

Laboratorio 1

Esercizio 1

Avviate il browser. Visitare i seguenti siti di interesse didattico per questo corso e per ciascun sito memorizzare un segnalibro.

- Sito Moodle dell'insegnamento, accessibile da: <http://stem.elearning.unipd.it/>. Consultare tutte le informazioni nella sezione introduttiva. Poi consultare tutte le informazioni e i materiali nella sezione "Informazioni di carattere generale sull'insegnamento di Fondamenti di Informatica, Canale B".
- Sito del DEI (<http://dei.unipd.it/>). Consultare le informazioni contenute nella sezione <https://www.dei.unipd.it/orientamento>
- Sito dell'Ateneo: <https://www.unipd.it/target/studentesse-e-studenti>

Esercizio 2 (svolgere a casa sul proprio computer)

Configurare i propri strumenti di lavoro.

- Scegliere e installare sul proprio computer un editor di testo adatto alla programmazione. Accertarsi almeno che:
 - Mostri i numeri di riga
 - Sia in grado di evidenziare la sintassi Java

Esercizio 3

Addestrarsi a lavorare dalla riga di comandi svolgendo [questa esercitazione](#).

I comandi funzionano in sistemi Linux, se svolgete questa esercitazione in ambienti diversi (Windows), dovete **adattare i comandi al sistema operativo che state utilizzando** (a casa o in Taliercio).

Esercizio 4

Usando uno degli editor di testo disponibili sul sistema, scrivere il file **HelloTester.java**, copiando con precisione quanto segue:

```
public class HelloTester
{
    public static void main(String[] args)
    {
        // visualizza un messaggio in output
        System.out.println("Ciao mondo!");
    }
}
```

Terminata la copiatura, salvare il file e uscire dall'editor. Aprire un terminale ed eseguire il comando (di compilazione Java):

javac HelloTester.java

Se il comando di compilazione non va a buon fine e vengono segnalati errori, usare di nuovo l'editor per correggere il file HelloTester.java. Se la compilazione va a buon fine (ovvero non viene segnalato alcun errore), verificare che sia stato creato il file HelloTester.class. Aprire un secondo terminale ed eseguire il comando (di esecuzione Java):

java HelloTester

Verificare che sulla finestra del terminale, subito dopo la riga in cui è avvenuta l'esecuzione del comando, venga visualizzato il messaggio:

Ciao mondo!

Esercizio 5

Eliminare usando comandi del terminale il file HelloTester.class e provare a eseguire di nuovo il comando di esecuzione Java:

java HelloTester

verificando che si ottiene una segnalazione d'errore.

Generare nuovamente il file HelloTester.class mediante l'esecuzione del compilatore Java:

```
javac HelloTester.java
```

quindi tentare di nuovo l'esecuzione (con **java HelloTester**), verificando che venga visualizzato il messaggio:

Ciao mondo!

Esercizio 6

Scaricare l'archivio .zip "SorgentiConErrori-20221010" (memorizzato nella cartella delle soluzioni e file utili, sezione attività laboratoriali, nostro moodle).

Tentare di compilare (ed eseguire?) tutti i file. Correggere gli errori e compilare (ed eseguire) i sorgenti.

Esercizio 7

Scrivere un programma che stampi il vostro nome incorniciato come segue:

```
-----  
|   nome   |  
-----
```

Esercizio 9

Scrivere un programma che disegni delle facce stilizzate usando i caratteri della tastiera, per esempio come queste (e magari anche migliori!).

NB: Attenzione ai caratteri <\> e <">.

```
\\|||///   ###|||###   " " " " " "  
( | 0 0 | ) ( | 0 0 | ) ((~~~~~))  
(  v  )   |  8  |   ))^  ^((  
(  < _ > ) (  VwwwV  ) ((  . .  ))  
<_____>  <_____>  )) -- ((  
                               ((    ))
```

Esercizio 10

Rispondere alle domande di autovalutazione proposte a lezione

Laboratorio 2

Esercizio 1

Argomento: errori di compilazione, errori di esecuzione

Scrivere, compilare ed eseguire alcuni degli esempi visti a lezione, introducendo anche errori di sintassi e/o logici e osservando con attenzione i messaggi d'errore generati dal compilatore e, eventualmente, dall'interprete.

Esercizio 2

Argomento: rappresentazione numeri in java, conversioni ed arrotondamenti

Scaricare, compilare ed eseguire la classe [NumLimitTester](#). Osservare i risultati, ripassare i limiti di variabilità per i tipi di dati fondamentali in java.

Scaricare, compilare ed eseguire la classe [NumTypeTester](#). Osservare i risultati, manipolare i dati secondo quanto suggerito nei commenti interni alla classe, rispondere alle domande.

Scaricare, compilare ed eseguire la classe [DiscountTester](#). Osservare i risultati, manipolare i dati secondo quanto suggerito nei commenti interni alla classe, rispondere alle domande.

Esercizio 3

Argomento: uso della documentazione java; conversioni tra stringhe e tipi numerici

Esplorare la Documentazione API di Java, rintracciabile sia [sul web](#) che sul [sito dell'Aula Taliercio](#) (-> Guide on-line -> Java openjdk 1.8.0 API).

Cercare la documentazione delle classi **String**, **Integer**, **Double** (possibilmente con la funzione di ricerca del vostro Browser), studiarne la descrizione, in particolare studiare la descrizione dettagliata dei:

- Metodi della classe **String**: length, substring, toUpperCase, toLowerCase, ...
- Metodi delle classi **Integer** e **Double**: parseInt, parseDouble, toString, ...

Scaricare, compilare ed eseguire la classe [StringNumTester](#). Osservare i risultati, manipolare i dati secondo quanto suggerito nei commenti interni alla classe, rispondere alle domande.

Esercizio 4

Argomento: rappresentazione dei numeri

Scrivere il codice di una classe **IntegerDivider** che definisce due numeri interi m e n e invia a standard output il risultato e il resto della divisione m/n. Compilare e eseguire. Che cosa succede se il divisore è zero?

Scrivere poi il codice di una classe **DoubleDivider** che definisce due numeri reali x e y e invia a standard output il risultato della divisione x/y. Compilare ed eseguire. Esegue correttamente? Provare con le coppie di numeri:

x = 7.0	y = 0.0
x = -7.0	y = 0.0
x = 0.0	y = 0.0

Esercizio 5

Argomento: rappresentazione dei numeri; divisione intera `</>` e modulo `<%>`

Scrivere un programma che

- definisce un numero intero positivo avente al massimo cinque cifre
- visualizza sull'output standard le singole cifre del numero, separandole con uno spazio

Ad esempio, il numero **14356** verrà visualizzato come **1 4 3 5 6**. Il numero **367** verrà visualizzato come **0 0 3 6 7**.

Suggerimento: applicare la definizione di notazione posizionale in base 10, studiare le proprietà della divisione intera in Java e del relativo operatore %. Verificare il corretto funzionamento del programma in casi diversi (in particolare, con numeri aventi meno di cinque cifre) e verificare il comportamento del programma in caso di valori di ingresso non ammessi

- numeri con più di cinque cifre
- numeri negativi
- numeri in virgola mobile
- valore di ingresso non numerico (ad esempio: Ciao)

Esercizio 6

Argomento: *rappresentazione dei numeri; divisione intera </> e modulo <%>*

Scrivere un programma che definisca un numero razionale positivo n nell'intervallo $[0, 32)$ e ne stampi a standard output la codifica binaria in formato a virgola fissa, usando cinque cifre binarie per la parte intera e cinque per quella frazionaria. Esempio: 8.5 in base dieci = 01000.10000 in base due.

Suggerimento: si usino gli algoritmi di conversione da decimale a binario della parte intera e della parte frazionaria presentati a lezione.

Esercizio 7

Argomento: *divisione intera </> e modulo <%>*

Scrivere un programma che

- definisce due orari nel formato "24 ore", ciascuno di quattro cifre (ad esempio, 0900 oppure 1730)
- visualizza sull'output standard il numero di ore e di minuti (separatamente) che intercorrono fra i due orari

secondo il seguente esempio:

primo orario: 0900

secondo orario: 1730

Tempo trascorso: 8 ore e 30 minuti

Modificare poi il programma in modo che funzioni correttamente anche se il secondo orario è inferiore al primo (per intervalli di tempo che comprendono la mezzanotte):

primo orario: 1730

secondo orario: 0900

Tempo trascorso: 15 ore e 30 minuti

Esercizio 8

(la classe **Rectangle** è stata solo accennata in classe, ma è comunque descritta esaurientemente nella documentazione API di Java)

Argomento: *uso della documentazione java*

Esplorare la Documentazione API di Java, rintracciabile sia [sul web](#) che sul [sito dell'Aula Taliercio](#) (-> Guide on-line -> Java openjdk 1.8.0 API).

Cercare la documentazione della classe **Rectangle** (possibilmente con la funzione di ricerca del vostro Browser), studiarne la descrizione, studiare la descrizione dettagliata dei suoi metodi. In particolare:

- Identificare il metodo **void add(int newx, int newy)**
- Leggere con attenzione la documentazione
- Stabilire il risultato dell'esecuzione degli enunciati seguenti

```
Rectangle box = new Rectangle(5, 10, 20, 30);
box.add(0, 0);
```

Esercizio 9

(la classe `Rectangle` è stata solo accennata in classe. E il suo metodo `intersection` non è stato presentato in classe. Entrambi sono comunque descritti esaurientemente nella documentazione API di Java)

Argomento: uso di classi (in particolare uso di classi della libreria standard)

Il metodo **`intersection`** della classe `Rectangle` calcola l'intersezione di due rettangoli e viene invocato in questo modo

```
Rectangle r3 = r1.intersection(r2);
```

(si veda anche la Documentazione API).

Scrivere un programma che

- costruisca due oggetti rettangolo, definendo per ciascuno di essi le coordinate del vertice in alto a sinistra, altezza e larghezza
- stampi vertici e dimensioni dei due rettangoli appena costruiti
- stampi vertici e dimensioni della loro intersezione.

Cosa succede quando i rettangoli non si sovrappongono?

Esercizio 10

Convertire un numero 'x' (a piacere) di secondi in ore, minuti e secondi.

Esercizio 11

Calcolare il numero di monete da 2 euro e da 1 euro necessarie per una certa somma 'x'.

Esercizio 12

Convertire un numero 'x' (a piacere) di giorni in settimane e giorni.

Esercizio 13

Calcolare il numero di pagine complete e il numero di righe sulla pagina incompleta, dato un numero totale di righe 'x' (a piacere) e il numero di righe per pagina (a piacere).

Esercizio 14

Convertire un numero di minuti (a piacere) in giorni, ore e minuti.

Laboratorio 3

Esercizio 1

Argomento: uso dell'input standard

Modificare l'**Esercizio 7** del Laboratorio 2 in maniera tale che il programma legga dall'input standard i due orari nel formato "24 ore", ciascuno di quattro cifre (ad esempio, 0900 oppure 1730).

Esercizio 2

Argomento: uso dell'input standard

Scrivere un programma che

- acquisisce da standard input una riga contenente tre parole o *token* (ovvero tre stringhe separate da un carattere di spaziatura)
- stampa le tre parole a standard output, una per riga nell'ordine in cui sono state acquisite

Esempio: se si introduce la stringa "uno due tre", l'esecuzione della classe produce le seguenti stampe:

```
uno
due
tre
```

Esercizio 3

Argomento: uso dell'input standard; uso di caratteri di escape

Modificare il programma dell'**Esercizio 2** in maniera tale che

- le tre stringhe vengano stampate a standard output nell'ordine inverso rispetto a quello in cui sono state inserite
- la stampa delle tre stringhe venga effettuata utilizzando una singola invocazione del metodo **print** o del metodo **println**

Esercizio 4

Argomento: uso dell'input standard

Scrivere un programma che

- acquisisce tre numeri in virgola mobile da standard input
- stampa a standard output i tre numeri e la loro somma.

Esempio: se si introduce la stringa "1.0 2.0 3.0", il programma produce la seguente stampa:

```
1.0 + 2.0 + 3.0 = 6.0
```

Suggerimento : se state lavorando su un sistema operativo che usa la lingua italiana, e usate il metodo **nextDouble** di **Scanner** per leggere i numeri in virgola mobile da standard input, i numeri vanno inseriti usando la virgola come carattere di separazione fra parte intera e parte frazionaria. Il programma invia a standard output numeri in virgola mobile in cui il carattere di separazione fra parte intera e frazionaria è il punto. Nel riquadro più sotto viene spiegato il perché.

Impostazioni locali nell'uso dell'input standard

A partire dalla JDK5.0 e nelle seguenti versioni, alcune funzioni di ingresso/uscita sono dipendenti da convenzioni in uso localmente, dette *impostazioni locali*. Esempi di impostazioni locali sono il formato dell'ora e il carattere di separazione nei numeri espressi in virgola mobile (come e' noto, nel mondo anglosassone il carattere di separazione fra la parte intera e la parte frazionaria e' il carattere '.', mentre in Italia e' il carattere ',').

Quando si usano oggetti di classe **java.util.Scanner** per acquisire dati in ingresso, il programmatore puo` decidere esplicitamente l'impostazione locale. Se non sceglie, viene automaticamente usata l'impostazione dell'elaboratore su cui viene eseguito il programma, detta *impostazione di default*.

La scelta dell'impostazione locale si fa invocando sull'oggetto di classe **Scanner** il metodo **useLocale(Locale locale)**, che accetta come parametro un oggetto di classe **Locale** del pacchetto **java.util**.

La classe **java.util.Locale** contiene la definizione di comodi oggetti costanti da usare come parametro; ad esempio:

- **Locale.US** oggetto costante per le impostazioni locali americane (USA)
- **Locale.ITALY** oggetto costante per le impostazioni locali italiane

Di seguito un esempio di uso esplicito di impostazioni locali:

```
import java.util.Scanner;
import java.util.Locale;

...
    Scanner in = new Scanner(System.in);
    in.useLocale(Locale.US);           // usa la convenzione USA
....
```

Approfondimento

Anche gli oggetti di classe **java.io.PrintStream** come **System.out** possono adoperare impostazioni locali. In questo caso si usa il metodo **printf(Locale locale, String format, Object ...args)** che invia a standard output la stringa *format*. Si tratta di una variante del metodo **printf(String format, Object ...args)**, gia` esaminato, in cui pero` c'e` un parametro esplicito aggiuntivo: il parametro **Locale locale**. Per il resto, questo due varianti del metodo **printf** si usano allo stesso modo, e in particolare usano le stesse convenzioni di formattazione.

Ad esempio, per stampare a standard output un numero in virgola mobile con una impostazione locale italiana si scrive:

```
import java.util.Locale;

...
double d = 12.3;
System.out.printf(Locale.ITALY, "%f", d);
...
```

Esercizio 5

Argomento: uso dell'input standard: localizzazione

Modificare l'**Esercizio 4** in maniera tale che il programma accetti in ingresso solo numeri in formato virgola mobile contenenti come separatore il punto (anziche` la virgola).

Modificarlo poi in maniera tale che il programma invece accetti in ingresso e **scriva in uscita** solo numeri in formato virgola mobile contenenti come separatore la virgola (anziche` il punto).

Esercizio 6

Argomento: uso dell'input standard; uso della classe Math;

Scrivere un programma che

- acquisisce da standard input il valore del raggio di un cerchio (numero in virgola mobile)
- stampa a stardand output il perimetro e l'area del cerchio corrispondente

Suggerimento: ricordate che la classe **Math** contiene metodi e anche *costanti numeriche* utili.

Esercizio 7

Argomento: uso dell'input standard; uso della classe Math; uso del metodo printf; stampa di caratteri non ASCII

Scrivere un programma che

- acquisisce da standard input i valori in centimetri dei due cateti di un triangolo rettangolo (due numeri in virgola mobile, uno per riga)
- stampa a standard output i seguenti numeri, tutti approssimati a due cifre decimali e incolonnati correttamente
 - il valore dell'ipotenusa
 - il perimetro e l'area del triangolo
 - i valori in gradi dei due angoli non retti del triangolo

Esempio: se si introducono i due numeri 3.0 e 4.0, l'esecuzione produce le seguenti stampe:

```
Ipotenusa:  5.00 cm
Perimetro: 12.00 cm
Area:       6.00 cm2
Angolo1:    37.00°
Angolo2:    53.00°
```

Esercizio 8

Argomento: uso dell'input standard; uso della classe Math; uso del metodo printf; stampa di caratteri non ASCII

Scrivere un programma che

- acquisisce da standard input il valore **in gradi** di un angolo (un numero in virgola mobile)
- stampa a standard output i seguenti numeri, tutti approssimati a tre cifre decimali
 - il valore dell'angolo, espresso in radianti, normalizzato nell'intervallo $(-2\pi, 2\pi)$ e visualizzato come multiplo di π
 - il valore del seno dell'angolo
 - il valore del coseno dell'angolo
 - il valore della tangente dell'angolo

Esempio: se si introduce il numero 405.0 (che quindi corrisponde a 45°), l'esecuzione produce le seguenti stampe:

```
Angolo = 0.250*pi rad
sin(0.250*pi) = 0.707
cos(0.250*pi) = 0.707
tan(0.250*pi) = 1.000
```

Suggerimento: studiare la documentazione della classe **Math** e trovare i metodi utili ad effettuare la conversione, prima da gradi a radianti, e poi a radianti normalizzati nell'intervallo desiderato.

SOLUZIONE

Esercizio 9

Argomento: uso dell'input standard; manipolazione di stringhe (uso della classe String)

Scrivere un programma che

- riceve in ingresso (su una sola riga) una frase nel formato "Articolo Sostantivo Verbo Aggettivo"
- ne stampa a standard output la traduzione nella lingua di Eta Beta (ovvero antepone il carattere 'p' al sostantivo e all'aggettivo), usando un carattere maiuscolo ad inizio frase.

Ad esempio, se viene inserita la stringa "la vita è bella" il programma stamperà a standard output la stringa "La pvita è pbella".

Esercizio 10

Argomento: uso dell'input standard; manipolazione di stringhe (uso della classe String)

Scrivere un programma che

- riceve in ingresso una stringa rappresentante un aggettivo qualificativo maschile o femminile (terminante con il carattere 'o' o con il carattere 'a')

- stampa a standard output l'aggettivo inserito, con solo il primo carattere maiuscolo
- stampa a standard output la forma diminutiva (-ino / -ina) dell'aggettivo inserito
- stampa a standard output il grado superlativo assoluto (-issimo / -issima) dell'aggettivo inserito

Ad esempio, se viene inserita la stringa "bello", il programma visualizzerà l'output

Aggettivo inserito:	Bello
Forma diminutiva:	Bellino
Superlativo assoluto:	Bellissimo

Se invece viene inserita la stringa "cara", il programma visualizzerà l'output

Aggettivo inserito:	Cara
Forma diminutiva:	Carina
Superlativo assoluto:	Carissima

Suggerimento: studiare la documentazione della classe **String** e trovare i metodi utili a risolvere il problema.

Laboratorio 4

Esercizio 1

Argomento: decisioni; confronto tra stringhe

Scrivere un programma che

- chiede all'utente di inserire tre stringhe (una per riga)
- visualizza le stringhe in ordine lessicografico crescente (una per riga)

Esercizio 2

Argomento: decisioni; confronto tra numeri reali

Scrivere un programma che

- legge due numeri in virgola mobile
- visualizza un messaggio che dice se i due numeri sono o meno *approssimativamente* uguali

I due numeri sono, in questo (semplificato) caso, da considerarsi *approssimativamente* uguali se, quando vengono arrotondati con due cifre decimali dopo la virgola, differiscono per meno di **0.01**.

Esempio: i due numeri **2.0** e **1.99998** vengono considerati approssimativamente uguali. I due numeri **0.999** e **0.991** non vengono considerati approssimativamente uguali.

Esercizio 3

Argomento: decisioni e operatori booleani

Scrivere un programma che segnala all'utente se il numero intero positivo che ha introdotto corrisponde ad un anno bisestile oppure no

Suggerimento:

- un anno bisestile è divisibile per 4. Fanno eccezione gli anni divisibili per 100, che non sono bisestili, e gli anni divisibili per 400, che invece sono bisestili: tali eccezioni esistono però solo dopo l'adozione del calendario gregoriano, che avvenne nel 1582.

Esercizio 4

Argomento: decisioni e operatori booleani

Scrivere un programma **SimpleTriangoloTester**, che riceve da standard input tre numeri interi positivi che rappresentano le lunghezze dei lati di un triangolo e che invia a standard output una stringa contenente le seguenti informazioni:

- relativamente ai lati: equilatero, isoscele, scaleno
- relativamente agli angoli: acutangolo, rettangolo, ottusangolo

Esempi

- se vengono inseriti i numeri "3 4 5", il programma visualizzerà la stringa "triangolo scaleno rettangolo".
- se vengono inseriti i numeri "5 7 7", il programma visualizzerà la stringa "triangolo isoscele acutangolo".
- se vengono inseriti i numeri "5 3 3", il programma visualizzerà la stringa "triangolo isoscele ottusangolo".
- se vengono inseriti i numeri "3 3 3", il programma visualizzerà la stringa "triangolo equilatero" (non serve l'informazione relativa agli angoli perché i triangoli equilateri sono sempre acutangoli avendo tre angoli uguali pari a $\pi/3$).
- se vengono inseriti i numeri "3 4 8", il programma visualizzerà la stringa "non è un triangolo" (non sempre tre lati rappresentano un triangolo).

Alcuni suggerimenti (da non leggere subito ma da consultare solo in caso di difficoltà) sono disponibili [a questo link](#).

Esercizio 5

Argomento: decisioni e operatori booleani

Scrivere un programma che riceve da standard input la data di compleanno dell'utente (in formato giorno mese), e visualizza a standard output l'oroscopo corrispondente. Il programma deve gestire correttamente anche il caso in cui l'input non corrisponda al formato prescritto.

Esempio: Se viene inserito l'input "26 7" (ovvero, 26 luglio), il programma potrà visualizzare la seguente stringa

```
LEONE
Amore:    4/5
Amicizia: 3/5
Lavoro:   3/5
```

Se invece l'input inserito non è interpretabile come una data in formato giorno mese (ad esempio, l'input "43 21", oppure l'input "32 1", oppure l'input "30 2"), il programma dovrà visualizzare la stringa

```
L'input inserito non è una data.
```

Suggerimenti: le associazioni tra date di nascita e segni zodiacali possono essere reperite in rete, ad esempio [qui](#). Le stringhe contenenti gli oroscopi di ciascun segno possono venire inventate a vostro piacimento (come si fa solitamente con gli oroscopi...?), l'unico requisito è che ciascuna di esse deve contenere il nome del segno corrispondente.

Attenzione: gli esercizi seguenti hanno come argomento la progettazione ed il collaudo di classi. E' bene seguire le seguenti fasi:

1. Progettare la classe

- Stabilire quali sono le caratteristiche essenziali degli oggetti della classe, e fare un elenco delle operazioni che sarà possibile compiere su di essi: processo di *astrazione*
- Definire e scrivere l'*interfaccia pubblica*. Ovvero, scrivere l'istestazione della classe, definire i costruttori ed i metodi pubblici da realizzare, e scrivere la *firma* (specificatore di accesso, tipo di valore restituito, nome del metodo, eventuali parametri espliciti) di ciascuno di essi.

Consiglio: se un metodo non restituisce valori (ovvero il tipo del valore restituito è void), scrivete inizialmente un corpo *vuoto*, ovvero `{}`. Se un metodo restituisce valori non void, scrivete inizialmente un corpo fittizio contenente solo un enunciato di return: ad esempio `{return 0;}` per metodi che restituiscono valori numerici o char, `{return false;}` per metodi che restituiscono valori booleani, `{return null;}` per metodi che restituiscono riferimenti ad oggetti. Con questi accorgimenti il codice compilerà correttamente fin dall'inizio del vostro lavoro. Quando poi scriverete i metodi, modificherete le istruzioni di return secondo quanto richiesto da ciascun metodo.

- Definire le *variabili di esemplare* ed eventuali *variabili statiche*. È necessario individuare tutte le variabili necessarie (quante? quali? dipende dalla classe!). Per ciascuna di esse si deve, poi, definire tipo e nome.

2. Realizzare la classe

- Verificate le impostazioni del vostro editor di testi, al fine rendere più efficiente il vostro lavoro di programmazione. In particolare, verificate ed eventualmente modificate le impostazioni per i rientri di tabulazione (valori tipici sono di 3 o 4 caratteri), e visualizzate i numeri di riga.
- Scrivere il codice dei metodi.

Consiglio: non appena si è realizzato un metodo, si deve compilare e correggere gli errori di compilazione (se il corpo del metodo è particolarmente lungo e complicato, compilare anche prima di terminare il metodo). Non aspettate di aver codificato tutta la classe per compilare! Altrimenti vi troverete, molto probabilmente, a dover affrontare un numero elevato di errori, con il rischio di non riuscire a venirne a capo in un tempo ragionevole (come quello a disposizione per la prova d'esame...).

3. Collaudare la classe. Ovvero scrivere una *classe di collaudo* contenente un metodo main, all'interno del quale vengono definiti e manipolati oggetti appartenenti alla classe da collaudare.

- E' possibile scrivere ciascuna classe in un file diverso. In tal caso, ciascun file avrà il nome della rispettiva classe ed avrà l'estensione .java. Tutti i file vanno tenuti nella stessa cartella, tutti i file vanno compilati separatamente, solo la classe di collaudo (contenente il metodo main) va eseguita.
- E' possibile scrivere tutte le classi in un unico file. In tal caso, il file .java deve contenere una sola classe public. In particolare, la classe contenente il metodo main deve essere public mentre la classe (o le classi) da collaudare non deve essere public (non serve scrivere private, semplicemente non si indica l'attributo public). Il file .java deve avere il nome della classe public

Esercizio 6

Argomento: progettazione e collaudo di classi

Progettare una versione modificata della classe **BankAccount** vista a lezione. L'interfaccia pubblica della versione modificata della classe deve contenere, oltre ai metodi precedentemente definiti e realizzati, anche il seguente metodo:

```
public void addInterest(double rate)
```

Questo metodo aggiunge al saldo del conto gli interessi, calcolati al tasso percentuale **rate** specificato come parametro esplicito. Il metodo deve essere realizzato in maniera tale che il parametro esplicito **rate** non possa essere un valore negativo. Ad esempio, dopo l'esecuzione di questi enunciati

```
BankAccount account = new BankAccount(2000);  
account.addInterest(2); //interessi al 2%
```

il saldo di **account** è di 2040€.

Inoltre il costruttore **BankAccount(double initialBalance)** e i metodi **deposit(double amount)** e **withdraw(double amount)** devono essere realizzati in maniera tale che

- il loro parametro esplicito (**initialBalance** o **amount**) non possa essere un valore negativo
- il saldo del conto corrente non possa diventare negativo dopo un prelievo.

Progettate una versione modificata della classe **BankAccountTester** vista a lezione, che

- riceva da input standard due numeri in virgola mobile, che specificano rispettivamente il saldo iniziale e il tasso percentuale di interesse
- crei un conto bancario accreditando un saldo iniziale ricevuto da standard input come numero in virgola mobile
- usi il metodo **addInterest** per accreditare sul conto bancario gli interessi di due anni, calcolati su un tasso di interesse ricevuto da standard input come numero in virgola mobile
- visualizzi il saldo del conto dopo due anni
- effettui un prelievo dal conto, pari ad una cifra ricevuta da standard input come numero in virgola mobile
- visualizzi il saldo del conto dopo il prelievo.

Verificare in particolare il funzionamento del programma quando vengono inseriti valori non validi (saldo iniziale negativo, tasso di interesse negativo, prelievo negativo o superiore al saldo).

Esercizio 7

Argomento: progettazione e collaudo di classi

Realizzare una classe **Product** (ovvero, prodotto).

Un prodotto è caratterizzato da un nome e da un prezzo in €. La classe **Product** deve contenere il costruttore

```
public Product(String name, double price)
```

che crea un nuovo prodotto caratterizzato dal nome **name** e dal prezzo in euro **price**. La classe deve inoltre contenere i seguenti metodi pubblici:

```
public String getName()  
public double getPrice()  
public void reducePrice(double rate)
```

I primi due sono *metodi di accesso*, e restituiscono rispettivamente il nome e il prezzo del prodotto. Il terzo è un *metodo modificatore*, che riduce il prezzo del prodotto secondo lo sconto percentuale definito dal parametro esplicito **rate**.

Scrivere una classe di collaudo che

- crea due prodotti ricevendo i parametri di creazione da standard input
 - stampa i due prodotti (ovvero il loro nome e prezzo) a standard output, in ordine decrescente di prezzo
 - applica uno sconto al prodotto più caro, secondo una percentuale ricevuta da standard input
 - stampa nuovamente i due prodotti (ovvero il loro nome e prezzo) a standard output, in ordine di prezzo
-

Esercizio 8 (impegnativo)

Argomento: progettazione e collaudo di classi

Si scriva la classe **Cerchio**, che descrive un cerchio, la cui interfaccia pubblica è specificata [a questo link](#).

Un cerchio è caratterizzato da due dati: il centro e il raggio. A sua volta il centro, essendo un punto di un piano, è caratterizzato da due dati: ascissa e ordinata.

Suggerimento: dati due cerchi, siano $O=(x,y)$ e R il centro e il raggio del cerchio C , e $O'=(x',y')$ e R' il centro e il raggio del cerchio C' . Allora

- C' è interno a C se $OO' < R - R'$
- C' è tangente interno a C se $OO' = R - R'$
- C' è secante a C se $R - R' < OO' < R + R'$
- C' è tangente esterno a C se $OO' = R + R'$
- C' è esterno a C se $OO' > R + R'$

Collaudare la classe **Cerchio** con la classe di prova **CerchioTester**, che legge i dati da standard input, e che è disponibile [a questo link](#).

Esercizio 9

Argomento: enum, switch

Crea un enum chiamato "Emozione" con almeno 5 emozioni diverse (es. FELICE, TRISTE, ARRABBIATO, SORPRESO, ANNOIATO). Scrivi un metodo che accetta un'emozione come input e restituisce un'emoji corrispondente usando uno **switch**. Poi, crea un main (nello stesso file .java) che permette all'utente di inserire un'emozione e visualizza l'emoji corrispondente.

Esercizio 10

Argomento: enum, switch

Il Gioco dei Pianeti

Sviluppa un programma che calcola il peso di una persona su diversi pianeti del sistema solare. Usa i seguenti fattori di gravità relativi alla Terra:

Mercurio: 0.38

Venere: 0.91

Marte: 0.38

Giove: 2.34

Saturno: 1.06

Urano: 0.92

Nettuno: 1.19

Segui questi passaggi:

Definisci un **enum** chiamato Pianeta che include questi pianeti del sistema solare con i loro fattori di gravità.

Implementa un **metodo** nell'enum che calcola il peso su un pianeta dato il peso sulla Terra.

Nel metodo **main**:

Chiedi all'utente di inserire il suo peso sulla Terra.

Chiedi all'utente di scegliere un pianeta.

Calcola e mostra quanto peserebbe l'utente su quel pianeta.

Usa uno **switch** per gestire i diversi pianeti nel calcolo del peso.

Laboratorio 5

Laboratorio 5

Esercizio 1

Argomento: gestione dell'input tramite "ciclo e mezzo"

Scrivere un programma che legga una sequenza di valori in virgola mobile inseriti dall'utente: il numero di valori non è predeterminato, dopo l'introduzione di ciascun valore l'utente deve premere il tasto "Invio" e la sequenza di valori termina non appena l'utente introduce una sequenza predeterminata di caratteri che non rappresenta un numero in virgola mobile.

Dopo aver letto tutti i valori inseriti dall'utente, il programma ne deve visualizzare il valore medio e la deviazione standard.

Suggerimento: Il valore medio di una sequenza di valori è (ovviamente) uguale alla somma di tutti i valori divisa per il numero di valori. La deviazione standard D si calcola con la formula seguente:

$$D = \text{Math.sqrt}((A - B*B/n)/(n-1))$$

dove n è il numero di valori, B è la somma di tutti i valori e A è la somma dei quadrati di tutti i valori. La formula è valida soltanto per $n > 1$; per $n = 1$ la deviazione standard vale 0, per definizione.

Gestire correttamente anche il caso in cui il numero di valori è zero: in tal caso il valore medio dei valori si assume essere convenzionalmente uguale a zero, così come la deviazione standard.

Esercizio 2

Argomento: cicli, decisioni, confronto tra numeri interi

Scrivere un programma che calcoli il massimo comun divisore (MCD) fra due numeri interi positivi m e n acquisiti da standard input, e visualizzi il risultato a standard output.

Si usi il ben noto **Algoritmo di Euclide** per il calcolo del MCD tra due numeri interi positivi m ed n (con $m > n$):

- finché il resto della divisione di m per n è diverso da zero
 - il nuovo valore di m è il precedente valore di n
 - il nuovo valore di n è il resto della divisione del precedente valore di m per il precedente valore di n
- Quando il resto è zero, allora il MCD è l'attuale valore di n

Esercizio 3

Argomento: gestione dell'input tramite "ciclo e mezzo", cicli annidati

Scrivere un programma che calcoli e visualizzi in ordine crescente, un numero per riga, tutti i numeri primi fino ad un valore massimo introdotto dall'utente; se l'utente inserisce un numero intero negativo, il programma deve ripetere la richiesta di inserzione finché non viene introdotto un numero intero positivo.

Suggerimento: Ricordare che un numero intero positivo è primo se non ha divisori (ovvero non è divisibile con resto nullo) oltre al numero uno e se stesso (si noti che, quindi, uno è il più piccolo numero primo).

Approfondimento: È possibile dimostrare che $n > 0$ è un numero primo se e solo se non ha divisori nell'insieme $[2, \text{radice di } n]$ (invece che in tutto l'insieme $[2, n-1]$).

Esercizio 4

Argomento: cicli, decisioni e variabili booleane; manipolazione di stringhe

Scrivere un programma che verifica se una stringa, introdotta dall'utente tramite lo standard input (un'intera riga), è palindroma. **Suggerimento:**

- Una stringa è palindroma se è composta da una sequenza di caratteri (anche non alfabetici) che possa essere letta allo stesso identico modo anche al contrario (es. "radar", "anna", "inni", "xyz%u%zyx").

Il programma deve eseguire il minimo numero di confronti necessari per determinare se la stringa è una palindroma.

Verificare il corretto funzionamento del programma con:

- una stringa palindroma di lunghezza pari
- una stringa palindroma di lunghezza dispari
- una stringa non palindroma di lunghezza pari
- una stringa non palindroma di lunghezza dispari
- una stringa di lunghezza unitaria (che è ovviamente palindroma)
- una stringa di lunghezza zero (che è ragionevole definire palindroma)

Esercizio 5

Argomento: gestione dell'input tramite "ciclo e mezzo", cicli, decisioni e variabili booleane;

Scrivere un programma che

- riceva da standard input un numero intero positivo **n** (se l'utente inserisce un numero intero negativo, il programma deve ripetere la richiesta di inserzione finché non viene introdotto un numero intero positivo.)
- calcoli e mostri a standard output la scomposizione in fattori primi di **n**.

Esempio: se viene inserito il numero 12, il programma deve visualizzare a standard output il messaggio "12 = 1 * 2 * 2 * 3".

Attenzione: gli esercizi seguenti hanno come argomento la progettazione ed il collaudo di classi. E' bene seguire le seguenti fasi:

1. Progettare la classe

- Stabilire quali sono le caratteristiche essenziali degli oggetti della classe, e fare un elenco delle operazioni che sarà possibile compiere su di essi: processo di *astrazione*
- Definire e scrivere l'*interfaccia pubblica*. Ovvero, scrivere l'intestazione della classe, definire i costruttori ed i metodi pubblici da realizzare, e scrivere la *firma* (specificatore di accesso, tipo di valore restituito, nome del metodo, eventuali parametri espliciti) di ciascuno di essi.
Consiglio: se un metodo non restituisce valori (ovvero il tipo del valore restituito è void), scrivete inizialmente un corpo vuoto, ovvero {}. Se un metodo restituisce valori non void, scrivete inizialmente un corpo fittizio contenente solo un enunciato di return: ad esempio {return 0;} per metodi che restituiscono valori numerici o char, {return false;} per metodi che restituiscono valori booleani, {return null;} per metodi che restituiscono riferimenti ad oggetti. Con questi accorgimenti il codice compilerà correttamente fin dall'inizio del vostro lavoro. Quando poi scriverete i metodi, modificherete le istruzioni di return secondo quanto richiesto da ciascun metodo.
- Definire le *variabili di esempio* ed eventuali *variabili statiche*. È necessario individuare tutte le variabili necessarie (quante? quali? dipende dalla classe!). Per ciascuna di esse si deve, poi, definire tipo e nome.

2. Realizzare la classe

- Verificate le impostazioni del vostro editor di testi, al fine rendere più efficiente il vostro lavoro di programmazione. In particolare, verificate ed eventualmente modificate le impostazioni per i rientri di tabulazione (valori tipici sono di 3 o 4 caratteri), e visualizzate i numeri di riga. A questo proposito si vedano anche i **Consigli per la produttività 5.1** sul libro di testo)
- Scrivere il codice dei metodi.
Consiglio: non appena si è realizzato un metodo, si deve compilare e correggere gli errori di compilazione (se il corpo del metodo è particolarmente lungo e complicato, compilare anche prima di terminare il metodo). Non aspettate di aver codificato tutta la classe per compilare! Altrimenti vi troverete, molto probabilmente, a dover affrontare un numero elevato di errori, con il rischio di non riuscire a venire a capo in un tempo ragionevole (come quello a disposizione per la prova d'esame...).

3. Collaudare la classe. Ovvero scrivere una *classe di collaudo* contenente un metodo main, all'interno del quale vengono definiti e manipolati oggetti appartenenti alla classe da collaudare.

- E' possibile scrivere ciascuna classe in un file diverso. In tal caso, ciascun file avrà il nome della rispettiva classe e avrà l'estensione .java. Tutti i file vanno tenuti nella stessa cartella, tutti i file vanno compilati separatamente, solo la classe di collaudo (contenente il metodo main) va eseguita.
- E' possibile scrivere tutte le classi in un unico file. In tal caso, il file .java deve contenere una sola classe public. In particolare, la classe contenente il metodo main deve essere public mentre la classe (o le classi) da collaudare non deve essere public (non serve scrivere private, semplicemente non si indica l'attributo public). Il file .java deve avere il nome della classe public

Esercizio 6

Argomento: progettazione e collaudo di classi

Si scriva la classe **MyComplex**, che rappresenta un numero complesso, la cui interfaccia pubblica è specificata [a questo link](#) (e contiene come commenti le uniche nozioni sui numeri complessi che è necessario conoscere ai fini di questo esercizio).

Collaudare la classe **MyComplex** con la classe di prova **MyComplexTester**, disponibile [a questo link](#). Il programma riceve due numeri complessi da standard input, uno per riga, nel formato "x y ", dove x (parte reale) e y (parte immaginaria) sono separate dal carattere spazio ' '. Fornisce a standard output somma, sottrazione, prodotto, divisione, elementi inversi e coniugati dei numeri. Provare con numeri semplici, in modo da verificare i risultati ottenuti.

Effettuare il collaudo in due modi: ciascuna delle due classi in un file .java ad essa omonimo (con entrambe le classi public), oppure entrambe le classi in un solo file **MyComplexTester.java** (con **MyComplexTester** public e **MyComplex** no)

Esercizio 7 (quasi impegnativo)

Argomento: progettazione e collaudo di classi

Si scriva la classe **Triangolo**, che descrive un triangolo, la cui interfaccia pubblica è specificata [a questo link](#). Alcuni suggerimenti (da non leggere subito ma da consultare solo in caso di difficoltà) sono disponibili [a questo link](#).

Collaudare la classe **Triangolo** con la classe di prova **TriangoloTester**, che legge i dati da standard input, e che è disponibile [a questo link](#). Come per l'esercizio precedente, effettuare il collaudo in due modi: ciascuna delle due classi in un file .java ad essa omonimo (con entrambe le classi public), oppure entrambe le classi in un solo file **TriangoloTester.java** (con **TriangoloTester** public e **Triangolo** no)

Esercizio 8 (impegnativo)

Argomento: progettazione e collaudo di classi

Scrivere un programma per la risoluzione di equazioni di secondo grado $ax^2 + bx + c = 0$. La realizzazione del programma va articolata in due fasi:

1. Progettare e scrivere una classe **QuadraticEquation**, la cui interfaccia pubblica è specificata [a questo link](#). Questa interfaccia lascia aperte alcune scelte di progetto, che ciascuno potrà effettuare in autonomia, eventualmente aggiungendo altri metodi e/o campi di esemplare alla classe. In particolare
 - Si può decidere che un oggetto di tipo **QuadraticEquation** sia *immutabile*, ovvero non possa più essere modificato una volta creato. Oppure si può dare la possibilità (tramite scrittura di opportuni *metodi modificatori*) di modificare oggetti di tipo **QuadraticEquation** già esistenti
 - Si può prevedere la presenza di *metodi di accesso*, che restituiscano informazioni sullo stato di un oggetto di tipo **QuadraticEquation**
 - Si può migliorare il progetto in modo da gestire correttamente i seguenti casi degeneri
 - $a=b=0$ e $c \neq 0$ (l'equazione non ha soluzioni)
 - $a=0$, $b \neq 0$ (l'equazione ha una soluzione)
 - $a=b=c=0$ (l'equazione ha infinite soluzioni)
2. Scrivere un programma di collaudo **QuadraticEquationTester** che utilizzi la classe **QuadraticEquation**. Il programma deve
 - leggere in ingresso i valori dei parametri a, b, c
 - (se esistono soluzioni reali) visualizzare le due soluzioni reali
 - (se non esistono soluzioni reali) visualizzare un messaggio di segnalazione

Come per gli esercizi precedenti, effettuare il collaudo in due modi: ciascuna delle due classi in un file .java ad essa omonimo (con entrambe le classi public), oppure entrambe le classi in un solo file omonimo della classe di test, con la classe di test pubblica e l'altra no.

Esercizio 9 (molto più semplice)

Argomento: progettazione e collaudo di classi

1. Creare una classe chiamata "Libro" con i seguenti attributi privati:
 - titolo (String)
 - autore (String)

- numeroPagine (int)
- prezzoBase (double)

2. Implementare i seguenti metodi nella classe Libro:

- Un costruttore che accetta tutti gli attributi come parametri
- Metodi getter per tutti gli attributi (getTitolo, getAutore, ecc.)
- Un metodo "calcolaPrezzo" che restituisce il prezzo del libro:
 - Se il numero di pagine è superiore a 300, aggiungere un 10% al prezzo base
- Un metodo "toString" che restituisce una stringa con tutte le informazioni del libro

3. Creare una classe di test chiamata "LibroTest" con un metodo main che:

- Crea almeno due oggetti Libro con dati diversi
- Stampa le informazioni di ciascun libro
- Calcola e stampa il prezzo di ciascun libro

Laboratorio 6

Esercizio 1

Argomento: gestione delle eccezioni; uso della terminazione di input; lettura/scrittura file

Modificare l'esercizio 1 del Laboratorio 5 come segue.

Il programma che calcola il valore medio e la deviazione standard dei dati inseriti dall'utente non dovrà accettare il dato in ingresso quando questo:

- non è un numero in virgola mobile
- non può essere memorizzato in una variabile di tipo double

In tali casi, il programma dovrà riproporre all'utente di inserire un numero in virgola mobile, e continuare così finché ciò non accade. Fare in modo che il programma riceva in input i dati scritti nel file di testo [input_double.txt](#) (contenente un insieme di dati di ingresso, numeri in virgola mobile e non solo, uno per riga) e ne calcoli valore medio e deviazione standard.

Modificare l'esercizio 3 del Laboratorio 4 come segue. Il programma che segnala all'utente se il numero intero positivo che ha introdotto corrisponde o meno ad un anno bisestile dovrà continuare a chiedere l'inserimento di un nuovo numero finché non viene inserito il valore "0" (valore sentinella). Inoltre il programma non dovrà accettare il dato in ingresso quando questo:

- non è un numero intero
- non è un numero positivo
- non può essere memorizzato in una variabile di tipo int

In tali casi, il programma dovrà riproporre all'utente di inserire un numero positivo, e continuare così finché ciò non accade. Usare il meccanismo di reindirizzamento dell'input standard in maniera tale che il programma riceva in input i dati scritti nel file di testo: [input_years.txt](#) (contenente un insieme di dati di ingresso, numeri interi e non solo, uno per riga) e segnali se corrispondono o meno ad anni bisestili.

Esercizio 2

Argomento: Lettura e scrittura di file, gestione delle eccezioni, "tokenizzazione" di stringhe

Scrivere un programma che

- Riceva dall'input standard due nomi di file di testo, uno in lettura e uno in scrittura
- Apra in lettura il primo file e ne legga il contenuto
- Crei e apra in scrittura il secondo file
- Copi nel secondo file il contenuto del primo, opportunamente modificato in modo che tutte le parole abbiano la prima lettera maiuscola e le seguenti minuscole

Provare il programma usando il file [vispateresa.txt](#) come file di input e creando (ad esempio) il file vispateresa2.txt in output.

Approfondimento: modificare il programma in modo che riconosca come due parole distinte anche quelle separate da un apostrofo. Ad esempio, se il file in lettura contiene le parole

LA vispA teresa AVEA tra l'erBETTa

al termine dell'esecuzione il secondo file dovrà contenere il testo

La Vispa Teresa Avea Tra L'Erbetta

Suggerimento importante: studiare la **documentazione di Scanner**, e verificare che usando opportuni metodi è possibile **usare un insieme di caratteri delimitatori diverso da quello di default**.

Esercizio 3

Argomento: Manipolazione di stringhe e caratteri, cicli annidati

Scrivere un programma che

- chiede all'utente di introdurre due stringhe (una per riga), **s1** e **s2**; ciascuna stringa è costituita da tutti i caratteri presenti sulla riga, compresi eventuali spazi iniziali, finali e/o intermedi
- verifica se la seconda stringa, **s2**, è una sottostringa di **s1**, ossia se esiste una coppia di numeri interi, x e y, per cui **s1.substring(x, y)** restituisce una stringa uguale a **s2** (al termine della verifica viene visualizzato un messaggio opportuno)

Attenzione:

Il programma può usare, della classe

String,

i soli metodi:

charAt

e

length

(in particolare, quindi, non va utilizzato il metodo

substring

appena menzionato).

Verificare che il programma gestisca correttamente la situazione in cui **s2** è la stringa vuota (che, in base alla definizione precedente, è sottostringa di qualsiasi stringa).

Esercizio 4 (impegnativo)

Argomento: Manipolazione di stringhe e caratteri, cicli annidati

Scrivere un programma che

- chiede all'utente di introdurre due stringhe (una per riga), s1 e s2; ciascuna stringa è costituita da tutti i caratteri presenti sulla riga, compresi eventuali spazi iniziali, finali e/o intermedi
- verifica se la seconda stringa **s2** è una sottosequenza di **s1** (al termine della verifica viene visualizzato un messaggio opportuno).

Il programma può usare, della classe **String**, i soli metodi **charAt** e **length**.

Una stringa **s2** è una sottosequenza di un'altra stringa **s1** se e solo se tutti i caratteri di **s2** sono presenti in **s1** nello stesso ordine (anche se in posizioni diverse). Ad esempio, **gatto** è una sottosequenza di **gratto** e **xyz** è una sottosequenza di **2xpppyqz** ma non di **yxz**.

Ovviamente, una sottostringa e' anche una sottosequenza, ma non viceversa.

Esercizio 5 (lungo... ma non impegnativo)

Argomento: Classi e variabili statiche; gestione di input ed eccezioni, "tokenizzazione" di stringhe

Modificare la classe **BankAccount** vista a lezione, con le seguenti diverse specifiche:

- L'interfaccia pubblica della classe è costituita dal costruttore di default, dai due metodi modificatori **deposit** e **withdraw** (che effettuano versamenti e prelievi), e dai due metodi di accesso **getBalance** e **getAccountNumber** (che restituiscono valore del saldo e numero del conto corrente).
 - Un conto corrente possiede un numero di conto progressivo: ad un nuovo conto corrente viene assegnato il primo numero intero disponibile (ovvero 1+ il numero dell'ultimo conto creato in precedenza).
- I metodi

deposit

e

withdraw

restituiscono un valore di tipo logico, true se e solo se l'operazione è ammissibile e va a buon fine (ma non devono visualizzare nessun messaggio d'errore); se l'operazione non è ammissibile, il saldo del conto non deve essere modificato

- un versamento va a buon fine se e solo se l'importo che viene versato è maggiore di zero
- un prelievo va a buon fine se e solo se l'importo che viene prelevato è maggiore di zero e minore o uguale al saldo del conto

Scrivere poi un programma **BankAccountTester** che crea un conto bancario (esemplare della classe appena progettata) e accetta ripetutamente comandi dall'utente, introdotti da tastiera, finché l'utente non introduce il comando **Q** di terminazione del programma. I comandi disponibili sono:

Q	Quit: termina il programma
B	Balance: visualizza il saldo del conto
D x	Deposit: versa nel conto la somma x
W x	Withdraw: preleva dal conto la somma x
A x	Add interest: accredita sul conto gli interessi, calcolati in base alla percentuale x del saldo attuale (x deve essere positivo)

Nei comandi che necessitano di un argomento, questo è separato dal comando mediante uno o più spazi (utilizzando, per esempio, un ciclo **for** opportuno per esaminare tutti i caratteri). La lettera che indica il comando può essere maiuscola o minuscola.

Un esempio di esecuzione del programma è il seguente (in corsivo i comandi introdotti dall'utente, in grassetto ciò che viene scritto dal programma):

```
Comando? (Q, B, D, W, A)
B
Saldo attuale: 0.0
Comando? (Q, B, D, W, A)
w 10
Prelievo non autorizzato
Comando? (Q, B, D, W, A)
D 10
Versamento effettuato: 10.0
Comando? (Q, B, D, W, A)
D 0
Versamento non corretto
Comando? (Q, B, D, W, A)
w 5
Prelievo effettuato: 5.0
Comando? (Q, B, D, W, A)
b
Saldo attuale: 5.0
Comando? (Q, B, D, W, A)
Z2
Comando? (Q, B, D, W, A)
A 5
Interessi calcolati e accreditati: 0.25
Comando? (Q, B, D, W, A)
B
Saldo attuale: 5.25
Comando? (Q, B, D, W, A)
q 5
Comando? (Q, B, D, W, A)
Q
Arrivederci
```

Tale esempio va anche inteso come specifica dei messaggi che devono essere visualizzati dal programma. Il programma deve gestire tutte le possibili condizioni di dati in ingresso senza essere mai interrotto da eccezioni; nei casi in cui manchino specifiche di comportamento del programma, assumere ipotesi ragionevoli.

Esercizio 6

Argomento: Progettare e collaudare classi, lanciare e catturare eccezioni

Realizzare una classe **FactorGenerator** che effettui la scomposizione di un numero intero positivo nei suoi fattori primi, con un comportamento simile a quello della classe **Scanner** per "tokenizzare" stringhe. La classe deve avere la seguente interfaccia pubblica:

- un costruttore che riceve come unico parametro un numero intero e verifica che sia positivo e maggiore di uno (lanciando **IllegalArgumentException** in caso contrario)
- un metodo non statico **nextFactor** che non riceve parametri espliciti e che, ad ogni successiva invocazione, restituisce uno dei fattori primi in cui viene scomposto il numero fornito nel costruttore; invocando un numero sufficiente di volte tale metodo si ottengono tutti e soli i fattori primi del numero (eventualmente ripetuti), in modo che moltiplicandoli si ottenga il numero originario (ad esempio, il numero 150 viene scomposto nei suoi fattori primi 2, 3, 5, 5); se il metodo viene invocato dopo aver ottenuto tutti i fattori primi, esso lancia **IllegalStateException**
- un metodo non statico **hasNextFactor** che non riceve parametri espliciti e che restituisce il valore booleano **true** se e solo se esistono fattori primi non ancora restituiti da **nextFactor**

Scrivere un programma **FactorGeneratorTester** che:

- legge dallo standard input un numero intero maggiore di uno
- crea un esemplare di **FactorGenerator** fornendo come parametro il numero ricevuto
- visualizza sullo standard output tutti i fattori primi del numero

Esercizio 7

Argomento: Collaudare classi, catturare eccezioni

Usando la classe **FactorGenerator** sviluppata nell'esercizio precedente, scrivere un programma **PrimeNumberTester** che:

- legge dallo standard input un numero intero positivo
- visualizza sull'uscita standard un messaggio che specifica se il numero introdotto è un numero primo oppure no (ricordare che un numero è primo se i suoi unici fattori primi sono 1 e il numero stesso)

Si faccia attenzione alla gestione del valore **1**, che è accettabile (ed è un numero primo per definizione) ma che non viene gestito dalla classe **FactorGenerator**.

Dopo la fase di lettura del dato in ingresso, il programma non deve avere cicli.

Laboratorio 7

Esercizio 1

Argomento: algoritmi su array

Studiare la classe `ArrayAlgs` vista a lezione. Testarne i metodi scrivendo una classe di collaudo `ArrayAlgsTester` che

- crea un array di numeri interi casuali, la cui dimensione e intervallo di variabilità è specificata dall'utente
- accetta ripetutamente comandi dall'utente, introdotti da tastiera, finché l'utente non introduce il comando di terminazione del programma.

I comandi disponibili sono:

Q	Quit: termina il programma
P	Print: stampa il contenuto dell'array
m	min: calcola il minimo valore contenuto nell'array
M	Max: calcola il massimo valore contenuto nell'array
r i	remove index: rimuove dall'array l'elemento di indice i
R i	Remove-sorted index: rimuove dall'array l'elemento di indice i, mantenendo l'ordine degli altri elementi
I i v	Insert index value: inserisce il valore value nella posizione specificata dall'indice index

Esercizio 2

Argomento: uso di array

Il Crivello di Eratostene è un noto algoritmo per la ricerca dei numeri primi minori di un certo valore massimo MAX, ed è così specificato:

- si predispose un array di MAX valori booleani. Ogni elemento dell'array "rappresenta" il numero intero corrispondente al proprio indice nell'array.
- Se e solo se l'elemento è **true**, allora il numero corrispondente è stato eliminato dall'insieme dei numeri primi, cioè non è un numero primo.
- All'inizio si suppone che tutti i numeri siano primi; successivamente, si considera ciascun numero intero maggiore di uno, in ordine crescente, e si eliminano tutti i numeri che ne sono multipli, contrassegnandoli opportunamente nell'array.
- Al termine, i numeri rimasti (ovvero quelli i cui corrispondenti elementi nell'array valgono ancora **false**) sono tutti e soli i numeri primi cercati, non essendo multipli di alcun numero.

Scrivere un programma che realizza il Crivello di Eratostene per identificare i numeri primi minori di un valore (intero positivo) MAX fornito dall'utente attraverso l'ingresso standard. Verificare il corretto funzionamento del programma con:

- MAX = 1 (non viene visualizzato nessun numero, perché non esiste nessun numero primo minore di 1)
- MAX = 2 (viene visualizzato soltanto il numero 1)
- MAX = 3 (vengono visualizzati soltanto i numeri 1 e 2)
- MAX = 4 e MAX = 5 (vengono visualizzati soltanto i numeri 1, 2 e 3)
- MAX = 6 (vengono visualizzati soltanto i numeri 1, 2, 3 e 5)

Esercizio 3

Argomento: manipolazione di stringhe, passaggio di parametri dalla riga di comando

Scrivere un programma che identifichi la più lunga sottostringa appartenente a due stringhe ricevute come argomenti sulla riga di comando.

Esempio: la più lunga sottostringa comune alle due stringhe

```
PippoPluto2Paperino
MinnieAiuzzto2Zo
```

e` la stringa

to2

Osservazione: questo è un caso particolare di un problema più generale, ovvero la ricerca della più lunga sottosequenza comune tra due stringhe. Si tratta di un problema molto interessante in applicazioni biologiche, in particolare nello studio del DNA. Un filamento di DNA è composto da molecole chiamate basi, e le basi possibili sono quattro: adenina (abbreviata con la lettera A), citosina (C), guanina (G) e timina (T). Dunque un filamento di DNA è rappresentabile come una stringa composta con l'alfabeto dei quattro caratteri {A,C,G,T}. Una buona misura della somiglianza tra due filamenti di DNA (ovvero della somiglianza tra due organismi) è data proprio dalla lunghezza della massima sottosequenza comune tra i due filamenti.

Esercizio 4 (impegnativo)

Argomento: uso di array riempiti solo in parte, array paralleli, ricerca lineare in array, gestione delle eccezioni

Scrivere una classe eseguibile **StudentManager** che gestisce un elenco di studenti.

In una prima fase il programma riceve dall'input standard e memorizza un elenco di dati che rappresentano

- i nomi di un insieme di studenti
- il voto della prova scritta
- il voto della prova orale

I dati di ciascuno studente sono inseriti in una riga separati da uno spazio (prima il nome, poi il voto scritto, poi il voto orale). I dati sono terminati da una riga vuota.

Al termine dell'inserimento, il programma chiede all'utente di inserire un comando per identificare l'elaborazione da svolgere.

- se l'utente inserisce il comando Q l'esecuzione termina
- se l'utente inserisce il comando S il programma
 - chiede all'utente di inserire il cognome di uno studente
 - se lo studente è presente nell'insieme, ne visualizza il voto finale (calcolato come media aritmetica tra i voti della prova scritta e della prova orale), altrimenti segnala che il cognome e` errato
 - chiede un nuovo inserimento di comando (Q o S)
- se il comando inserito è diverso da Q o S, il programma segnala che il comando è errato e chiede un nuovo inserimento di comando

Suggerimenti:

1. Evitare l'uso di array paralleli. Scrivere una classe **Student**, che rappresenta il tipo di dato "studente" (specificato da nome, voto scritto, e voto orale), e usare un array di oggetti **Student** per memorizzare l'insieme di studenti nel programma.
2. La parte piu` impegnativa di questo esercizio e` la scrittura della classe **StudentManager**. Si consiglia di individuare le principali operazioni che la classe deve realizzare (ad esempio, creazione di un oggetto **Student**, ricerca di un oggetto **Student** nell'array, stampa della media dei voti) e per ciascuna di esse scrivere un metodo statico ausiliario invocato dal metodo **main**.
3. Scrivere la classe in due fasi: in una prima fase realizzare le operazioni richieste senza curarsi di precondizioni e situazioni inaspettate; solo in una seconda fase affrontare il problema della gestione delle eccezioni (in particolare i metodi scritti potranno sia lanciare che catturare eccezioni).

Esercizio 5 (impegnativo)

Argomento: progettazione di classi, uso di array riempiti solo in parte, ricerca lineare in array, manipolazione di stringhe

Scrivere la classe **TextContainer** la cui interfaccia pubblica è definita [a questo link](#). La classe memorizza un testo e rende disponibili metodi per semplice analisi dei testi.

Suggerimento: memorizzare un testo in un array di stringhe e usare la tecnica degli array riempiti solo in parte. Ridimensionare dinamicamente gli array in modo che la classe possa memorizzare un numero indefinito di stringhe.

Scrivere poi la classe eseguibile **TextAnalyzer** che:

- legga un testo da standard input una riga alla volta e memorizzi le righe in un oggetto di classe **TextContainer**.
- esegua la seguente elaborazione
 - separi il testo in parole
 - elimini dalle parole del testo i caratteri non alfabetici

- renda minuscoli tutti i caratteri del testo
- scriva a standard output:
 - il testo originale
 - il numero delle parole contenute nel testo
 - la prima e l'ultima parola secondo l'ordine lessicografico
 - le 5 parole che compaiono più frequentemente nel testo, in ordine decrescente di occorrenza.

Provare la classi con re-indirizzamento dello standard input sul file [brickinthewall.txt](#), o su un qualsiasi altro file di testo.

Esercizio 6 (impegnativo)

Argomento: progettazione di classi, uso di array bidimensionali

Scrivere un programma per giocare a "tris" (tic-tac-toe in inglese), il classico gioco in cui due giocatori dispongono alternativamente un proprio contrassegno in una casella di una scacchiera 3 x 3 finché uno dei due non pone tre contrassegni in una fila orizzontale, verticale o diagonale.

X	O	
	X	O
		X

Il programma inizia visualizzando la scacchiera vuota, come segue (ogni puntino rappresenta una casella vuota)

```
|...|
|...|
|...|
```

e chiedendo al primo giocatore di inserire le coordinate della casella in cui vuole porre il suo contrassegno (che sarà un carattere X).

Le coordinate si indicano con due numeri interi (valori ammessi: 0, 1 o 2), il primo numero essendo l'indice di riga (a partire dall'alto) ed il secondo l'indice di colonna (a partire da sinistra).

Il programma deve verificare se la casella richiesta è libera oppure no. Nel primo caso visualizza la scacchiera aggiornata e chiede all'altro giocatore di inserire la propria mossa (verrà usato il contrassegno O). Nel secondo caso, invece, il programma fornisce una segnalazione d'errore, visualizza nuovamente la stessa scacchiera visualizzata in precedenza e chiede al giocatore di inserire una nuova mossa; analogo comportamento si verifica se il giocatore introduce delle coordinate non valide, ma il messaggio d'errore deve essere diverso.

Il programma deve essere in grado di segnalare la vittoria di uno dei due giocatori qualora questa avvenga, oppure di porre fine alla partita quando la scacchiera è piena senza che uno dei due giocatori abbia raggiunto la vittoria.

Al termine di una partita, il programma chiede al giocatore se intende giocare un'altra partita oppure no: nel secondo caso il programma termina.

Risolvere il problema in due passi

1. Scrivere una classe **Tris** che rappresenti la scacchiera e la cui interfaccia pubblica è definita [a questo link](#).
2. Scrivere poi una classe eseguibile che usi **Tris**, e che gestisca una partita a tris tra due giocatori, seguendo il comportamento descritto sopra.

Esercizio 7

Argomento: ricorsione semplice, argomenti sulla riga dei comandi

Scrivere una classe eseguibile **RecStringReverser** capace di invertire una stringa in modo ricorsivo (ad es., l'inverso della stringa "Hello" è la stringa "olleH").

- La stringa viene passata come argomento nella riga di comando
- Il risultato dell'inversione viene visualizzato a standard output

Esempio di uso:

```
$java RecStringReverser Hello
```

Suggerimento: scrivere un metodo statico *ricorsivo* **reverseString** che realizza l'algoritmo ricorsivo di inversione di stringhe, poi scrivere un metodo **main** che realizza il comportamento sopra specificato invocando il metodo **reverseString**. Il metodo **main** non deve eseguire alcuna azione se non riceve parametri.

Esercizio 8

Argomento: *ricorsione semplice, argomenti sulla riga dei comandi*

Scrivere una classe eseguibile **RecNumberPrinter**, che effettui la stampa dei primi **n** numeri interi secondo un algoritmo ricorsivo.

- Il massimo numero intero **n** (positivo) da stampare viene passato come argomento nella riga di comando
- Il risultato della stampa viene visualizzato a standard output

Suggerimento: scrivere un metodo statico *ricorsivo* **listNumbers(int n)** che realizza l'algoritmo ricorsivo restituendo una stringa contenente i numeri da 1 a n, in ordine crescente, separati da uno spazio. Poi scrivere un metodo **main** che realizza il comportamento sopra specificato invocando il metodo **listNumbers**. Il metodo **main** non deve eseguire alcuna azione se non riceve parametri.

Esempio: l'invocazione **listNumbers(5)** restituisce la stringa "1 2 3 4 5"

Laboratorio 8

Laboratorio 8

Attenzione: gli esercizi seguenti richiedono di risolvere vari problemi algoritmici (alcuni dei quali già visti) usando algoritmi ricorsivi. Lo scopo di questi esercizi è di farvi abituare a "**pensare ricorsivo**", ossia a entrare nella logica di una formulazione ricorsiva di un algoritmo. In generale, l'impostazione mentale da seguire è quella di cercare di demandare a "qualcun altro" il lavoro difficile, e risolvere il problema solo nei casi semplici. Ad esempio, nel calcolo di $n!$ visto in aula abbiamo risolto il problema solo per $n=0$ (ovvero, $0!=1$), poi abbiamo demandato a "qualcun altro" (ovvero ad una invocazione ricorsiva) la maggior parte del lavoro (ovvero il calcolo di $(n-1)!$) e abbiamo osservato che per ottenere $n!$ basta moltiplicare $(n-1)!$ per n .

Esercizio 1

Argomento: ricorsione semplice, argomenti sulla riga di comando, lancio/cattura di eccezioni

Scrivere una classe eseguibile avente il funzionamento seguente:

- se sulla linea di comando vengono forniti più o meno di due parametri, il programma termina con una segnalazione di errore
- altrimenti
 - se uno dei due parametri ricevuti non è un numero intero positivo, il programma termina con una segnalazione di errore
 - se entrambi i parametri ricevuti sono numeri interi positivi, il programma visualizza sull'uscita standard il **M.C.D.** tra i due numeri ricevuti, calcolato con un **algoritmo ricorsivo**.

Si scriva un metodo statico ausiliario, `recursiveMCD`, invocato dal metodo `main` per realizzare il comportamento sopra indicato. Tale metodo calcola ricorsivamente il massimo comun divisore (MCD) fra due numeri interi positivi m ed n (con $m > n$), ricevuti come parametri espliciti, usando il ben noto **Algoritmo di Euclide**:

- se n è un divisore di m , allora n è il **M.C.D.** tra m ed n
- altrimenti, il **M.C.D.** di m ed n è uguale al **M.C.D.** di n e del resto della divisione intera di m per n

Si confronti questa definizione ricorsiva dell'algoritmo di Euclide con la definizione iterativa data nell'Esercizio 2 del Laboratorio 5, e si verifichi che le due sono equivalenti

Esercizio 2

Argomento: ricorsione doppia, rilevazione delle prestazioni

Scrivere una classe eseguibile il cui metodo `main`

- riceva un numero intero dalla riga di comando, oppure (nel caso in cui non vengano forniti argomenti sulla riga di comando) chieda all'utente un numero intero n
- visualizzi l' n -esimo numero di Fibonacci, calcolato usando un algoritmo iterativo, e di seguito visualizzi il tempo impiegato per il calcolo (espresso in secondi, con precisione di millisecondi)
- visualizzi l' n -esimo numero di Fibonacci, calcolato usando un algoritmo ricorsivo, e di seguito visualizzi il tempo impiegato per il calcolo (espresso in secondi, con precisione di millisecondi)

Suggerimenti:

1. Ripassare quanto detto a lezione sulla serie di Fibonacci. In particolare si ricorda che la serie di Fibonacci è così definita:

```
Fib( 1 ) = 1
Fib( 2 ) = 1
Fib( n ) = Fib(n-2) + Fib(n-1) per ogni n > 2
```

2. Si consiglia di scrivere due metodi ausiliari statici, **recursiveFib** e **iterativeFib**, invocati dal metodo **main** per realizzare il comportamento sopra indicato. Entrambi i metodi ricevono un parametro n di tipo `int` e (dopo aver verificato la pre-condizione che n non sia negativo) restituiscono un valore di tipo `long` che rappresenta l' n -esimo numero **Fib(n)** nella sequenza di Fibonacci.

- Il metodo **recursiveFib** calcola il valore da restituire usando la ricorsione doppia, implementando direttamente la definizione della serie
 - Il metodo **iterativeFib** deve calcolare il valore da restituire senza usare la ricorsione e senza usare strutture dati di memorizzazione (ossia senza array, ma usando soltanto variabili semplici).
3. Osservare il rapidissimo aumento del tempo di calcolo richiesto dal metodo **recursiveFib** all'aumentare di n (dipendentemente dalla velocità del calcolatore, il calcolo ricorsivo non è più "istantaneo" a partire da valori di n superiori a 30).

Esercizio 3

Argomento: ricorsione semplice, argomenti sulla riga di comando

Scrivere una classe eseguibile **RecSubstringGenerator** capace di generare tutte le sottostringhe di una stringa, usando un **algoritmo ricorsivo**. Ad esempio, l'insieme delle sottostringhe della stringa "abc" è il seguente: {a, ab, abc, b, bc, c}.

- La stringa viene passata come argomento nella riga di comando
- L'insieme delle sottostringhe viene visualizzato a standard output

Esempio di uso:

```
$java RecSubstringGenerator abc
```

Suggerimento: costruire l'insieme delle sottostringhe come unione tra:

1. l'insieme delle sottostringhe che contengono il primo carattere (n sottostringhe, se n è il numero di caratteri della stringa; ad es. se la stringa è "Roma" gli elementi di questo sottoinsieme sono: "R" "Ro" "Rom" "Roma") e
2. l'insieme delle sottostringhe che non contengono il primo carattere (ovvero le sottostringhe della stringa privata del primo carattere; ad es. se la stringa è "Roma" gli elementi di questo sottoinsieme sono le sottostringhe della stringa "oma").

Esercizio 4

Argomento: ricorsione semplice, argomenti sulla riga di comando

Scrivere una classe eseguibile che verifica se una stringa, fornita come parametro sulla riga di comando, è palindroma. La verifica che una stringa sia o meno palindroma deve essere realizzata con un **algoritmo ricorsivo**.

Si ricordi che una stringa è palindroma se è composta da una sequenza di caratteri (anche non alfabetici) che possa essere letta allo stesso modo anche al contrario (es. "radar", "anna", "inni", "xyz%u%zyx").

Attenzione: Il programma *NON deve avere alcun costrutto iterativo* (ossia **non** deve contenere cicli).

Verificare il corretto funzionamento del programma con:

- una stringa palindroma di lunghezza pari
- una stringa palindroma di lunghezza dispari
- una stringa non palindroma di lunghezza pari
- una stringa non palindroma di lunghezza dispari
- una stringa di lunghezza unitaria (che è ovviamente palindroma)
- una stringa di lunghezza zero (che è ragionevole definire palindroma); per fornire come parametro sulla riga di comando una stringa di lunghezza zero si indica il parametro ""

Approfondimento: il programma può essere esteso in maniera tale da verificare se intere frasi sono palindrome, usando le seguenti regole:

- Si trascurano differenze tra maiuscole e minuscole
- Si ignorano gli spazi e altri delimitatori
- Si ignorano i segni di punteggiatura

Secondo queste regole, la frase "Madam, I'm Adam!" è palindroma (ovvero è la più antica palindroma del mondo...). Per questo approfondimento si consultino anche i *Consigli Pratici 12.1* del libro di testo.

Attenzione: per fornire come parametro sulla riga di comando una stringa che comprende anche separatori bisogna racchiuderla tra virgolette (altrimenti ogni token della stringa verrebbe assegnato a un elemento diverso dell'array **args**). Ad esempio:

```
$java RecPalindromeTester "Madam, I'm Adam!"
```

Notate inoltre che nella shell bash (in Linux) il punto esclamativo ha un significato particolare. Per questo motivo nella shell bash il comando scritto sopra genera un errore e bisogna invece scrivere

```
$java RecPalindromeTester "Madam, I'm Adam"\!
```

Attenzione: gli esercizi seguenti riprendono i problemi di ordinamento e ricerca visti a lezione. E' fondamentale prendere confidenza con tutte le realizzazioni di tutti gli algoritmi di ordinamento e ricerca presenti nella classe **ArrayAlgs**, ed essere in grado di riprodurle autonomamente. Ne avrete bisogno molto spesso, sia negli esercizi dei prossimi laboratori sia nelle prove d'esame...

Esercizio 5

Argomento: array riempiti solo in parte, ordinamento di array, ricerca binaria in un array

Studiare la classe **ArrayAlgs** vista a lezione. Studiare con attenzione gli algoritmi di ordinamento e ricerca.

Approfondimento: testare gli algoritmi di ordinamento e ricerca scrivendo una classe di collaudo **ArrayAlgsTester** che:

- riceve da standard input la dimensione e l'intervallo di variabilità di un array di numeri interi casuali
- collauda gli algoritmi di ordinamento tramite un ciclo in cui
 - si crea un array di numeri interi casuali
 - si sceglie l'algoritmo di ordinamento tramite comando introdotto da tastiera (S=SelectionSort, I=InsertionSort, M=MergeSort)
 - si esce quando l'utente introduce il comando Q.
- collauda gli algoritmi di ricerca tramite un ciclo in cui
 - si introduce un valore intero positivo da cercare nell'ultimo array ordinato e creato nella fase precedente
 - si sceglie l'algoritmo di ricerca tramite comando introdotto da tastiera (L=linearSearch, B=binarySearch)
 - si segnala se il valore è stato trovato
 - si esce quando l'utente introduce il comando Q.

Esercizio 6

Argomento: array riempiti solo in parte, ricerca di valore minimo/massimo in un array, ricorsione semplice

Scrivere una classe eseguibile che

- riceve da riga di comando due numeri interi **dim** e **n**;
- crea un array di dimensione **dim**, contenente numeri interi casuali compresi tra 1 e **n**, e lo visualizza a standard output;
- cerca il valore minimo tra quelli contenuti nell'array, tramite un **algoritmo ricorsivo**.

Si consiglia di scrivere un metodo ricorsivo statico che effettui la ricerca e che restituisca un numero intero ≥ 0 rappresentante il valore minimo trovato.

Suggerimento: in alternativa al metodo **Math.random()** visto a lezione, si può utilizzare il metodo **int nextInt(int k)** della classe **java.util.Random**. Prima di usarlo, ovviamente dovete leggere attentamente l'interfaccia pubblica della classe **java.util.Random**.

Esercizio 7

Argomento: array riempiti solo in parte, ordinamento di array, ricerca binaria in un array

Scrivere la classe **SortedArray**, la cui interfaccia pubblica è documentata [a questo link](#), in grado di memorizzare numeri interi mantenendo l'insieme ordinato.

Scrivere successivamente una classe di collaudo **IntSorter** che:

1. riceve due parametri dalla riga di comando: un numero intero N ed una stringa
2. genera N numeri casuali fra 1 e N compresi (N è il primo parametro passato dalla riga di comando) memorizzandoli in un oggetto di tipo **SortedArray**
3. calcola la media dei valori dell'oggetto **SortedArray** e la invia a standard output
4. esegue il seguente ciclo
 - acquisizione da standard input di un numero intero value fra 1 e N
 - ricerca del numero i all'interno dell'oggetto di tipo **SortedArray**
 - invio di un messaggio a standard output del tipo "value: non presente" oppure "value: trovato in indice j"
5. esce dal suddetto ciclo quando viene inserita la stringa "Q"
6. memorizza in ordine decrescente i valori dell'oggetto **SortedArray** in un file di testo (il cui nome e' stato passato come secondo parametro dalla riga di comando), 10 numeri per riga in colonne equispaziate

Esempio di uso:

```
$java IntSorter < dimensione > < nomefile >
```

Esercizio 8

Argomento: array riempiti solo in parte, ordinamento di array, ricorsione semplice

Scrivere una classe eseguibile che

- riceve da riga di comando due numeri interi **dim** e **n**;
- crea un array di dimensione **dim**, contenente numeri interi casuali compresi tra 1 e **n**, e lo visualizza a standard output;
- ordina l'array utilizzando l'algoritmo di ordinamento **selectionSort**, realizzato secondo una **formulazione ricorsiva**, e lo visualizza a standard output.

Si consiglia di scrivere un metodo ricorsivo statico **recSelectionSort**, che realizzi **selectionSort** secondo una formulazione ricorsiva.

Suggerimento: ripassate la formulazione di **selectionSort** mostrata a lezione e verificate che è già presentata in modo da prestarsi a una formulazione ricorsiva (per realizzare **selectionSort** bisogna posizionare nel primo elemento il valore minimo trovato nell'array, e poi...).

Laboratorio 9

Esercizio 1

Argomento: algoritmi di ricerca, ricerca su stringhe.

Realizzare un algoritmo di ricerca binaria (`binarySearch`) che funzioni in modo iterativo anziché ricorsivo.

Dopo avere scritto un metodo statico **`iterativeBinSearch`** che realizza l'algoritmo per la ricerca in un array di stringhe, inserirlo in una classe **`StringArrayAlgs`** che conterrà metodi statici di utilità per l'elaborazione di array di stringhe (analogamente a quanto fatto in aula con la classe **`ArrayAlgs`** e i suoi metodi per l'elaborazione di array di numeri interi).

Esercizio 2

Argomento: algoritmi di ordinamento, ordinamento di stringhe

Realizzare un algoritmo di ordinamento per dati di un array che procede in questo modo:

- si scandisce l'array a partire dall'ultimo indice fino al primo indice, e si confronta ogni coppia di elementi adiacenti, scambiandoli se non rispettano il loro ordinamento relativo (ossia se il valore nell'elemento di indice superiore è maggiore di quello nell'elemento di indice inferiore)
- Al termine di questa prima scansione, il valore minimo contenuto nell'intero array è contenuto nell'elemento di indice 0 dell'array
- Il passo successivo è identico al precedente, ma termina dopo aver esaminato l'elemento di indice 1: al termine di questa seconda scansione, anche l'elemento di indice 1 dell'array contiene l'elemento corretto
- Si itera il procedimento fino al completo ordinamento dell'array

Questo algoritmo prende il nome di **ordinamento a bolle (`bubbleSort`)**. Infatti, se si considera l'array come se fosse disposto verticalmente, con la cella di indice 0 in alto, lo spostamento di valori effettuato dall'algoritmo è simile al movimento di bolle che gorgogliano verso l'alto.

Dopo avere scritto un metodo statico che realizza l'algoritmo **`bubbleSort`** per ordinare **array di stringhe**, inserirlo nella classe **`StringArrayAlgs`** definita nell'Esercizio 1.

Esercizio 3 (impegnativo)

Argomento: algoritmi di ordinamento, ordinamento di stringhe

Realizzare un algoritmo di ordinamento per fusione (`mergeSort`) che funzioni in modo iterativo anziché ricorsivo.

Suggerimenti:

1. Realizzare una prima versione del programma che funzioni correttamente soltanto quando la dimensione dell'insieme dei dati (ossia la lunghezza n dell'array) è una potenza di 2.
 - Alla prima iterazione del ciclo principale, l'array da ordinare viene considerato (dal punto di vista logico) come costituito da n sotto-array di dimensione 1
 - Tali sotto-array elementari vengono fusi a coppie, ciascun array in posizione pari (a partire dalla posizione 0) fuso con l'array seguente in posizione dispari. L'algoritmo di fusione di due array ordinati è identico a quello visto per la versione ricorsiva di `MergeSort`
 - Al termine di tale prima iterazione l'array originale è costituito da $n/2$ sotto-array consecutivi, ciascuno dei quali contiene due elementi ordinati.
 - La successiva iterazione del ciclo principale fonde tali sotto-array a coppie, come nella prima iterazione, cosicché al termine l'array originale è costituito da $n/4$ sotto-array consecutivi, ciascuno dei quali contiene quattro elementi ordinati
 - L'algoritmo applica iterativamente questa procedura fino al completo ordinamento del vettore.
2. Modificare la prima versione in modo da gestire insiemi di dati di lunghezza arbitraria, eliminando quindi il vincolo che tale lunghezza sia una potenza di 2, dopo aver individuato le necessarie modifiche da apportare all'algoritmo sopra delineato.

Dopo avere scritto un metodo statico **iterativeMergeSort**, che realizza l'algoritmo sopra descritto per ordinare **array di stringhe**, inserirlo nella classe **StringArrayAlgs** definita nell'Esercizio 1.

Esercizio 4

Argomento: algoritmi di ordinamento e ricerca su stringhe

Scrivere una classe eseguibile per il collaudo della classe **StringArrayAlgs** (definita negli esercizi precedenti e contenente gli algoritmi di ordinamento **bubbleSort** e **iterativeMergeSort**, nonché l'algoritmo di ricerca **iterativeBinSearch**). Il programma

- legge dallo standard input un insieme di dimensione non predeterminata di stringhe, e le memorizza in un array **di stringhe** (l'inserimento termina con la terminazione dello standard input)
- ordina l'array appena costruito utilizzando **bubbleSort**, o in alternativa **iterativeMergeSort** (la scelta dell'algoritmo da utilizzare viene effettuata dall'utente tramite standard input)
- riceve da standard input una stringa da cercare all'interno dell'array ed effettua la ricerca con il metodo **iterativeBinSearch**.
- ripete questa operazione per un numero non predeterminato di volte (le ricerche terminano con la terminazione dello standard input).

Esercizio 5

Argomento: ereditarietà, sovrascrivere i metodi della classe *Object*

Si consideri la gerarchia di ereditarietà costituita dalla superclasse **BankAccount** (disponibile [a questo link](#)) e dalle sue due sottoclassi **SavingsAccount** ([a questo link](#)) e **CheckingAccount** ([a questo link](#)).

Si completi la scrittura di queste tre classi sovrascrivendo in ciascuna di esse i metodi **toString** e **equals**, ereditati dalla superclasse universale **Object**. Per il metodo **toString** in particolare si richiede di adottare la convenzione della libreria standard.

Si verifichi il corretto funzionamento di tutti i metodi delle tre classi usando la classe di collaudo **AccountTester** (disponibile [a questo link](#)). In particolare, i metodi **toString** sono corretti se le stampe a standard output dei vari oggetti rispettano le convenzioni della libreria standard; i metodi **equals** sono corretti se i controlli di uguaglianza forniscono i risultati attesi.

Esercizio 6

Argomento: ereditarietà, sovrascrivere metodi, metodi polimorfici

Si consideri nuovamente la gerarchia di ereditarietà vista nell'esercizio precedente, costituita dalla superclasse **BankAccount**, e dalle sue due sottoclassi **SavingsAccount** e **CheckingAccount**.

Si richiede di aggiungere a questa gerarchia di ereditarietà una nuova classe **TimeDepositAccount** ("conto di deposito vincolato"). Un conto di deposito vincolato è identico a un conto di risparmio, a parte il fatto che l'intestatario si impegna a lasciare il denaro sul conto per un certo numero di mesi (definito all'apertura del conto insieme al tasso di interesse), senza fare prelievi durante questo periodo iniziale. È prevista una penale di 20€ in caso di prelievo anticipato.

Completare la classe **TimeDepositAccount** sovrascrivendo in essa i metodi **toString** e **equals**, ereditati dalla sua superclasse. Effettuare il collaudo modificando opportunamente la classe **AccountTester** (fornita a corredo dell'esercizio precedente): il metodo **testAccount** deve prevedere anche il caso in cui l'oggetto puntato dalla variabile **a** sia di tipo **TimeDepositAccount**, e in tal caso deve effettuare delle operazioni di collaudo.

Suggerimenti (da non leggere subito):

- La definizione di **TimeDepositAccount** appena data ci fa capire come questa nuova classe si deve inserire nella gerarchia di ereditarietà. Sarà una sottoclasse di ...
- Scrivere almeno due costruttori di **TimeDepositAccount**: uno che consente di inizializzare il tasso di interesse e i mesi di deposito vincolato, e uno che oltre a questi due parametri consente di inizializzare anche il saldo ad un valore non nullo.
- Il metodo **addInterest** viene invocato alla fine di ogni mese per accreditare gli interessi. Quindi il numero di mesi di deposito vincolato si deve ridurre ad ogni invocazione di questo metodo.
- Se al momento di un prelievo il numero di mesi di deposito vincolato è ancora positivo, allora bisogna addebitare la penale

Esercizio 7

Argomento: ereditarietà, sovrascrivere metodi, metodi polimorfici

Realizzare una classe **Square** che estenda la classe **Rectangle** della libreria standard. La classe deve realizzare i seguenti comportamenti:

- Il costruttore crea un quadrato ricevendo come parametri espliciti le coordinate (**x,y**) del centro del quadrato, e la dimensione del quadrato (ovvero la lunghezza del lato). Per realizzare il costruttore sarà necessario invocare costruttori e/o metodi della classe **Rectangle** (si consiglia in particolare di studiare la documentazione dei metodi **setLocation** e **setSize** della classe **Rectangle**). Osservare anche che, dal momento che i campi **x**, **y**, **width**, **height** di **Rectangle** sono tutti di tipo **int**, il posizionamento del centro del quadrato avrà una approssimazione di ± 1 .
- La classe possiede un nuovo metodo **getArea()**, che calcola e restituisce l'area del quadrato.
- La classe sovrascrive il metodo **setSize(int width, int height)** della classe **Rectangle** (studiarne la documentazione): se **width=height**, il metodo esegue correttamente ridimensionando il quadrato alla nuova dimensione **width**, altrimenti lancia un'eccezione di tipo **IllegalArgumentException**.
- La classe possiede un nuovo metodo **setSize(int dim)**, che esegue il ridimensionamento del quadrato sulla base dell'unico parametro esplicito **dim**.

Collaudare la classe scrivendo un programma di test che

- Riceve da standard input due triple di numeri interi (una tripla per riga), rappresentanti le coordinate (x,y) del centro e la dimensione di ciascuno dei due quadrati.
- Crea due oggetti di tipo **Square** usando tali valori, e stampa gli oggetti a standard output in ordine di area (il primo oggetto stampato è quello di area più piccola).
- Riceve da standard input due coppie di numeri interi (una coppia per riga), rappresentanti i nuovi valori (width,height) di larghezza e altezza per ciascuno dei due quadrati.
- ridimensiona i due quadrati usando il metodo

setSize(int width, int height)

sovrascritto nella classe

Square

- Se **setSize** termina correttamente la propria esecuzione (ovvero se **width=height**), stampa nuovamente i due oggetti a standard output in ordine di area; altrimenti segnala l'errore e termina l'esecuzione.

Suggerimenti: non dimenticate che la classe **Rectangle** ha già sovrascritto i metodi **toString** e **equals** di **Object**. Inoltre, poniamoci questa domanda: la nostra classe **Square** ha oppure no necessità di avere dei nuovi campi di esemplare rispetto a **Rectangle**? Quindi è necessario oppure no sovrascrivere **equals**?