LABORATORIO DI PROGRAMMAZIONE
SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE
UNIVERSITÀ DEGLI STUDI DI MILANO

2025–2026

15.12.2025

Indice

Avvertenze. L'esame avrà una durata di 3.5h ed è diviso in tre parti, che peseranno quasi equamente sulla valutazione finale.

La prima parte, in linguaggio `C`, servirà a valutare le vostre competenze di base di programmazione. La seconda parte, in linguaggio `C`, e la terza, in linguaggio `Java`, serviranno a valutare le competenze specifiche sui due linguaggi.

Questa simulazione include due sotto-esami, ciascuno comprendente parte 3.

Si raccomanda di leggere interamente il tema d'esame con attenzione prima di cominciare a scrivere le soluzioni dei singoli esercizi, in modo da avere chiaro l'obiettivo finale.

Ultima revisione: 14 dicembre 2025.

Sotto-Esame 1

Pare 3 – (Linguaggio `Java`).

Esercizio 1 - Macchinetta del Caffè.

Si implementino le seguenti classi, per ognuna delle quali vengono specificati gli attributi e metodi richiesti. Ulteriori attributi, nonché i metodi `getter`, `setter` e `toString()` saranno a vostra discrezione.

- **Classe** `Prodotto`: Classe astratta che descrive un prodotto venduto dalla macchinetta. Il prodotto possiede due attributi: `costo` (un intero) e `nome` (una stringa).
- **Classe** `Bevanda`: Sottoclasse di `Prodotto` le cui istanze rappresentano una bevanda. Possiede l'attributo addizionale `volume` (un intero).
- **Classe** `Merendina`: Sottoclasse di `Prodotto` le cui istanze rappresentano una merendina. Ha un attributo addizionale `calorie`.
- **Classe** `Macchinetta`: Classe che rappresenta una macchinetta per le merendine. Ha un attributo `prodotti` che contiene una lista di prodotti. Fornisce i seguenti metodi:
  - `void aggiungiProdotto(Prodotto)`: aggiunge il prodotto dato alla macchinetta.
  - `Merendina compraMerendina(int soldi, int calorie)`: restituisce il primo elemento nella lista dei prodotti che è una merendina (usate `instanceof`) il cui costo è minore o uguale ai soldi disponibili e le cui calorie sono minori o uguali a quelle date. Se non è disponibile nessuna merendina che soddisfa i requisiti restituisce `null`.
  - `Bevanda compraBevanda(int soldi, int volume)`: restituisce il primo elemento nella lista dei prodotti che è una bevanda il cui costo è minore o uguale ai soldi disponibili e il cui volume è maggiore o uguale al volume dato. Se non è disponibile nessuna bevanda che soddisfa i requisiti restituisce `null`.

Il materiale fornito vi spiega come usare `instanceof`.

Scrivere una classe `Solver` che crea una macchinetta e tiene traccia in una variabile dei soldi che l'utente ha a disposizione (inizialmente, zero). Quindi, la classe legge da standard input una sequenza di righe con uno dei seguenti formati:

```
B nome costo volume
M nome costo calorie
S soldi
CB volume
CM calorie
SORT
F
```

Le righe che iniziano con `B` richiedono di inserire una bevanda con i parametri dati tramite `aggiungiProdotto(Prodotto)`.

Le righe che iniziano con `M` richiedono di inserire una merendina con i parametri dati tramite `aggiungiProdotto()`.

Le righe che iniziano con `S` richiedono di aggiungere i soldi specificati ai soldi disponibili. Se la quantità di soldi inserita è negativa, il programma termina con una eccezione.

Le righe che iniziano con `CB` richiedono di utilizzare il metodo `compraBevanda()` per comprare una bevanda con i soldi disponibili e con il volume dato: se si riesce a comprare qualcosa, la bevanda viene stampata e il costo sottratto dai soldi disponibili.

Analogamente le righe che iniziano con `CM` richiedono di utilizzare il metodo `compraMerendina()` per comprare una merendina con i soldi disponibili e le calorie date: se si riesce a comprare qualcosa, il nome della merendina viene stampato e il costo sottratto dai soldi disponibili.

Le righe che iniziano con `SORT` richiedono di stampare la lista dei prodotti disponibili nella macchinetta in ordine crescente di costo. Implementate questa funzionalità sfruttando quanto offerto dal linguaggio `Java`.

Le righe con una sola `F` terminano l'inserimento.

Per questo tipo di input è conveniente utilizzare un ciclo basato su uno `Scanner`: fin tanto che `Scanner.hasNext()` restituisce vero leggete il file di input utilizzando `Scanner.next()` per le stringhe e `Scanner.nextInt()` per gli interi.

Non è necessario fare controlli sulla correttezza dell'input. Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da standard input, possibilmente utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Sfruttate la redirezione: rende il testing del vostro esercizio molto più veloce.

Esempio Se l'utente inserisse le seguenti linee:

```
B coca 10 10
B cocagrande 15 30
B cocamedia 10 15
M fiesta 10 20
M duplo 5 10
M crackers 5 6
CB 1
S 14
CB 1
CM 5
CM 10
S 20
CM 10
CB 1
S 10
CM 10
F
```

allora il comando `java Es1 <in1.txt` deve stampare

```
Ho bevuto coca
Ho mangiato duplo
Ho bevuto cocagrande Ho mangiato crackers
```

Sotto-Esame 2

Pare 3 – (Linguaggio `Java`).

Esercizio 2 - Noleggio Auto.

Si implementino le seguenti classi, per ognuna delle quali vengono specificati gli attributi e metodi richiesti. Ulteriori attributi, nonché i metodi `getter`, `setter` e `toString()` saranno a vostra discrezione.

- **Classe** `Veicolo`: Classe astratta che descrive un veicolo disponibile per noleggio. Il veicolo possiede quattro attributi:
  - `targa`: una stringa che identifica un veicolo.
  - `modello`: una stringa, e.g., `"OPEL"` o `"Yamaha"`, che per comodità sarà una parola, ovvero una stringa senza spazi.
  - `anno`: un intero che rappresenta l'anno di immatricolazione. L'anno deve essere compreso tra il 1800 e il 2025; se così non fosse il valore dell'attributo anno è assegnato di default al 2025.
  - `numNoleggi` che registra il numero di noleggi del veicolo.
- **Classe** `Moto`: Sottoclasse di `Veicolo` le cui istanze rappresentano una moto. Una moto ha un attributo addizionale `cavalli`, un intero che rappresenta i cavalli della moto e che non può essere inferiore a 200 e non deve superare il valore 2000. Se il numero inserito non fosse nel range, deve essere inserito il valore 1000 di default.
- **Classe** `Auto`: Sottoclasse di `Veicolo` le cui istanze rappresentano una auto. Oltre alla targa, e l'anno di immatricolazione una auto è caratterizzata dall'attributo `numeroPax` che deve essere tra 2 e 8 e identifica il numero di persone per cui l'auto è immatricolata. Se il numero non è compreso tra 2 e 8, viene assegnato di default il valore 5.
- **Classe** `Noleggio`: Classe che rappresenta un noleggio auto. Ha un solo attributo `listaVeicoli` che contiene una lista dei veicoli disponibili. Fornisce i seguenti metodi per aggiunta/noleggio/restituzione veicoli:
  - `void aggiugiVeicolo(Veicolo)`: aggiunge un nuovo `Veicolo` (auto o moto) alla lista dei Veicoli, ovvero un veicolo immatricolato nel 2025.
  - `Auto noleggiaAuto(int numeroPax, int anno)`: restituisce il primo veicolo che è di classe `Auto` (usate `instanceof`), può caricare almeno `numeroPax` persone ed è immatricolato durante o dopo l'anno `anno`. Se non è disponibile nessuna auto che soddisfa i requisiti restituisce `null`; altrimenti, il veicolo viene tolto dalla lista di veicoli.
  - `Moto noleggiaMoto(String modello, int cavalli)`: restituisce la prima moto con modello `modello` e con un numero di cavalli superiore o uguale a `cavalli`. Se non viene trovata nessuna moto che rispecchia il requisito il metodo restituisce `null`; altrimenti, il veicolo viene tolto dalla lista di veicoli.
  - `void restituisciVeicolo(Veicolo)`: rappresenta la restituzione di un veicolo. Lo riaggiunge quindi alla lista dei Veicoli.
  - `void sort(void)`: ordina i prodotti per ordine crescente di anno di immatricolazione.

Scrivere una classe `Solver` che crea un Noleggio. Quindi, la classe legge da standard input una sequenza di righe che comincia con i seguenti comandi (pensate opportunamente il resto della riga di comando)

```
ADD ....
R ....
NAuto ....
NMoto ...
S...
END...
```

Le righe che iniziano con `ADD` richiedono di inserire un nuovo veicolo. Dopo l'inserimento, viene stampato il veicolo (con una opportuna `toString()`).

Le righe che iniziano con `R` restituiscono il veicolo. Dopo la restituzione viene stampata la lista dei veicoli, ordinata per numero di noleggi.

Le righe che iniziano con `NAuto` o `NAuto` richiedono di noleggiare una auto o una moto. Il risultato del noleggio deve produrre in stampa il veicolo noleggiato, o deve fornire una stringa del tipo `Auto/Moto non disponibile`.

Le righe che iniziano con `S` richiedono ordinare la lista di veicoli per anno di immatricolazione e di stamparla.

Il programma termina quando viene inserita una riga che comincia con `END`.

Per questo tipo di input è conveniente utilizzare un ciclo basato su uno `Scanner`: fin tanto che `Scanner.hasNext()` restituisce `true` leggete il prossimo input utilizzando `Scanner.next()` per le parole e `Scanner.nextInt()` per gli interi.

Non vanno fatte assunzioni sul numero di linee in input. L'input va letto da standard input, possibilmente utilizzando la redirezione fornita da shell (si veda l'esempio sotto).

Sfruttate la redirezione: rende il testing del vostro esercizio molto più veloce.

Istruzioni per la consegna

- Come consegnare:
  –
  – Per gli esercizi in linguaggio `java`, consegnate tutti i file `.java` implementati, contenenti le varie classi e il `main`.

  Consegnate tutte le funzioni ausiliarie da voi implementate. Commentare il codice per spiegare le scelte fatte è fortemente consigliato. Il codice **deve essere indentato e leggibile**.
- Sito per la consegna:

  `https://upload.di.unimi.it`
- Autenticarsi con le proprie credenziali di posta elettronica d'ateneo.
- Consegnare solo il codice sorgente (file `*.java`), NON l'eseguibile.
- Ogni file consegnato **non compilabile** non verrà corretto, anche se alcune sue parti sono giuste. Questo vuol dire che se all'interno del file vi fossero degli esercizi svolti correttamente, ma il file non è compilabile, questi non verranno valutati.