# BBB Testing Documentation

Group 9: Philipp Uhl, Christoph Schwägerl

*[2018-11-26 Mon]*

## Contents

# 1 Testing approach

The application is composed of five main components:

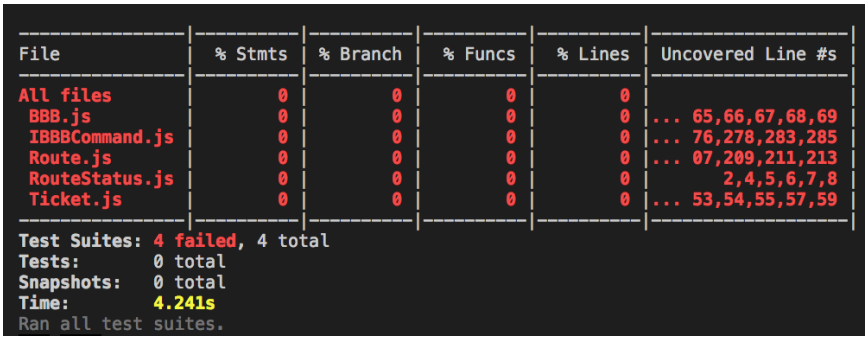
- `Ticket`: Class that holds ticket information
- `Route`: Class that holds route information as well as associated tickets, available seats, etc.
- `RouteStatus`: Enumaration that indicates the status of a `Route`
- `IBBBCommand`: Interface and several implementations that each fulfill one specific funcitonality such as creating a new route, purchasing a ticket, etc.
- `BBB`: Class that uses as the composition of overall functionality which parses the given commands and loads/stores routes from/in a file

We tried to reach 100% branch coverage in the code by the use of unit tests. For this, all components that we created were tested individually until 100% branch coverage was reached. For every component we looked at each method, property or constructor to identify the branches to be tested. This allowed us to look at a single functionality in each test case and made it a lot easier to identify all the necessary test cases for each component. We decided not to target all components in the first sprint because we expected a high number of required test cases which would have made it a bit more unstructured and harder to understand. Instead we targeted the classes `Ticket` and `Route` in the first iteration. In the second iteration we added the test cases for `Ticket` and `Route` that we identified missing and targeted `IBBBCommand` and `BBB` as well. The third iteration was the last one and only included test cases that were identified to be missing in iteration two. The `RouteStatus` enumeration is not targeted individually because this is done implicitly when testing the `Route` class.

We used the testing framework Jest that enabled us to run automated tests on individual files. We did not find any non-reachable code but we found several faults that we fixed as described in the documented below.

## 2 Description of Coverage Screenshot

The coverage tool shows a table with 6 columns. Each row shows a file that has been tested and below the table there is some additional information about how many passed/failed test suites and about how many tests out of these test suits passed/failed. Each row in the table shows coverage information of a certain file and includes percentage of covered statements, branches, functions and lines. Given any lines are still uncovered after running the tests, those lines are shown in the most right column. The screenshot below shows an example of the coverage output where no tests have been executed.



| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
|---|---|---|---|---|---|
| All files | 0 | 0 | 0 | 0 | |
| BBB.js | 0 | 0 | 0 | 0 | ... 65,66,67,68,69 |
| IBBBCommand.js | 0 | 0 | 0 | 0 | ... 76,278,283,285 |
| Route.js | 0 | 0 | 0 | 0 | ... 07,209,211,213 |
| RouteStatus.js | 0 | 0 | 0 | 0 | 2,4,5,6,7,8 |
| Ticket.js | 0 | 0 | 0 | 0 | ... 53,54,55,57,59 |

Test Suites: 4 failed, 4 total
Tests: 0 total
Snapshots: 0 total
Time: 4.241s
Ran all test suites.

The goal for this assignment is to reach a value of 100% in the column `% Branch` for all files without any tests failing.

# 3 Notes

We found that the document was better readable when not using tables and decided to use a list formatting for better readability. Some test cases required the targeted class to be in a certain state. Therefore, we added a `Precondition` to those test cases. For instance, the precondition `Route{ id: ‘‘R1’’, source: "Madrid", destination: "Toledo", ...}` means that a `Route` instance should be initialized with the stated properties in the arrangement of the test case. If no class name is stated at the beginning, it means that we are referring to a simple object representation or single variables instead of a class instance. A value of `N/A` indicates that no precondition or input is required. For shortening the definition of a list of `Tickets` we used values as for instance T1_R1_9 which corresponds to a `Ticket` instance on `Route` R1 with seat 9 which is not boarded, e.g. Ticket { id: "T1", seat: 9, boarded: false }.

The complete source code of the tested application is shown in the attachment.

# 4 Iteration 1

Iteration 1 targets the classes `Ticket` and `Route`

## 4.1 Specify Test Cases

### 4.1.1 Class Ticket

**TC_Ticket_1** initializes correctly

**Goal** Test that the `Ticket` class initializes correctly

**Class** Ticket

**Method** constructor

**Precondition** N/A

**Input** { id: “T1”, seat: 1 }

**Expected Output** Ticket{ id: “T1”, seat: 1, boarded: false }

**TC_Ticket_2** throws error for invalid id

**Goal** Test that the `Ticket` class fails the initialization when an invalid id is passed

**Class** Ticket

**Method** constructor

**Precondition** N/A

**Input** { id: " ", seat: 1 }

**Expected Output** Error("Invalid id")

**TC_Ticket_3** throws error for invalid seat

**Goal** Test that the `Ticket` class fails the initialization when an invalid seat number is passed

**Class** Ticket

**Method** constructor

**Precondition** N/A

**Input** { id: "T1", seat: -1 }

**Expected Output** Error("Invalid seat")

**TC_Ticket_4** changes value correctly

**Goal** Test that the `boarded` property changes it's value correctly

**Class** Ticket

**Method** setter boarded

**Precondition** Ticket{ boarded: false }

**Input** true

**Expected Output** Ticket{ boarded: true }

**TC_Ticket_5** creates object correctly

**Goal** Test that the `toObject()` method creates a correct object representation of the `Ticket`

**Class** Ticket

**Method** toObject

**Precondition** Ticket{ id: "T1", seat: 1, boarded: false }

**Input** N/A

**Expected Output** Object{id: "T1", seat: 1, boarded: false }

**TC_Ticket_6** creates ticket correctly

**Goal** Test the the `fromObject()` method creates a correct `Ticket` instance from it's object representation

**Class** Ticket

**Method** fromObject

**Precondition** N/A

**Input** Object{ id: "T1", seat: 1, boarded: false }

**Expected Output** Ticket{id: "T1", seat: 1, boarded: false }

**TC_Ticket_7** throws error for invalid ticket object

**Goal** Test that the `fromObject()` method throws an error if an invalid object representation is passed

**Class** Ticket

**Method** fromObject

**Precondition** N/A

**Input** Object{ id_X: "T1", seat: 1, boarded: false }

**Expected Output** Error("Invalid object")

#### 4.1.2 Class Route

**TC_Route_1** initializes correctly

**Goal** Test that the `Route` class initializes correctly

**Class** Route

**Method** constructor

**Precondition** N/A

**Input** { id: "R1", source: "Madrid", destination: "Toledo", capacity: 10 }

**Expected Output** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}

**TC_Route_2** throws error on invalid id

**Goal** Test that the `Route` class fails initialization if an invalid id is passed

**Class** Route

**Method** constructor

**Precondition** N/A

**Input** { id: “ ”, source: “Madrid”, destination: “Toledo”, capacity: 10 }

**Expected Output** Error(“Invalid id”)

**TC_Route_3** throws error on invalid source

**Goal** Test that the `Route` class fails initialization if an invalid source is given

**Class** Route

**Method** constructor

**Precondition** N/A

**Input** { id: “R1”, source: “ ”, destination: “Toledo”, capacity: 10 }

**Expected Output** Error(“Invalid source”)

**TC_Route_4** throws error on invalid destination

**Goal** Test that the `Route` class fails initialization if an invalid destination is given

**Class** Route

**Method** constructor

**Precondition** N/A

**Input** { id: “R1”, source: “Madrid”, destination: null, capacity: 10 }

**Expected Output** Error(“Invalid source”)

**TC_Route_5** throws error on invalid capacity

**Goal** Test that the `Route` class fails initialization if an invalid capacity is given

**Class** Route

**Method** constructor

**Precondition** N/A

**Input** { id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: -1 }

**Expected Output** Error(“Invalid capacity”)

**TC_Route_6** returns status “travelling” on travelling

**Goal** Test that the property `status` returns "travelling" if it has departed

**Class** Route

**Method** getter status

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 9]}

**Input** N/A

**Expected Output** “travelling”

**Note** The date set for departed is an example. For the test the current date and time will be set

**TC_Route_7** returns status “empty” on empty

**Goal** Test that the property `status` returns "empty" if it has not departed and no ticket has been purchased

**Class** Route

**Method** getter status

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

**Input** N/A

**Expected Output** “empty”

**TC_Route_8** returns status “available” on available

**Goal** Test that the property `status` returns "available" if it has not departed and at least one ticket has been purchased

**Class** Route

**Method** getter status

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}

**Input** N/A

**Expected Output** “available”

**TC_Route_9** returns status “full” on full

**Goal** Test that the property `status` returns "full" if it has not departed and all available tickets have been purchased

**Class** Route

**Method** getter status

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9, . . . , T_R1_0], departed: null, availableSeats: []}

**Input** N/A

**Expected Output** “full”

**TC_Route_10** successfully purchase ticket

**Goal** Test that the method `purchaseTicket()` successfully creates a new `Ticket` instance and removes one available seat

**Class** Route

**Method** purchaseTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

**Input** N/A

**Expected Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: false } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9], departed: null, availableSeats: [0, . . . , 8]}

**TC_Route_11** purchase ticket fails on no available tickets

**Goal** Test that the method `purchaseTicket()` fails if there are no available seats left

**Class** Route

**Method** purchaseTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

**Input** N/A

**Expected Output** { success: false, reason: “No tickets available” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**TC_Route_12** successfully board ticket

**Goal** Test that the method `boardTicket()` successfully changes the property "boarded" of the corresponding `Ticket` to "true" and does not alter any other `Ticket`

**Class** Route

**Method** boardTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: false }

**Input** { ticketId: “T1_R1_9” }

**Expected Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: true } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**TC_Route_13** board ticket fails for invalid ticketId

**Goal** Test that the method `boardTicket()` fails if the passed `ticketId` does not match any `Ticket`

**Class** Route

**Method** boardTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**Input** { ticketId: “T1_R1_XXX” }

**Expected Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**TC_Route_14** board ticket fails for already boarded ticketId

**Goal** Test that the method `boardTicket()` fails if the property `boarded` of the corresponding `Ticket` is already set to true

**Class** Route

**Method** boardTicket

**Precondition** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: "T1_R1_9", seat: 9, boarded: true }

**Input** { ticketId: "T1_R1_9" }

**Expected Output** { success: false, reason: "Ticket is already boarded" }, Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: "T1_R1_9", seat: 9, boarded: true }

**TC_Route_15** successfully cancel ticket

**Goal** Test that the method `cancelTicket()` successfully removes the corresponding `Ticket` from the list of `Tickets` and adds the seat of the `Ticket` back to the list of the available seats.

**Class** Route

**Method** cancelTicket

**Precondition** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: "T1_R1_9", seat: 9, boarded: false }

**Input** { ticketId: "T1_R1_9" }

**Expected Output** { success: true, ticket: Ticket{ id: "T1_R1_9", seat: 9, boarded: false } }, Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_8, ... T1_R1_0], departed: null, availableSeats: [9]}

**TC_Route_16** cancel ticket fails for invalid ticketId

**Goal** Test that the method `cancelTicket()` fails if the passed `ticketId` does not match any `Ticket`

**Class** Route

**Method** cancelTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

**Input** { ticketId: “T1_R1_XXX” }

**Expected Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

**TC_Route_17** cancel ticket fails for already boarded ticketId

**Goal** Test that the method `cancelTicket()` fails if the property `boarded` of the corresponding `Ticket` is already set to true

**Class** Route

**Method** cancelTicket

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

**Input** { ticketId: “T1_R1_9” }

**Expected Output** { success: false, reason: “Ticket is already boarded” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

**TC_Route_18** depart successfully sets departure time

**Goal** Test that the method `depart()` successfully sets the departure of the `Route` with a current timestamp

**Class** Route

**Method** depart

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

**Input** N/A

**Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 9]}

**Note** The date set for departed is an example. For the test the current date and time will be set

**TC_Route_19** hasArrived successfully resets the Route

**Goal** Test that the method `hasArrived()` successfully resets the departure to null if the departure is set and at least 10 seconds have been passed since the departure

**Class** Route

**Method** hasArrived

**Precondition** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: "2008-09-15T15:53:00", availableSeats: []}

**Input** N/A

**Expected Output** true, Route{ id: "R1", source: "Toledo", destination: "Madrid", capacity: 10, tickets: [], departed: null, availableSeats: [0, ..., 9]}

**Note** The date set for departed is an example. For the test the current date and time will be set

**TC_Route_20** hasArrived does not reset the Route if no departed yet

**Goal** Test that the method `hasArrived()` does nothing if no departure is set

**Class** Route

**Method** hasArrived

**Precondition** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**Input** N/A

**Expected Output** false, Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: null, availableSeats: []}

**TC_Route_21** hasArrived does not reset the Route if still travelling

**Goal** Test that the method `hasArrived()` does nothing if the departure is set but less than 10 seconds have been passed since departure

**Class** Route

**Method** hasArrived

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: “2008-09-15T15:53:00”, availableSeats: []}

**Input** N/A

**Expected Output** false, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_0], departed: “2008-09-15T15:53:00”, availableSeats: []}

**Note** The date set for departed is an example. For the test the current date and time will be set so that the 10 seconds have not passed yet

**TC_Route_22** fromObject successfully creates new Route with set departure

**Goal** Test that the `fromObject()` method successfully creates a `Route` instance from it's object representation that has a departure set

**Class** Route

**Method** fromObject

**Precondition** N/A

**Input** { id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**Note** The date set for departed is an example

**TC_Route_23** fromObject successfully creates new Route without set departure and tickets

**Goal** Test that the `fromObject()` method successfully creates a `Route` instance from it's object representation that does not have a departure set

**Class** Route

**Method** fromObject

**Precondition** N/A

**Input** { id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

**Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

**TC_Route_24** toObject successfully creates new Object with set departure

**Goal** Test that the `toObject()` method successfully creates a object representation of the `Route` that has a departure set

**Class** Route

**Method** toObject

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**Input** N/A

**Expected Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**TC_Route_25** toObject successfully creates new Object without departure

**Goal** Test that the `toObject()` method successfully creates a object representation of the `Route` that does not have a departure set

**Class** Route

**Method** toObject

**Precondition** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_3], departed: null, availableSeats: [0, 1, 2]}

**Input** N/A

**Expected Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_3], departed: null, availableSeats: [0, 1, 2]}

## 4.2 Run Test Cases

### 4.2.1 Class Ticket

- **TC_Ticket_1**

  **Expected Output** Ticket{ id: “T1”, seat: 1, boarded: false }
  **Observed Output** Ticket{ id: “T1”, seat: 1, boarded: false }
  **Failure** None

- **TC_Ticket_2**

  **Expected Output** Error(“Invalid id”)
  **Observed Output** Error(“Invalid id”)
  **Failure** None

- **TC_Ticket_3**

  **Expected Output** Error(“Invalid seat”)
  **Observed Output** Error(“Invalid seat”)
  **Failure** None

- **TC_Ticket_4**

  **Expected Output** Ticket{ boarded: true }
  **Observed Output** Ticket{ boarded: true }
  **Failure** None

- **TC_Ticket_5**

  **Expected Output** Object{id: “T1”, seat: 1, boarded: false }
  **Observed Output** Object{id: “T1”, seat: 1, boarded: false }
  **Failure** None

- **TC_Ticket_6**

  **Expected Output** Ticket{id: “T1”, seat: 1, boarded: false }
  **Observed Output** Ticket{id: “T1”, seat: 1, boarded: false }
  **Failure** None

- **TC_Ticket_7**

  **Expected Output** Error(“Invalid object”)
  **Observed Output** Error(“Invalid object”)
  **Failure** None

#### 4.2.2 Class Route

- **TC_Route_1**

  **Expected Output** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}

  **Observed Output** Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}

  **Failure** None

- **TC_Route_2**

  **Expected Output** Error("Invalid id")

  **Observed Output** Error("Invalid id")

  **Failure** None

- **TC_Route_3**

  **Expected Output** Error("Invalid source")

  **Observed Output** Error("Invalid source")

  **Failure** None

- **TC_Route_4**

  **Expected Output** Error("Invalid source")

  **Observed Output** Error("Invalid source")

  **Failure** None

- **TC_Route_5**

  **Expected Output** Error("Invalid capacity")

  **Observed Output** Error("Invalid capacity")

  **Failure** None

- **TC_Route_6**

  **Expected Output** "travelling"

  **Observed Output** 0

  **Failure** Yes

- **TC_Route_7**

  **Expected Output** “empty”

  **Observed Output** 1

  **Failure** Yes

- **TC_Route_8**

  **Expected Output** “available”

  **Observed Output** 3

  **Failure** Yes

- **TC_Route_9**

  **Expected Output** “full”

  **Observed Output** 2

  **Failure** Yes

- **TC_Route_10**

  **Expected Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: false } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9], departed: null, availableSeats: [0, . . . , 8]}

  **Observed Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: false } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9], departed: null, availableSeats: [0, . . . , 8]}

  **Failure** None

- **TC_Route_11**

  **Expected Output** { success: false, reason: “No tickets available” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Observed Output** { success: false, reason: “No tickets available” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

**Failure** None

- **TC_Route_12**

  **Expected Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: true } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Observed Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: true } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Failure** None

- **TC_Route_13**

  **Expected Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Observed Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Failure** None

- **TC_Route_14**

  **Expected Output** { success: false, reason: “Ticket is already boarded” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

  **Observed Output** { success: false, reason: “Ticket is already boarded” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

  **Failure** None

- **TC_Route_15**

**Expected Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: false } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_8, . . . T1_R1_0], departed: null, availableSeats: [9]}

**Observed Output** { success: true, ticket: Ticket{ id: “T1_R1_9”, seat: 9, boarded: false } }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_8, . . . T1_R1_0], departed: null, availableSeats: []}

**Failure** Yes

- **TC_Route_16**

  **Expected Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Observed Output** { success: false, reason: “Ticket does not exist” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Failure** None

- **TC_Route_17**

  **Expected Output** { success: false, reason: “Ticket is already boarded” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

  **Observed Output** { success: false, reason: “Ticket is already boarded” }, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}, T1_R1_9{ id: “T1_R1_9”, seat: 9, boarded: true }

  **Failure** None

- **TC_Route_18**

  **Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 9]}

**Observed Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 9]}

**Failure** None

- **TC_Route_19**

  **Expected Output** true, Route{ id: “R1”, source: “Toledo”, destination: “Madrid”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

  **Observed Output** true, Route{ id: “R1”, source: “Toledo”, destination: “Madrid”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}

  **Failure** None

- **TC_Route_20**

  **Expected Output** false, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Observed Output** false, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: null, availableSeats: []}

  **Failure** None

- **TC_Route_21**

  **Expected Output** false, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: “2008-09-15T15:53:00”, availableSeats: []}

  **Observed Output** false, Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_0], departed: “2008-09-15T15:53:00”, availableSeats: []}

  **Failure** None

- **TC_Route_22**

  **Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, . . . T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**Observed Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

**Failure** None

- **TC_Route_23**

  **Expected Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, ..., 9]}

  **Observed Output** Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, ..., 9]}

  **Failure** None

- **TC_Route_24**

  **Expected Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

  **Observed Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: “2008-09-15T15:53:00”, availableSeats: [0, 1, 2]}

  **Failure** None

- **TC_Route_25**

  **Expected Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: null, availableSeats: [0, 1, 2]}

  **Observed Output** Object{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T1_R1_9, ... T1_R1_3], departed: null, availableSeats: [0, 1, 2]}

  **Failure** None

## 4.3 Check Coverage

```
-------------|---------|----------|---------|---------|-------------------
File         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------|---------|----------|---------|---------|-------------------
All files    |   36.06 |    39.22 |   35.37 |   36.24 |
 BBB.js      |       0 |        0 |       0 |       0 | ... 58,61,66,68,69
 IBBBCommand.js |     0 |        0 |       0 |       0 | ... 78,280,285,287
 Route.js    |   98.18 |    95.92 |     100 |   98.11 |         191,200
 RouteStatus.js |  100 |      100 |     100 |     100 |
 Ticket.js   |     100 |      100 |     100 |     100 |
-------------|---------|----------|---------|---------|-------------------
Test Suites: 1 failed, 1 passed, 2 total
Tests:       5 failed, 27 passed, 32 total
Snapshots:   0 total
Time:        3.741s, estimated 4s
Ran all test suites.
```

Using the coverage tool we identified following lines/branches not covered (red line means the line was not executed, "I" indicates that the "if" path was never taken). The test cases covering those lines will be added in iteration 2.

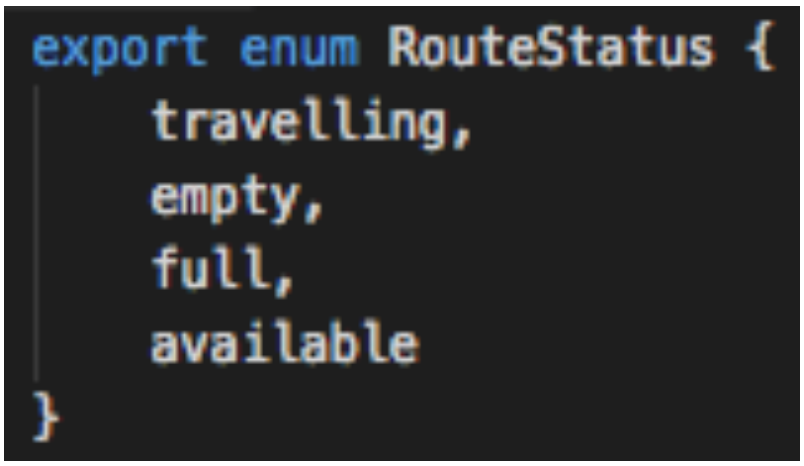
```
Route.fromObject = function (object) {
    I if (!object.hasOwnProperty('id') ||
        !object.hasOwnProperty('source') ||
        !object.hasOwnProperty('destination') ||
        !object.hasOwnProperty('capacity') ||
        !object.hasOwnProperty('departed') ||
        !object.hasOwnProperty('availableSeats') ||
        !object.hasOwnProperty('tickets')) {
        throw new Error('Invalid object');
    }
    var route = new Route(object.id, object.source, object.destination, object.capacity
    if (object.departed === null) {
        route._departed = null;
    }
    else {
        route._departed = moment(object.departed);
        I if (!route._departed.isValid()) {
            throw new Error('Invalid departed time');
        }
    }
    route._availableSeats = object.availableSeats;
    for (var i in object.tickets) {
        var ticket = Ticket_1.Ticket.fromObject(object.tickets[i]);
        route._tickets.push(ticket);
    }
    return route;
};
return Route;
```

## 4.4 Trace failures to faults

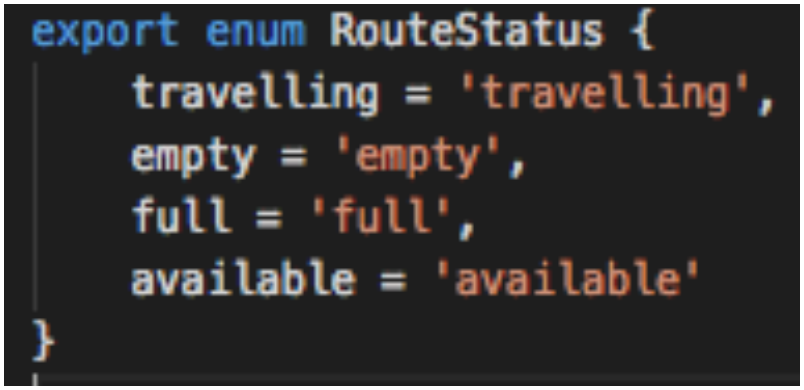
### 4.4.1 TC_Route_6, TC_Route_7, TC_Route_8, TC_Route_9

**Failure** The output of the `status` property of the `Route` class returns an `int` value instead of a meaningful `string` value

**Fault** The `RouteStatus` enumeration uses `int` representation (default behavior) instead of `string` representations




**Fix** Assign `string` values to `RouteStatus` enumeration:




#### 4.4.2 TC_Route_15

**Failure** When cancelling a `Ticket` the seat that is available again is not added again to the list of available seats

**Fault** The `cancelTicket()` method misses the necessary statements that push the seat of the cancelled `Ticket` back onto the `availableSeats` list


**Fix** Added the seat of the ticket to the list of available seats:


# 5 Iteration 2

Iteration 2 first specifies the test cases that were identified missing from iteration 1. Then `IBBBCommand` and `BBB` are targeted.

## 5.1 Specify Test Cases

### 5.1.1 Class Route (Identified to be missing in last iteration)

**TC_Route_26** fromObject fails on invalid object

**Goal** Test that the `fromObject()` method throws an error if an invalid object representation is passed

**Class** Route

**Method** fromObject

**Precondition** N/A

**Input** { id_X: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}

**Expected Output** Error(‘Invalid object’)

**Note** The date set for departed is an example

**TC_Route_27** fromObject fails on invalid departure time

**Goal** Test that the `fromObject()` method throws an error if departed is set to an invalid value

**Class** Route

**Method** fromObject

**Precondition** N/A

**Input** { id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: “4711”, availableSeats: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}

**Expected Output** Error(‘Invalid departed time’)

### 5.1.2 Class IBBBCommand

**TC_RegisterRouteCommand_1** returns correct id

**Goal** Test that the `commandId` of the `RegisterRouteCommand` returns the correct value

**Class** RegisterRouteCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** ‘registerroute’

**TC_RegisterRouteCommand_2** fails for invalid number of arguments

**Goal** Test that the `RegisterRouteCommand` displays the correct error message if an invalid number of arguments is given

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** []

**Expected Output** BBB{ _routes: [] } Console: 'Invalid number of arguments given'

**TC_RegisterRouteCommand_3** fails for invalid route

**Goal** Test that the `RegisterRouteCommand` displays the correct error message if an invalid value for route is given

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** [" ", "Madrid", "Toledo", 10]

**Expected Output** BBB{ _routes: [] } Console: 'Invalid value for route given'

**TC_RegisterRouteCommand_4** fails for invalid source

**Goal** Test that the `RegisterRouteCommand` displays the correct error message if an invalid value for source is given

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** ["R1", null, "Toledo", 10]

**Expected Output** BBB{ _routes: [] } Console: 'Invalid value for source given'

**TC_RegisterRouteCommand_5** fails for invalid destination

**Goal** Test that the `RegisterRouteCommand` displays the correct error message if an invalid destination is given

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** ["R1", "Madrid", undefined, 10]

**Expected Output** BBB{ _routes: [] } Console: 'Invalid value for destination given'

**TC_RegisterRouteCommand_6** fails for invalid capacity

**Goal** Test that the `RegisterRouteCommand` displays the correct error message if an invalid capacity is given

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** ["R1", "Madrid", "Toledo", "asdf"]

**Expected Output** BBB{ _routes: [] } Console: 'Invalid value for capacity'

**TC_RegisterRouteCommand_7** succeeds for valid input

**Goal** Test that the `RegisterRouteCommand` successfully registers a new `Route`

**Class** RegisterRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [] }

**Input** ["R1", "Madrid", "Toledo", 10"]

**Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]} Console: "Created route R1 from Madrid to Toledo with 10 seats"

**TC_DeleteRouteCommand_1** returns correct id

**Goal** Test that the `commandId` of the "DeleteRouteCommand" returns the correct value

**Class** DeleteRouteCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** 'deleteroute'

**TC_DeleteRouteCommand_2** fails for invalid number of arguments

**Goal** Test that the `DeleteRouteCommand` displays the correct error message if an invalid number of arguments is given

**Class** DeleteRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** []

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]} Console: ‘Invalid number of arguments given’

**TC_DeleteRouteCommand_3** fails for invalid route

**Goal** Test that the `DeleteRouteCommand` displays the correct error message if an invalid value for route is given

**Class** DeleteRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** [“ ”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]} Console: ‘Invalid value for route given’

**TC_DeleteRouteCommand_4** fails for route with purchased tickets

**Goal** Test that the `DeleteRouteCommand` does not delete a `Route` that includes already purchased `Tickets`

**Class** DeleteRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** [“R1”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]} Console: “Cannot delete route R1 because there are 1 tickets booked”

**TC_DeleteRouteCommand_5** succeeds for valid input

**Goal** Test that the `DeleteRouteCommand` successfully deletes a `Route` that has no purchased `Tickets`

**Class** DeleteRouteCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** [“R1”]

**Expected Output** BBB{ _routes: [] } Console: “Successfully deleted route R1”

**TC_DepartCommand_1** returns correct id

**Goal** Test that the `commandId` of the "DepartCommand" returns the correct value

**Class** DepartCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** ‘depart’

**TC_DepartCommand_2** fails for invalid number of arguments

**Goal** Test that the `DepartCommand` displays the correct error message if an invalid number of arguments is given

**Class** DepartCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** []

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]} Console: ‘Invalid number of arguments given’

**TC_DepartCommand_3** fails for invalid route

**Goal** Test that the `DepartCommand` displays the correct error message if an invalid value for route is given

**Class** DepartCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** [“R_X”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]} Console: ‘Invalid value for route given’

**TC_DepartCommand_4** succeeds for valid route

**Goal** Test that the `DepartCommand` successfully sets the departure of a `Route`

**Class** DepartCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}

**Input** [“R1”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 8]}]} Console: ‘R1 departed’

**TC_StatusCommand_1** returns correct id

**Goal** Test that the `commandId` of the "StatusCommand" returns the correct value

**Class** StatusCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** 'status'

**TC_StatusCommand_2** fails for invalid number of arguments

**Goal** Test that the `StatusCommand` displays the correct error message if an invalid number of arguments is given

**Class** StatusCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}

**Input** ["A", "B"]

**Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]} Console: 'Invalid number of arguments given'

**TC_StatusCommand_3** fails for specifying not existing route

**Goal** Test that the `StatusCommand` does print the correct error message when specifying a not existing `Route`

**Class** StatusCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}

**Input** [“R3”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: ‘Route R3 does not exist’

**TC_StatusCommand_4** prints status of one specified route successfully

**Goal** Test that the `StatusCommand` prints the correct status of a given `Route`

**Class** StatusCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** [“R2”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: ‘R2: empty’

**TC_StatusCommand_5** prints status without specified route successfully

**Goal** Test that the `StatusCommand` prints the correct status of all `Routes` if no `Route` was given

**Class** StatusCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** []

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: “R1: available R2: empty’”

**TC_BuyCommand_1** returns correct id

**Goal** Test that the `commandId` of the "BuyCommand" returns the correct value

**Class** BuyCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** ‘buy’

**TC_BuyCommand_2** fails for not existing route

**Goal** Test that the `BuyCommand` does print the correct error message when specifying a not existing `Route`

**Class** BuyCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** [“R3”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: ‘Route R3 does not exist’

**TC_BuyCommand_3** fails for sold out route

**Goal** Test that the `BuyCommand` does not buy a `Ticket` if the `Route` is already sold out

**Class** BuyCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9, . . . T_R1_0], departed: null, availableSeats: []}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** [“R1”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9, . . . T_R1_0]], departed: null, availableSeats: []}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: ‘Sorry! You were too late! Tickets are sold out!’

**TC_BuyCommand_4** succeeds for valid route

**Goal** Test that the `BuyCommand` successfully buys a `Ticket` if the `Route` is not sold out

**Class** BuyCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}

**Input** [“R1”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9, T_R1_8], departed: null, availableSeats: [0, . . . , 7]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: ‘Successfully purchased ticket T_R1_8 on route R1 from Madrid to Toledo’

**TC_CheckinCommand_1** returns correct id

**Goal** Test that the `commandId` of the "CheckinCommand" returns the correct value

**Class** CheckinCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** 'checkin'

**TC_CheckinCommand_2** fails for invalid number of arguments

**Goal** Test that the `CheckinCommand` displays the correct error message if an invalid number of arguments is given

**Class** CheckinCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}]}, Ticket{ id: "T_R1_9", seat: 9, boarded: false }

**Input** []

**Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}]}, Ticket{ id: "T_R1_9", seat: 9, boarded: false } Console: "Invalid number of arguments given"

**TC_CheckinCommand_3** fails for invalid value for ticket

**Goal** Test that the `CheckinCommand` displays the correct error message if an invalid `Ticket` is specified

**Class** CheckinCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}]}, Ticket{ id: "T_R1_9", seat: 9, boarded: false }

**Input** [" "]

**Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}]}, Ticket{ id: "T_R1_9", seat: 9, boarded: false } Console: "Invalid value for ticket given"

**TC_CheckinCommand_4** fails for not existing ticket

**Goal** Test that the `CheckinCommand` displays the correct error message if a not existing `Ticket` is specified

**Class** CheckinCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** [“T_R1_X”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false } Console: “Ticket with id T_R1_X does not exist”

**TC_CheckinCommand_5** fails already boarded ticket

**Goal** Test that the `CheckinCommand` fails if a `Ticket` is specified that has already been boarded

**Class** CheckinCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true }

**Input** [“T_R1_9”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true } Console: “Unable to checkin ticket T_R1_9: Ticket is already boarded”

**TC_CheckinCommand_6** succeeds for valid ticket

**Goal** Test that the `CheckinCommand` successfully boards a `Ticket` that has not been boarded yet

**Class** CheckinCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** [“T_R1_9”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true } Console: “Successfully checked in ticket T_R1_9 on route R1 from Madrid to Toledo and assigned seat 9”

**TC_CancelCommand_1** returns correct id

**Goal** Test that the `commandId` of the "CancelCommand" returns the correct value

**Class** CancelCommand

**Method** commandId get

**Precondition** N/A

**Input** N/A

**Expected Output** ‘cancel’

**TC_CancelCommand_2** fails for invalid number of arguments

**Goal** Test that the `CancelCommand` displays the correct error message if an invalid number of arguments is given

**Class** CancelCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** []

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false } Console: “Invalid number of arguments given”

**TC_CancelCommand_3** fails for invalid value for ticket

**Goal** Test that the `CancelCommand` displays the correct error message if an invalid `Ticket` is specified

**Class** CancelCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** [“ “]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false } Console: “Invalid value for ticket given”

**TC_CancelCommand_4** fails for not existing ticket

**Goal** Test that the `CancelCommand` displays the correct error message if a not existing `Ticket` is specified

**Class** CancelCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** [“T_R1_X”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false } Console: “Ticket with id T_R1_X does not exist”

**TC_CancelCommand_5** fails already boarded ticket

**Goal** Test that the `CancelCommand` fails if the specified `Ticket` has already been boarded

**Class** CancelCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true }

**Input** [“T_R1_9”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true } Console: “Unable to cancel ticket T_R1_9: Ticket is already boarded”

**TC_CancelCommand_6** succeeds for valid ticket

**Goal** Test that the `CancelCommand` successfully cancels a `Ticket` das has not been boarded yet

**Class** CancelCommand

**Method** execute

**Precondition** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }

**Input** [“T_R1_9”]

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]} Console: “Cancelled ticket T_R1_9 on route R1 from Madrid to Toledo”

### 5.1.3 Class BBB

**TC_BBB_1** successfully writes file

**Goal** Test that the method `saveRoutes()` successfully creates a database file persisting the existing `Routes`

**Class** BBB

**Method** saveRoutes

**Precondition** routes: [{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [{id: “T_R1_9”, “seat”: 9, “boarded”: false}], departed: null, availableSeats: [0, . . . , 8]}, { id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]

**Input** N/A

**Expected Output** file: [{ “id”: “R1”, “source”: “Madrid”, “destination”: “Toledo”, “capacity”: 10, “tickets”: [{id: “T_R1_9”, “seat”: 9, “boarded”: false}], “departed”: null, “availableSeats”: [0, . . . , 8]}, { “id”: “R2”, “source”: “Barcelona”, “destination”: “Valencia”, “capacity”: 10, “tickets”: [], “departed”: null, “availableSeats”: [0, . . . , 9]}]

**TC_BBB_2** successfully reads file with routes

**Goal** Test that the method `loadRoutes()` successfully reads and initilaizes the `Routes` from an existing database file

**Class** BBB

**Method** loadRoutes

**Precondition** routes: undefined file: [{ “id”: “R1”, “source”: “Madrid”, “destination”: “Toledo”, “capacity”: 10, “tickets”: [{id: “T_R1_9”, “seat”: 9, “boarded”: false)}], “departed”: null, “availableSeats”: [0, . . . , 8]}, { “id”: “R2”, “source”: “Barcelona”, “destination”: “Valencia”, “capacity”: 10, “tickets”: [], “departed”: null, “availableSeats”: [0, . . . , 9]}]

**Input** N/A

**Expected Output** routes: [{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, { id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]

**TC_BBB_3** successfully reads without routes

**Goal** Test that the method `loadRoutes()` successfully creates a empty list of `Routes` if a database without `Routes` is read

**Class** BBB

**Method** loadRoutes

**Precondition** routes: undefined file: []

**Input** N/A

**Expected Output** routes: []

**TC_BBB_4** does not read not existing file

**Goal** Test that the method `loadRoutes()` successfully creates a empty list of `Routes` if no database file is available

**Class** BBB

**Method** loadRoutes

**Precondition** routes : undefined, filePath: "./test/db"

**Input** N/A

**Expected Output** routes: []

**TC_BBB_5** fails for no arguments given

**Goal** Test that the method `parseCommand()` displays the correct error message if no arguments are given

**Class** BBB

**Method** parseCommand

**Precondition** N/A

**Input** args: []

**Expected Output** Console: "No argument was given"

**TC_BBB_6** fails for not existing command

**Goal** Test that the method `parseCommand()` displays the correct error message if a not existing `Command` is specified

**Class** BBB

**Method** parseCommand

**Precondition** N/A

**Input** args: ["asdf"]

**Expected Output** Console: "Command asdf does not exist"

**TC_BBB_7** succeeds for existing command

**Goal** Test that the method `parseCommand()` executes the `execute()` method of the specified `Command`

**Class** BBB

**Method** parseCommand

**Precondition** N/A

**Input** args: ["status"]

**Expected Output** _commands["status"].execute was called

## 5.2 Run Test Cases

### 5.2.1 Class Route

- **TC_Route_26**

  **Expected Output** Error('Invalid object')

  **Observed Output** Error('Invalid object')

  **Failure** None

- **TC_Route_27**

  **Expected Output** Error('Invalid departed time')

  **Observed Output** Route { id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: "4711-01-01T00:00:00.000Z", availableSeats: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}

  **Failure** Yes

### 5.2.2 Class IBBBCommand

- **TC_RegisterRouteCommand_1**

  **Expected Output** 'registerroute'

  **Observed Output** 'registerroute'

  **Failure** None

- **TC_RegisterRouteCommand_2**

  **Expected Output** BBB{ _routes: [] }
  Console: 'Invalid number of arguments given'

  **Observed Output** BBB{ _routes: [] }
  Console: 'Invalid number of arguments given'

  **Failure** None

- **TC_RegisterRouteCommand_3**

  **Input** [" ", "Madrid", "Toledo", 10]

  **Expected Output** BBB{ _routes: [] }
  Console: 'Invalid value for route given'

  **Observed Output** BBB{ _routes: [] }
  Console: 'Invalid value for route given'

**Failure** None

- **TC_RegisterRouteCommand_4**

  **Expected Output** Console: 'Invalid value for source given'

  **Observed Output** TypeError('Cannot read property 'trim' of null')

  **Failure** Yes

- **TC_RegisterRouteCommand_5**

  **Expected Output** Console: 'Invalid value for destination given'

  **Observed Output** TypeError('Cannot read property 'trim' of undefined')

  **Failure** Yes

- **TC_RegisterRouteCommand_6**

  **Expected Output** Console: 'Invalid value for capacity'

  **Observed Output** RangeError(Invalid array length)

  **Failure** Yes

- **TC_RegisterRouteCommand_7**

  **Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: "Created route R1 from Madrid to Toledo with 10 seats"

  **Observed Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: "Created route R1 from Madrid to Toledo with 10 seats"

  **Failure** None

- **TC_DeleteRouteCommand_1**

  **Expected Output** 'deleteroute'

  **Observed Output** 'deleteroute'

  **Failure** None

- **TC_DeleteRouteCommand_2**

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: ‘Invalid number of arguments given’

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: ‘Invalid number of arguments given’

**Failure** None

- **TC_DeleteRouteCommand_3**

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: ‘Invalid value for route given’

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: ‘Invalid value for route given’

**Failure** None

- **TC_DeleteRouteCommand_4**

**Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: “Cannot delete route R1 because there are 1 tickets booked”

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
Console: “Cannot delete route R1 because there are 1 tickets booked”

**Failure** None

- **TC_DeleteRouteCommand_5**

**Expected Output** BBB{ _routes: [] }
Console: “Successfully deleted route R1”

**Observed Output** BBB{ _routes: [] }
Console: “Successfully deleted route R1”

**Failure** None

- **TC_DepartCommand_1**

  **Expected Output** ‘depart’

  **Observed Output** ‘depart’

  **Failure** None

- **TC_DepartCommand_2**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
  Console: ‘Invalid number of arguments given’

  **Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}
  Console: ‘Invalid number of arguments given’

  **Failure** None

- **TC_DepartCommand_3**

  **Expected Output** Console: ‘Invalid value for route given’

  **Observed Output** Console: ‘Route R_X does not exist’

  **Failure** Yes

- **TC_DepartCommand_4**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 8]}]}
  Console: ‘R1 departed’

  **Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: “2008-09-15T15:53:00”, availableSeats: [0, . . . , 8]}]}
  Console: ‘R1 departed’

  **Failure** None

- **TC_StatusCommand_1**

**Expected Output** 'status'

**Observed Output** 'status'

**Failure** None

- **TC_StatusCommand_2**

  **Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: 'Invalid number of arguments given'

  **Observed Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: 'Invalid number of arguments given'

  **Failure** None

- **TC_StatusCommand_3**

  **Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: 'Route R3 does not exist'

  **Observed Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]}
  Console: 'Route R3 does not exist'

  **Failure** None

- **TC_StatusCommand_4**

  **Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed:

null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
Console: ‘R2: empty’

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
Console: ‘R2: empty’

**Failure** None

- **TC_StatusCommand_5**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: “R1: available R2: empty’”

  **Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: “R1: available R2: empty’”

  **Failure** None

- **TC_BuyCommand_1**

  **Expected Output** ‘buy’

  **Observed Output** ‘buy’

  **Failure** None

- **TC_BuyCommand_2**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null,

availableSeats: [0, . . . , 9]}]}
Console: 'Route R3 does not exist'

**Observed Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
Console: 'Route R3 does not exist'

**Failure** None

- **TC_BuyCommand_3**

  **Expected Output** Console: 'Sorry! You were too late! Tickets are sold out!'

  **Observed Output** TypeError(Cannot read property 'id' of undefined)

  **Failure** Yes

- **TC_BuyCommand_4**

  **Expected Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9, T_R1_8], departed: null, availableSeats: [0, . . . , 7]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: 'Successfully purchased ticket T_R1_8 on route R1 from Madrid to Toledo'

  **Observed Output** BBB{ _routes: [Route{ id: "R1", source: "Madrid", destination: "Toledo", capacity: 10, tickets: [T_R1_9, T_R1_8], departed: null, availableSeats: [0, . . . , 7]}, Route{ id: "R2", source: "Barcelona", destination: "Valencia", capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: 'Successfully purchased ticket T_R1_8 on route R1 from Madrid to Toledo'

  **Failure** None

- **TC_CheckinCommand_1**

  **Expected Output** 'checkin'

  **Observed Output** 'checkin'

  **Failure** None

- **TC_CheckinCommand_2**

  **Expected Output** Console: “Invalid number of arguments given”

  **Observed Output** Console: “Invalid number of arguments given” “Ticket with id null does not exist”

  **Failure** Yes

- **TC_CheckinCommand_3**

  **Expected Output** Console: “Invalid value for ticket given”

  **Observed Output** Console: “Invalid value for ticket given” “Ticket with id null does not exist”

  **Failure** Yes

- **TC_CheckinCommand_4**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }
  Console: “Ticket with id T_R1_X does not exist”

  **Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }
  Console: “Ticket with id T_R1_X does not exist”

  **Failure** None

- **TC_CheckinCommand_5**

  **Expected Output** Console: “Unable to checkin ticket T_R1_9: Ticket is already boarded”

  **Observed Output** TypeError(Cannot read property ‘seat’ of undefined)

  **Failure** Yes

- **TC_CheckinCommand_6**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat:

9, boarded: true }
Console: “Successfully checked in ticket T_R1_9 on route R1 from Madrid to Toledo and assigned seat 9”

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: true }
Console: “Successfully checked in ticket T_R1_9 on route R1 from Madrid to Toledo and assigned seat 9”

**Failure** None

- **TC_CancelCommand_1**

  **Expected Output** ‘cancel’

  **Observed Output** ‘cancel’

  **Failure** None

- **TC_CancelCommand_2**

  **Expected Output** Console: “Invalid number of arguments given”

  **Observed Output** Console: “Invalid number of arguments given”
  Console: “Ticket with id null does not exist”

  **Failure** Yes

- **TC_CancelCommand_3**

  **Expected Output** Console: “Invalid value for ticket given”

  **Observed Output** Console: “Invalid value for ticket given”
  Console: “Ticket with id null does not exist”

  **Failure** Yes

- **TC_CancelCommand_4**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }
  Console: “Ticket with id T_R1_X does not exist”

**Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, . . . , 8]}]}, Ticket{ id: “T_R1_9”, seat: 9, boarded: false }
Console: “Ticket with id T_R1_X does not exist”

**Failure** None

- **TC_CancelCommand_5**

  **Expected Output** Console: “Unable to cancel ticket T_R1_9: Ticket is already boarded”

  **Observed Output** Console: “Unable to cancel ticket T_R1_9: Ticket is already boarded”
  Console: “Cancelled ticket T_R1_9 on route R1 from Madrid to Toledo”

  **Failure** Yes

- **TC_CancelCommand_6**

  **Expected Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: “Cancelled ticket T_R1_9 on route R1 from Madrid to Toledo”

  **Observed Output** BBB{ _routes: [Route{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]}
  Console: “Cancelled ticket T_R1_9 on route R1 from Madrid to Toledo”

  **Failure** None

#### 5.2.3 Class BBB

**TC_BBB_1** successfully writes file

**Expected Output** file: [{ “id”: “R1”, “source”: “Madrid”, “destination”: “Toledo”, “capacity”: 10, “tickets”: [{id: “T_R1_9”, “seat”: 9, “boarded”: false}], “departed”: null, “availableSeats”: [0, . . . , 8]}, { “id”: “R2”, “source”: “Barcelona”, “destination”: “Valencia”, “capacity”: 10, “tickets”: [], “departed”: null, “availableSeats”: [0, . . . , 9]}]

**Observed Output** file: [{ “id”: “R1”, “source”: “Madrid”, “destination”: “Toledo”, “capacity”: 10, “tickets”: [{id: “T_R1_9”, “seat”: 9, “boarded”: false}], “departed”: null, “availableSeats”: [0, ... , 8]}, { “id”: “R2”, “source”: “Barcelona”, “destination”: “Valencia”, “capacity”: 10, “tickets”: [], “departed”: null, “availableSeats”: [0, ... , 9]}]

**Failure** None

**TC_BBB_2** successfully reads file with routes

**Expected Output** routes: [{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, { id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]

**Observed Output** routes: [{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [T_R1_9], departed: null, availableSeats: [0, ... , 8]}, { id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, ... , 9]}]

**Failure** None

**TC_BBB_3** successfully reads without routes

**Expected Output** routes: []

**Observed Output** routes: []

**Failure** None

**TC_BBB_4** does not read not existing file

**Expected Output** routes: []

**Observed Output** routes: []

**Failure** None

**TC_BBB_5** fails for no arguments given

**Expected Output** Console: “No argument was given”

**Observed Output** Console: “No argument was given”

**Failure** None

**TC_BBB_6** fails for not existing command

**Expected Output** Console: “Command asdf does not exist”

**Observed Output** Console: “Command asdf does not exist”

**Failure** None

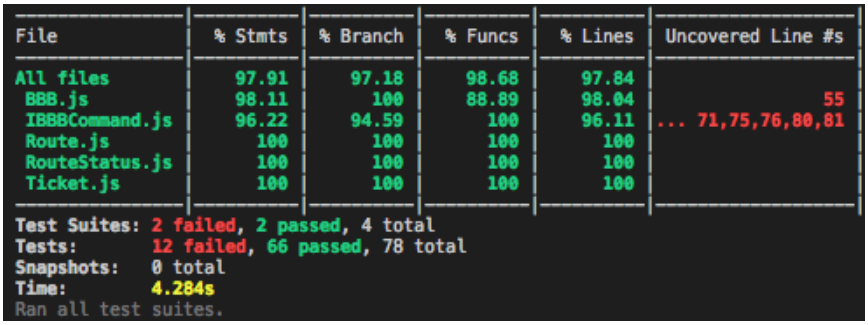
**TC_BBB_7** succeeds for existing command

**Expected Output** _commands[“status”].execute was called

**Observed Output** _commands[“status”].execute was called

**Failure** None

## 5.3 Check Coverage



| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
|---|---|---|---|---|---|
| All files | 97.91 | 97.18 | 98.68 | 97.84 | |
| BBB.js | 98.11 | 100 | 88.89 | 98.04 | 55 |
| IBBBCommand.js | 96.22 | 94.59 | 100 | 96.11 | ... 71,75,76,80,81 |
| Route.js | 100 | 100 | 100 | 100 | |
| RouteStatus.js | 100 | 100 | 100 | 100 | |
| Ticket.js | 100 | 100 | 100 | 100 | |

Test Suites: 2 failed, 2 passed, 4 total
Tests: 12 failed, 66 passed, 78 total
Snapshots: 0 total
Time: 4.284s
Ran all test suites.

Using the coverage tools we checked the lines/branches that are not covered.

```
var BBBCommandBase = /** @class */ (function () {
    function BBBCommandBase(bbb) {
        var _this = this;
        this.getRouteFromArgs = function (args) {
            if (args.length !== 1) {
                console.log('Invalid number of arguments given');
                return null;
            }
            var routeId = args[0].trim();
            if (!routeId || routeId.length === 0) {
                console.log('Invalid value for route given');
                return null;
            }
            var routeIndex = _this._bbb.routes.map(function (r) { return r.id; }).indexOf(routeId);
            if (routeIndex === -1) {
                console.log("Route " + routeId + " does not exist");
                return null;
            }
            return _this._bbb.routes[routeIndex];
        };
        this.getTicketIdFromArgs = function (args) {
            if (args.length !== 1) {
                console.log('Invalid number of arguments given');
                return null;
            }
            var ticketId = args[0].trim();
            if (!ticketId || ticketId.length === 0) {
                console.log('Invalid value for ticket given');
                return null;
            }
            return ticketId;
        };
        this.getRouteFromTicketId = function (ticketId) {
            var routes = _this._bbb.routes.filter(function (route) { return route.tickets.map(function (t
            if (routes.length === 0) {
                console.log("Ticket with id " + ticketId + " does not exist");
                return null;
            }
            return routes[0];
        };
        if (bbb === null) {
            throw new Error('Invalid bbb');
        }
        this._bbb = bbb;
    }
    return BBBCommandBase;
}());
```

For the constructor a test case is missing that tests if the initialization fails if an invalid `BBB` instance is passed.

```
exports.BBBCommandBase = BBBCommandBase;
var RegisterRouteCommand = /** @class */ (function (_super) {
    tslib_2.__extends(RegisterRouteCommand, _super);
    function RegisterRouteCommand(bbb) {
        var _this = _super.call(this, bbb) || this;
        _this.execute = function (args) {
            if (args.length !== 4) {
                console.log('Invalid number of arguments given');
                return;
            }
            var routeId = args[0].trim();
            if (!routeId || routeId.length === 0) {
                console.log('Invalid value for route given');
                return;
            }
            var source = args[1].trim();
            I if (!source || source.length === 0) {
                console.log('Invalid value for source given');
                return;
            }
            var destination = args[2].trim();
            I if (!destination || destination.length === 0) {
                console.log('Invalid value for destination given');
                return;
            }
            var capacity = Number(args[3]);
            I if (capacity === NaN || capacity < 1) {
                console.log('Invalid value for capacity given');
                return;
            }
            var route = new Route_1.Route(routeId, source, destination, capacity);
            _this._bbb.routes.push(route);
            console.log("Created route " + routeId + " from " + source + " to " + d
        };
        return _this;
    }
```

For the `RegisterRouteCommand` the not covered lines are a result from the failing test cases. The first two if's are not covered because an exception is thrown in TC_RegisterRouteCommand_4 and TC_RegisterRouteCommand_5 on calling trim() on a null value. Therefore the test fails before going to the if. The third if is never true because of the failure detected in TC_RegisterRouteCommand_6. A comparison using === and "NaN" is always false and therefore the if branch is never taken. Thus it is not necessary to introduce additional test cases after fixing those three test cases.

```
    Object.defineProperty(BBB.prototype, "routes", {
        get: function () {
            return this._routes;
        },
        set: function (newRoutes) {
            this._routes = newRoutes;
        },
        enumerable: true,
        configurable: true
    });
    return BBB;
}());
```

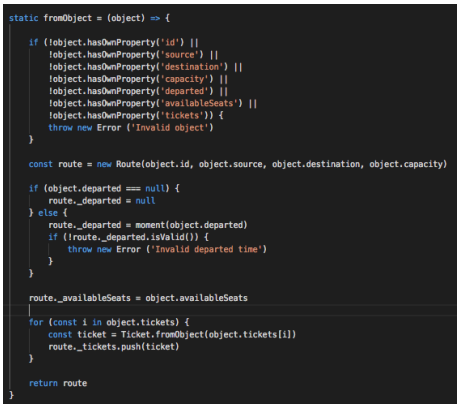
In the class `BBB` the setter for setting the routes is never called. Therefore, an additional test for the setter has to be created.

## 5.4 Trace failures to faults

### 5.4.1 TC_Route_27

**Failure** Instead of throwing an error because of the invalid value for departed a new `Route` instance is created

**Fault** Departed is not parsed enforcing ISO_8601 date format



```
static fromObject = (object) => {

    if (!object.hasOwnProperty('id') ||
        !object.hasOwnProperty('source') ||
        !object.hasOwnProperty('destination') ||
        !object.hasOwnProperty('capacity') ||
        !object.hasOwnProperty('departed') ||
        !object.hasOwnProperty('availableSeats') ||
        !object.hasOwnProperty('tickets')) {
        throw new Error ('Invalid object')
    }

    const route = new Route(object.id, object.source, object.destination, object.capacity)

    if (object.departed === null) {
        route._departed = null
    } else {
        route._departed = moment(object.departed)
        if (!route._departed.isValid()) {
            throw new Error ('Invalid departed time')
        }
    }

    route._availableSeats = object.availableSeats

    for (const i in object.tickets) {
        const ticket = Ticket.fromObject(object.tickets[i])
        route._tickets.push(ticket)
    }

    return route
}
```

**Fix** Ensure that the parsing is done enfocring ISO_8601 date format by specifying the format in the constructor

```
static fromObject = (object) => {

    if (!object.hasOwnProperty('id') ||
        !object.hasOwnProperty('source') ||
        !object.hasOwnProperty('destination') ||
        !object.hasOwnProperty('capacity') ||
        !object.hasOwnProperty('departed') ||
        !object.hasOwnProperty('availableSeats') ||
        !object.hasOwnProperty('tickets')) {
        throw new Error ('Invalid object')
    }

    const route = new Route(object.id, object.source, object.destination, object.capacity)

    if (object.departed === null) {
        route._departed = null
    } else {
        if (!moment(object.departed, moment.ISO_8601, true).isValid()) {
            throw new Error ('Invalid departed time')
        }
        route._departed = moment(object.departed, moment.ISO_8601, true)
    }

    route._availableSeats = object.availableSeats

    for (const i in object.tickets) {
        const ticket = Ticket.fromObject(object.tickets[i])
        route._tickets.push(ticket)
    }

    return route
}
```

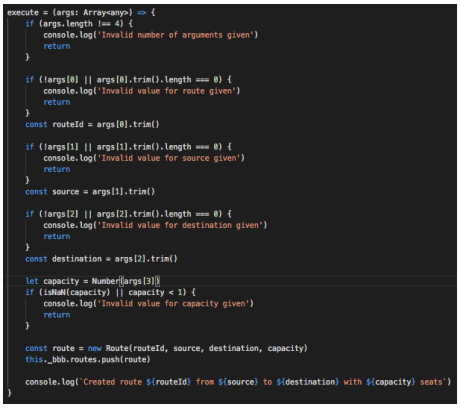
### 5.4.2 TC_RegisterRouteCommand_4

**Failure** Instead of showing a meaningful error message a `TypeError` is thrown

**Fault** The method `trim()` is called on the first argument `args[0]` which is `null`

```
execute = (args: Array<any>) => {
    if (args.length !== 4) {
        console.log('Invalid number of arguments given')
        return
    }

    const routeId = args[0].trim()
    if (!routeId || routeId.length === 0) {
        console.log('Invalid value for route given')
        return
    }

    const source = args[1].trim()
    if (!source || source.length === 0) {
        console.log('Invalid value for source given')
        return
    }

    const destination = args[2].trim()
    if (!destination || destination.length === 0) {
        console.log('Invalid value for destination given')
        return
    }

    let capacity = Number(args[3])
    if (capacity === NaN || capacity < 1) {
        console.log('Invalid value for capacity given')
        return
    }

    const route = new Route(routeId, source, destination, capacity)
    this._bbb.routes.push(route)

    console.log(`Created route ${routeId} from ${source} to ${destination} with ${capacity} seats`)
}
```

**Fix** Ensure that `args[1]` is not null before using the `trim()` method




### 5.4.3 TC_RegisterRouteCommand_5

The same failure and fault as in TC_RegisterRouteCommand_4 but with second argument `args[2]`. Is fixed the same way as TC_RegisterRouteCommand_4 and already shown in the previous screenshotss.

### 5.4.4 TC_RegisterRouteCommand_6

**Failure** Instead of showing a meaninigful error message a `RangeError` is thrown in the constructor of the `Route`

**Fault** The check if an invalid capacity has been given is done using the condition `capacity === NaN` but performing a `===` check on `NaN` always yields false

**Fix** Use the method `isNaN()` for checking for an invalid capacity (the fault and fix are also shown in the screenshots from TC_RegisterRouteCommand_4)

### 5.4.5 TC_DepartCommand_3

**Failure** Message "Invalid value for route given" is shown instead of the message "Route R_X does not exist"

**Fault** Actually, the observed output is correct and it is the test case that was specified wrongly

**Fix** Update the test case so that the expected output is a console message "Route R_X does not exist" and the title states "fails for not existing route"

### 5.4.6 TC_BuyCommand_3

**Failure** Instead of showing an error message saying the tickets are sold out a `TypeError` is thrown because it is tried to access the property `id` of undefined

**Fault** After checking if the purchase of a `Ticket` was unsuccessful a return statement is missing

```
execute = (args: Array<any>) => {

    const route = this.getRouteFromArgs(args)
    if (route === null) {
        return
    }

    const result = route.purchaseTicket()

    if (!result.success) {
        console.log('Sorry! You were too late! Tickets are sold out!')
    }

    const ticket = result.ticket

    console.log(`Successfully purchased ticket ${ticket.id} on route ${route.id} from ${route.source} to ${route.destination}`)
    return
}
```

**Fix** Add the return statement in the case of an unsuccessful purchase attempt

```
execute = (args: Array<any>) => {

    const route = this.getRouteFromArgs(args)
    if (route === null) {
        return
    }

    const result = route.purchaseTicket()

    if (!result.success) {
        console.log('Sorry! You were too late! Tickets are sold out!')
        return
    }

    const ticket = result.ticket

    console.log(`Successfully purchased ticket ${ticket.id} on route ${route.id} from ${route.source} to ${route.destination}`)
    return
}
```

### 5.4.7 TC_CheckinCommand_2

**Failure** In addition to the "Invalid number of arguments given" error message "Ticket with id null does exist" is shown

**Fault** After parsing the `ticketId` from the arguments it is not checked whether the "ticketId" is null in order to return

```
execute = (args: Array<any>) => {

    const ticketId = this.getTicketIdFromArgs(args)
    const route = this.getRouteFromTicketId(ticketId)
    if (route == null) {
        return
    }

    const result = route.boardTicket(ticketId)

    if (!result.success) {
        console.log(`Unable to checkin ticket ${ticketId}: ${result.reason}`)
    }

    const ticket = result.ticket

    console.log(`Successfully checked in ticket ${ticketId} on route ${route.id} from ${route.source} to ${route.destination} and
    assigned seat ${ticket.seat}`)
    return
}
```

**Fix** Added the missing `null` check and return statement before proceeding with finding the `Route` with the `ticketId`

### 5.4.8 TC_CheckinCommand_3

The same failure and fault as in TC_CheckinCommand_2. Is fixed the same way.

### 5.4.9 TC_CheckinCommand_5

**Failure** Instead of showing the expected error message saying that the ticket is already boarded a `TypeError` is thrown

**Fault** After checking if the checkin of the `Ticket` was unsuccessful a return statement is missing

```
execute = (args: Array<any>) => {

    const ticketId = this.getTicketIdFromArgs(args)
    const route = this.getRouteFromTicketId(ticketId)
    if (route == null) {
        return
    }

    const result = route.boardTicket(ticketId)

    if (!result.success) {
        console.log(`Unable to checkin ticket ${ticketId}: ${result.reason}`)
    }

    const ticket = result.ticket

    console.log(`Successfully checked in ticket ${ticketId} on route ${route.id} from ${route.source} to ${route.destination} and
    assigned seat ${ticket.seat}`)
    return
}
```

**Fix** Added the return statement in the cases of an unsuccessful checkin attempt

```
execute = (args: Array<any>) => {

    const ticketId = this.getTicketIdFromArgs(args)
    if (ticketId === null) {
        return
    }

    const route = this.getRouteFromTicketId(ticketId)
    if (route === null) {
        return
    }

    const result = route.boardTicket(ticketId)

    if (!result.success) {
        console.log(`Unable to checkin ticket ${ticketId}: ${result.reason}`)
        return
    }

    const ticket = result.ticket

    console.log(`Successfully checked in ticket ${ticketId} on route ${route.id} from ${route.source} to ${route.destination} and
    assigned seat ${ticket.seat}`)
    return
}
```

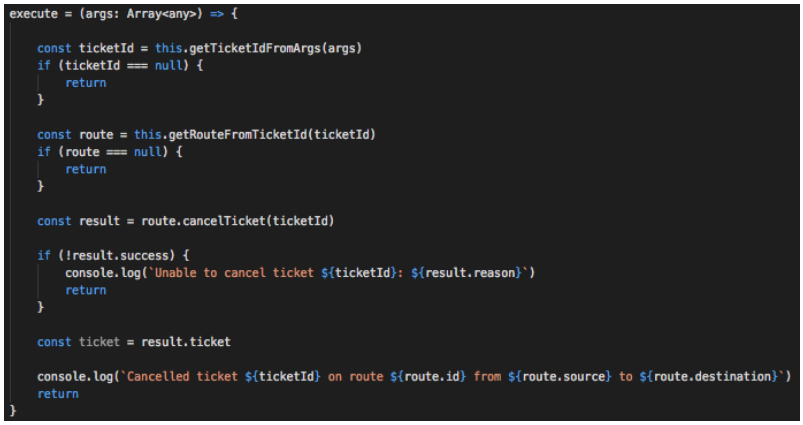
### 5.4.10 TC_CancelCommand_2

**Failure** In addition to the expected "Invalid number of arguments given" error message the message "Ticket with id null does not exist" is shown

**Fault** After parsing the `ticketId` from the arguments it is not checked whether the "ticketId" is null in order to return

```
execute = (args: Array<any>) => {

    const ticketId = this.getTicketIdFromArgs(args)
    const route = this.getRouteFromTicketId(ticketId)
    if (route == null) {
        return
    }

    const result = route.cancelTicket(ticketId)

    if (!result.success) {
        console.log(`Unable to cancel ticket ${ticketId}: ${result.reason}`)
    }

    const ticket = result.ticket

    console.log(`Cancelled ticket ${ticketId} on route ${route.id} from ${route.source} to ${route.destination}`)
    return
}
```

**Fix** Added the missing `null` check and return statement before proceeding with finding the `Route` with the `ticketId`



```
execute = (args: Array<any>) => {

    const ticketId = this.getTicketIdFromArgs(args)
    if (ticketId === null) {
        return
    }

    const route = this.getRouteFromTicketId(ticketId)
    if (route === null) {
        return
    }

    const result = route.cancelTicket(ticketId)

    if (!result.success) {
        console.log(`Unable to cancel ticket ${ticketId}: ${result.reason}`)
        return
    }

    const ticket = result.ticket

    console.log(`Cancelled ticket ${ticketId} on route ${route.id} from ${route.source} to ${route.destination}`)
    return
}
```

#### 5.4.11 TC_CancelCommand_3

The same failure and fault as in TC_CancelCommand_2. Is fixed the same way and already shown in the previous screenshots from TC_CancelCommand_2.

#### 5.4.12 TC_CancelCommand_5

**Failure** After showing the expected unable to cancel ticket message the message for successfully canceled the ticket is shown

**Fault** After checking if the cancelation of the `Ticket` was unsuccessful a return statement is missing (the fault is already shown in the screenshot from TC_CancelCommand_2)

**Fix** Added the return statement in the cases of an unsuccessful cancel attempt (the fix is already shown in the screenshot from TC_CancelCommand_2)

## 6 Iteration 3

Iteration 3 does not target a new component and only adds test cases that are missing to get full coverage.

### 6.1 Specify Test Cases

#### 6.1.1 Class IBBBCommand

**TC_CancelCommand_7** fails when initialized with invalid bbb

**Goal** Test that the constructor of an `IBBBCommand` fails when initialized with an invalid `BBB`

**Class** CancelCommand

**Method** constructor

**Precondition** N/A

**Input** null

**Expected Output** Error("Invalid bbb")

### 6.1.2 BBB Class

**TC_BBB_8** set routes

**Goals** Test that the property `routes` can be set correctly

**Class** BBB

**Method** set route

**Precondition** routes: [{ id: “R1”, source: “Madrid”, destination: “Toledo”, capacity: 10, tickets: [{id: “T_R1_9”, “seat”: 9, “boarded”: false}], departed: null, availableSeats: [0, . . . , 8]}, { id: “R2”, source: “Barcelona”, destination: “Valencia”, capacity: 10, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]

**Input** routes: [{ id: “R1”, source: “Berlin”, destination: “Toledo”, capacity: 11, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]

**Expected Output** routes: [{ id: “R1”, source: “Berlin”, destination: “Toledo”, capacity: 11, tickets: [], departed: null, availableSeats: [0, . . . , 9]}]

## 6.2 Run Test Cases

### 6.2.1 Class IBBBCommand

**TC_CancelCommand_7** fails when initialized with invalid bbb

**Expected Output** Error("Invalid bbb")

**Observed Output** Error("Invalid bbb")

**Failure** None

### 6.2.2 Class BBB

**TC_BBB_8** set routes

**Expected Output** routes: [{ id: "R1", source: "Berlin", destination: "Toledo", capacity: 11, tickets: [], departed: null, availableSeats: [0, ... , 9]}]

**Observed Output** routes: [{ id: "R1", source: "Berlin", destination: "Toledo", capacity: 11, tickets: [], departed: null, availableSeats: [0, ... , 9]}]

**Failure** None

## 6.3 Check Coverage

```
----------------|----------|----------|----------|----------|-------------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------------|----------|----------|----------|----------|-------------------|
All files       |      100 |      100 |      100 |      100 |                   |
 BBB.js         |      100 |      100 |      100 |      100 |                   |
 IBBBCommand.js |      100 |      100 |      100 |      100 |                   |
 Route.js       |      100 |      100 |      100 |      100 |                   |
 RouteStatus.js |      100 |      100 |      100 |      100 |                   |
 Ticket.js      |      100 |      100 |      100 |      100 |                   |
----------------|----------|----------|----------|----------|-------------------|

Test Suites: 4 passed, 4 total
Tests:       80 passed, 80 total
Snapshots:   0 total
Time:        1.89s
Ran all test suites.
```

We reached 100% branch coverage for all files without any tests failing. Therefore the white box assignment is completed. Since we used a unit test approach, we tested each component of the application individually and isolated. Even though we reached the full branch coverage which was the aim of this assignment, we would usually also implement integration tests to ensure that the application works correctly when all components are used jointly.

# Attachment – Source Code

## Ticket

```
export class Ticket {

    private _id: string
    private _seat: number
    private _boarded: boolean

    constructor(id: string, seat: number) {

        if (!id || id.trim().length === 0) {
            throw new Error('Invalid id')
        }

        if (seat < 0) {
            throw new Error ('Invalid seat')
        }

        this._id = id
        this._seat = seat
        this._boarded = false
    }

    get id(): string {
        return this._id
    }

    get seat(): number {
        return this._seat
    }

    get boarded(): boolean {
        return this._boarded
    }

    set boarded(boarded: boolean) {
        this._boarded = boarded
    }

    static fromObject = (object) => {
        if (!object.hasOwnProperty('id') ||
            !object.hasOwnProperty('seat') ||
            !object.hasOwnProperty('boarded')) {
            throw new Error ('Invalid object')
        }

        let ticket = new Ticket(object.id, object.seat)
        ticket.boarded = object.boarded

        return ticket
    }

    toObject = () => {
        return {
            id: this._id,
            seat: this._seat,
            boarded: this._boarded
        }
    }

}
```

## Route

```
import {Ticket} from './Ticket'
import {RouteStatus} from './RouteStatus'
import * as moment from 'moment'

export class Route {

    private _id: string
    private _source: string
    private _destination: string
    private _capacity: number
    private _departed: moment.Moment
    private _availableSeats: Array<number>
    private _tickets: Array<Ticket>

    constructor(id: string, source: string, destination: string, capacity: number) {
        if (!id || id.trim().length === 0) {
            throw new Error('Invalid id')
        }

        if (!source || source.trim().length === 0) {
```

```
            throw new Error('Invalid source')
        }

        if (!destination || destination.trim().length === 0) {
            throw new Error('Invalid destination')
        }

        if (capacity < 1) {
            throw new Error('Invalid capacity')
        }

        this._id = id
        this._source = source
        this._destination = destination
        this._capacity = capacity
        this._tickets = new Array()
        this._departed = null

        this.initializeSeats()
    }

    get id(): string {
        return this._id
    }

    get source(): string {
        return this._source
    }

    get destination(): string {
        return this._destination
    }

    get capacity(): number {
        return this._capacity
    }

    get tickets(): Array<Ticket> {
        return this._tickets
    }

    get status(): RouteStatus {
        if (this._departed !== null) {
            return RouteStatus.travelling
        } else {
            if (this._availableSeats.length === this.capacity) {
                return RouteStatus.empty
            } else if (this._availableSeats.length === 0) {
                return RouteStatus.full
            } else {
                return RouteStatus.available
            }
        }
    }

    private initializeSeats = () => {
        this._availableSeats = new Array(this._capacity)
        for (let i = 0; i < this._capacity; i++) {
            this._availableSeats[i] = i
        }
    }

    purchaseTicket = () => {

        if (this._availableSeats.length === 0) {
            return {
                success: false,
                reason: 'No tickets available'
            }
        }

        const nextSeat = this._availableSeats.pop()
        const ticket = new Ticket(`T_${this._id}_${nextSeat}`, nextSeat)
        this._tickets.push(ticket)

        return {
            success: true,
            ticket: ticket
        }
    }

    boardTicket = (ticketId: string) => {
        const ticketIndex = this._tickets.map((t) => t.id).indexOf(ticketId)

        if (ticketIndex === -1) {
            return {
                success: false,
                reason: 'Ticket does not exist'
            }
        }

        const ticket = this._tickets[ticketIndex]

        if (ticket.boarded === true) {
```

```
            return {
                success: false,
                reason: 'Ticket is already boarded'
            }
        }

        ticket.boarded = true

        return {
            success: true,
            ticket: ticket
        }
    }

    cancelTicket = (ticketId: string) => {
        const ticketIndex = this._tickets.map((t) => t.id).indexOf(ticketId)

        if (ticketIndex === -1) {
            return {
                success: false,
                reason: 'Ticket does not exist'
            }
        }

        const ticket = this._tickets[ticketIndex]

        if (ticket.boarded === true) {
            return {
                success: false,
                reason: 'Ticket is already boarded'
            }
        }

        this._tickets = this._tickets.filter((t) => t.id !== ticketId)

        const seat = ticket.seat
        this._availableSeats.push(seat)

        return {
            success: true,
            ticket: ticket
        }
    }

    depart = () => {
        this._departed = moment()
    }

    hasArrived = () => {
        if (this._departed === null) {
            return false
        }

        const now = moment()
        if (now.isBefore(this._departed.add(10, 'seconds'))) {
            return false
        }

        const source = this._source
        this._source = this._destination
        this._destination = source
        this._tickets = new Array()
        this._departed = null
        this.initializeSeats()

        return true
    }

    static fromObject = (object) => {

        if (!object.hasOwnProperty('id') ||
            !object.hasOwnProperty('source') ||
            !object.hasOwnProperty('destination') ||
            !object.hasOwnProperty('capacity') ||
            !object.hasOwnProperty('departed') ||
            !object.hasOwnProperty('availableSeats') ||
            !object.hasOwnProperty('tickets')) {
            throw new Error ('Invalid object')
        }

        const route = new Route(object.id, object.source, object.destination, object.capacity)

        if (object.departed === null) {
            route._departed = null
        } else {
            if (!moment(object.departed, moment.ISO_8601, true).isValid()) {
                throw new Error ('Invalid departed time')
            }
            route._departed = moment(object.departed, moment.ISO_8601, true)
        }

        route._availableSeats = object.availableSeats

        for (const i in object.tickets) {
```

```
            const ticket = Ticket.fromObject(object.tickets[i])
            route._tickets.push(ticket)
        }

        return route
    }

    toObject = () => {

        let departedString = null
        if (this._departed !== null) {
            departedString = this._departed.toISOString()
        }

        const ticketObjects = this._tickets.map((t) => t.toObject())

        return {
            id: this._id,
            source: this._source,
            destination: this._destination,
            capacity: this._capacity,
            availableSeats: this._availableSeats,
            tickets: ticketObjects,
            departed: departedString
        }

    }
}
```

## RouteStatus

```
export enum RouteStatus {
    travelling = 'travelling',
    empty = 'empty',
    full = 'full',
    available = 'available'
}
```

## IBBBCommand

```
import { BBB } from "./BBB";
import { Route } from "./Route";
import { RouteStatus } from "./RouteStatus"
import { basename } from "path";
import { stringify } from "querystring";
var tslib_1 = require("tslib");

export interface IBBBCommand {
    commandId: string,
    execute: (args: Array<any>) => any
}

export abstract class BBBCommandBase {

    protected _bbb: BBB

    constructor(bbb: BBB) {

        if (bbb === null) {
            throw new Error('Invalid bbb')
        }

        this._bbb = bbb
    }

    protected getRouteFromArgs = (args: Array<any>) => {
        if (args.length !== 1) {
            console.log('Invalid number of arguments given')
            return null
        }

        if (!args[0] || args[0].trim().length === 0) {
            console.log('Invalid value for route given')
            return null
        }
        const routeId = args[0].trim()

        const routeIndex = this._bbb.routes.map(r => r.id).indexOf(routeId)
        if (routeIndex === -1) {
            console.log(`Route ${routeId} does not exist`)
            return null
        }

        return this._bbb.routes[routeIndex]
    }

    protected getTicketIdFromArgs = (args: Array<any>) => {
```

```
        if (args.length !== 1) {
            console.log('Invalid number of arguments given')
            return null
        }

        if (!args[0] || args[0].trim().length === 0) {
            console.log('Invalid value for ticket given')
            return null
        }
        const ticketId = args[0].trim()

        return ticketId
    }

    protected getRouteFromTicketId = (ticketId: string) => {

        const routes = this._bbb.routes.filter(route => route.tickets.map(ticket => ticket.id).indexOf(ticketId)
!== -1)
        if (routes.length === 0) {
            console.log(`Ticket with id ${ticketId} does not exist`)
            return null
        }

        return routes[0]
    }
}

export class RegisterRouteCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    get commandId(): string {
        return 'registerroute'
    }

    execute = (args: Array<any>) => {
        if (args.length !== 4) {
            console.log('Invalid number of arguments given')
            return
        }

        if (!args[0] || args[0].trim().length === 0) {
            console.log('Invalid value for route given')
            return
        }
        const routeId = args[0].trim()

        if (!args[1] || args[1].trim().length === 0) {
            console.log('Invalid value for source given')
            return
        }
        const source = args[1].trim()

        if (!args[2] || args[2].trim().length === 0) {
            console.log('Invalid value for destination given')
            return
        }
        const destination = args[2].trim()

        let capacity = Number(args[3])
        if (isNaN(capacity) || capacity < 1) {
            console.log('Invalid value for capacity given')
            return
        }

        const route = new Route(routeId, source, destination, capacity)
        this._bbb.routes.push(route)

        console.log(`Created route ${routeId} from ${source} to ${destination} with ${capacity} seats`)
    }
}

export class DeleteRouteCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'deleteroute'
    }

    execute = (args: Array<any>) => {

        const route = this.getRouteFromArgs(args)
        if (route === null) {
            return
        }

        if (route.tickets.length > 0) {
            console.log(`Cannot delete route ${route.id} because there are ${route.tickets.length} tickets booked`)
```

```
            return
        }

        this._bbb.routes = this._bbb.routes.filter(r => r.id !== route.id)

        console.log(`Successfully deleted route ${route.id}`)
        return
    }
}

export class DepartCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'depart'
    }

    execute = (args: Array<any>) => {

        const route = this.getRouteFromArgs(args)
        if (route === null) {
            return
        }

        route.depart()

        console.log(`${route.id} departed`)
        return
    }
}

export class StatusComamnd extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'status'
    }

    execute = (args: Array<any>) => {

        let routesToDisplay: Array<Route> = new Array()

        if (args.length === 0) {
            routesToDisplay = this._bbb.routes
        }
        else if (args.length === 1)
        {
            const route = this.getRouteFromArgs(args)
            if (route == null) {
                return
            }

            routesToDisplay.push(route)
        }
        else
        {
            console.log('Invalid number of arguments given')
            return
        }

        routesToDisplay.forEach(route => console.log(`${route.id}: ${route.status}`))

        return
    }
}

export class BuyCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'buy'
    }

    execute = (args: Array<any>) => {

        const route = this.getRouteFromArgs(args)
        if (route === null) {
            return
        }

        const result = route.purchaseTicket()

        if (!result.success) {
            console.log('Sorry! You were too late! Tickets are sold out!')
            return
```

```
        }

        const ticket = result.ticket

        console.log(`Successfully purchased ticket ${ticket.id} on route ${route.id} from ${route.source} to
${route.destination}`)
        return
    }
}

export class CheckinCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'checkin'
    }

    execute = (args: Array<any>) => {

        const ticketId = this.getTicketIdFromArgs(args)
        if (ticketId === null) {
            return
        }

        const route = this.getRouteFromTicketId(ticketId)
        if (route === null) {
            return
        }

        const result = route.boardTicket(ticketId)

        if (!result.success) {
            console.log(`Unable to checkin ticket ${ticketId}: ${result.reason}`)
            return
        }

        const ticket = result.ticket

        console.log(`Successfully checked in ticket ${ticketId} on route ${route.id} from ${route.source} to
${route.destination} and assigned seat ${ticket.seat}`)
        return
    }
}

export class CancelCommand extends BBBCommandBase implements IBBBCommand {

    constructor(bbb: BBB) {
        super(bbb)
    }

    public get commandId(): string {
        return 'cancel'
    }

    execute = (args: Array<any>) => {

        const ticketId = this.getTicketIdFromArgs(args)
        if (ticketId === null) {
            return
        }

        const route = this.getRouteFromTicketId(ticketId)
        if (route === null) {
            return
        }

        const result = route.cancelTicket(ticketId)

        if (!result.success) {
            console.log(`Unable to cancel ticket ${ticketId}: ${result.reason}`)
            return
        }

        const ticket = result.ticket

        console.log(`Cancelled ticket ${ticketId} on route ${route.id} from ${route.source} to
${route.destination}`)
        return
    }
}
```

## BBB

```
import {Route} from './Route'
import {IBBBCommand, RegisterRouteCommand, DeleteRouteCommand, DepartCommand, StatusComamnd, BuyCommand,
CheckinCommand, CancelCommand} from './IBBBCommand'
```

```
import * as fs from 'fs'

export class BBB {

    _routes: Array<Route>
    _commands: Array<IBBBCommand>
    _filePath: string

    constructor (filePath: string) {

        this._filePath = filePath

        this._commands = new Array()
        this._commands.push(new RegisterRouteCommand(this))
        this._commands.push(new DeleteRouteCommand(this))
        this._commands.push(new DepartCommand(this))
        this._commands.push(new StatusComamnd(this))
        this._commands.push(new BuyCommand(this))
        this._commands.push(new CheckinCommand(this))
        this._commands.push(new CancelCommand(this))
    }

    get routes(): Array<Route> {
        return this._routes
    }

    set routes(newRoutes: Array<Route>) {
        this._routes = newRoutes
    }

    public saveRoutes = () => {
        const routeObjects = this._routes.map((r) => r.toObject())
        const json = JSON.stringify(routeObjects)

        fs.writeFileSync(this._filePath, json)
    }

    public loadRoutes = () => {
        this._routes = new Array()

        if (fs.existsSync(this._filePath)) {
            const input = fs.readFileSync(this._filePath)
            const routeObjects: Array<any> = JSON.parse(input.toString())

            for (const index in routeObjects) {
                const route = Route.fromObject(routeObjects[index])
                route.hasArrived()

                this._routes.push(route)
            }
        }
    }

    public parseCommand = (args: Array<any>) => {

        if (args.length === 0) {
            console.log('No argument was given')
            return
        }

        const commandId = args.shift()
        const commandIndex = this._commands.map((c) => c.commandId).indexOf(commandId)

        if (commandIndex === -1) {
            console.log(`Command ${commandId} does not exist`)
            return
        }

        const command = this._commands[commandIndex]
        command.execute(args)
    }

}

let args = process.argv
args.shift()
args.shift()

const bbb = new BBB('./.bbb_data')
bbb.loadRoutes()
bbb.parseCommand(args)
bbb.saveRoutes()
```