Attachment – Source Code

Ticket.ts

```typescript
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.ts

```
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.ts

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

IBBBCommand.ts

```
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()
```

```typescript
        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.ts

```
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()
```