**Airport**

In your IDE of choice, create a new JavaScript file named airport.js and make it so that all code written in the file follows JS strict mode.

Create an anonymous immediately-invoking function that will hold the main execution of all program calls. Writing a simple command console.log(“Hi”) inside this function and running the code should output “Hi“ in the console.

Create constructor functions with properties representing the following:

* **Person** - name, surname
* **Seat** - number, category (can be **"b"** for business or **"e"** for economy)
* **Passenger** - person (instance of Person), seat (instance of Seat)
* **Flight** - relation, date and list of passengers (initially empty)
* **Airport** - name (should be a hard-coded value "Nikola Tesla"), list of flights

If category is not provided for Seat, assume it’s economy (“e”).

If seat number is not provided, assign a random number between 10 and 100.

Add **getData** method to **Person**. It should return a formatted string containing the name and surname of the person.

"John", "Snow" -> "John Snow”

Add **getData** method to **Seat**. It should return a formatted string containing a seat number and category.

12, "B" -> "12, B”

Add **getData** method to **Passenger**. It should return a formatted string containing a seat number, category letter (always in uppercase), a name and surname. To display seat data, use **getData** method of the Seat object. The same goes for the person data.

personObj, seatObj -> 12, B, John Snow

Add **addPassenger** method to **Flight**. It should receive **Passenger** and add the passenger to the passenger list of a given flight.

Add **addFlight** method to **Airport**. It should add the flight to the list of flights of the airport.

Add **getData** method to **Flight**. It should return a formatted string containing a date, relation, all data related to the flight and list of passengers with their data.

25.10.2017, Belgrade - Paris

1, B, John Snow

2, E, Cersei Lannister

Inside your immediately-invoking function, add **createFlight** function that receives a relation (ex. Belgrade - New York) and date as parameters and returns a created **Flight**.

Inside your immediately-invoking function, add **createPassenger** function that receives a first name, last name, seat number and category and returns a created **Passenger**.

In you main program function, create an instance of the **Airport** object.

Create two instances of the **Flight** object using the **createFlight** function and the following data:

relation: "Belgrade - New York" date: "Oct 25 2017"

relation: "Barcelona - Belgrade" date: "Nov 11 2017"

Create four instances of the **Passenger** object using the **createPassenger** function with the following data:

Name: "John" surname: "Snow" seat number: 1 category: "b"

Name: "Cersei" surname: "Lannister" seat number: 2 category: "b"

Name: "Daenerys" surname: "Targaryen" seat number: 14

Name: "Tyrion" surname: "Lannister"

Add the first two passengers to the first flight and the second two to the second flight using the Flight’s **addPassenger** method.

Add the created flight instances to the airport using Airport’s **addFlight** method.

Call Airport’s **getData** method to display the airport data output in the following manner:

Airport: Nikola Tesla, total passengers: 4

25.10.2017, Belgrade - Paris

1, B, John Snow

2, E, Cersei Lannister

11.11.2017, Barcelona - Belgrade

3, E, Daenerys Targaryen

4, E, Tyrion Lannister

**Hints**

* List is a synonym for array, so when we say a list of flights, it’s actually an array of flight objects
* Use JS built-in Date()object to parse input date
* Use \t and \n special strings to format output
* Use built-in String methods to transform text from lowercase to uppercase
* Use Array’s built-in methods to add and remove elements from an array

**Extra**

Modify **Flight** **getData** method to return a formatted string as date and relation (as the first and the last consonant of the starting location - the first and the last consonant of the destination location) of the current flight.

"Belgrade - Paris", "25.09.2017" -> 25.09.2017 BD - PS

While adding passenger with **addPassenger** method, make sure that:

* Two passengers can not have the same seat number;
* The flight can not have more than 100 passengers;
* If a passenger with the same full name exists in a flight list, you should replace the existing passenger’s data with new data (e.g. it can happen when a passenger changes seats).

Modify Passenger’s **getData** method to return full category name “business” for “b” and “economy” for “e”.

Add the total number of passengers in business category for each flight and the total number of business category passengers for the airport to final output.