ESAME DI FONDAMENTI DI INFORMATICA T-2 del 14/6/2022
Proff. E. Denti – R. Calegari – A. Molesini
Tempo a disposizione: 3h30

NOME PROGETTO ECLIPSE: CognomeNome-matricola (es. RossiMario-0000123456)
NOME CARTELLA PROGETTO: CognomeNome-matricola (es. RossiMario-0000123456)
NOME ZIP DA CONSEGNARE: CognomeNome-matricola.zip (es. RossiMario-0000123456.zip)
NOME JAR DA CONSEGNARE: CognomeNome-matricola.jar (es. RossiMario-0000123456.jar)

Si devono consegnare DUE FILE: *l'intero progetto Eclipse* e *il JAR eseguibile*

Si ricorda che compiti *non compilabili* o *palesemente lontani da 18/30* NON SARANNO CORRETTI e causeranno la verbalizzazione del giudizio "RESPINTO"

È stato richiesto di sviluppare un'applicazione che consenta ai clienti di una banca di tenere sott'occhio il saldo, giorno per giorno, di un conto corrente bancario.

DESCRIZIONE DEL DOMINIO DEL PROBLEMA

Un ***conto corrente*** è uno strumento bancario per effettuare e ricevere pagamenti, nonché per tenere depositato denaro presso la banca stessa. L'operare del cliente sul conto è descritto da un insieme ordinato di ***movimenti,*** che possono essere di diverse ***tipologie***:

- **Accrediti**: rappresentano versamenti di denaro sul conto
- **Addebiti**: rappresentano prelievi di denaro sul conto o spese addebitate sul conto
- **Nulli**: rappresentano movimenti che coinvolgono (in addebito o accredito) un importo pari a zero
- **Saldo**: rappresentano l'ammontare di denaro esistente sul conto

Ogni movimento è caratterizzato da una serie di proprietà:

- **data contabile** in cui l'operazione viene registrata sul conto
- **data valuta** in cui i denari vengono effettivamente prelevati/versati sul conto
- **importo** del movimento stesso
- **causale** ossia una descrizione testuale del motivo del prelievo/versamento/operazione.

Logicamente, il saldo delle operazioni a una certa data D2 è pari al saldo in una precedente data D1 più tutti gli accrediti e meno tutti gli addebiti nel frattempo intervenuti.

Il file **Movimenti.txt** contiene l'elenco dei movimenti di un conto corrente**,** nel formato descritto più oltre.

TEMPO STIMATO PER SVOLGERE L'INTERO COMPITO: 2h15 – 3h

PARTE 1 – Modello dei dati: Punti 12 [TEMPO STIMATO: 60-75 minuti]

PARTE 2 – Persistenza: Punti 7 [TEMPO STIMATO: 30-45 minuti]

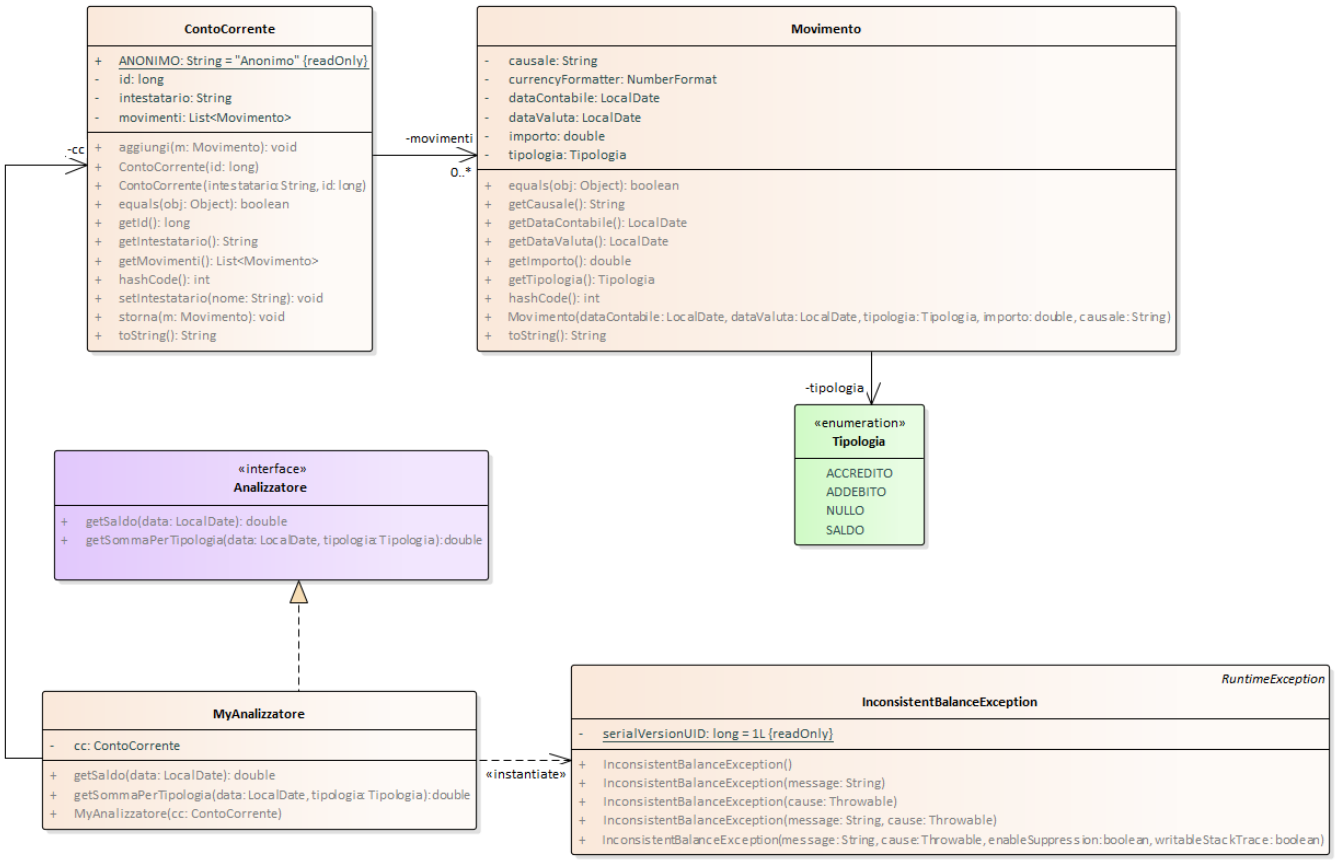
PARTE 3 – Grafica: Punti 11 [TEMPO STIMATO: 45-60 minuti]

JAVAFX – Parametri run configuration nei LAB

```
--module-path "C:\applicativi\moduli\javafx-sdk-17.0.2\lib"
--add-modules javafx.controls
```

Parte 1 – Modello dei dati (punti: 12)

Package: contocorrente.model **[TEMPO STIMATO: 60-75 minuti]**



SEMANTICA:

a) l'enumerativo ***Tipologia*** (fornito) elenca le quattro possibili tipologie di movimento

b) la classe ***Movimento* (da completare nel costruttore)** rappresenta un movimento come descritto nel Dominio del Problema: appositi accessor consentono di estrarre le proprietà rilevanti, e idonee *equals*, *hashCode* e *toString* completano l'implementazione. *Il costruttore deve verificare con cura gli argomenti ricevuti*, verificando in particolare che non siano nulli (per il valore double, che non sia *NaN*) e che l'importo – positivo, negativo o nullo – sia coerente con la tipologia di movimento specificata. Le formattazioni degli importi devono utilizzare apposito formatter con convenzioni culturali italiane.

c) la classe ***ContoCorrente* (da completare nel metodo aggiungi)** rappresenta il conto corrente, caratterizzato da *lista movimenti*, *intestatario* (una stringa) e *identificativo univoco* (un valore long). Anche in questo caso appositi accessor consentono di estrarre le proprietà rilevanti mentre idonee *equals*, *hashCode* e *toString* completano l'implementazione. Il conto corrente mantiene al suo interno la lista dei movimenti costantemente ordinata per data contabile crescente e in subordine per data valuta crescente: essa è manipolabile attraverso la coppia di metodi *aggiungi(Movimento) e storna(Movimento)* che rispettivamente inseriscono/rimuovono il movimento dato (se non nullo) dalla lista. In particolare, *il metodo aggiungi deve garantire che l'inserimento del movimento avvenga in modo da mantenere la lista movimenti costantemente ordinata come sopra specificato*.

d) l'interfaccia ***Analizzatore*** (fornita) rappresenta il componente che analizza ed elabora i movimenti, esponendo a tal fine i due metodi *getSaldo(LocalDate)*, che restituisce il saldo del conto alla data contabile specificata, e *getSommaPerTipologia(LocalDate, Tipologia)* che restituisce la somma dei movimenti della tipologia data alla data contabile specificata.

e) la classe ***MyAnalizzatore*** (**da completare implementando i due metodi**), il cui costruttore riceve il ***ContoCorrente*** su cui operare, implementa la precedente:

- il metodo *getSaldo* effettua la somma algebrica dei movimenti effettuati entro la data contabile specificata, *avendo cura però di escludere le righe relative ai saldi diversi dal saldo iniziale* (altrimenti, gli importi verrebbero sommati più volte e il totale sarebbe inconsistente); in presenza di un saldo intermedio deve invece verificare che la somma dei movimenti fino a quel momento coincida con quella dichiarata nel saldo stesso (a meno di € 0.01): ove così non sia, deve lanciare l'apposita ***InconsistentBalanceException*** (fornita)
- il metodo *getSommaPerTipologia* procede similmente, limitando tuttavia la somma ai soli movimenti della tipologia specificata, che può essere solo **ACCREDITO** o **ADDEBITO:** in caso contrario, dev'essere lanciata ***IllegalArgumentException***

Parte 2 – Persistenza (punti: 7)

Package: contocorrente.persistence **[TEMPO STIMATO: 30-45 minuti]**

La prima riga del file di testo specifica il numero del conto corrente, secondo il seguente formato:

- la sigla "CC", seguita da uno o più spazi
- il numero del conto nella forma "N.123455667", dove il valore numerico è un *long* ma occorre tenere in considerazione che potrebbero essere presenti spazi dopo "N."

Entrambe le sigle "N." e "CC" sono case-insensitive.

Le righe successive descrivono ciascuna un movimento, tramite quattro elementi separati da tabulazioni:

- la data contabile, nel formato italiano SHORT
- la data valuta, anch'essa nel formato italiano SHORT
- l'importo espresso come numero positivo o negativo, senza simbolo di valuta, ma con l'utilizzo dei separatori italiani per le migliaia e per la parte decimale; tuttavia, in questa banca l'eventuale segno "-" negli importi negativi potrebbe essere seguito da spazi prima del valore assoluto del numero
- la causale, intesa come stringa libera che può contenere spazi

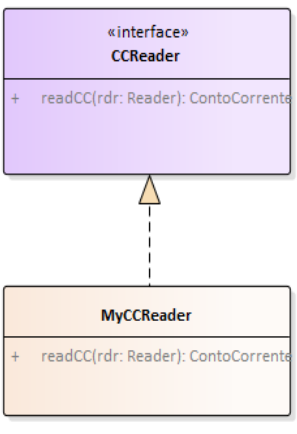
ESEMPIO:

```
CC N.123456789
31/01/22    31/01/22      5.317,81      SALDO INIZIALE A VOSTRO CREDITO
02/02/22    02/02/22      - 2,90        IMP.BOLLO CC
07/02/22    07/02/22      - 61,59       SPESE GESTIONE
09/02/22    08/02/22      - 29,14       PAGAMENTO POS CARBURANTI BOLOGNA
...
21/02/22    20/02/22      - 31,57       PAGAMENTO POS SUPERMARKET BOLOGNA
21/02/22    20/02/22      - 24,90       PAGAMENTO POS TECNOLOGIA FIRENZE
22/02/22    21/02/22      - 28,56       PAGAMENTO POS CARBURANTI BOLOGNA
25/02/22    25/02/22      2.144,58      RETRIBUZIONE
27/02/22    27/02/22      - 21,11       UTENZE
28/02/22    28/02/22      6.855,30      SALDO FINALE AL 28/02/2022
02/03/22    02/03/22      - 2,62        IMP.BOLLO CC
07/03/22    05/03/22      - 49,97       PAGAMENTO POS CARBURANTI MODENA
...
```

SEMANTICA:

a) L'eccezione ***BadFileFormatException*** (fornita) esprime l'idea di file formattato in modo scorretto

b) L'interfaccia ***CCReader*** (fornita) dichiara il metodo *readCC,* che legge da un *Reader* (ricevuto come argomento) i dati di un conto corrente, configurando e restituendo l'opportuno oggetto ***ContoCorrente***



c) La classe ***MyCCReader*** (**da realizzare**) implementa ***CCReader***: non prevede costruttori, si limita a implementare il metodo *readCC* come sopra specificato. In caso di problemi di I/O deve essere propagata l'opportuna ***IOException,*** mentre in caso di ***Reader*** nullo o altri problemi di formato dei file deve essere lanciata una opportuna ***BadFileFormatException***, il cui messaggio dettagli l'accaduto.
Per la prima riga, il reader deve verificare, lanciando ***BadFileFormatException*** in caso contrario, che:

   i. la riga contenga o due o tre elementi
   ii. il primo elemento sia "CC" (case-insensitive)
   iii. il secondo elemento inizi per "N." (case-insensitive) e, nel caso non sia presente un terzo elemento, contenga il numero del conto espresso come intero long
   iv. l'eventuale terzo elemento, se non già presente nel secondo, contenga il numero del conto come intero long

d) Per le righe successive, il reader deve verificare, sempre lanciando ***BadFileFormatException*** in caso contrario, che:

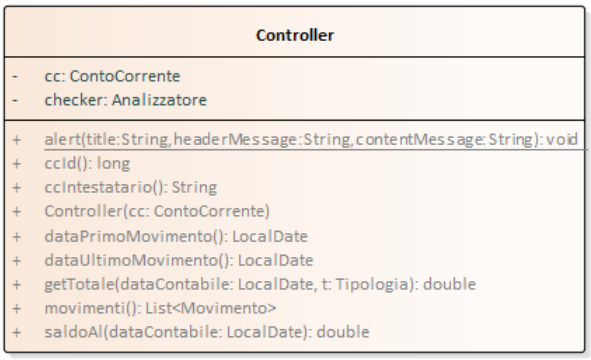
   i. ogni riga contenga esattamente quattro elementi (tab-separated)
   ii. le date siano espresse secondo le convenzioni italiane (formato SHORT)
   iii. l'importo segua le regole sopra descritte

   Sulla base dei dati letti deve essere costruito un nuovo oggetto ***Movimento***, avendo cura di attribuirgli la corretta ***Tipologia*** in base all'importo (positivo, negativo, nullo) e alla causale (per distinguere il saldo iniziale, da trattare come accredito, dagli altri saldi).

Parte 3 (punti: 11)

Package: contocorrente.controller *(punti 0)*

Il **Controller** (fornito) è organizzato secondo il diagramma UML nella figura seguente: esso lavora su un ***ContoCorrente*** ricevuto all'atto della costruzione ed utilizza internamente un ***Analizzatore*** per operare su esso.



SEMANTICA:

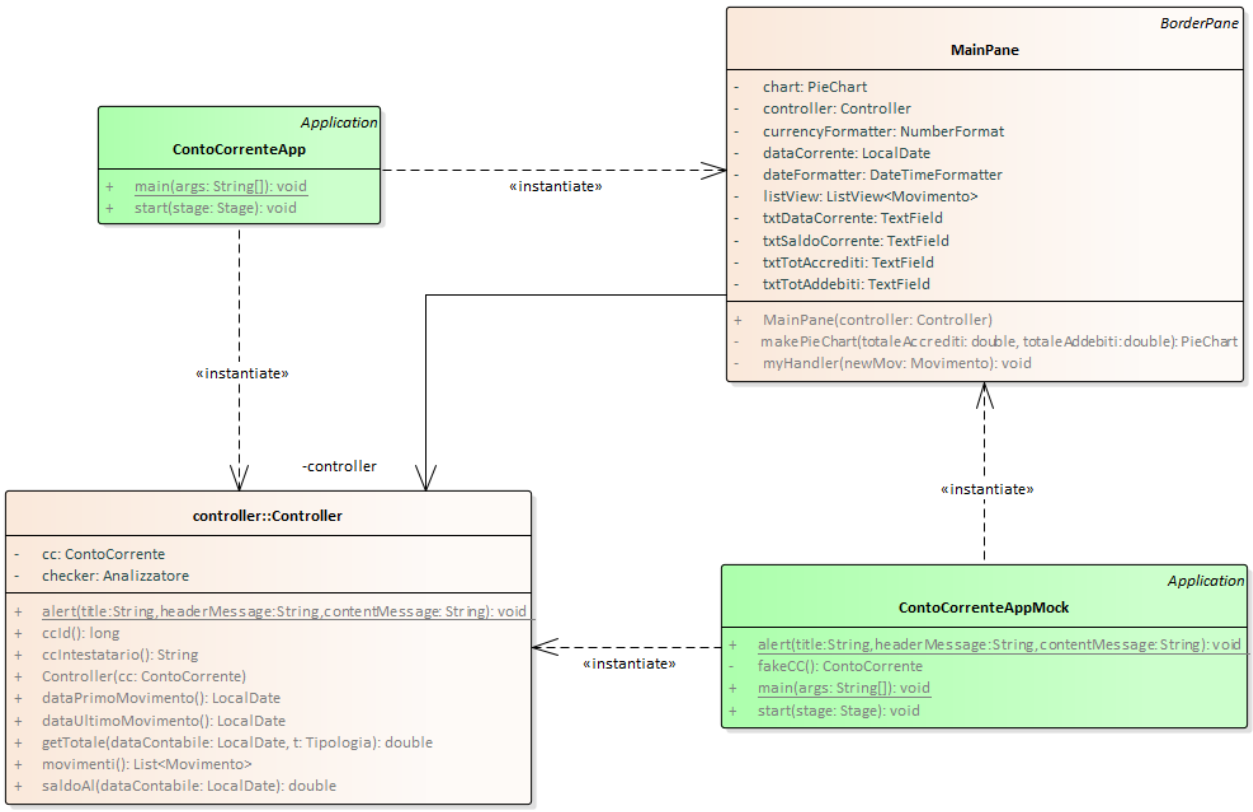
- il costruttore riceve il ***ContoCorrente*** su cui opererà e istanzia l'opportuno ***Analizzatore***;
- i metodi accessor, dall'ovvia denominazione, fanno da ponte verso gli omonimi metodi del model
- la classe contiene anche il **metodo statico ausiliario *alert***, utile per mostrare avvisi all'utente.

Package: contocorrente.ui **[TEMPO STIMATO: 45-60 minuti]** *(punti 11)*

La classe ***ContoCorrenteApp*** (fornita) costituisce l'applicazione JavaFX che si occupa di aprire i file, creare il controller e incorporare il ***MainPane***. Per consentire di collaudare la GUI anche in assenza / in caso di

malfunzionamento della parte di persistenza, è possibile avviare l'applicazione mediante la classe ***ContoCorrenteAppMock***.

L'interfaccia utente è illustrata nelle figure seguenti e segue il modello sotto illustrato*:*



L'interfaccia grafica si presenta come segue:

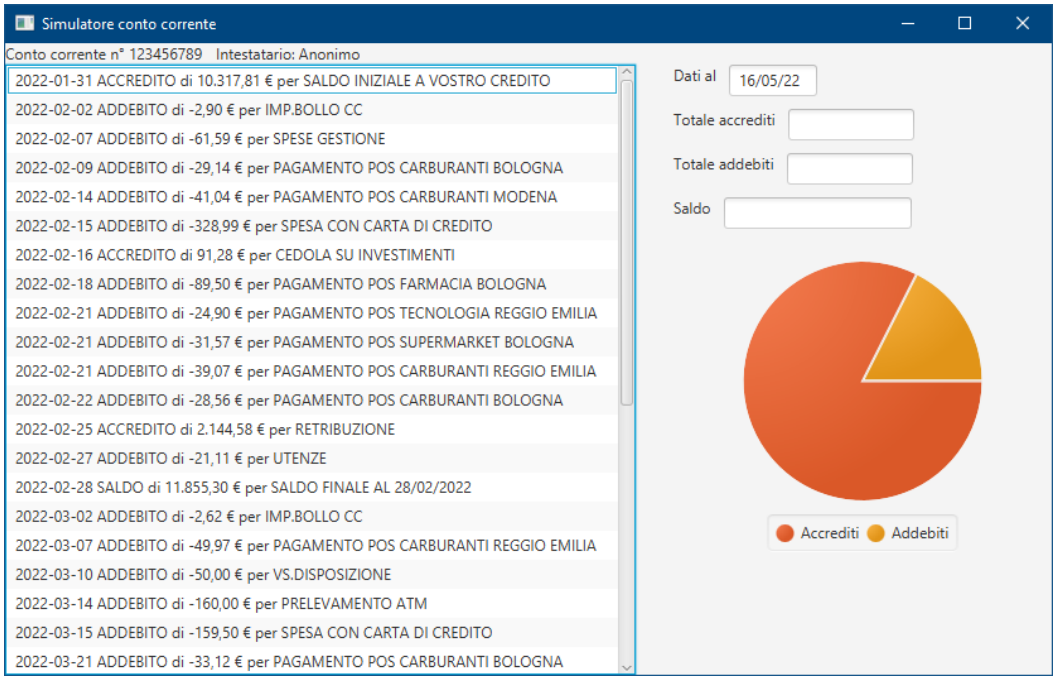
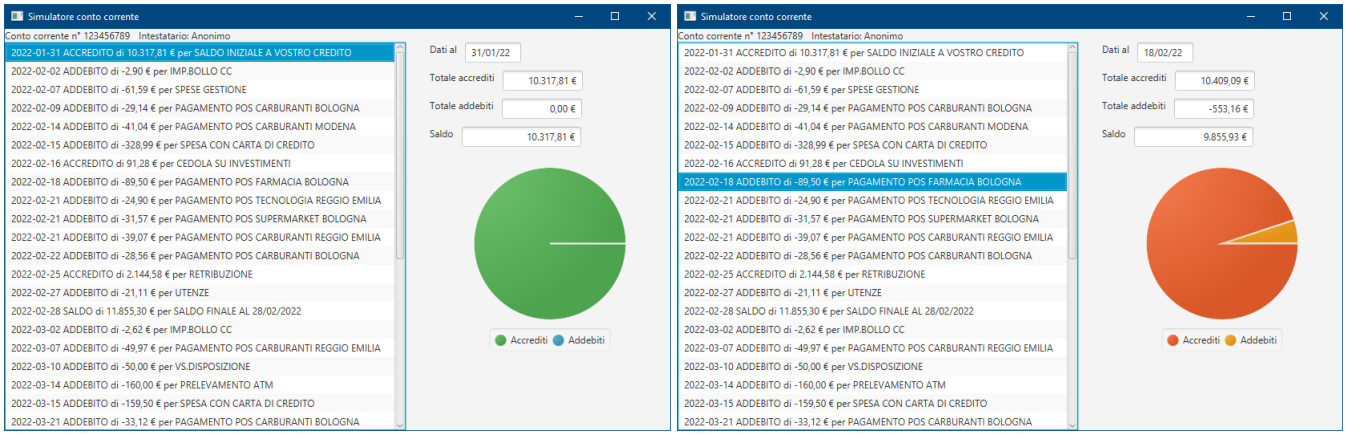


Fig. 1: la situazione iniziale della GUI, con data contabile impostata di default alla *data odierna*

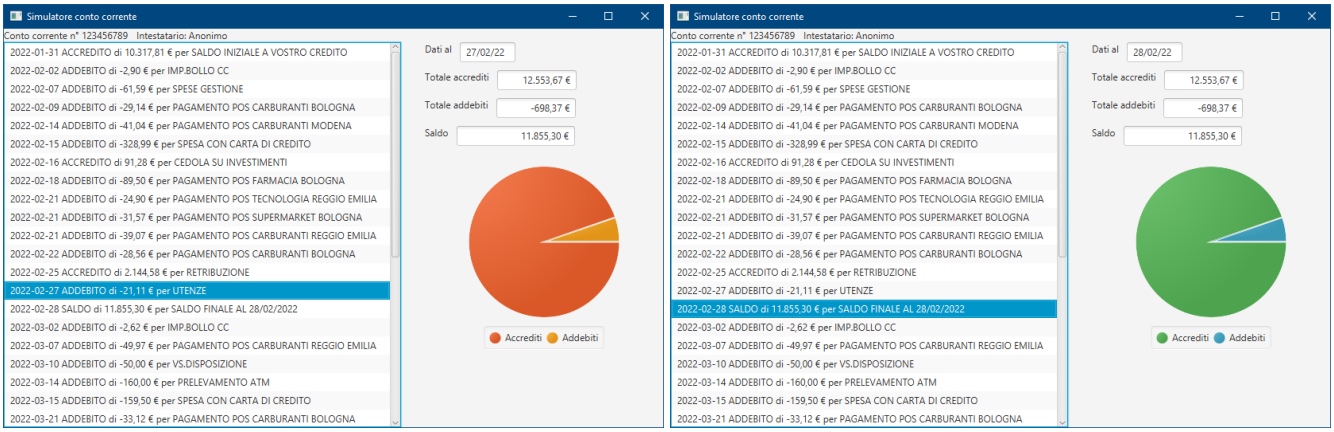
- in alto, una etichetta riporta numero e intestatario del conto corrente
- a sinistra, una **ListView** mostra i movimenti del conto corrente

- a destra, una serie di etichette e campi di testo (non editabili e posti verticalmente in una VBox) mostrano i dati del conto corrente a una certa data, formattata secondo le convenzioni italiane in stile SHORT. Sotto, un *grafico a torta* mostra il *rapporto fra accrediti e addebiti* alla data sopra indicata.

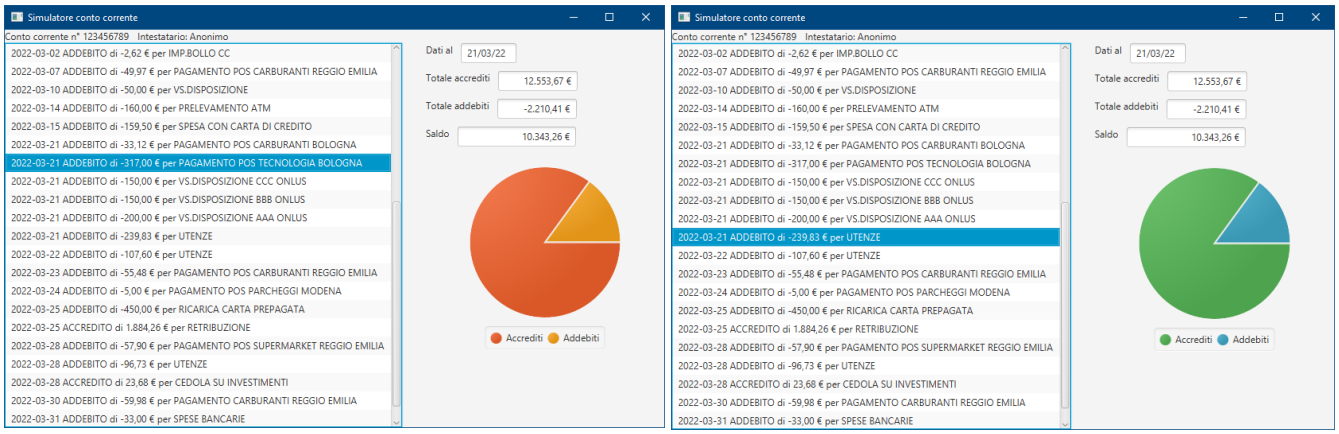
L'utente può interagire unicamente selezionando le varie righe sulla sinistra: immediatamente, la data sulla destra viene aggiornata alla data contabile del movimento selezionato e, subito a seguire, vengono di conseguenza aggiornati anche il saldo e i totali accrediti/addebiti, nonché il grafico a torta.



Figg. 2 / 3: la GUI dopo la selezione della prima riga (a sinistra) e di una riga successiva (a destra)



Figg. 4 / 5: il saldo e i totali non cambiano selezionando una riga di saldo intermedio, che viene però verificata



Figg. 6 / 7: nel caso di più movimenti nella stessa data contabile, i totali non variano, indipendentemente da quale riga si selezioni

Il MainPane è fornito *parzialmente realizzato*: è presente buona parte dell'impostazione strutturale, mentre sono da completare la configurazione di alcuni componenti e la gestione degli eventi.

La classe ***MainPane*** (**da completare**) estende ***BorderPane*** e prevede:

1) in alto, una ***HBox*** con dentro due etichette affiancate
2) a sinistra, una ***ListView*** di larghezza 500 e con righe impostate di adeguata altezza, per leggibilità
3) a destra, una ***VBox*** con le varie etichette e campi di testo (inserite a coppie in altrettante piccole ***HBox*** per garantire un gradevole allineamento) e, sotto, il grafico a torta.

La **parte da completare** riguarda:

1) la configurazione iniziale dei formattatori di valuta e data e della data corrente
2) l'aggancio alla ***ListView*** dell'opportuno listener incapsulato nel metodo ausiliario *myHandler*
3) la logica di gestione dell'evento, incapsulata nel metodo privato *myHandler*
4) la costruzione del grafico a torta, di dimensioni 250x250, demandata al metodo ausiliario *mkPieChart*

In particolare, la gestione dell'evento deve:

- recuperare dal movimento selezionato la data contabile da utilizzare, e impostarla nell'apposito campo di testo, opportunamente formattata
- utilizzare tale data per recuperare dal controller i vari dati (saldo, totale accrediti e addebiti) e mostrarli, opportunamente formattati, negli appositi campi di testo
- aggiornare il grafico a torta, ricalcolandolo coi nuovi totali accrediti e addebiti appena ottenuti

Cose da ricordare

- salva costantemente il tuo lavoro: l'informatica a volte può essere "subdolamente ostile"..
- in particolare: se ora compila e stai per fare modifiche, salva la versione attuale (non si sa mai)

Checklist di consegna

- Hai fatto un **JAR eseguibile**, che contenga cioè l'indicazione del main?
- Hai controllato che **si compili** e **ci sia tutto**? [NB: non includere il PDF del testo]
- Hai **rinominato** IL PROGETTO, lo ZIP e il JAR esattamente come richiesto?
- Hai **chiamato** la cartella del progetto esattamente come richiesto?
- Hai fatto un **unico file ZIP (NON .7z, rar o altri formati)** contenente l'intero progetto? In particolare, ti sei assicurato di aver incluso tutti i file .java (e non solo i .class)?
- **Hai consegnato DUE file distinti, ossia lo ZIP col progetto e il JAR eseguibile?**
- Su EOL, hai **premuto** il tasto "CONFERMA" per inviare il tuo elaborato?