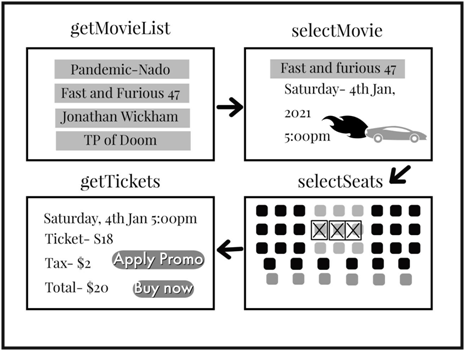
**Chaining multiple CompletableFutures: Non-Blocking**

**1. Purchasing movie tickets online**



1. View the movie list and select a particular movie

2. Select seats

3. Apply any available promotional codes and retrieve the final ticket price

4. Make payment

*// Get Movies playing for the selected showtime (date and time)*CompletableFuture<List<Movie>> getMovieList(String day){  
 **return** CompletableFuture.*supplyAsync*( ()-> {  
 List movieList = **new** ArrayList<Movie>();  
 *//getMovieList from backend* **return** movieList;  
 });  
}*// Select seats for the movie  
//ShowDetails includes movie selected, date and time of the movie, along with seats selected for that show*CompletableFuture<ShowDetails> selectSeats (ShowTime showTime) {  
 **return** CompletableFuture.*supplyAsync*(() -> {   
 ShowDetails showDetails = new ShowDetails();  
 showDetails.setSeats(selectSeatsForShow());  
 **return s**howDetails;  
 });  
 }*//Customer selects a movie from the movie list*CompletableFuture<Movie> selectMovie(List<Movie> movies){  
 *//user selects movie* **return** CompletableFuture.*supplyAsync*(() -> {  
 movie = getCustomerSelectedMovie();  
 **return movie**;  
 });  
}*//Calculate ticket price*CompletableFuture<TicketPrice> getTicketPrice (ShowDetails showdetails){  
 **return** CompletableFuture.*supplyAsync*(() -> {  
 ticketPrice = getTotalTicketPrice();   
 **return** ticketPrice; *//final price*  
 });  
}*// Apply promo code if available*ShowDetails applyPromoCode (ShowDetails showdetails, String promoCode){  
 showdetails.setFinalDiscount(getDiscount(promoCode));  
 **return** showdetails;  
}

***Chaining multiple futures to book the Movie Show***

**public void** bookMyShow(ShowDetails showDetails, String promoCode){  
CompletableFuture result = getMovieList(showDetails.getShowTime().getDay())  
 .thenCompose(movies -> selectMovie(movies))  
 .thenCompose(movie -> selectSeats(showDetails.getShowTime())  
 .thenApply(showDetails1 -> applyPromoCode(showDetails1,promoCode))  
 .thenCompose(showDetails2 -> getTicketPrice(showDetails2)));  
}

In the above example, we take the output of one future and pass it as input to the next in the chain.

**<U> [CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \t "_blank)<U> [thenApply](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \l "thenApply-java.util.function.Function-" \t "_blank)(**[**Function**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)**<? super**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)**,? extends U> fn):**The input is a function to use to compute the value of the returned CompletionStage.

**<U> [CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \t "_blank)<U> [thenCompose](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \l "thenCompose-java.util.function.Function-" \t "_blank)(**[**Function**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)**<? super**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)**,? extends [CompletionStage](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html" \t "_blank)<U>> fn) :**The input is a function returning a new CompletionStage.

**Step by step guide through the example:**

Step 1: *getMovieList()*returns a CompletableFuture. This is the first CompletableFuture in the chain and created with a *supplyAsync()*

Step 2: The next step is selecting a particular movie from the list retrieved, *thenCompose()* indicates that we want to execute another CompletableFuture and get its completion result

Step 3: Once the movie is selected, we now select seats by chaining each of the asynchronous tasks through *thenCompose()*

Step 4: Before calculating the total price of the tickets, apply any promotional code through *applyPromoCode()*

Notice *applyPromoCode()* passed to the *thenApply()* does not return a Completable future. This task is synchronous and returns an object instead.

If this task were to return a CompletableFuture, the result of *thenApply()*would be a nested CompletableFuture —

< CompletableFuture<CompleteableFuture>>

This is an important difference between *thenApply()*and*thenCompose()*where the latter returns a flattened result, synonymous to the difference between a *map()* and *flatMap()*

Step 5: The last step is to calculate the final ticket price and purchase the tickets

**2. Ordering snacks at the movies**



1. At the movie, you order popcorn and a soda

2. The manager takes the order and lets his staff know. One staff member preps the popcorn and the other fills the soda. Basically, both do the tasks **simultaneously —***your order is not fully complete until both these tasks are done*

3. Both get their orders to the manager

4. You are now ready to enjoy your refreshments!

***Multiple independent futures running in parallel***

CompletableFuture<String> getPopCorn(){  
 CompletableFuture<String> future = CompletableFuture.*supplyAsync*(() -> {  
 **return**(**"Popcorn ready"**);  
 });  
 **return** future;  
}CompletableFuture<String> getDrink(){  
 CompletableFuture<String> future = CompletableFuture.*supplyAsync*(() -> {  
 **return**(**"Drink ready"**);  
 });  
 **return** future;  
}**public** String snackReady(){  
 **return "Order is ready - Enjoy your movie snacks"**;  
}//snacks are ready when popcorn and drink are ready  
**public void** getSnacksForMovie(){  
 CompletableFuture snacks = getPopCorn()  
 .thenCombine(getDrink(),(str1,str2)->{**return** snackReady();}) ;  
}

**<U,V> [CompletableFuture](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \t "_blank)<V> [thenCombine](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html" \l "thenCombine-java.util.concurrent.CompletionStage-java.util.function.BiFunction-" \t "_blank)([CompletionStage](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html" \t "_blank)<? extends U> other, [BiFunction](https://docs.oracle.com/javase/8/docs/api/java/util/function/BiFunction.html" \t "_blank)<? super**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)**,? super U,? extends V> fn):** The input is a second CompletionStage and a supplier function that executes with the two results as arguments. The result returned is a new CompletionStage.

Both the *getPopCorn()* and *getDrink()* are executed in parallel. Once both of them have completed, they are passed as input to our supplier function to print the result *snackReady().*

CompletableFuture API is constantly updated with newer, more powerful improvements and methods with every version of Java. Suffice to say that we’ve barely just scratched the surface here.