Eccezioni

Laboratorio

Esercizio 1

- Modificare la classe **`BankAccount`** in modo che lanci un'eccezione quando viene istanziato un conto con saldo negativo, quando viene versata una somma negativa e quando si tenta di prelevare una somma non compresa tra 0 e il saldo del conto
- Definire tre eccezioni diverse una per ogni situazione descritta al punto precedente (una deve essere controllata e le altre non controllate)
- Scrivere un programma di test che prende in input a scelta dell'utente le operazioni da eseguire
- Il programma di test deve catturare e gestire una delle eccezioni non controllate e lasciare le altre due non gestite

Esercizio 2

- Modificare la classe **`DataSetReader`** in modo che non invochi **`hasNextInt`** né **`hasNextDouble`**. Semplicemente, lasciate che **`nextInt`** e **`nextDouble`** lancino un'eccezione di tipo **`InputMismatchException`** oppure di tipo **`NoSuchElementException`**, catturandola nel metodo **`main`**.

Esercizio 3

- Scrivere un programma che legga un insieme di descrizioni di monete da un file avente il formato seguente:

  `nomeMoneta1 valoreMoneta1`

  `nomeMoneta2 valoreMoneta2`

  ……

- Aggiungere alla classe **`Coin`** il metodo

  `void read(Scanner in) throws IOException`

  che lanci un'eccezione se la riga letta non ha il formato corretto. Quindi, realizzate il metodo

  `static ArrayList<Coin> readFile(String filename) throws IOException`

- Nel metodo **`main`** invocate **`readFile`**. Se viene lanciata un'eccezione, date all'utente la possibilità di selezionare un altro file. Se tutte le monete vengono lette correttamente, visualizzate il loro valore totale.

Esercizio 4

- Progettate una classe **`Bank`** che contenga un certo numero di conti bancari. Ciascun conto ha un numero di conto e un saldo. Aggiungete un campo **`accountNumber`** alla classe **`BankAccount`** e memorizzate i conti bancari in un vettore. Scrivete un metodo **`readFile`** per la classe **`Bank`** che legga un file scritto con il seguente formato:

  `numeroDiConto1 saldo1`

  `numeroDiConto2 saldo2`

  ……

- Realizzate metodi **`read`** nelle classi **`Bank`** e **`BankAccount`**. Scrivete un programma di prova che legga un file di conti correnti, per poi visualizzare il conto con il saldo maggiore. Se il file non contiene dati corretti, date all'utente la possibilità di selezionare un diverso file.

Esercizio 5

- Si definisca la classe "**utente**".
  - Ciascun "utente" ha nome, cognome, login e password.
  - La password e la login **devono essere lunghi almeno 5 caratteri e non possono contenere spazi**.
  - Definire quindi un costruttore per la classe utente che prenda in input i dati dell'utente stesso e lo crei.
  - **Se la password o la login non hanno il formato specificato il costruttore deve lanciare l'eccezione** predefinita IllegalArgumentException contenente il messaggio "Login dell'utente con formato errato" oppure "Password dell'utente con formato errato". Lo stessa tecnica si deve usare se uno dei campi in input è null oppure se ha lunghezza zero.
- Si definisca la classe **studente** estendendo la classe **utente.**
  - La classe studente è infatti identica alla classe utente ma ha in piu' un array bidimensionale e una variabile per memorizzare la matricola dello studente.
  - Per ogni riga *i* dell' array si avrà in posizione 0 il nome di un esame ed in posizione 1 il voto (memorizzato come stringa) che lo studente ha preso a quell'esame.
  - Il numero degli esami che lo studente ha sostenuto (e quindi anche un'indicazione dell'ultimo indice valido dell'array) sarà memorizzato nella variabile **numero_esami_superati**.
  - Per lo studente definire inoltre il metodo: **aggiungiEsame (String nome_Esame, String voto)** tale metodo aggiungerà l'esame all'array e aggiornerà il numero degli esami superati.

    Se il voto passato in input al metodo non rappresenta un intero tale metodo lancerà un eccezione del tipo (NumberFormatException). In realtà non lancerà tale eccezione ma la propagherà (utilizzare **Integer.parseInt**).
  - Per lo studente definire inoltre il metodo: **getMediaEsami()**

    Se lo studente non ha sostenuto esami restituire zero.

Esercizio 5 cont.

- Si definisca la classe **docente** estendendo la classe **utente.** La classe docente è infatti identica alla classe utente ma ha in piu' il numero dello studio del docente e il dipartimento di appartenenza.
- **Scrivere infine un programma di prova per collaudare le classi e i metodi.**
- Nel main prendere in input da tastiera i dati di una serie di studenti.
- Se vengono inseriti una login o una password aventi un formato numerico errato il main dovrà catturare l'eccezione e indicare all'utente che ha commesso un errore e che deve re-inserire lo studente.
- Dopo aver preso i dati generici il programma richiederà uno alla volta esame e voto di tutti gli esami che lo studente ha superato.
- Se viene passato in input un voto non numerico o minore di 18 il main dovrà catturare l'eccezione e indicare all'utente che ha commesso un errore e deve re-inserire l'esame errato.
- L'inserimento dovrà terminare se si inserisce un esame vuoto.
- Al termine dell'inserimento di ciascuno studente visualizzare a video le informazioni dello studente e gli esami superati con relativo voto.
- Creare inoltre un file "matricolastudente.txt" in cui si scrivono nella prima riga i dati dello studente e su ciascuna riga successiva una coppia esame – voto.
- Uno studente con matricola vuota indica che il programma deve terminare.
- Fare l'interfaccia utente in modo che l'utente del sistema sappia sempre cosa deve fare. Ad esempio indicargli che se vuole terminare il processo di inserimento studenti deve inserire una matricola vuota.
- Prima di terminare stampare a video un messaggio di saluto.

Esercizio 6

- Si definisca una classe “bilancio” le cui istanze conservano informazioni relative alla gestione di un bilancio. La classe deve fornire i seguenti costruttori e metodi:
  - un costruttore con argomento intero e un oggetto “utente” che rappresenta l'ammontare iniziale a disposizione e l’utente a cui si riferisce il bilancio;
  - un metodo void receiveFrom (int amount, String source) che registra una nuova entrata di amount ricevuta da source (ad esempio, "stipendio");
  - un metodo void spendFor (int amount, String reason) che registra una nuova uscita di amount spesa per reason (ad esempio, "affitto");
  - un metodo int cashOnHand () che restituisce il contante disponibile;
  - un metodo int totalReceivedFrom (String source) che restiuisce il totale delle entrate ricevute da source (0 di default);
  - un metodo int totalSpentFor (String reason) che restituisce il totale delle uscite spese per reason (0 di default).
  - un metodo che restituisca una sorta di “estratto conto”. Per ogni entrata o uscita in tale estratto conto deve esserci la source o reason e l’ammontare.
  - un metodo che scriva tutte le informazioni su un file (prima riga informazioni utente e righe successive entrate / uscite).
  - un metodo che legga tutte le informazioni da un file
- Continua…

Esercizio 6

- NB: le liste delle entrate e delle uscite possono essere implementate utilizzando la classe predefinita ArrayList e utilizzando una classe che rappresenti un entrata o un uscita. Notare che entrate e uscite sono costituite entrambe da una stringa e un importo quindi si può fare un'unica classe Movimento contenente al suo interno (oltre alla reason/source e all'importo) una stringa o un intero che permetta di distinguere le entrate dalle uscite.

- Si aggiunga un trattamento con eccezioni delle seguenti situazioni:
  1) passaggio come parametro di un ammontare negativo (per un entrata o un uscita) o non numerico;
  2) "bilancio in rosso" (cioe' si tenta di spendere piu' di quel che si ha).

- Si definisca infine una classe per testare FinancialHistory. In tale classe si richiede il nome di un file (in cui è contenuta il bilancio di un utente). Dopo la lettura del file il programma deve dare all'utente la possibilità di inserire nuove entrate/uscite, visualizzare i vari importi (totalReceivedFrom, totalSpentFor), l'estratto conto, salvare gli aggiornamenti su file, inserire il nome di un altro file (cambiando quindi conto), terminare.

Java.io

Esercizi

Esercizio 1 - IO

- Scrivere un programma che
  - prende in input il nome di un file e una parola chiave
  - da in output in un file tutte le righe che contengono la parola chiave

Esercizio 2 - IO

- Scrivere un programma capace di cifrare e decifrare file di testo. La cifratura avviene aggiungendo un intero k ad ogni byte del file. Il programma dovrà chiedere all'utente di inserire
  - 1) il nome del file di input
  - 2) il nome del file di output
  - 3) la chiave di cifratura k,
- effettuerà la cifratura del file di input salvando il risultato nel file di output.
- La decifratura corrisponde a cifrare con -k

Esercizio 3 - IO

- Scrivere un programma che
  - prende in input il nome di un file
  - modifica il file in modo che ogni riga è scritta al contrario (due esecuzioni di questo programma in cascata non modificano il file originale)
- Utilizzare file ad accesso casuale