INLÄMNINGSUPPGIFT 2

Prague Parking V2

Prague Parking 2.0 – OOP-version

Bakgrund

Parkeringshuset i Prag behöver uppdateras och förbättras. Den är lite knölig att underhålla och stränghanteringen av data i arrayen har visat sig vara inflexibel när kunden säger att man i framtiden kommer att ta emot olika sorters fordon och kanske även olika stora parkeringsrutor.

Grundförutsättningarna är fortfarande desamma. Samma funktionalitet, endast parkering av MC och bilar i nuläget. Kunden har dock pratat om bussar och cyklar. Systemet skall registrera när fordonen kommer in och tala om hur länge de stått parkerade när de lämnas ut.

Kundens nya krav på systemet

- Data skall sparas så att de inte försvinner när programmet stängs av.
    - Filen läses in när programmet startar.
    - Filformatet bör vara JSON.
- Det skall gå att hantera P-platser av olika storlek så att även stora fordon (bussar) och små fordon (cyklar) kan hanteras.
- En karta över P-huset skall visas, så att man enkelt kan se beläggningen. Det skall gå att stå en bit ifrån skärmen och se översiktligt hur många platser som är lediga
- Det skall finnas en prislista i form av en textfil. Filen läses in vid programstart och kan vid behov läsas in på nytt via ett menyval. Filen skall gå att ändra även för en med låg IT-kunskap, så formatet behöver vara lätt att förstå. (**TIPS**: om allt efter ”#” på en rad är kommentarer, kan man ha dokumentation inne i själva filen)
- Det skall finnas en textfil med konfigurationsdata för systemet. Filformatet bör vara JSON.
- I konfigurationsfilen skall man kunna konfigurera
    - Antal P-platser i garaget
    - Fordonstyper (Bil och MC, men det kan komma fler)
    - Antal fordon per P-plats för respektive fordonstyp
- Prisstrukturen är till en början
    - Bil: 20 CZK per påbörjad timme
    - MC: 10 CZK per påbörjad timme
    - De första tio minuterna är gratis
- Filerna för prislista och konfiguration kan gärna kombineras till en fil
- Applikationen skall ha ett grafiskt användargränssnitt
    - Konsolapplikation ska använda **Spectre.Console**

Tekniska krav, nya i version 2.0

Alla fordon skall nu hanteras som objekt. Det skall finnas minst fem klasser:

1. Fordon

2. Bil, som ärver från Fordon
3. MC, som ärver från Fordon
4. ParkeringsPlats
5. ParkeringsHus

Om ni föredrar engelska namn på saker och ting:

1. Vehicle
2. Car, som ärver från Vehicle
3. MC, som ärver från Vehicle
4. ParkingSpot
5. ParkingGarage

Det kan mycket väl visa sig att det behövs fler klasser. Det kan också behövas en enumerator eller två.

P-huset kan med fördel innehålla en lista av P-platser (dvs `List<ParkingSpot>`). Fortfarande gäller att det finns 100 P-rutor som standardvärde, men det kan ändras med konfigurationsfilen.

Vidare skall lösningen innehålla ett separat projekt (dvs klassbibliotek) för dataåtkomst. Där bor alla klasser och metoder som har med dataåtkomst att göra, dvs Läsa fil, Spara fil och vad annat du kan behöva. Förslagsvis har du en klass, till exempel `MinaFiler`, med ett antal metoder, som `Read`, `Save` och så vidare.

Det kan bli nödvändigt att läsa på om hur man skapar och använder klassbibliotek. Det finns även en bra film av Tim Corey som beskriver klassbibliotek.

Slutligen ska det finnas ett project med **enhetstester**. Det räcker med två tester. Exakt vad som testas spelar ingen roll. Ni ska använda MSTest.

Github

All inlämning ska göras via Github.

Gör ditt repo **publikt**, så att läraren (och eventuellt klasskamraterna) kan läsa vad som finns. Privata repon gör det väldigt besvärligt för läraren att granska ert arbete.

Kom ihåg att du ska skicka in länken till ditt repo, inte till någon enskild fil. Om du har problem så fråga läraren eller klasskamraterna.

Inlämning

Inlämning sker indivuduellt. Dock är det inget som hindrar att ni arbetar tillsammans.

Inlämning görs i lärplattformen. Bifoga en länk till ditt repo i GitHub.

Dessutom ska du, liksom i tidigare projekt, föra en loggbok under projektet. Loggboken blir en del av inlämningen.

Betygskrav

För G

- För betyget G skall alla kraven ovan vara uppfyllda.
- Uppgiften löses med den kunskap vi hittills lärt oss, såsom klasser med arv, kollektioner, loopar och listor
- Data om de parkerade fordonen lagras i en JSON-fil
    - Datafilen fungerar samtidigt som testdata
    - Datafilen skall uppdateras varje gång det sker en förändring av fordonen i P-huset (incheckning, utcheckning, förflyttning)
- Vid uppstart skall programmet läsa in inställningar för programmet från konfigurationsfilen
- Vid uppstart skall programmet läsa in data om parkerade fordon från datafilen
- Inlämning skall ske via GitHub-länk
- Applikationen skall gå att köra på en dator annan än din egen (dvs på lärarens dator)
- Om det behövs några speciella handgrepp för att köra applikationen (utöver att trycks F5 eller Ctrl-F5 i Visual Studio) så skall dessa dokumenteras. Använd README.MD i GitHub för ändamålet.

Prague Parking 2.1 - För dig som vill ha VG på uppgiften.

När du är "klar", dvs har ett fungerande program med en meny, med Prague Parking 2.0:

Skapa ett nytt projekt i din Solution med lämpligt namn. Det skall givetvis framgå tydligt att det är version 2.1

Du kan, om du vill, kopiera innehållet från 2.0-projektet till 2.1-projektet och fortsätta att arbeta därifrån.

Uppgiften löses med den kunskap vi hittills lärt oss, såsom klasser med arv, kollektioner, loopar och listor.

Vidare skall det finnas minst två gränssnitt (interface) i systemet. Det kan till exempel vara `IVehicle` och `IParkingSpot`. Kodning skall primärt göras mot dessa gränssnitt och inte de konkreta klasserna.

Lägg till nedanstående funktioner.

1. **P-huset skall kunna hantera andra fordonstyper**
    a. Bussar, som tar fyra gånger så mycket plats som en bil. Endast de första 50 platserna har tillräckligt högt i tak för att bussar ska kunna komma in.
    b. Cyklar, som tar hälften så mycket plats som en MC
    c. Om en cykel har storleken 1, så har alltså
        i. MC 2
        ii. Bil 4
        iii. Buss 16

   d. En P-ruta har då storleken 4
   e. Fordonstyper och deras storlekar sparas i konfigurationsfilen
   f. Timtaxan för cyklar är CZK 5/timma, för bussar är det CZK 80 per timma. (Priset är alltså relaterat till fordonsstorlek – det kan med fördel utnyttjas)
2. **Programmet skall byggas så att datafilen automatiskt skapas och fylls på med testdata**, om den inte redan finns.
3. **Programmet skall ha en funktion för att läsa in nya konfigurationsdata, som kan användas när som helst**. Den behöver således vara synlig i menyn. Kontroll av de nya data skall göras så att man inte kan mata in ogiltiga värden. Exempelvis skall det inte gå att ta bort P-platser (dvs minska på antalet P-platser) om det står fordon parkerade på någon av dem. Den nya konfigurationen skall återspeglas i användargränssnittet.

Ytterligare krav för VG

För VG i betyg krävs det, förutom en fungerande version 2.1 av systemet enligt ovan, att du skriver en personlig reflektion över projektet och kursen. Reflektionen kan lämpligen struktureras enligt nedan:

1. Sammanfattning
2. Hur jag löste uppgiften
3. Utmaningar i uppgiften och hur de löstes
4. Metoder och modeller som använts för att lösa uppgiften
5. Hur du skulle lösa uppgiften nästa gång, givet vad du vet nu
6. Slutsats hemuppgift
7. Slutsats kurs

En sådan reflektion behöver inte bli särskilt omfattande, men ett par – tre sidor är normalt.

Tips

Det går att skapa riktigt snygga konsolapplikationer. Om man utnyttjar vad som faktiskt går att göra så kan man få till applikationer som inte skäms för sig. I detta projektet **ska** du använda biblioteket Spectre.Console. Här är några länkar till tips, verktyg och filmer som kan hjälpa dig en bit på väg

- Spectre.Console, på GitHub
  - Dokumentationen är utmärkt
- Tim Corey har nyligen (sommaren 2025) släppt en serie filmer om Spectre.Console. Det är sexton stycken små tiominuters filmer. Efter att ha sett dem och arbetat dig igenom hans exempel, har du ett tillräckligt bra grepp om Spectre.Console för att kunna skriva ditt eget gränssnitt.
- Awesome console apps in C#, en artikel om hur man kan göra snygga konsolapplikationer
  - Med ett kodexempel
- Create Better Looking Console Applications With Spectre.Console. En välskriven artikel från CodeMaze
- Det finns flera andra filmer om Spectre.Console på YouTube.

Tips 2

Att spara ett C# objekt till en fil, eller att läsa in till ett C# objekt från en fil är inte svårt. Det finns en liten film på YouTube som visar hur man gör: Using JSON IN C#! Serialization & Deserialization made easy!

Det finns även en liten kurs hos Microsoft: Store and retrieve JSON files.