Corso di Metodologie di Programmazione

Raccolta di esercizi proposti a lezione
(Credits: Prof. Roberto Navigli, Prof. Stefano Faralli)

Versione: February 2023

1 Argomento 1 - Introduzione

Esercizio L.1.1 Quanto fa $5 - 2 - 3/2.0 + 2 * 2 - 5\%2/2.0$?

Esercizio L.1.2 Si riscriva l'espressione bBisestile utilizzando un'unica istruzione di assegnazione.

```
public class AnnoBisetile
{
 public static void main(String[] args)
 {
   int anno=Integer.parseInt(args[0]);
   boolean bBisestile;
   bBisestile= anno % 4 == 0;
   bBisestile= bBisestile && ( anno % 100 != 0);
   bBisestile= bBisestile ||  (anno % 400 ==0);

   System.out.println("L'anno "+anno+" e' bisestile? "+bBisestile);
 }
}
```

Esercizio L.1.3 Siano date **a** e **b** variabili di tipo double; si scriva l'istruzione che definisce **c** pari al valore true se **a** è maggiore di **b**, false altrimenti.

Esercizio L.1.4 Siano date una variabile intera **a** e una variabile double **b**; si scriva l'istruzione che assegna a **c** la parte intera della differenza tra **a** e **b**.

Esercizio L.1.5 Qual è il valore di x dopo l'esecuzione di quanto segue?

```
int b=10;
double a=5;
int x = (int)a/b + 5;
```

Esercizio L.1.6 Scrivere una classe Moltiplica che, dati in input 2 numeri interi, ne restituisca a video il prodotto.

Esercizio L.1.7 Scrivere una classe StampaNome che, dato in input un nome, lo stampi tra due righe di trattini.
Ad es.:

```
+--------------------+
Pippo
+--------------------+
```

Esercizio L.1.8 Scrivere una classe Variabili che, all'interno del metodo main, dichiari una variabile intera i, una variabile di tipo stringa s e una variabile double d. Quindi vengono svolte le seguenti tre operazioni:

La stringa viene inizializzata al valore del primo argomento fornito in input al main. All'intero viene assegnato il valore intero della stringa. Al double viene assegnata la metà del valore di i (ad es. se i è pari a 3, d sarà pari a 1.5). I valori di s, i e d vengono stampati a video.

Esercizio L.1.9 Progettare una classe i cui oggetti contengono tre elenchi di parole l1, l2 e l3 La classe è in grado di emettere nuove espressioni costruite creando stringhe del tipo "a b c" scegliendo casualmente dai tre rispettivi elenchi a ∈ l1, b ∈ l2, c ∈ l3.

Ad esempio, dati i seguenti elenchi: l1 = { "salve", "ciao", "hello", "buongiorno", "scialla" }
l2 = { "egregio", "eclettico", "intelligentissimo", "astutissimo" }
l3 = { "studente", "ragazzo", "giovane", "scapestrato", "fannullone", "studioso" }

Esempi di output sono:
"salve egregio fannullone"
"ciao eclettico scapestrato"

2 Argomento 2 - Classi e Oggetti

Esercizio L.2.1 Progettare una classe Persona i cui oggetti rappresentano una persona e ne memorizzano il nome e il cognome. La classe espone un metodo main che crea un'istanza della Persona. La classe espone anche un metodo stampa che visualizza nome e cognome della persona.

Esercizio L.2.2 Progettare una classe Quadrato, i cui oggetti sono costruiti con il lato dello stesso. La classe è dotata di un metodo getPerimetro che restituisce il perimetro. E di un metodo main che crea un quadrato di lato 4 e ne stampa a video il valore del perimetro.

Esercizio L.2.3 Progettare una classe Cerchio i cui oggetti rappresentano un cerchio. La classe è dotata dei metodi getCirconferenza e getArea (si usi la costante Math.PI). La classe espone anche un metodo main che crea due cerchi (di raggio 1 e di raggio 5) e ne stampa la circonferenza (per il primo) e l'area

(per il secondo).

Esercizio L.2.3 Progettare una classe BarraDiCompletamento i cui oggetti rappresentano una barra di caricamento. Gli oggetti vengono costruiti con la percentuale di partenza. La classe espone un metodo incrementa che, data una percentuale in input, incrementa la percentuale di partenza con quella fornita in input (ad es. new BarraDiCompletamento(5).incrementa(10) porta la barra al 15%). Il metodo toString dell'oggetto restituisce una stringa contenente la percentuale di completamento arrotondata con Math.round(). Il metodo main che crea una barra di completamento che parte da 0, la incrementa prima di 20 punti percentuale e poi di altri 25 e quindi stampa la rappresentazione stringa della barra.

Esercizio L.2.4 Progettare una classe Rettangolo i cui oggetti rappresentano un rettangolo e sono costruiti a partire dalle coordinate x, y e dalla lunghezza e altezza del rettangolo. La classe implementa i seguenti metodi:
– trasla che, dati in input due valori x e y, trasla le coordinate del rettangolo dei valori orizzontali e verticali corrispondenti.
– toString che restituisce una stringa del tipo "(x1, y1)-¿(x2, y2)" con i punti degli angoli in alto a sinistra e in basso a destra del rettangolo.
Implementare una classe di test TestRettangolo che verifichi il funzionamento della classe Rettangolo sul rettangolo in posizione (0, 0) e di lunghezza 20 e altezza 10, traslandolo di 10 verso destra e 5 in basso.

Esercizio L.2.5 Progettare una classe Colore i cui oggetti rappresentano un colore in modalità RGB e che sono costruiti a partire da tre valori: R (rosso), G (verde), B (blu), ognuno dei quali ammette un valore intero nell'intervallo 0-255.

La classe Colore espone anche due costanti BIANCO e NERO.

Fare in modo che ogni Rettangolo (vedere Esercizio L.precedente) abbia associato un Colore di base NERO e che sia possibile impostare il colore di un rettangolo mediante un apposito metodo.

Esercizio L.2.6 Progettare una classe Programmatore i cui oggetti rappresentano persone che svolgono il lavoro di sviluppo presso un'azienda La classe implementa i seguenti metodi:
- un costruttore con il nome e cognome della persona.
- un metodo setAzienda che imposta il nome dell'azienda per cui la persona lavora.
- un metodo addLinguaggio che aggiunge un linguaggio di programmazione a quelli inizialmente usati dal programmatore.
- i metodi getNome, getCognome e getAzienda che restituiscono i valori corrispondenti.
- un metodo getLinguaggi che restituisce la stringa dei linguaggi (separati da spazio) noti al programmatore.

Implementare un main all'interno della stessa classe in modo che il seguente codice funzioni correttamente:

```
Programmatore p1 = new Programmatore("Bjarne","Stroustrup");
Programmatore p2 = new Programmatore("Brian","Kernighan");
Programmatore p3 = new Programmatore("James","Gosling");

p1.addLinguaggio("C");
p1.addLinguaggio("C++");
p1.setAzienda("Morgan Stanley");

p2.addLinguaggio("C");
p2.addLinguaggio("AWK");

p3.addLinguaggio("Java");
p3.addLinguaggio("Oracle");

// Stampa Morgan Stanley
System.out.println(p1.getAzienda());

//Stampa C AWK
System.out.println(p2.getLinguaggi());
```

Esercizio L.2.7 Progettare una classe che costituisca un modello di registratore di cassa.
La classe deve consentire a un cassiere di inserire i prezzi degli articoli e, data la quantità di denaro pagata dal cliente, calcolare il resto dovuto.
Quale stato (campi) dovete prevedere?
Quali metodi vi servono?
- Registra il prezzo di vendita per un articolo. - Conclude la transazione, calcola il resto dovuto al cliente nel momento in cui effettua il pagamento e restituendolo in uscita.

Esercizio L.2.8 Progettare una classe Punto per la rappresentazione di un punto nello spazio tridimensionale.
E una classe Segmento per rappresentare un segmento nello spazio tridimensionale.
Scrivere una classe di test che crei:
due oggetti della classe Punto con coordinate (1, 3, 8) e (4, 4, 7).
un oggetto della classe Segmento che rappresenti il segmento che unisce i due punti di cui sopra.

Esercizio L.2.9 Progettare una classe Punto per la rappresentazione di un punto nello spazio tridimensionale.

E una classe Segmento per rappresentare un segmento nello spazio tridimensionale.
Scrivere una classe di test che crei:
- due oggetti della classe Punto con coordinate (1, 3, 8) e (4, 4, 7)
- un oggetto della classe Segmento che rappresenti il segmento che unisce i due punti di cui sopra.

Raffigurare l'evoluzione dello stato della memoria, distinguendo tra stack , heap e metaspace.

3 Argomento 3 - Istruzioni di Controllo e Array

Esercizio L.3.1 Scrivere un metodo che, presi in ingresso un testo sotto forma di stringa e una parola w, trasformi il testo in parole (token) e conti le occorrenze di w nel testo.

Esercizio L.3.2 Scrivere un metodo che legge una stringa da console (ovvero da input args) e la stampa in verticale un carattere per linea.
Ad esempio, dato in input "ciao", viene stampato:
c
i
a
o

Esercizio L.3.3 Scrivere un metodo che riceve tre stringhe e le stampa in verticale una accanto all'altra. Ad esempio: date "ciao", "buondì", "hello", stampa:

```
cbh
iue
aol
onl
 do
 ì
```

Esercizio L.3.4 Scrivere un metodo che riceve una stringa e stampa a video il conteggio delle vocali in essa contenute.
Ad esempio: data la stringa "le aiuole sono pulite", il metodo stampa:
a=1 e=3 i=2 o=3 u=2

Esercizio L.3.5 Scrivere un metodo che, dato un intero positivo n in ingresso, stampi i divisori propri di n (ovvero i divisori ¡ n).
Ad esempio, dato l'intero 20, il metodo stampa:
1, 2, 4, 5, 10
Esercizio L.3.6 Scrivere un metodo che, dati in ingresso due interi a e b e un terzo intero N, stampi a e b e gli N numeri della sequenza in cui ogni numero è

la somma dei due precedenti.
Ad esempio, dati gli interi 2, 3 e 6, il metodo stampa:
2, 3, 5, 8, 13, 21, 34, 55

Esercizio L.3.7 Progettare una classe per la conversione di base dei numeri interi.
Ogni oggetto della classe viene costruito con un intero o con una stringa che contiene l'intero.
La classe è in grado di convertire l'intero nella base desiderata (restituito sotto forma di stringa).
Esercizio L.3.8 Una stringa è palindroma se rimane uguale quando viene letta da sinistra verso destra o da destra verso sinistra.
Scrivere un metodo che dica che una stringa è palindroma.
Scrivere anche una classe di collaudo che testi il metodo in diverse situazioni.
Ad esempio, data in ingresso le stringhe "angelalavalalegna" o "itopinonave-vanonipoti", il metodo deve segnalare che la stringa è palindroma.

Esercizio L.3.9 Scrivere un metodo che, dato un intero N in ingresso, stampi una cornice NxN.
Ad esempio: dato l'intero 5 in ingresso il metodo stampa:

```
*****
*   *
*   *
*   *
*****
```

Esercizio L.3.10 Scrivere un metodo che, dato un intero N e una stringa in ingresso, stampi una cornice NxN all'interno della quale venga visualizzata la stringa (eventualmente suddivisa per righe).
Ad esempio: dato l'intero 6 e la stringa "Cornici in Java" in ingresso il metodo stampa:

```
******
*Corn*
*ici *
*in J*
*ava *
******
```

Esercizio L.3.11 Una terna pitagorica è una tripla di numeri interi a, b, c tali che:
1 ¡= a ¡= b ¡= c e $a^2+b^2 = c^2$ Ovvero a e b sono i lati di un triangolo rettangolo e c l'ipotenusa.
Scrivere un metodo che legge un intero N e stampa tutte le triple pitagoriche con c ¡= N.
Ad esempio:

dato N=15 il metodo stampa:

```
a=3 b=4 c=5
a=6 b=8 c=10
a=5 b=12 c=13
a=9 b=12 c=15
```

Esercizio L.3.12 Scrivere un metodo che, dato un intero positivo dispari N¿1, stampi un triangolo isoscele la cui base è costituita da N caratteri e l'altezza da N-2. Ad esempio, dato l'intero 5, il metodo stampa:

```
  *
 ***
*****
```

Esercizio L.3.13 Progettare una classe Sequenza i cui oggetti si costruiscono con un array di interi La classe espone i seguenti metodi:
- getMediana() che restituisce l'elemnto centrale dell'array ordinato (si utilizzi Arrays.sort per ordinare l'array e Arrays.copyOf per effettuarne una copia).
- getMedia() che restituisce la media degli elementi dell'array.
- getModa() che restituisce il valore più frequente nella sequenza.

Esercizio L.3.14 Esercizio: da numeri romani a numeri interi.
Progettare una classe NumeroRomano i cui oggetti si costruiscono con una stringa contenente un numero romano M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, I = 1
La classe espone il metodo toInteger() che restituisce il valore intero corrispondente.
Ad esempio, new NumeroRomano("MMXIX").toInteger()
restituisce 2019.

Esercizio L.3.15 Scrivere un metodo che, dato un array di stringhe e una stringa in input, restituisca true se l'array contiene la stringa, false altrimenti.
Scrivere una seconda versione del metodo che restituisca la posizione della stringa trovata, -1 altrimenti.

Esercizio L.3.16 Scrivere un metodo che, dato un array di double, restituisca il valore massimo dell'array.

Esercizio L.3.17 Progettare una classe MioArray i cui oggetti vengono costruiti con un array di interi (int[ ]).
La classe implementa i seguenti metodi:
- **contiene**, che dati in ingresso una posizione e un intero, restituisce true o false se l'intero è contenuto in quella posizione nell'array.
- **somma2**, che restituisce la somma dei primi due elementi dell'array. Se l'array

è di lunghezza inferiore (info: la lunghezza dell'array a si ottiene con il campo speciale length, quindi a.length), restituisce il valore del primo elemento oppure 0 se l'array è vuoto.
- **scambia**, che date in ingresso due posizioni intere, scambia i valori presenti nelle due posizioni dell'array (es. scambia(1, 3) trasforma l'array { 1, 2, 3, 4, 5 } in { 1, 4, 3, 2, 5 }).
- **maxTripla**: che restituisce il valore massimo tra il primo, l'ultimo e il valore in posizione intermedia dell'array (es. restituisce 3 se l'oggetto è costruito con { 1, 7, 5, 3, 0, 2, 2 }, le posizione esaminate sono in grassetto).
- **falloInDue**: che restituisce un array di due interi, il primo è il primo elemento dell'array dell'oggetto, il secondo è l'ultimo elemento dell'array dell'oggetto.

Esercizio L.3.18 Progettare una classe Istogramma che rappresenta la distribuzione di dati (es. voti degli studenti) in un intervallo da i a j fornito in input (es. da 0 a 31 (trenta e lode)).
La classe permette di incrementare il conteggio in corrispondenza di ciascun elemento dell'intervallo (es. memorizzando così un nuovo voto di uno studente).
La classe può stampare a video l'istogramma corrispondente.
Più facile in orizzontale
Provate a stampare in verticale!!!

Esercizio L.3.19 Progettare una classe Carta che rappresenti una singola carta da gioco (con seme e valore).
La classe deve restituire su richiesta la propria rappresentazione sotto forma di stringa.
Progettare quindi una classe MazzoDiCarte che rappresenti un intero mazzo da 52 carte.
La classe deve implementare i seguenti metodi:
- **mescola** il mazzo di carte
- **distribuisci** la prossima carta
Infine si progetti una classe di collaudo che crea un mazzo, mescoli le carte e ne distribuisca carte fino ad esaurimento del mazzo.

Esercizio L.3.20 Scrivere un metodo che prenda in ingresso una stringa contenente cifre e restituisca una stringa in cui ciascuna cifra è stata trasformata nella parola corrispondente.
Ad esempio, data in input la stringa "8452", il metodo restituisce "otto quattro cinque due".
Viceversa, scrivere un metodo che prenda in ingresso una stringa contenente cifre scritte a lettere e restituisca una stringa contenente le cifre corrispondenti.
Ad esempio, data in input la stringa "otto quattro cinque due", il metodo restituisce "8452".
Nota: è conveniente utilizzare gli array di stringhe String[]!

Esercizio L.3.21 Come l'esercizio precedente, ma stampando (o leggendo) a lettere tenendo conto della posizione delle cifre (occhio ai casi speciali: undici,

dodici, ecc. . .).
Ad esempio, “8452” viene trasformato in “ottomila quattrocento cinquanta due”.

Esercizio L.3.22 Progettare una classe Filtro costruita con un array di interi. La classe implementa operazioni che permettono di ottenere nuovi sotto-array dell'array iniziale:
-**passaBasso**: restituisce tutti gli elementi ¡= k nell'ordine iniziale.
- **passaAlto**: restituisce tutti gli elementi ¿= k nell'ordine iniziale.
filtra: restituisce l'array iniziale da cui sono state eliminate tutte le occorrenze dell'intero passato in input.
- **filtra**: una seconda versione del metodo che restituisce l'array iniziale da cui vengono eliminate tutte le occorrenze di interi presenti nell'array passato in input.

Se Filtro viene costruito con l'array 1, 2, 10, 2, 42, 7, 8 :
passaBasso(8) restituisce 1, 2, 2, 7, 8
passaAlto(9) restituisce 10, 42
filtra(2) restituisce 1, 10, 42, 7, 8
filtra(new int[] 2, 7, 42) restituisce 1, 10, 8

Esercizio L.3.23 Implementare un metodo statico copyOf che, analogamente a java.util.Arrays.copyOf, copi un array in un nuovo array delle dimensioni specificate (troncando l'array in input, se più grande).

Esercizio L.3.24 Progettare una classe SequenzaDiCifre che espone un metodo che, data in input una stringa e un intero N, aggiunga alla sequenza inizialmente vuota (rappresentata mediate un array) le prime N cifre contenute nella stringa (si assuma che ne contenga comunque almeno N). La classe espone anche un metodo toString che fornisce una rappresentazione sotto forma di stringa della sequenza.
Ad esempio:

```
SequenzaDiCifre s = new SequenzaDiCifre();\\
s.aggiungiCifre("abc1--23", 2);\\
s.aggiungiCifre("xx0a8b76543100", 4);\\
System.out.println(s.toString());\\
```

stampa: [1,2,0,8,7,6]

Esercizio L.3.25 Che cos'è una lista? E' una sequenza di oggetti.
Implementare una classe ListaDiInteri che permetta le seguenti operazioni:
- **Restituisce** l'elemento i-esimo della lista.
- **Restituisce** l'indice della posizione dell'intero fornito in input.
- **Restituisce** una stringa formattata contenente la lista di interi.
- **Restituisce** la dimensione della lista.

- **Contiene** un determinato intero (true o false)?.
- **Aggiungi** un intero in coda alla lista.
- **Aggiungi** un intero nella posizione specificata.
- **Elimina** la prima occorrenza di un intero dalla lista.
- **Elimina** l'elemento i-esimo della lista.

Esercizio L.3.26

Scrivere una classe che rappresenti la tavola pitagorica NxN (dove l'intero N è un parametro di costruzione della classe).
La classe deve, su richiesta, restituire il valore della tabella in corrispondenza della posizione (i, j).
La classe deve poter stampare l'intera tavola.

Esercizio L.3.27 Progettare una classe ScacchieraTris che implementi la scacchiera del gioco del tris La classe deve memorizzare la scacchiera i cui elementi possono essere:
- " " (se non è stata ancora occupata la casella).
- "X" oppure "O" (secondo il giocatore che ha occupato la casella).
La classe deve stampare in qualsiasi momento la situazione della scacchiera.
Deve permettere di occupare una casella con un simbolo "X" o "O".
Progettare quindi una classe Tris che implementi il gioco utilizzando la scacchiera appena progettata.

4 Argomento 4 - Oggetti e Classi 2, Enumerazioni

Esercizio L.4.1 Implementate le classi Carta e MazzoDiCarte utilizzando le enumerazioni invece degli interi per rappresentare semi e valori delle carte.

Esercizio L.4.2 Progettare la classe CampoMinato che realizzi il gioco del campo minato http://it.wikipedia.org/wiki/Campo_minato_(videogioco).
Il costruttore deve inizializzare il campo NxM (dove N e M sono interi forniti in ingresso al costruttore insieme al numero m di mine) piazzando casualmente le m mine nel campo
Implementare un metodo scopri() che, dati x e y in ingresso, scopre la casella e restituisce un intero pari a:
- -1 se la casella contiene una mina.
- La quantità di caselle adiacenti contenenti mine (incluse quelle in diagonale).
- 0 se la caselle adiacenti non contengono mine. In quest'ultimo caso, vengono scoperte anche le caselle adiacenti finché non si incontra un numero ¿ 0 (richiede la ricorsione!).
Implementare un metodo toString() che restituisce la situazione attuale del gioco.

Implementare un metodo vinto() che restituisce lo stato del gioco: perso, vinto, in gioco.

Esercizio L.4.2 Progettare la classe GiocoDelQuindici che realizzi il gioco del quindici (http://it.wikipedia.org/wiki/Gioco_del_quindici).

Il costruttore deve inizializzare una tabellina 4x4 in cui sono posizionate casualmente 15 tessere quadrate (da 1 a 15).
Implementare un metodo privato mischia che posiziona le caselle casualmente nella tabella (usato anche dal costruttore).
Implementare un metodo scorri che prende in ingresso la posizione x e y della casella e la direzione in cui spostare la casella.
Implementare un metodo vinto che restituisce un booleano corrispondente alla vincita del giocatore.

5 Argomento 5 - Ereditarietà

Esercizio L.5.1 Creare una classe BarraDiEnergia costruita con un intero che ne determina la lunghezza massima. Inizialmente la barra è vuota. La barra è dotata di un metodo per l'incremento unitario del suo livello di riempimento e di un metodo toString che ne fornisca la rappresentazione sotto forma di stringa (es. se il livello è 3 su 10, la stringa sarà "OOO=======".
Creare quindi una seconda classe BarraDiEnergiaConPercentuale che fornisce una rappresentazione sotto forma di stringa come BarraDiEnergia ma stampando in coda alla stringa la percentuale del livello di riempimento. Per esempio, se il livello è 3 su 10, la stringa sarà "OOO======= 30%".

Esercizio L.5.2 Implementare una classe ListaDiInteri mediante un array (con i metodi specificati in fondo alle diapositive della terza parte: "controllo e array").
Implementare quindi una classe ListaOrdinataDiInteri per creare liste di interi ordinati in modo crescente. La classe ridefinisce i seguenti 3 metodi di aggiunta:
- **aggiungiInCoda(k):** Aggiungi un intero in coda alla lista: l'aggiunta avviene solo se l'intero preserva l'ordine della lista.
- **aggiungi(k, j):** Aggiungi un intero nella posizione specificata: come sopra, l'aggiunta avviene solo se l'intero preserva l'ordine degli interi della lista.
- **aggiungi(k):** Aggiungi un intero: l'intero viene inserito nella posizione appropriata, in modo da preservare l'ordine degli interi della lista.
L'array non deve essere ordinato con metodi di sorting, quali Arrays.sort (né vostri metodi di sorting *completo* dell'array).

Extra: permettere di specificare un parametro da passare opzionalmente al costruttore di ListaOrdinataDiInteri per stabilire l'ordine della lista (crescente o decrescente; per default, crescente)

Esercizio L.5.3 Progettare (diagramma delle classi) ed implementare la classe Animale che rappresenti un generico animale.
La classe possiede i metodi emettiVerso() e getNumeroDiZampe().
Possiede inoltre il metodo getTaglia() che restituisce un valore scelto tra: piccola, media e grande.
Progettare (diagramma delle classi) ed implementare quindi le classi Mammifero, Felino, Gatto (taglia piccola), Tigre (grande), Cane, Chihuahua (piccola), Beagle (media), Terranova (grande), Uccello, Corvo (media), Passero (piccola), Millepiedi (piccola).
Personalizzare in modo appropriato la taglia, il numero di zampe e il verso degli animali.

Esercizio L.5.4 Progettare la classe ContoBancario che rappresenti un conto con informazioni relative al denaro attualmente disponibile, il codice IBAN.
Modellare quindi una generica operazione bancaria Operazione che disponga di un metodo esegui() Modellare quindi i seguenti tipi di operazione:
- **PrelevaDenaro**: preleva una specificata quantità di denaro da un dato conto.
- **SvuotaConto**: preleva tutto il denaro da un dato conto.
- **VersaDenaro**: versa del denaro sul conto specificato.
- **SituazioneConto**: stampa l'attuale saldo del conto. - **Bonifico**: preleva del denaro da un conto e lo versa su un altro.

Specificare un metodo nella classe ContoBancario che restituisca l'elenco delle operazioni svolte in ordine temporale.

Esercizio L.5.5
Progettare una classe Prodotto con un prezzo e tre tipi diversi di prodotto: BottigliaDAcqua, BarraDiCioccolato, GommeDaMasticare.
Progettare la classe DistributoreAutomatico che rappresenti un distributore automatico costruito con un intero N che determina il numero di prodotti nel distributore.
La classe prevede i seguenti metodi:
- un metodo carica() che inserisce N prodotti di tipo e ordine casuale.
- un metodo inserisciImporto() che permette di inserire un importo nella macchinetta.
- un metodo getProdotto() che, dato in ingresso un numero di prodotto, restituisca il prodotto
- associato a quel numero e decrementi il saldo disponibile nel distributore.
- Un metodo getSaldo() che restituisca il saldo attuale del distributore.
- un metodo getResto() che restituisca il resto dovuto e azzeri il saldo.

Esercizio L.5.6 Progettare una serie di classi che modellino le espressioni matematiche secondo la seguente definizione: - Una costante di tipo double è un'espressione
- Una variabile con nome di tipo stringa e valore double è un'espressione
- Se e1 è un'espressione, allora -e1 è un'espressione

- Se e1, e2 sono espressioni, allora e1 op e2 è un'espressione dove op può essere l'operatore +, -, *, /, %
Ogni tipo di espressione (costante, variabile, espressioni composte) deve essere modellata mediante una classe separata.

Ogni espressione dispone del metodo getValore() che restituisce il valore che quell'espressione possiede in quel momento

Costruire quindi l'espressione –(5+(3/2)-2)*x e calcolarne il valore quando la variabile x vale 3 e quando la variabile x vale 6.

Suggerimenti: la variabile può modificare il proprio valore nel tempo; servirà veramente salvare un valore nella superclasse?

Alternative: progettarlo usando l'ereditarietà (meglio) e mediante enum per le espressioni binarie.

Esercizio L.5.6 Progettare il Gioco dell'Oca modellando:
Il Giocatore, che mantiene l'informazione sulla posizione nel tabellone e i punti accumulati e implementa il metodo tiraDadi().
Il Tabellone come sequenza di caselle costruita a partire da un intero N e da un elenco di giocatori; la classe dispone dell'operazione di posizionamento dei giocatori tenendo conto dell'effetto "gioco dell'oca" in cui, arrivati alla fine, si torna indietro.
Diversi tipi di caselle ciascuna con un diverso effetto a seguito del posizionamento del giocatore su quella casella:
- una CasellaVuota (nessun effetto sul giocatore).
- una CasellaSpostaGiocatore che sposta il giocatore di x caselle (avanti se x ¿ 0 o indietro se x ¡ 0).
- una CasellaPunti che ha l'effetto di far guadagnare o perdere un certo numero di punti al giocatore.
- la classe GiocoDellOca che, dato un intero N e dati i giocatori, che inizializza un tabellone di lunghezza N e implementa il metodo giocaUnTurno() che fa effettuare una mossa a ognuno dei giocatori.

Esercizio L.5.7 Progettare il gioco del Tetris modellando la classe Pezzo con le seguenti operazioni:
- Left : sposta a sinistra il pezzo – Right: sposta a destra il pezzo.
- Rotate: ruota il pezzo in senso orario.
- Down: manda giù il pezzo.
Progettare anche la classe di ciascun pezzo (a forma di L, a forma di T, a serpente, a forma di I e cubo).
Progettare infine la classe Tetris che somministra i pezzi, permette al giocatore di muoverli, gestisce lo spazio libero e calcola i punteggi del giocatore.

6 Argomento 7 - Interfacce e classi interne

Esercizio L.7.1 Progettare tre classi che realizzano diversi tipi di successioni:
- La successione i^2 (es. 0, 1, 4, 9, 16, ecc.)
- La successione casuale (es. -42, 2, 5, 18, 154, ecc.)
- La successione di Fibonacci (es. 1, 1, 2, 3, 5, 8, 13, ecc.)
- La successione infinita 4 8 15 16 23 42 4 8 15 16 23 42 ecc.

Elaborare un meccanismo generale per la generazione della successione indipendentemente dall'implementazione specifica.

Esercizio L.7.2 Scrivere una classe che implementa un array di interi iterabile. Utilizzare l'interfaccia Iterable¡Integer¿.

Esercizio L.7.3 Scrivere una classe che implementa una stringa iterabile utilizzando Iterable¡Character¿.

Esercizio L.7.4 Implementare l'esercizio sulle successioni (L.7.1) utilizzando il meccanismo standard di Java per l'iterazione.

Esercizio L.7.5 Implementare una lista linkata di interi in modo che implementi l'interfaccia java.util.List e rendendo la lista iterabile con il costrutto for each di Java.
Se si sono viste le classi interne, implementare l'iterazione mediante classi interne.

Esercizio L.7.6 Progettare una gerarchia di classi dei seguenti animali, ciascuno con determinate caratteristiche:
- Uccello (vola, becca)
- Pinguino (becca, nuota)
- Aquila (vola, becca)
- Pesce (nuota)
- Pesce volante (nuota, vola)
- Cane (salta, corre, fedele a, domestico)
- Felino (salta, corre, fa le fusa)
- Gatto (salta, corre, fa le fusa, domestico)
- Uomo (salta, corre, pensa, nuota, vola?)

Esercizio L.7.7 Reimplementare le liste linkate di interi in modo da nascondere all'esterno l'implementazione del singolo elemento della lista.
Rendere iterabile la lista in modo che il seguente codice funzioni correttamente:

```
ListaLinkata l = new ListaLinkata(4, 8, 15, 16); l.add(23);
l.add(42);
// stampa gli elementi uno per ogni riga
for (int k : l) System.out.println(k);
```

```
// richiama l.toString()
System.out.println(l);
```

Esercizio L.7.8 Modellare uno scontro su campo tra personaggi Disney e personaggi Marvel.
Squadra Disney = Paperinik, Ciccionik, Superpippo
– Paperinik, Ciccionik e Superpippo sono la versione supereroe dei corrispettivi personaggi (Paperino, Ciccio e Pippo).
Squadra Marvel = Spiderman, La cosa, Magneto
– Notare che Spiderman è la versione supereroe di Peter Parker, noto fotografo del Daily Bugle; La cosa e Magneto invece sono supereroi 24h.

I personaggi con una doppia vita devono esporre un'interfaccia DoppiaVita con i seguenti metodi:
- assumiIdentitaSegreta(): consente di trasformarsi in supereroe.
- assumiIdentitaPubblica(): consente di assumere le sembianze 'umane'.

Ogni supereroe implementa l'interfaccia Supereroe con:
- attacca(): sferra l'attacco specifico del supereroe.

Creare una partita su campo (ovvero una classe di test) in cui le due squadre si alternano in attacchi micidiali.
Il nemico viene scelto casualmente dalla lista dei supereroi avversari.
Hint per versione semplificata: Senza utilizzare classi interne, ma solo interfacce ed ereditarietà.
Hint per versione elaborata: Per riflettere il fatto che nessuno è a conoscenza dei superpoteri in possesso dalle controparti umane, i supereroi, laddove possibile, dovrebbero estendere le classi umane corrispondenti ed essere implementati come loro classi interne private (ad es. Paperinik estende Paperino, ma solo Paperino potrà restituire un'istanza di Paperinik tramite l'interfaccia DoppiaVita).

7 Argomento 7 - no esercizi proposti

8 Argomento 8 - Strutture Dati

Esercizio L.8.1 Si definisca una classe Canzone i cui oggetti sono costruiti con nome e autore della canzone.
Si crei una classe Raccolta che contiene (ovvero i cui oggetti sono costruiti con) un insieme di canzoni.
In una prima versione, si utilizzi HashSet per memorizzare l'insieme di canzoni. Si implementi correttamente equals e hashCode in modo da evitare duplicati di canzoni sostanzialmente uguali (ovvero con lo stesso nome e autore).
La classe è dotata di un metodo toString che restituisce la rappresentazione sotto forma di stringa dell'insieme di canzoni (analogamente, le canzoni sono

dotato dello stesso metodo).
Si implementi quindi la classe RaccoltaOrdinata che utilizza TreeSet per mantenere un ordine sulle canzoni.
Implementare un ordine sulle canzoni basato sul nome della canzone e, se questo è uguale, sul nome dell'autore.
Esercizio L.8.2 Creare una multimappa (ovvero una mappa che permette, a fronte della stessa chiave, di memorizzare più valori) esponendo i metodi:
- **get e put:** Si faccia uso dei metodi più avanzati disponibili in Java 8 (ad es. getOrDefault o merge).
- **keySet e values** (quest'ultimo restituisce l'elenco di tutti i valori presenti nella mappa, anche con ripetizione).
- una versione di **values** che restituisce una lista di elementi ordinata secondo una comparazione ad hoc.
- **valueSet** che restituisce l'insieme di tutti i valori contenuti nella multimappa.

Esercizio L.8.2 Creare una classe ElencoDiRoutine i cui oggetti sono costruiti con una lista di funzioni da stringa a intero da eseguire in sequenza una dopo l'altra.
Queste funzioni sono implementazioni dell'interfaccia java.util.function.Function.
La classe dispone inoltre del metodo esegui che prende in input un parametro di tipo stringa ed esegue le funzioni in sequenza stampandone l'output.
Costruire quindi un'istanza della classe con le seguenti funzioni (usando riferimenti a metodi laddove possibile):
- data in input una stringa x, restituisce la lunghezza di x.
- data una stringa x, restituisce il numero di occorrenze del carattere 'y'.
- data una stringa x, restituisce il corrispondente intero (assumendo che x contenga solo cifre).
- data una stringa x, restituisce la somma dei caratteri contenuti nella stringa.

Esercizio L.8.3 Partendo dall'esempio sugli alberi binari delle diapositive precedenti, scrivere una classe Albero che rappresenta un albero n-ario (con un numero variabile di figli per ciascun nodo) i cui nodi contengono un valore intero.
La classe espone il metodo contains che, dato in input un intero, restituisce true se un nodo dell'albero contiene l'intero, false altrimenti.
Sono implementati due meccanismi per aggiungere nodi all'albero:
- Mediante classe annidata Nodo, che espone un metodo add il quale aggiunge il nodo in input tra i propri figli.
- Mediante un metodo Albero.add(intero1, intero2, ..., interon) che aggiunge il cammino degli interi all'albero, verificando che il primo intero corrisponda con quello della radice e sollevando eccezione altrimenti; per gli altri interi, se non esistono vengono aggiunti come nodi intermedi.
La classe Albero espone il metodo remove che, dato in input un intero, rimuove il nodo corrispondente dall'albero aggiungendo tutti i suoi figli come figli del padre; nel caso il nodo fosse la radice, il metodo prende il suo primo figlio e lo sostituisce alla radice stessa.

9 Argomento 9 - Tipi Generici

Esercizio L.9.1 Progettare una classe generica Pila implementata mediante lista di elementi (provate anche con l'array).
La classe è costruita con la dimensione iniziale dell'array ed implementa i seguenti metodi:
- push: inserisce un elemento in cima alla pila.
- pop: elimina e restituisce l'elemento in cima alla pila.
- peek: restituisce l'elemento in cima alla pila.
- isEmpty: restituisce true se la pila è vuota.

Esercizio L.9.2 Implementare i seguenti due metodi generici statici: inverti, che data in input una lista di elementi di tipo generico, restituisca un'altra lista con gli elementi in ordine invertito max, che data in input una lista di elementi di tipo generico, ne restituisca il valore massimo.
Nota: per il secondo metodo è necessario utilizzare il costrutto ¡T extends InterfacciaOClasse¿, che impone un vincolo sul supertipo di T.
Esercizio L.9.3 Scrivere una classe SuccessioniNumeriche, i cui oggetti sono inizialmente vuoti La classe espone un metodo addSuccessione che, preso in input un nome e una sequenza numerica, consente di aggiungere una successione (di lunghezza finita) del tipo numerico generico specificato (Integer, Float, Long, ecc.).
La classe ha inoltre un metodo getSuccessione(String nome) con cui è possibile recuperare una successione a partire dal nome mnemonico.
Prevedere le seguenti successioni:
– Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34
– 1/n: 1, 1/2, 1/3, 1/4, 1/5, 1/6
– RandomLong: contiene oggetti Long costruiti casualmente
Hint: utilizzare la classe Number come superclasse del tipo generico.

Esercizio L.9.4 Una multimappa è una mappa che ammette più valori a fronte di una data chiave.
Creare una classe MultiMappa generica sul tipo di chiavi e valori.
La classe implementa i seguenti metodi:
- put(k, v) che associa il valore alla chiave specificata
- putAll(MultiMappa) che aggiunge tutti gli elementi della multimappa in input alla mappa corrente.
- removeAll(MultiMappa) che rimuove tutte le chiavi della multimappa in input dalla mappa corrente.
- get(k) che restituisce l'insieme dei valori associati alla chiave.
- get(k, p), come sopra ma restituisce solo i valori che soddisfano il predicato p.
- values() che restituisce l'elenco (con duplicati) dei valori contenuti nella multimappa.
- valueSet() che restituisce l'insieme dei valori contenuti nella multimappa.
- transfomToMultiMappa che restituisce una multimappa in cui le coppie (k, v) sono trasformate in (k, z) secondo una funzione (k, v) -> z (z può essere di tipo

diverso rispetto a quello di v)
- mapEach che sostituisce ciascun valore v con un valore dello stesso tipo secondo una funzione (k, v) -> v'
- la classe è iterabile sulle coppie (k, v) mediante una classe interna Elemento.

10 Argomento 10 - Ricorsione

Esercizio L.10.1 Si realizzi un metodo ricorsivo che, dato in input un array di stringhe, ne restituisca la loro concatenazione.
Ad esempio, concatena(new String[] { "abc", "de", "f" }) restituisce "abcdef".

Esercizio L.10.2 Si realizzi una classe Cartella, costruita con il percorso di una cartella reale, dotata di un metodo toString() che ne visualizzi ricorsivamente il contenuto.
Avete bisogno della classe java.io.File.

Esercizio L.10.3 Si realizzi una classe Cartella, costruita con il percorso di una cartella reale, dotata dei seguenti metodi:
- Cerca un file all'interno di una cartella.
- Cerca all'interno della cartella tutti i file con l'estensione specificata.
- Cerca all'interno della cartella tutti i file con le estensioni fornite in input.
Si faccia uso al meglio dell'overloading.

Esercizio L.10.4 Scrivere un metodo ricorsivo che stampi tutte le stringhe binarie di lunghezza k.
Ad esempio, genera(3) stampa:
000
001
010
011
100
101
110
111

Esercizio L.10.5 Scrivere una classe costruita con un insieme di caratteri e dotata di un metodo genera che, dato in input un intero k, restituisca l'elenco completo delle stringhe di lunghezza k utilizzando caratteri nell'insieme in input.
– ad esempio, per l'insieme {'a', 'b', 'c'}, genera(2) restituisce l'elenco:
["*aa*", "*ab*", "*ac*", "*ba*", "*bb*", "*bc*", "*ca*", "*cb*", "*cc*"]

Come procedere? E' necessario scomporre il problema in sottoproblemi.
Iterare sui caratteri validi per ottenere il primo carattere della stringa ed ef-

fettuare, per ciascuno di essi, una chiamata ricorsiva genera(k-1), finché non si arriva alla stringa vuota.
Extra: creare una versione sovraccaricata di genera in cui si passa un predicato che determina se la stringa deve essere generata oppure no.

Esercizio L.10.6 Scrivere una classe costruita con una stringa e dotata di un metodo generaPermutazioni che restituisce l'elenco completo delle permutazioni della stringa.
ad esempio, per la stringa "abc" genera l'elenco ["*abc*","*acb*","*bac*","*bca*","*cab*","*cba*"]
Come procedere? E' necessario scomporre il problema in sottoproblemi.
Fissato un carattere della stringa iniziale, lo poniamo all'inizio della stringa e permutiamo la sottostringa rimanente.
stesso problema, ma su una sequenza di caratteri più piccola!

Esercizio L.10.7 Si realizzi una classe Labirinto dotata di un solo ingresso a un corridoio. Ciascun corridoio fornisce zero o più ingressi ad altri corridoi. Ogni corridoio è dotato di un metodo contieneMinotauro che specifica se nel corridoio si trova il Minotauro. La classe Labirinto è dotata di un metodo percorri() che, partendo dal corridoio iniziale, cerca il Minotauro nel labirinto.
Avanzato: Teseo, che fa il suo ingresso nel Labirinto, una volta trovato il Minotauro, stampa il percorso a ritroso mediante il "filo di Arianna"...

Esercizio L.10.8 Si realizzi una classe LabirintoMatrice in cui il labirinto è rappresentato mediante una matrice NxM (due valori in input al costruttore).
Ogni cella della matrice è un'istanza di una classe Cella.
Una cella o è una stanza o è un muro.
Una stanza può o meno contenere il minotauro.
La classe LabirintoMatrice è dotata di un metodo percorri() che, partendo dalla cella (x,y) specificata in ingresso al metodo, cerca il Minotauro nel labirinto.
Avanzato: stampare il percorso a ritroso.

Esercizio L.10.9 Scrivere una classe SommaSottrai che, costruita con una lista di interi, espone un metodo ricorsivo sommaSottrai che restituisce la somma dei valori della lista in posizione dispari da cui vengono sottratti i valori in posizione pari.
– Ad esempio, data la lista $[2, 5, 3, 7, 11, 1]$ il metodo restituisce $2 - 5 + 3 - 7 + 11 - 1 = 3$.

Esercizio L.10.10 Si progetti un'interfaccia Valutabile che espone un metodo valuta() il quale calcola il valore dell'oggetto e restituisce un booleano.
Si progetti quindi una classe ValutaValutabili i cui oggetti sono costruiti con un array di oggetti "valutabili".
La classe espone un metodo valutaInAnd il quale restituisce il valore booleano dell'and tra tutti i valori degli oggetti nell'array.
Analogamente, realizzare un metodo valutaInOr che mette in or i valori degli oggetti nell'array.

Esercizio L.10.11 In un tempio dell'estremo oriente alcuni monaci devono spostare una pila di dischi d'oro da un piolo di diamante a un altro.
La pila iniziale comprende 64 dischi, tutti infililati nello stesso piolo dal basso verso l'alto in ordine di grandezza decrescente.
I monaci devono spostare l'intera pila su un altro piolo, rispettando il vincolo di muovere un solo disco per volta e non ponendo mai un disco più grande sopra uno più piccolo.
Sono disponibili 3 pioli e uno può essere usato come supporto temporaneo.
Si realizzi la classe TorreDiHanoi che risolva il problema.
Tenete bene a mente che per risolvere un problema ricorsivamente è necessario:
- ridurlo a un problema più piccolo
- identificare il caso base
Posso riformulare il problema di spostare n dischi nel problema in cui:
- Sposto n-1 dischi dal piolo 1 al piolo 2 (usando il piolo 3 come supporto temporaneo)
- Sposto l'ultimo disco (il più grande) dal piolo 1 al piolo 3
- Sposto n-1 dischi dal piolo 2 al piolo 3, usando il piolo 1 come supporto temporaneo

11 Argomento 11 - Eccezioni

Esercizio L.11.1 Definiamo una sequenza "a gradini" come una successione in cui ciascun elemento dista esattamente 1 dal precedente (la sequenza "7 8 9 10 11 12 11 10 11 10 9 8 7" è a gradini, mentre "1 2 8 7 6 5 42 9 20" non lo è).
Costruire una sequenza a gradini inizialmente vuota.
Inserire successivamente altri elementi nella sequenza tramite il metodo aggiungi(int x).
Nel caso il prossimo numero aggiunto violi il vincolo della sequenza a gradini, va notificato un errore al metodo chiamante.
Prevedere inoltre un metodo che stampi la sequenza finora memorizzata. La sequenza, inoltre, non ha vincoli di lunghezza (potenzialmente infinita).

Esercizio L.11.2 Implementare un floppy disk da 3.5 pollici
Il floppy disk è un supporto magnetico che contiene dati e può essere acceduto in lettura e scrittura
Esso ha una capacità pari a 1.474.560 bytes (ovvero 1.44 MB)
Il floppy disk possiede anche un blocco scrittura che è possibile attivare o disattivare; questo meccanismo, se attivato, impedisce la scrittura di dati
Implementare inoltre i seguenti metodi:
- posizionaTestina(): posiziona la testina in posizione k-esima
- leggi(): legge i prossimi x byte
- scrivi(): scrive i byte passati in input
- formatta(): cancella tutti i dati presenti sul floppy disk

- attivaBloccoScrittura(): attiva il blocco scrittura
- disattivaBloccoScrittura(): disattiva il blocco scrittura

Gestire inoltre tutte le situazioni di errore, ad esempio:
- se si cerca di scrivere o formattare mentre è presente il blocco scrittura.
- se si cerca di scrivere ma il disco è pieno.
- si cerca di leggere ma non sono presenti sufficienti dati sul disco.

Esercizio L.11.3 Scrivere un'interfaccia Dizionario dotata dei metodi:
- Elemento search(Chiave k): cerca l'elemento associato alla chiave k nella struttura dati
- void add(Chiave k, Elemento e): aggiunge l'elemento e con chiave k nella struttura dati - Elemento delete(Chiave k): rimuove l'oggetto associato alla chiave k dalla struttura dati.
- int size(): restituisce la taglia del dizionario.

Implementare una coppia (k, e) chiave-elemento, costruita a partire da una Chiave k ed un Elemento e ad essa associato.
Implementare la struttura dati Mappa che rappresenta una collezione di coppie senza ripetizione di chiavi.
Prevedere il sollevamento delle seguenti eccezioni:
- ElementNotFoundException: lanciata nel caso la chiave da cercare o rimuovere non è contenuta nella struttura dati.
- ElementAlreadyContainedException: lanciata nel caso in cui la chiave da aggiungere all'insieme sia già contenuta.

Esercizio L.11.4 Nel 1825 a causa del Miramichi Fire in Canada morirono 160 persone. La squadra di volontari FIRE (Fuoco Immaginario o Reale Estinguesi) è specializzata nell'estinguere qualunque tipo di fuoco, sia esso fuoco fatuo o un devastante incendio naturale.

Dal gruppo emergono per competenza due figure:
- La volontaria Acquafredda: la volontaria più veloce nel fornire secchi d'acqua al resto del gruppo.
- Il volontario Fuoco: il volontario dalla mira migliore e in grado di spegnere anche le fiamme più alte.
- Il resto del gruppo è formato da semplici volontari uniti in una catena umana, ciascuno in grado di passare secchi d'acqua dal volontario alla sua sinistra a quello alla sua destra.

I volontari hanno a disposizione un unico secchio d'acqua che è possibile riempire o svuotare.
La volontaria Acquafredda è l'unica ad aver accesso all'acqua, ovvero l'unica in grado di riempire il secchio.
Il volontario Fuoco invece è l'unico in grado di osservare in maniera diretta l'incendio, l'unico a poterlo spegnere e l'unico anche a determinarne l'avvenuta

estinzione.
All'incendio è associato un intero compreso tra 1 e 10 che ne determina l'intensità nella scala "Fireneit" e corrisponde, sperimentalmente, al numero di secchi necessari alla sua estinzione.

Quando l'incendio si estingue il volontario Fuoco si accorge dell'evento FuocoEstintoException e, secondo il codice condiviso all'interno della FIRE, lancia il messaggio BastaAcquaException!'. Tale messaggio è molto importante ed è necessario che arrivi alle orecchie della volontaria Acquafredda il prima possibile.
Al ricevimento di tale messaggio, la volontaria Acquafredda dovrà smettere di riempire il secchio d'acqua, risorsa assai preziosa. Un obiettivo della missione dev'essere quello di non sprecare acqua ed utilizzarne la giusta quantità.

Costruire una catena di volontari avente in testa la volontaria Acquafredda, in coda il volontario Fuoco e in mezzo 3 volontari comuni.
I volontari avranno a disposizione un unico secchio, inizialmente messo a disposizione dalla volontaria Acquafredda, che viene passato da volontario a volontario tramite il metodo void estinguiIncendio(Secchio s).
Appiccare un Incendio doloso e mettere in moto la catena FIRE in modo che estingua l'incendio.

12 Argomento 12 - Input Output

Esercizio L.12.1 Si progetti una classe i cui oggetti sono costruiti a partire da un determinato file di testo f. La classe espone un metodo numera che:
crea un file il cui nome è dato da quello del file f cui viene concatenata l'estensione ".num"
contiene riga per riga le informazioni contenute nel file f cui viene anteposto il numero di riga.

Ad esempio, date le righe:
Questa è la prima riga del file
Questa è la seconda riga del file

salva nel nuovo file:
1: Questa è la prima riga del file
2: Questa è la seconda riga del file

13 Argomento 13 - Stream

Esercizio L.13.1

```
public class Titolo
{
    public enum Allineamento {CX, SX, DX}
    private Allineamento allineamento;
    private List<Riga> righe;

    public Titolo(Allineamento a) {this(a, new ArrayList<>());}
    public Titolo(Allineamento a,List<Riga> righe) {
    allinemanto=a;this.righe=righe;
    }
    public void add(Riga r) {righe.add(r);}
    public boolean isCentered(){return allineamento==Allineamento.CX;}
    @Override
    public String toString(){return righe.toString();}
    public Allineamento getAllineamento(){return allineamento;}
    public List<Riga> getRighe(){return new ArrayList<>(righe);}

    static public class Riga
    {
        private String riga;
        private int numero;
        public Riga(String riga, int numero){this.riga=riga;this.numero=numero;}
        public Riga(String riga){this.riga,-1);}
        @Override
        public String toString(){return (numero==-1?"":numero+":")+riga;}
    }
}
```

Utilizzare gli stream per ottenere i seguenti dati (svolgere ciascun punto sia con riferimenti a metodi che lambda).

A partire da una List¡Titolo¿, si calcolino senza precalcoli di strutture dati intermedie (ovvero, su una sola riga):
- l'insieme dei primi 5 titoli aventi al più una riga.
- la lista dei soli titoli centrati e in ordine alfabetico.
- mappa da allineamento a lista di titoli.
- mappa da allineamento a insieme di titoli.
- mappa da allineamento alla concatenazione delle stringhe dei titoli.
- mappa da allineamento alla lista delle stringhe dei titoli.
- insieme delle righe dei titoli ciascuna rappresentata sotto forma di stringa.
- mappa dei conteggi delle parole nelle righe di titoli calcolate al punto precedente (questo esercizio richiede l'utilizzo di flatMap).

1) Utilizzando il Collectors.groupingBy.
2) Utilizzando il Collectors.toMap.