# Esercizi Java.io

02/12/2021

# 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
- Modificare il programma precedente in maniera tale da effettuare la ricerca in una pagina web (il cui indirizzo è dato in input)

# 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

- Un sistema di messaggistica è in grado di ricevere messaggi SMS. Ogni messaggio ha un mittente, una data, un Identificativo (ID), un testo. Rappresentare mediante classi questo scenario implementando due metodi: getDate() e getSender.
- Realizzare un'agenda multimediale, in grado di raccogliere messaggi definiti in precedenza. Per l'agenda occorre implementare i metodi: addSMS che aggiunge un SMS all'agenda, getListByDate() getListBySender() che rispettivamente restituiscono la lista dei messaggi presenti ordinati per data o per mittente.
- Realizzare un metodo main che legge l'agenda multimediale da un file di oggetti, permette di aggiungere SMS, visualizzare gli SMS, salvare l'agenda nel file alla chiusura del programma.

# Esercizio 4

- 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 4

- 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.

# Esercizio 5

- I. la classe **Imbarcazione** che modella le tipologie di imbarcazioni che possono attraccare in un porto. Ogni imbarcazione è caratterizzato da targa, marca, lunghezza, anno di costruzione, data ormeggio, data partenza, e dalla variabile booleana cabinato. Corredare la classe con i metodi
  - check-in() che setta data ormeggio alla data attuale (se data ormeggio è già settata lancia l'eccezione RunTimeException).
  - check-out() che setta data partenza alla data attuale (se data partenza è già settata lancia l'eccezione RunTimeException, se data ormeggio non è settata lancia l'eccezione IllegalOperationException).
  - boolean cabinato() che restituisce true se l'imbarcazione è un cabinato.
- II. due sottoclassi
- 1) **EntroBordo** caratterizzata dalle variabili numeroCabine e altezza (in metri), e che fornisce il seguente metodo
  - int dammiCapienza() che restituisce il numero di persone che possono occupare l'imbarcazione ed è dato dalla forma (numeroCabine*2)+3.
- 2) **FuoriBordo** caratterizzata dalle variabili numeroCavalli, numeroPersone, e dalla variabile booleana gommone, e che fornisce il seguente metodo
  - double dammiPotenzaReale() che restituisce la potenza reale dell'imbarcazione in base alla seguente formula: numeroCavalli/numeroPersone. A tale valore bisogna aggiungere 50 nel caso in cui il fuoribordo è un gommone.
- III. Aggiungere un metodo *dammiCostoOrmeggio()* che restituisce il costo di ormeggio giornaliero di una Imbarcazione. In particolare, il costo di ormeggio giornaliero di un EntroBordo è dato da (altezza*capienza)*100, e il costo di ormeggio giornaliero di un fuoribordo è dato da (potenzaReale*lunghezza)*3.

# Esercizio 5

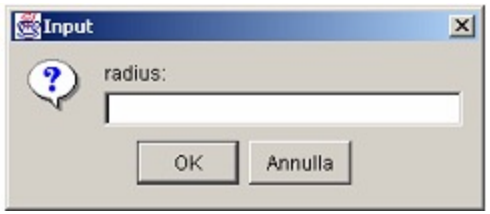
- Scrivere la classe **GestioneOrmeggi** che modella una collezione di imbarcazioni e fornisce i seguenti metodi:
  - void aggiungiImbarcazione(Imbarcazione i) che inserisce un imbarcazione nell'archivio.
  - double costoOrmeggioImbarcazione(int i) che restituisce la somma pagata dall'i-esima imbarcazione presente nell'archivio. Nel caso in cui l'imbarcazione non ha effettuato il check-in lanciare l'eccezione RunTimeException. Nel caso in cui l'imbarcazione non ha ancora effettuato il check-out restituire la somma che deve pagare fino alla data odierna.
  - double costoOrmeggi() che restituisce la somma incassata dalle imbarcazioni presenti nell'archivio che hanno effettuato il check-out.
  - void effettuaCheck-in(int i) che effettua il check-in sull'i-esima imbarcazione.
  - void effettuaCheck-out(int i) che effettua il check-out sull'i-esima imbarcazione.

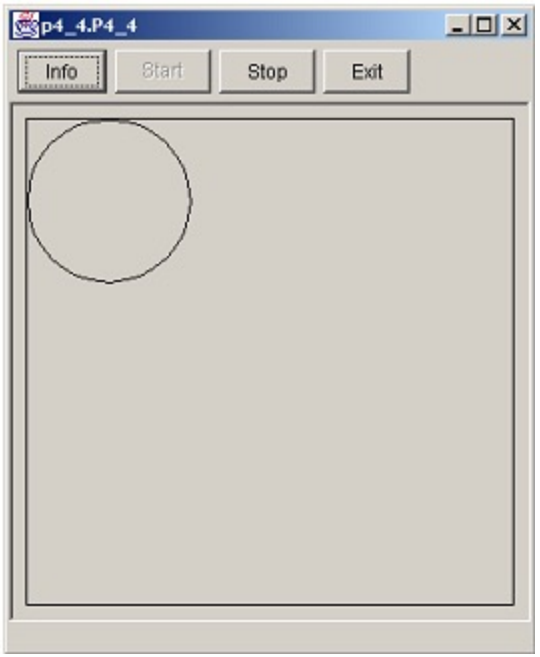
- Considerando le classi ai punti precedenti, scrivere un programma Java che realizzi un'interfaccia grafica per
  - caricare da un file una lista di imbarcazioni,
  - inserire nuove imbarcazioni,
  - stampare la lista delle imbarcazioni ed il totale incassato, e
  - salvare in un file la lista di imbarcazioni.

# Programmazione Grafica

# Esercizio 1 – Progr. Grafica

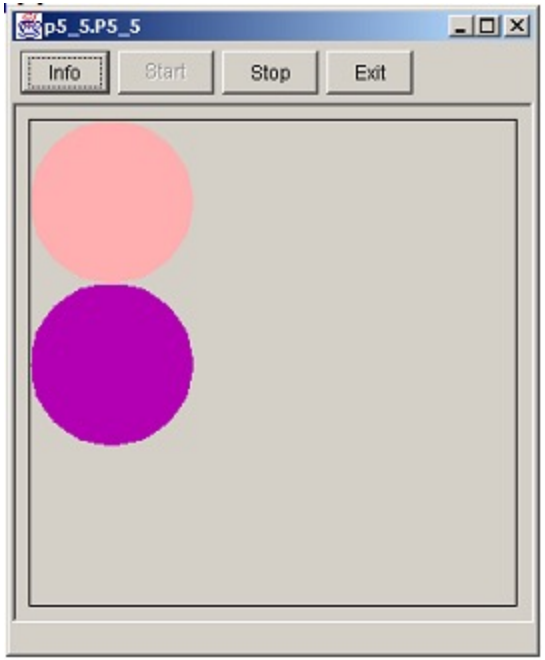
- Scrivere un'applicazione che chieda all'utente di inserire la misura di un raggio e tracci una circonferenza con quel raggio



# Esercizio 2 – Progr. Grafica

- Scrivere un'applicazione che disegni due cerchi pieni, uno rosa e uno viola. Usate un colore standard per uno e uno personalizzato per l'altro



# Esercizio 3 – Progr. Grafica

- Scrivere un'applicazione che disegni una grande ellisse, che
  - tocchi i bordi della finestra dell'applet
  - sia riempita con il vostro colore preferito
  - si ridimensioni automaticamente quando si ridimensiona la finestra
- Utilizzare i metodi getWidth()  e getHeight() per calcolare le dimensioni della finestra