



Restituire gli utenti che hanno pubblicato almeno un commento prima di una certa data

```
private static void main() {  
    LocalDate date = LocalDate.of(2021, 1, 3);  
    List<Utente> l = List.of(  
        new Utente("u1",  
            List.of(new Commento("c2", LocalDate.of(2021, 1, 2)),  
                    new Commento("c1", LocalDate.of(2021, 1, 1)),  
                    new Commento("c5", LocalDate.of(2021, 1, 5)))),  
        new Utente("u2",  
            List.of(new Commento("c4", LocalDate.of(2021, 1, 4)),  
                    new Commento("c3", LocalDate.of(2021, 1, 3)))));  
    // result = ["u1"]  
}
```

Stream()

filter()

anyMatch()

l1. stream()

.filter (s -> s.commenti().stream()
.anyMatch (w -> w.date().isBefore(date);

forEach (System.out::println);

}

l. stream()

.min (Comparator.comparing (Utente::getIs)
.orElse(null)

System.out.println;

Restituire l'utente che ha pubblicato il commento più recente

```
private static void es11() {  
    List<Utente> l = List.of(  
        new Utente("u1",  
            List.of(new Commento("c2", LocalDate.of(2021, 1, 2)),  
                    new Commento("c1", LocalDate.of(2021, 1, 1)),  
                    new Commento("c5", LocalDate.of(2021, 1, 5)))),  
        new Utente("u2",  
            List.of(new Commento("c4", LocalDate.of(2021, 1, 4)),  
                    new Commento("c3", LocalDate.of(2021, 1, 3))));  
    // result = "u1"  
}
```



es 1

`l.stream()`

`List<String> nomi =
nomi.stream()`

- `.filter(x -> x.age() < 30 && x -> x.equals(nomi))`
- `.map(Person::getName)`
- `.forEach(System.out::println);`

es 2

`List<Stan> nomi =`

- `.nomi.stream()`
- `.map(p -> p.role())`
- `.distinct()`
- `.forEach(System.out::println);`

es 3

```
List<String> l =  
final String string = "av"
```

```
l.stream()
```

- filter (x -> x.startsWith(string))
- ForEach(System.out.println)

es 4

```
final String out
```

```
l.stream()
```

- map (s -> s.substring(0, 1))
- reduce("", (v, e) -> v.concat(e))
- System.out.println;

Hai una LISTA di FRASI.

es 5

```
List<String> l1 = {.....}
```

```
String output = l1.stream()  
    .max(Comparator.comparing(Person::length),  
    System.out.println(output)
```



1. Nei diagrammi UML degli stati, i nomi degli stati sono indicati
 - (a) All'interno di rettangoli senza angoli arrotondati
 - ☒ (b) All'interno di rettangoli con angoli arrotondati
 - (c) Sopra i rettangoli con angoli arrotondati
 - (d) Alle estremità delle frecce delle transizioni
2. Il design pattern Observer è consigliabile quando
 - (a) Si vuol separare bene il codice di due classi
 - (b) Si vuol chiamare un metodo di un'altra classe
 - (c) Si vuol tenere coerente lo stato di una classe con quello di un'altra classe
 - (d) Si vuol tenere separato lo stato di due oggetti
3. Il design pattern Decorator permette
 - (a) Di aggiungere o togliere un piccolo compito ad una classe
 - (b) Solo di aggiungere un piccolo compito ad una classe
 - (c) Di togliere un metodo ad una classe
 - (d) Di ridefinire un metodo di una classe
4. Il design pattern Bridge permette alla classe client di
 - (a) Tenere alcuni riferimenti a oggetti sottotipi di Implementor
 - (b) Non tenere riferimenti
 - (c) Non dover chiamare i metodi di Implementor
 - (d) Tenere riferimenti a oggetti che sono sottotipi di Implementor
5. La classe che ha il ruolo Facade ha
 - (a) Una sola classe nel sottosistema
 - (b) Nessuna classe client, difatti il Facade potrebbe avere il metodo main
 - (c) Una sola classe client
 - (d) Un certo numero di classi client
6. Tipicamente per il processo XP, le release del software avvengono
 - (a) Una release alla settimana
 - (b) Una release ogni 2 settimane
 - (c) Una release ogni 3 settimane
 - (d) Una release ogni 4 settimane
7. La fase di progettazione produce e documenta
 - (a) La struttura del software che realizza le specifiche
 - (b) Le specifiche del software
 - (c) Le modalità di utilizzo del software
 - (d) Le funzionalità del software
8. Il design pattern Adapter serve a
 - (a) Convertire una interfaccia di una classe in un'altra
 - (b) Convertire il tipo di una variabile
 - (c) Cambiare l'interfaccia di una classe a runtime
 - (d) Usare due nomi per una stessa classe
9. Si abbia il frammento di codice:

```
public Libro getLibro() {  
    return new Book();  
}
```

 - (a) Il frammento sta svolgendo il ruolo di un Factory Method
 - (b) Il frammento sta svolgendo il ruolo di un Singleton
 - (c) Il codice non è compilabile
 - (d) Il codice dovrebbe essere all'interno di una superclasse

Per il design pattern Chain of Responsibility

 - (a) I possibili riceventi sono più di uno
 - (b) Le classi richiedenti devono essere più di una
 - (c) Si ha che il numero di richiedenti è pari al numero di riceventi
 - (d) Si richiede di implementare tanti metodi per ciascuna classe ricevente

Quale delle seguenti è uno svantaggio dell'uso del pattern Command?

 - (a) Gli stessi comandi potrebbero essere generati da elementi diversi dell'interfaccia
 - (b) Si deve usare il pattern Singleton per imporre un gestore unico dei comandi
 - (c) Si potrebbero penalizzare le prestazioni a causa dell'indirizione introdotta
 - (d) I comandi ancora non eseguiti potrebbero essere anche annullati
12. I design pattern sono strutture
 - (a) Create appositamente quando si risolve un nuovo problema
 - (b) Per un piccolo numero di classi
 - (c) Utili alla fase di raccolta ed analisi dei requisiti
 - (d) Per un grande numero di classi
13. Quale legge è soddisfatta quando si usa il Mediator?
 - (a) La legge di Murphy
 - (b) La prima legge di Lehman
 - (c) La legge di Demeter
 - (d) La legge sui grandi numeri
14. Per un sistema che è Spaghetti code
 - (a) Il codice è stato creato insieme al cliente
 - (b) Si hanno classi con tanti metodi
 - (c) E' stata seguita la progettazione ad oggetti
 - (d) Non è possibile far test
15. Per il design pattern Composite, si ha trasparenza quando il client
 - (a) Distingue oggetti semplici e composti
 - (b) Può usare oggetti semplici o composti
 - (c) Non deve distinguere fra oggetti semplici e composti
 - (d) Non può usare oggetti semplici o composti
16. Per eliminare le istruzioni condizionali
 - (a) Si possono inserire le precondizioni
 - (b) Si potrebbe usare il design pattern Facade
 - (c) Si potrebbe usare il design pattern Singleton
 - (d) Si potrebbe usare il design pattern State