C# Övning 5 - Garage 1.0

OBS - Resultatet av övningen skall visas för lärare och godkännas innan den kan anses vara genomförd.

Ett första övergripande projekt

För att koppla samman mycket av det ni lärt er, så skall vi nu bygga en garage applikation. Denna applikation skall tillhandahålla den funktionalitet som ett system kan behöva om det skall användas för att simulera ett enkelt garage. Det skall alltså gå att parkera fordon, hämta ut fordon, se efter vilka fordon som finns där och vilka egenskaper de har. Allt detta i en konsol applikation med huvudmeny och undermenyer.

Anledningen till att ni skall programmera ett garage är att det är enkelt att förankra uppdelningen i det hela. Vi kan huvudsakligen dela upp ett garage i följande delar:

Garaget: En representation av själva byggnaden. Garaget är en plats där en mängd av fordon kan förvaras. Garaget kan alltså representeras som en samling av fordon.

Fordon: Bilar, motorcyklar, enhjulingar eller vad för typ av fordon man nu vill ställa in i garaget.

Dessa är de två "objekttyper" som man ser i ett fysiskt garage. Men tittar vi närmare bör det också finnas subklasser till fordon, alltså att varje fordonstyp är en egen subklass i systemet. Utöver detta krävs det funktionalitet som hanterar att fordon ställs in i garaget, att fordon kan tas ut ur garaget, samt att vi kan få en presentation av vad som finns i garaget och söka i det.

I mer programmerings vänliga termer skall vi alltså som **minimum** ha:

- En *kollektion* av fordon; klassen `Garage`.
- En fordonsklass, klassen `Vehicle`.
- Ett antal subklasser till fordon.
- Ett användargränssnitt som låter oss använda funktionaliteten hos ett garage. Här sker all interaktion med användaren.
- En `GarageHandler`. För att abstrahera ett lager så att det inte finns någon direkt kontakt mellan användargränssnittet och garage klassen. Detta görs lämpligen genom en klass som hanterar funktionaliteten som gränssnittet behöver ha tillgång till.
- Vi programmerar inte direkt mot konkreta typer så vi använder oss av Interfaces för det tex `IUI, IHandler, IVehicle`. (Tips är att bryta ut till interface när implementationen är klar om man tycker den här delen är svår)

Kravspecifikation

Fordonen ska implementeras som klassen `Vehicle` och subklasser till den.

- `Vehicle` innehåller samtliga egenskaper som ska finnas i samtliga fordonstyper. T.ex. registreringsnummer, färg, antal hjul och andra egenskaper ni kan komma på.
- Registreringsnumret är unikt
- Det måste minst finnas följande subklasser:
    - `Airplane`
    - `Motorcycle`
    - `Car`
    - `Bus`
    - `Boat`
- Dessa skall implementera minst en egen egenskap var, t.ex:
    - *Number of Engines*
    - *Cylinder volume*
    - *Fueltype (Gasoline/Diesel)*
    - *Number of seats*
    - *Lenght*

Själva garaget ska implementeras som en generisk samling av fordon:

```
class Garage<T>
```

Dessutom ska den generiska typen begränsas med hjälp av en constraint:

```
class Garage<T> where ....
```

Vidare ska det gå att iterera över en instans av Garage med hjälp av foreach. Det innebär att Garage ska implementera den generiska varianten av interfacet IEnumerable:

```
class Garage<T> : ....
```

Klassen behöver inte ärva från någon annan klass eller implementera något annat interface.
Samlingen av fordon ska internt i klassen hanteras som en `array`. Den interna arrayen ska var **privat**. Vid instansieringen av ett nytt garage måste kapaciteten anges som argument till konstruktorn.

Vi ska **EJ** använda oss av en `List<Vehicle>` internt i Garage klassen!!!!

Funktionalitet

Det ska gå att:

- Lista samtliga parkerade fordon
- Lista fordonstyper och hur många av varje som står i garaget
- Lägga till och ta bort fordon ur garaget
- Sätta en kapacitet (antal parkeringsplatser) vid instansieringen av ett nytt garage
- Möjlighet att populera garaget med ett antal fordon från start.
- Hitta ett specifikt fordon via registreringsnumret. Det ska gå fungera med både ABC123 samt Abc123 eller AbC123.
- Söka efter fordon utifrån en egenskap eller flera (alla möjliga kombinationer från basklassen `Vehicle`). Exempelvis:
    - *Alla svarta fordon med fyra hjul.*
    - *Alla motorcyklar som är rosa och har 3 hjul.*
    - *Alla lastbilar*
    - *Alla röda fordon*
- Användaren ska få feedback på att saker gått bra eller dåligt. Till exempel när vi parkerat ett fordon vill vi få en bekräftelse på att fordonet är parkerat. Om det inte går vill användaren få veta varför.

Programmet ska vara en konsolapplikation med ett textbaserat användargränssnitt.

Från gränssnittet skall det gå att:

- Navigera till **samtlig** funktionalitet från garage via gränssnittet
- Skapa ett garage med en användar specificerad storlek
- Det skall gå att stänga av applikationen från gränssnittet

Applikationen skall fel hantera indata på ett robust sätt, så att den **inte kraschar** vid felaktig inmatning eller användning.

Unit testing

Testen ska skapas i ett eget testprojekt. Vi begränsar oss till att testa de publika metoderna i klassen `Garage`. (Att skriva test för hela applikationen ses som en extra uppgift om tid finns)

Testa gärna med att skriva testen före ni implementerat funktionaliteten! Använd er sedan ctrl . för att generera era objekt och metoder.
Flytta dessa genererade klasser till rätt projekt.
Implementera sen funktionaliteten tills testet går igenom.

Strukturera testen enligt principen.

1. **Arrange** här sätter ni upp det som ska testas, instansierar objekt och inputs
2. **Act** här anropar ni metoden som ska testas
3. **Assert** här kontrollerar ni att ni fått det förväntade resultatet

Tänk även på att namnge testen på ett förklarande sätt. När ett test inte går igenom vill vi veta vad som inte fungerat enbart genom att se på testmetod namnet.

Exempelvis:
[MethodName_StateUnderTest_ExpectedBehavior]

Public void Sum_NegativeNumberAs1stParam_ExceptionThrown()

Förslag på Extra funktionalitet (ej krav):

Möjlighet att också kunna söka på de fordonsspecifika egenskaperna.

Hantera flera garage som kan ha olika typer av fordon i sig exempelvis en hangar ett vanligt garage samt ett motorcykelgarage.
Detta kommer medföra att man ska kunna manövrera sig mellan dom olika garagen i ui:t för att kunna göra dom tidigare nämnda funktionerna de ska ske på bara det garaget som man har för tillfället valt.
Det ska tydligt visas vilket garage man för närvarande arbetar med.

Ett garage består inte längre av fordon utan av parkeringsplatser som i sin tur kan hålla fordon.

Det går att via C# skriva och läsa till filsystemet från er applikation. Ta reda på hur man gör för att kunna spara ert garage (via meny eller automatiskt vid avstängning) och ladda in ert garage (via meny eller automatiskt vid start av applikationen)

Olika fordon tar olika stor plats tex en bil tar 1plats en båt tar 2 platser ett flygplan kräver 3 platser osv en motorcykel tar endast 1/3 dels plats.

När man parkerar ska endast de fordon som garaget har plats för visas som allternativ.

Läsa in storleken på garaget via konfiguration.

Valfri funktionalitet ni tycker borde finnas.

Lycka till! 😊