DOCUMENTAZIONE PROGETTO BANCA PT.2 TPSIT

made by
Alvise Sacconato & Alessandro Zago

Classe `ContoBancario`

La classe rappresenta un conto bancario con funzionalità di gestione di un portafoglio, operazioni di deposito e prelievo, investimenti con vari tipi di durata e simulazione di guadagni o perdite. La classe gestisce anche l'avanzamento temporale mese per mese e il calcolo del saldo finale dell'investimento.

Attributi:

- **banca (`double`)**: Saldo del conto bancario.
- **portafoglio (`double`)**: Saldo disponibile nel portafoglio.
- **mese (`int`)**: Mese corrente del conto.
- **anno (`int`)**: Anno corrente del conto.
- **mesiInvestimento (`int`)**: Durata dell'investimento in mesi.
- **saldoFinale (`double`)**: Saldo finale dell'investimento.
- **variazione (`double`)**: Variazione di saldo dovuta all'investimento.
- **durataInvestimento (`boolean`)**: Indica se l'investimento è attivo o meno.
- **haGuadagnato (`boolean`)**: Indica se l'investimento ha generato guadagni.
- **rosso (`boolean`)**: Indica se l'investimento ha prodotto una perdita superiore all'importo investito.

Costruttore:

- **`ContoBancario()`**: Inizializza un oggetto e assegna i valori di default dell'oggetto:

Metodi:

- **getBanca() :**
  - Restituisce il saldo corrente in banca.
  - **Ritorna**: il saldo in banca.
- **setBanca(`double banca`) :**
  - Imposta il saldo del conto bancario.
  - **Parametri**:
    - `banca` (`double`): il nuovo saldo da impostare.

- **getPortafoglio()**:
  - Restituisce il saldo del portafoglio.
  - **Ritorna**: il saldo del portafoglio.
- **setPortafoglio(`double portafoglio`)**:
  - Imposta il saldo del portafoglio.
  - **Parametri**:
    - `portafoglio` (`double`): il nuovo saldo da impostare.
- **getMese()**:
  - Restituisce il mese corrente.
  - **Ritorna**: il mese corrente.
- **setMese(`int mese`)**:
  - Imposta il mese corrente.
  - **Parametri**:
    - `mese` (`int`): il nuovo mese da impostare.
- **getAnno()**:
  - Restituisce l'anno corrente.
  - **Ritorna**: l'anno corrente.
- **setAnno(`int anno`)**:
  - Imposta l'anno corrente.
  - **Parametri**:
    - `anno` (`int`): il nuovo anno da impostare.
- **getInvestimento()**:
  - Restituisce lo stato dell'investimento, se attivo o meno.
  - **Ritorna**: `true` se l'investimento è attivo, altrimenti `false`.
- **preleva(`double preleva`)**:
  - Preleva una somma dal conto bancario e la aggiunge al portafoglio.
  - **Parametri**:
    - `preleva` (`double`): l'importo da prelevare.
- **deposita(`double amount`)**:
  - Depone una somma dal portafoglio nel conto bancario.
  - **Parametri**:
    - `amount` (`double`): l'importo da depositare.
- **investe(`double importoInvestito, int investimentiMesi, Random random`)**:
  - Simula un investimento nel conto, con una durata specifica in mesi e un importo da investire. Il tipo di investimento viene determinato in base alla durata: breve, medio o lungo.
  - **Parametri**:
    - `importoInvestito` (`double`): l'importo da investire.
    - `investimentiMesi` (`int`): la durata dell'investimento in mesi.
    - `random` (`Random`): un oggetto per generare numeri casuali.
- **fineInvestimento()**:
  - Conclude l'investimento e aggiorna il saldo del conto.
- **nextMonth()**:
  - Avanza di un mese e aggiorna il portafoglio.

- **statoConto()**:
    - Restituisce lo stato corrente del conto, includendo saldo in banca e portafoglio, mese e anno.
    - **Ritorna**: una stringa che rappresenta lo stato attuale del conto.

Classe Utente

La classe `Utente` rappresenta un singolo utente nel sistema, con attributi per il nome, cognome, età, username, password e un oggetto `ContoBancario` per gestire le informazioni bancarie dell'utente.

Attributi:

- **Nome (`string`):** Una stringa che rappresenta il nome dell'utente.
- **Cognome (`string`):** Una stringa che rappresenta il cognome dell'utente.
- **eta (`int`):** Una stringa che rappresenta l'età dell'utente.
- **username (`string`):** Una stringa che rappresenta il nome utente.
- **password (`string`):** Una stringa che rappresenta la password dell'utente.
- **conto (`ContoBancario`):** Un oggetto che memorizza informazioni bancarie relative all'utente, come saldo del conto, portafoglio, mese e anno.

Costruttore:

- **Utente(`String Nome, String Cognome, String eta, String username, String password`):** Inizializza un oggetto `Utente` con il nome, cognome, età, username, password. Crea anche un nuovo oggetto `ContoBancario` associato all'utente.

Metodi:

- **getNome()**: Restituisce il nome dell'utente.
- **getCognome()**: Restituisce il cognome dell'utente.
- **getEta()**: Restituisce l'età dell'utente.
- **getUsername()**: Restituisce lo username dell'utente.
- **getPassword()**: Restituisce la password dell'utente.
- **getConto()**: Restituisce l'oggetto `ContoBancario` associato all'utente.
- **toCSV()**: Restituisce una stringa che rappresenta l'utente in formato CSV (nome, cognome, età, username, password, informazioni sul conto).
- **fromCSV(`String line`)**: Riceve una stringa in formato CSV e restituisce un oggetto `Utente` creato con i dati contenuti nella stringa. Se i dati sono incompleti o malformati, il metodo restituirà `null`.

Classe GestioneUtenti

La classe `GestioneUtenti` gestisce il caricamento e il salvataggio degli utenti, utilizzando un file CSV per memorizzare le informazioni. La classe consente di leggere gli utenti dal file, memorizzarli in una mappa, e successivamente salvare eventuali modifiche nel file.

Costanti:

- **`DIRECTORY_PATH`**: Percorso della cartella in cui si trova il file `utenti.csv`.
- **`FILE_PATH`**: Percorso completo del file CSV contenente gli utenti.

Metodi:

- **loadUtenti()**: Carica gli utenti dal file `utenti.csv`. Restituisce una mappa in cui le chiavi sono gli username e i valori sono gli oggetti `Utente`. Se il file non esiste, restituisce una mappa vuota.
    - **Restituisce**: Una mappa di tipo `Map<String, Utente>` contenente gli utenti caricati dal file.
- **saveUtenti(`Map<String, Utente> utenti`)**: Salva gli utenti nel file `utenti.csv`, sovrascrivendo il file esistente. Ogni utente viene serializzato in formato CSV e scritto nel file.
    - **Parametri**:
        - `utenti`: Una mappa di utenti da salvare nel file.

Classe Storico Transazioni

La classe `StoricoTransazioni` gestisce lo storico delle transazioni di ogni utente, salvando le transazioni in file di testo separati per ciascun utente e permettendo di visualizzare lo storico delle transazioni.

Costanti:

- **`DIRECTORY_PATH`**: La costante definisce il percorso della cartella in cui vengono salvati i file di storico delle transazioni per ciascun utente. La cartella si trova in `"File/Transazioni"`.

Metodi:

- **aggiungiTransazione(`String username, String transazione`):** Aggiunge una nuova transazione nel file di storico dell'utente specificato. Il metodo crea il file se non esiste già e aggiunge una riga con la data e l'ora della transazione. Il file di storico viene aggiornato in modalità append.
  - **Parametri**:
    - `username`: Il nome utente dell'utente per cui viene salvata la transazione.
    - `transazione`: Una stringa che descrive la transazione da aggiungere.
- **visualizzaStorico(`String username`):** Restituisce lo storico delle transazioni di un utente, leggendo il contenuto del file di storico associato. Se il file non esiste, restituisce un messaggio che indica che non ci sono transazioni per quell'utente.
  - **Parametri**:
    - `username`: Il nome utente per cui si vuole visualizzare lo storico delle transazioni.
  - **Restituisce**: Una stringa contenente lo storico delle transazioni dell'utente o un messaggio che indica che lo storico non è disponibile.

Classe `FileManager`

La classe `FileManager` fornisce metodi statici per la gestione di file e cartelle. Consente di creare cartelle, scrivere su file e leggere il contenuto di file di testo. È progettata per semplificare le operazioni di I/O (input/output) con file di testo, con particolare attenzione alla gestione degli errori.

Metodi:

- **creaCartella(`String DIRECTORY_PATH`)**:
  - Crea una cartella nel percorso specificato se non esiste già.
  - **Parametri**:
    - `DIRECTORY_PATH` (`String`): Il percorso della cartella da creare.
  - **Comportamento**:
    - Se la cartella non esiste nel percorso indicato, viene creata.
- **scriviSuFile(`String filePath, String contenuto, boolean append`)**:
  - Scrive il contenuto su un file. Se il file esiste, può aggiungere (append) o sovrascrivere il contenuto in base al parametro `append`.
  - **Parametri**:
    - `filePath` (`String`): Il percorso del file su cui scrivere.
    - `contenuto` (`String`): Il contenuto da scrivere nel file.
    - `append` (`boolean`): Se `true`, aggiunge il contenuto al file esistente; se `false`, sovrascrive il contenuto del file.
  - **Comportamento**:
    - Apre il file in modalità di scrittura. Se il file non esiste, viene creato. Se esiste e `append` è `true`, il contenuto viene aggiunto alla fine del file.
- **leggiDaFile(`String filePath`)**:
  - Legge il contenuto di un file e restituisce il contenuto come una stringa.
  - **Parametri**:
    - `filePath` (`String`): Il percorso del file da leggere.
  - **Ritorna**:
    - Una stringa che rappresenta il contenuto del file. Se il file non esiste o non è leggibile, restituisce una stringa vuota.
  - **Comportamento**:
    - Apre il file in modalità di lettura e legge riga per riga, concatenando il contenuto. Se il file non esiste, restituisce una stringa vuota.

Documentazione Classe `BankGUI`

La classe `BankGUI` è una GUI per la gestione di un'applicazione bancaria che permette agli utenti di eseguire operazioni bancarie come prelievi, depositi, investimenti, visualizzazione dello storico e avanzamento del mese. La GUI offre una schermata di login/registrazione e una schermata principale dove vengono visualizzati i dati utente e le azioni disponibili. La classe è basata su `JFrame` e utilizza il layout `CardLayout` per passare dalla vista di login alla vista principale.

Componenti Principali

- **Login Panel**: Un pannello con un `JTabbedPane` per la registrazione e il login dell'utente.
- **Main Panel**: Il pannello principale che visualizza le informazioni dell'utente e offre varie azioni tramite pulsanti.
- **CardLayout**: Usato per passare tra la schermata di login e quella principale.
- **Random**: Utilizzato per la generazione di numeri casuali, come per la gestione degli investimenti.

Struttura

Costruttore BankGUI()

Il costruttore inizializza la GUI, carica gli utenti, crea e configura i pannelli di login e principale, e imposta il layout della finestra.

Metodo createLoginPanel()

Questo metodo crea il pannello di login e registrazione, che include due tab: uno per il login e uno per la registrazione di un nuovo utente.

Funzionalità del login

- Verifica le credenziali dell'utente.
- Se le credenziali sono corrette, l'utente è autenticato e viene visualizzata la schermata principale.

Funzionalità della registrazione

- Permette all'utente di registrarsi fornendo nome, cognome, età, username e password.
- Se lo username è già presente, viene mostrato un messaggio di errore.
- Altrimenti, l'utente viene creato e salvato.

Metodo createMainPanel()

Crea il pannello principale che mostra le informazioni dell'utente e le opzioni per le azioni bancarie.

Funzionalità dei pulsanti

1. **Preleva dalla banca**: Consente all'utente di prelevare un importo dal proprio conto.
2. **Deposita in banca**: Consente all'utente di depositare un importo dal proprio portafoglio.
3. **Investi soldi**: Permette di investire una somma di denaro nel conto.
4. **Concludi investimento**: Conclude un investimento aperto.
5. **Avanza di un mese**: Avanza la simulazione di un mese nel conto dell'utente.
6. **Visualizza storico**: Visualizza le transazioni storiche dell'utente.
7. **Logout**: Permette all'utente di uscire dalla sessione e tornare al login.
8. **Esci**: Chiude l'applicazione.

Ogni bottone è associato a un `ActionListener` che gestisce l'evento corrispondente.

Metodo refreshUserInfo()

Aggiorna le informazioni utente mostrate nella parte superiore della finestra. Mostra il saldo del conto bancario, il saldo del portafoglio e lo stato degli investimenti.

Eccezioni e Gestione degli Errori

Durante le operazioni, vengono gestite le seguenti situazioni:

1. **Prelievo/Deposito**: Se l'utente cerca di prelevare o depositare una somma errata (es. valore negativo o superiore al saldo disponibile), viene visualizzato un messaggio di errore.
2. **Investimenti**: Se l'utente cerca di investire senza avere abbastanza fondi o inserisce un importo o una durata invalidi, viene mostrato un messaggio di errore.
3. **Login/Registrazione**: Se l'utente inserisce credenziali errate o un username già esistente, viene visualizzato un messaggio di errore.

Metodi Ausiliari

- **GestioneUtenti.loadUtenti()**: Carica gli utenti registrati dal file di dati.
- **GestioneUtenti.saveUtenti(`Map<String, Utente> utenti`)**: Salva gli utenti nel file di dati.
- **StoricoTransazioni.aggiungiTransazione(`String username, String transazione`)**: Aggiunge una transazione allo storico.
- **StoricoTransazioni.visualizzaStorico(`String username`)**: Visualizza lo storico delle transazioni dell'utente.

Dipendenze

La classe dipende dalle seguenti classi:

- `Utente`: La classe che rappresenta un utente con attributi come nome, cognome, username, password e conto bancario.
- `GestioneUtenti`: Gestisce l'archiviazione e il caricamento degli utenti.
- `Conto`: Gestisce le operazioni sul conto bancario dell'utente (prelievi, depositi, investimenti, ecc.).
- `StoricoTransazioni`: Gestisce lo storico delle transazioni dell'utente.