acquisto alimentari, pagamento F24, ...) e l’importo. Se il movimento è un prelievo, viene quindi visualizzato il beneficiario, altrimenti viene visualizzato il pagatore. Per facilitare l’utenza nel consultare i propri movimenti, sono disponibili nel sito diversi tipi di ricerca: per beneficiario, per tipologia di movimentazione e per importo. All’interno del sito, un utente può inoltre associare una carta di credito, inserendo il numero di carta, la data di scadenza e il codice CVV2. Un errore informa l’utente se le informazioni inserite non sono corrette. Il sito della banca offre ovviamente anche funzioni per l’utenza generica. E’ infatti possibile ricercare le filiali della banca per CAP o Città. È inoltre disponibile una sezione nella quale vengono descritti i prodotti della banca, quali mutui, prestiti e polizze vita / danni.

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

I sottocasi individuati come commenti devono essere sempre sviluppati durante la prova d’esame.


UC2 Ricerca per CAP
UC1 Ricerca filiali
UC3 Ricerca per città
Utente generico
UC4 Vis. sezione prodotti banca
Individuare sottocasi per:
- Mutui
- Prestiti
- Polizze
UC5 Autenticazione
Individuare sottocasi:
- Inserimento cod. titolare
- Inserimento PIN
- Inserimento codice
<<extend>>
UC6 Vis. errore autenticazione
Individuare sottocasi:
- Vis. elemento in lista
- Vis. categoria op.
- Vis. importo
- Vis. pagatore / benef.
UC7 Vis. lista movimenti
Utente autenticato
UC9 Ricerca per beneficiario
UC8 Ricerca movimento
UC10 Ricerca per tipo movimento
UC11 Ricerca per importo
UC12 Associazione carta di credito
Individuare sottocasi:
- Inserimento numero carta
- Inserimento data scadenza
- Inserimento CVV2
<<extend>>
UC13: Vis. errore associazione

Esercizio 2 (7 punti)

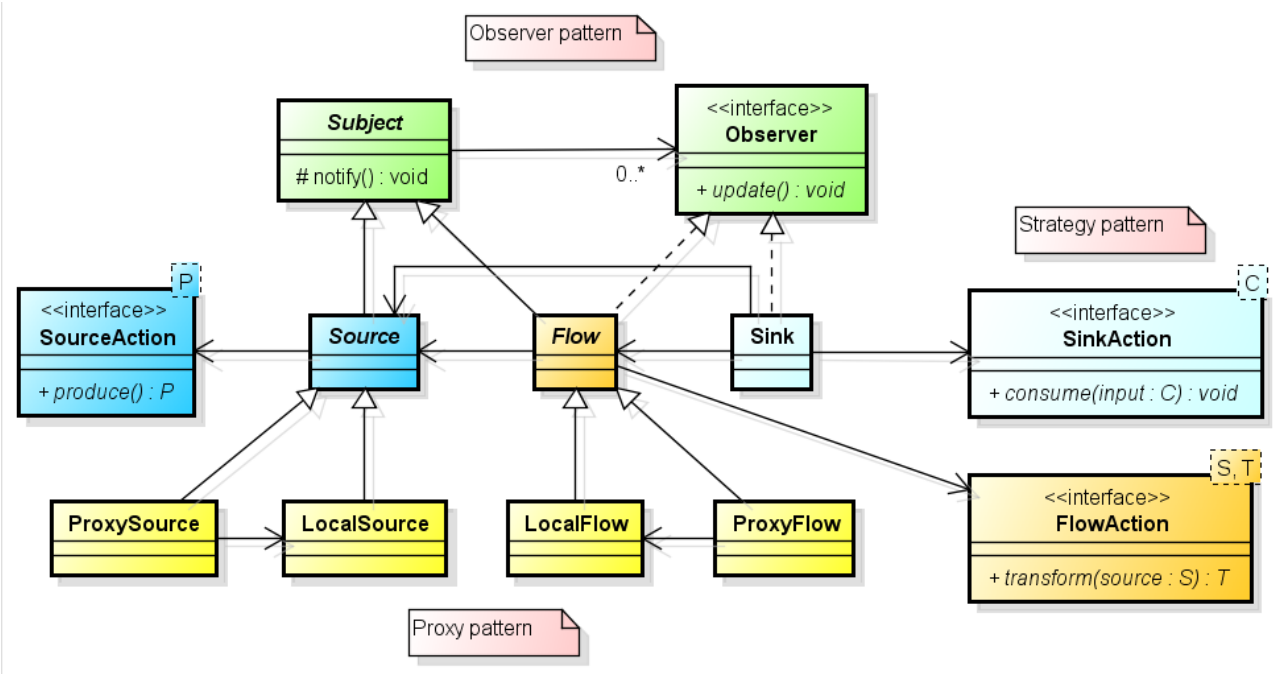
Descrizione

La programmazione moderna è sempre più orientata all'utilizzo di un modello ad eventi. Anziché controllare proattivamente se l'evento atteso sia disponibile, una risorsa viene informata di questo fatto. Questo tipo di programmazione è detta *reactive programming*. Le strutture che permettono di gestire gli eventi in modo appropriato vengono dette *stream*. Uno stream può essere di tre tipi differenti. Uno stream di tipo `Source` genera eventi. Uno stream di tipo `Sink` consuma eventi. Uno stream di tipo `Flow` consuma un evento e ne genera un altro. Anziché utilizzare un tipo dedicato per gli eventi, queste strutture sono generiche sui tipi. Quindi, un oggetto di tipo `Source` può essere osservato da oggetti di tipo `Sink` o `Flow`, mentre un oggetto di tipo `Flow` può essere osservato da oggetti di tipo `Sink`. `Source` e `Flow` possono ovviamente essere locali all'applicazione o remoti. Per mantere distinta la logica di controllo degli stream e l'effettiva implementazione della componente di business, ogni tipo di stream ha associato un tipo `*Action`. Abbiamo quindi `SouceAction`, `FlowAction` e `SinkAction`. Ognuno di questi tipi espone un unico opportuno metodo attraverso il quale è possibile implementare il comportamento dello stream.

Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza una catena di stream, che, leggendo un file di testo su un file system remoto, lo stampano su una console.

Soluzione

Design pattern utilizzati: Observer pattern, Proxy pattern, Strategy pattern.



Esercizio 3 (3 punti)

Descrizione

Siano prese in considerazione le seguenti classi, che permettono di copiare dei semplici caratteri inseriti da tastiera direttamente verso una stampante.

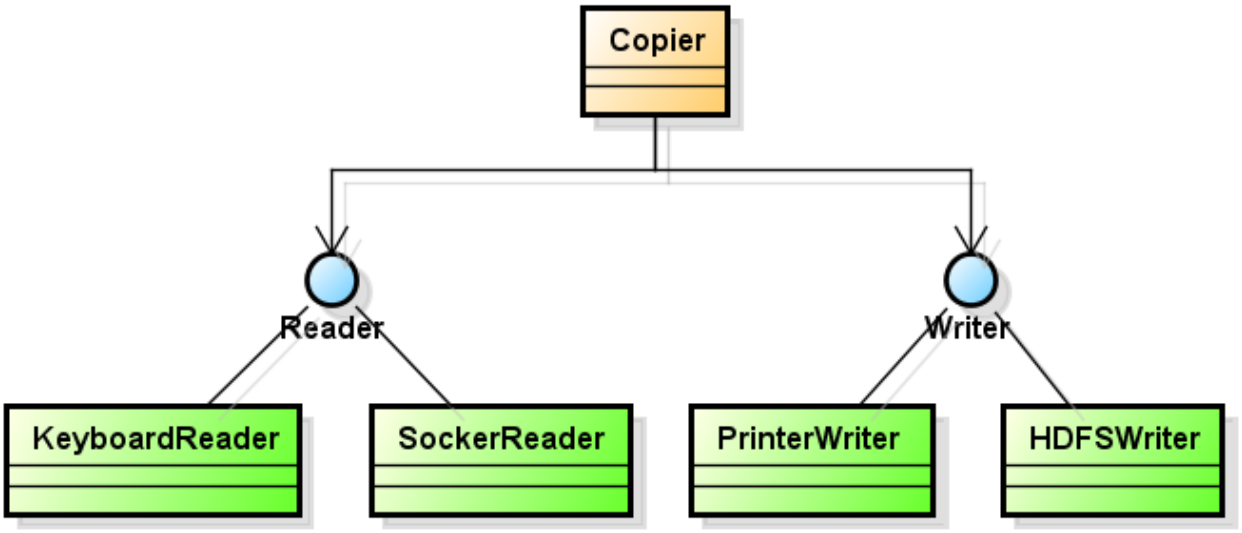
```
public class Copier {
    private KeyboardReader reader;
    private PrinterWriter writer;
    public Copier(KeyboardReader reader, PrinterWriter writer) {
         this.reader = reader;
         this.writer = writer;
    }
    public void copy(OutputDevice dev) {
        char c;
        while ((c = this.reader.read()) != EOF)
            if (dev == PRINTER)
                writer.write(c);
    }
}
public class KeyboardReader {
    public char read() {
        // Reads a character from the keyboard
    }
}
public class PrinterWriter {
    public void write(char c) {
        // Writes the character directly into the printer spool
    }
}
```

Si fornisca il diagramma delle classi di una soluzione che permetta di estendere le suddette classi in modo tale che la classe Copy possa essere riutilizzata per copiare caratteri da provenienti da un Socket direttamente su un file all'interno di un filesystem HDFS.

La soluzione dovrà tenere in considerazione il *Dependency Inversion Principle*.

Soluzione

Il livelli più alti dell'architettura non devono dipendere dalle implementazioni dei livelli più bassi. Copier ora dipendende solo da astrazioni, così come i *reader* e *writer* concreti.


Copier
Reader
Writer
KeyboardReader
SockerReader
PrinterWriter
HDFSWriter