Prague Parking 2

**Version 2.0**

Starta upp projektet

Jag börjar med att starta ett nytt projekt i Visual Studio med solution namn Prague Parking 2. Sedan lägger jag till 4 projekt:

* PragueParking 2.0 (Console.App) som kommer vara det synliga programmet som användaren kör.
* PragueParking.Domain (Class Library) som kommer vara ett klassbibliotek över olika klasser, metoder och modeller.
* PragueParking.Data (Class Library) som hanterar dataåtkomst – allt som rör filer.
* PragueParking.Tests (MSTest) För att kunna göra enhetstester.

Jag ser till att göra så projekten refererar till varandra på rätt sätt. PragueParking 2.0 refererar till Domain och Data. Domain refererar inte till någon av de andra projekten (för att undvika cirkelberoenden). Data refererar till Domain. Tests refererar till Domain.

Domain: modeller och grundlogik

I projektet Domain skapar jag klasser som representerar verkliga saker i garaget:

* **Vehicle** – en basklass som innehåller registreringsnummer och tidpunkten då fordonet checkar in.
* **Car** och **Motorcycle** – två klasser som ärver från Vehicle och skiljer på bil och motorcykel.
* **ParkingSpot** – en parkeringsruta som kan innehålla antingen en bil eller upp till två motorcyklar.
* **ParkingGarage** – själva parkeringshuset som består av flera parkeringsrutor och kan parkera, ta bort och söka fordon.

Jag skriver också in regler i koden som bestämmer hur fordon får stå. Till exempel att en bil alltid står ensam på en ruta, men att två motorcyklar kan dela på samma plats. Nu kan jag enkelt lägga till fler fordonstyper senare (t.ex. buss eller cykel) utan att behöva skriva om hela koden.

Data: filhantering

I projektet Data skapar jag klasserna GarageDto, SpotDto och VehicleDto.

Jag skapar också en GarageMapper-klass. Mappern har två metoder:

* ToDto() – konverterar programmets objekt till DTO-format för att kunna sparas som JSON.
* FromDto() – läser in JSON-data och skapar tillbaka rätt objekt i minnet.

Jag skapar sedan en klass som jag kallar FileManager. Den ska sköta allt som har med filer att göra, till exempel:

* läsa in sparade parkeringsplatser och fordon från en JSON-fil när programmet startar,
* spara nya förändringar (t.ex. när man parkerar eller tar bort ett fordon),
* kunna läsa in konfigurationsdata som antal platser och priser.

Jag kommer att använda System.Text.Json för att konvertera (”serialisera”) objekt till textfiler och tvärtom.

Jag planerar också att ha en separat JSON-fil för konfigurationen, där man kan ställa in antalet platser, pris per fordonstyp och regler.  
Den ska läsas in automatiskt när programmet startar.

På det här sättet blir systemet mycket mer flexibelt.  
Jag kan ändra priser eller antal platser direkt i filen utan att behöva ändra i koden.

App: huvudprogram & menyn

Jag börjar med att läsa in sparade data från filen garage.json genom att anropa FileManager.LoadGarage().

Om filen inte finns skapas ett nytt parkeringshus med 100 platser som standard.

Sedan har jag byggt en enkel meny med fyra huvudval:

1. Park vehicle
2. Retrieve vehicle
3. Search vehicle
4. Show spots

Användaren kan parkera en bil eller motorcykel genom att skriva in fordonstyp och registreringsnummer.

När ett fordon parkeras eller tas bort sparas uppdaterade data direkt till garage.json via FileManager.SaveGarage().

Vid uttag räknas även hur länge fordonet stått parkerat (skillnaden mellan nuvarande tid och incheckningstiden som lagras i objektet).

Om man söker efter ett registreringsnummer används garage.Find() för att se vilken plats fordonet står på.

Menyalternativet Show spots visar alla parkeringsrutor med status – tomma eller vilka fordon som står där.

Konfiguration (appsettings.json)

Jag lägger till en konfigurationsfil appsettings.json i App-projektet med standardvärden:

* Capacity: 100
* PricePerHourCar: 20
* PricePerHourMc: 10
* FreeMinutes: 10

I Data-projektet skapar jag AppConfig och ConfigManager som kan läsa/spara den här filen med System.Text.Json. I Program.cs laddar jag konfigurationen i början. Kapaciteten i garaget sätts sedan från konfigen när datafilen läses in. Detta gör jag för att kunna ändra regler (t.ex. priser, friminuter, antal platser) utan att göra om koden.

Prisberäkning vid uthämtning

I menyvalet Retrieve vehicle räknar jag ut parkeringstid och pris:

* Hämtar fordonets CheckInUtc från modellen.
* Räknar ut totala minuter parkerat.
* Jämför mot FreeMinutes (gratisperiod).
* Debiterar påbörjade timmar med rätt timpris (CAR/MC) från appsettings.json.

App: meny och flöde (uppdatering)

I Program.cs:

* Läser först appsettings.json och garage.json.
* Menyvalen hanterar parkering, uthämtning, sökning och utskrift av alla platser.
* Efter varje ändring sparas garaget tillbaka till garage.json.

Nytt menyval: flytta fordon

Jag inser att jag inte har något menyval för att flytta fordon. Jag går därför in i domain och lägger till en ny metod (Move vehicle) i klassen ParkingGarage. Sedan lägger jag till det som ett nytt alternativ i min meny och som ett nytt case i min switch-sats i program.cs.

Grafiskt användargränssnitt och konsolapplikation med Spectre.Console

Jag börjar med att installera Spectre.Console och lägger till using Spectre.Console; högst upp i program.cs. Sedan lägger jag till Show map som ett alternativ i menyn och ett nytt case i min switch-sats. Sedan lägger jag till en ny metod utanför (while-loopen) och i den lägger jag in färger +symboler för statusarna Empty, 1 MC, 2MC och Car.

Units-modell för olika storlekar på parkeringsplatser

För att systemet ska kunna hantera olika stora fordon och parkeringsplatser lägger jag till en units-modell. Jag börjar med att lägga till en ny rad i appsettings.json: "SpotMaxUnits": 2 (varje parkeringsruta har plats för 2 ”storleksenheter”. En bil tar 2 enheter, en MC tar 1).

I projektet Data uppdaterar jag klassen AppConfig med en ny egenskap SpotMaxUnits, så att värdet kan läsas in från konfigurationsfilen.

I projektet Domain lägger jag till ett nytt fält SizeUnits i basklassen Vehicle, som anger hur mycket plats ett fordon tar.

I ParkingSpot lägger jag till en ny egenskap MaxUnits (standardvärde 2) och ändrar metoden CanPark() så att den räknar ihop fordonens storleksenheter.  
Nu kan flera små fordon dela på en plats (t.ex. två MC), men en bil tar upp hela rutan själv.

Till sist uppdaterar jag Program.cs så att varje parkeringsruta får rätt värde för MaxUnits från konfigurationsfilen vid programstart.

Det här gör att systemet redan nu klarar olika storlekar på fordon och platser, och det går enkelt att lägga till t.ex. bussar eller cyklar senare.

Ändring i prisberäkning

För att göra programmet lite snyggare flyttar jag prislogiken från program.cs till domain:

jag skapar en ny klass PricingService i projektet Domain.  
Den innehåller en metod CalculateFee() som räknar ut priset för ett fordon när det lämnar parkeringen.

Metoden tar hänsyn till:  
• hur länge fordonet stått parkerat (utifrån incheckningstiden),  
• de första tio gratisminuterna,  
• fordonstypen (bil eller motorcykel) och respektive timpris från appsettings.json.

Tiden avrundas alltid uppåt till påbörjad timme.

Tidigare låg hela uträkningen direkt i menyvalet Retrieve vehicle, med flera rader som beräknade timmar, gratisminuter och olika priser för bil och MC.  
Jag tar bort den koden och ersätter den med ett enda anrop till:

var price = PricingService.CalculateFee(...);

Det gör programmet enklare att läsa och lättare att testa, eftersom all prislogik nu ligger på ett ställe.

Enhetstester prisberäkning:

Jag lägger till mitt första enhetstest i projektet PragueParking.Tests.  
Testet heter PricingServiceTests och kontrollerar att prisberäkningen med gratisminuter fungerar som den ska.

Testet CalculateFee\_LessOrEqual\_FreeMinutes\_ReturnsZero kontrollerar att parkeringen blir gratis om fordonet står 10 minuter eller mindre, vilket är gratisperioden enligt reglerna i appsettings.json.

Jag skapar en bil med ett incheckningstillfälle 10 minuter bakåt i tiden och kör metoden PricingService.CalculateFee(). Sedan kontrollerar jag att resultatet blir 0 CZK med Assert.AreEqual(0m, fee);.

Det här testet bekräftar att systemet hanterar gratisperioden korrekt och att metoden fungerar som förväntat.

Jag lägger även till ett andra test: CalculateFee\_Car\_65Minutes\_Returns40CZK.

Det testar att en bil som stått parkerad i 65 minuter (med 10 gratisminuter) får rätt avgift.

65 minuter räknas som två påbörjade timmar, och priset blir 40 CZK vid 20 CZK per timme.

När testerna körs visas Passed, vilket bekräftar att avrundningen och prisberäkningen fungerar som tänkt.

Nytt menyval: läsa om konfiguration och priser

Jag lade till ett nytt menyval i programmet för att kunna läsa om konfigurationsfilen utan att starta om programmet.  
Det här gör det möjligt att ändra priser, gratisminuter eller antal platser direkt i filen och uppdatera inställningarna under körning.

I menyn la jag till alternativ 7)Reload configuration (prices/capacity/units) och ett nytt case "7" i Program.cs.  
När användaren väljer det här alternativet anropas:

cfg = ConfigManager.Load(configPath);

Programmet läser då in konfigurationen på nytt och uppdaterar aktuella värden, till exempel:

* SpotMaxUnits (hur många fordon som får plats per ruta)
* PricePerHourCar och PricePerHourMc (timpris för bil och MC)
* FreeMinutes (gratisperiod)

Efter omladdningen visas de aktuella värdena i konsolen som en bekräftelse:

Configuration reloaded. Capacity: 100, SpotMaxUnits: 2, Prices: Car=20/h, MC=10/h, Free=10 min

Det här gör programmet mer användarvänligt och flexibelt. Användaren kan enkelt ändra priser eller regler direkt i textfilen utan att behöva stänga och starta om applikationen.