Færdighedsprøve - Svømmehold

D. 23. januar 2024 kl. 9-13.00

Regler for prøven

Der må anvendes alle hjælpemidler til eksamen undtagen **alle** former for kunstig intelligens, herunder Chat GPT og Co-pilot.
Koden må ikke lægges op i github.
Det er ikke tilladt at kommunikere med andre under prøven.

Domæne beskrivelse

Svømmeklubben (De Ydmyges Klub – DYK) er en mindre svømme klub i Roskilde. **DYK** har brug for et system til at holde styr på de forskellige svømmehold(team). Det endelige system skal kunne håndtere oplysninger om svømmehold herunder hjælpemidler(aid), medlemmer(member) og instruktører(trainer).

I systemet skal der oprettes en klasse **Team**, der indeholder oplysninger om det enkelte hold. Klassen skal indeholde følgende properties:

Property	***Type***	***Beskrivelse***
Id	int	Holdets nummer eks. **2501**
Day	string	Den dag, hvor holdet svømmer, eks. **mandag**
Time	int	Det tidspunkt holdet svømmer (starter altid på hele timer), eks. **10**
Limit	int	Det antal medlemmer der kan være på et hold eks. **22**

Bemærk. Alle hold (team) starter på hele klokkeslæt og træningen varer 1 time.

Udviklingen af systemet sker igennem 5 iterationer. For hver iteration skal der tegnes nogle UML-diagrammer indenfor analyse og design og en mindre del at systemet skal implementeres i C#. Inden for hver iteration kan opgaverne laves i en valgfri rækkefølge.

Tegningen af diagrammerne kan laves på papir og der skal så sættes et billede ind i det dokument, der afleveres. Det er også tilladt at tegne vha. Visio eller lignende system.
Programmet laves i et nyt projekt af typen .NET Core Console Application projekt og kald det **DYK**.

Iteration 0

I denne indledende iteration skal du lave en domænemodel og user stories for hele systemet.

På de enkelte svømmehold(team) kan der være én instruktør(trainer), samt et antal medlemmer(member) som deltager. Nogle hold laver vand-gymnastik, hvorfor der er behov for et eller flere hjælpemidler(aid) (’plader’,’svømmefødder’,’pølser’,’musik’ mm)

A. Tegn en lille domænemodel, med attributter, associationer og multipliciteter.

B. Skriv fem User stories med tilhørende acceptance kriterier.

fortsættes

Iteration 1

I systemet skal alle hold gemmes i en software klasse **TeamRepository**, der indeholder en liste af hold af typen **List<Team>,** den indeholder IKKE properties men en række CRUD-metoder til at håndtere samlingen af hold.

I denne iteration skal der modelleres, designes og implementeres en mindre del af systemet, som omhandler oprettelse af klasserne **Team** og **TeamRepository**.

A. Tegn et (Design) klassediagram, med attributter, associationer og multipliciteter, for iteration 1.

B. Implementer klassen **Team**
   - Implementer klassen med instans felter, properties og passende konstruktør(er).
   - Implementer ToString() metoden i klassen.
   - Test klassen ved at oprette nogle instanser af den i Program.cs og skriv dem ud til konsollen med Console.WriteLine

C. Implementer klassen **TeamRepository**.
   - Implementer klassen med instansfelter og passende konstruktør(er).
   - Implementer en metode **GetAllTeams**, der returnerer alle Teams fra listen af Team objekter.
     - Overvej parametre og return type.
   - Implementer en metode **GetTeamById**, der returnerer et Team fra listen af Team objekter, ud fra et teams id.
     - Overvej parametre og return type.
   - Implementer en metode **AddTeam**, der kan tilføje et Team til en liste af Team objekter.
     - Overvej parametre og return type.
   - Implementer en metode **DeleteTeam**, der kan slette et Team fra en liste af Team objekter, ud fra et teams id.
     - Overvej parametre og return type.
   - Implementer ToString() metoden i klassen. Den returnerede string skal rumme informationer fra alle Team objekter i listen.
   - Test klassen ved at oprette en instans af **TeamRepository** i Program.cs og skriv Team objekterne ud til konsollen med Console.WriteLine. Formateringen af output er ikke af betydning.

D. Redegør for forskellen mellem en Domænemodel og et (Design) klassediagram.

Iteration 2

I denne iteration skal der modelleres, designes og implementeres en del af systemet, som omhandler hjælpemidler, **Aid**, og tilknytning af hjælpemidler til et Team objekt. Til et hold (team) kan der forekomme mange hjælpemidler.

Klassen **Aid**

Property	***Type***	***Beskrivelse***
Id	int	Unikt id der identificerer hjælpemidlet, eks. **33**
Description	string	Beskrivelse af hjælpemidlet. Eks. **"Svømmefødder**"
Amount	int	Antal hjælpemidler på holdet, eks. **12**

E. Udvid og opdater (design) klassediagrammet.

F. Implementer klassen **Aid**
   - Implementer klassen med instansfelter, properties og passende konstruktør.
   - Implementer ToString() metoden i klassen.
   - Test klassen ved at oprette nogle instanser af den i Program.cs og skriv dem ud til konsollen med Console.WriteLine.

Hvert **Team** skal indeholde en liste af objekter af typen **Aid**

G. Udvid implementationen af klassen **Team** således, at der til et Team objekt kan tilføjes objekter af typen **Aid**.
   - I klassen **Team** skal der implementeres en metode **AddAid**.
     - Overvej returtype og hvilke parametre, der er nødvendige.
   - Udvid ToString() metoden i Team klassen, så den også returnerer alle hjælpemidler til holdet.
   - Test klassen ved at tilføje nogle instanser af **Aid** til et Team objekt og skriv dem ud til konsollen med Console.WriteLine i program.cs.

H. Tegn et sekvensdiagram der viser, hvorledes der tilføjes og oprettes et nyt Team objekt og der tilføjes et nyt objekt af typen Aid til Team objektet, som til sidst indsættes i TeamRepository.
    - Sekvensdiagrammet skal tegnes med udgangspunkt i program.cs. Se bl.a. koden nedenfor:

```
Team novice = new Team(2501, “mandag”, 10, 22);
Aid noviceAid1 = new Aid(33, ”Svømmefødder”, 12);
novice.AddAid(noviceAid1);
repo.AddTeam(novice); // repo er et objekt af klassen TeamRepository
```

fortsættes

Iteration 3

I denne iteration skal systemet udvides med klasser til at håndtere medlemmer (**Member**) på holdet og instruktøren (**Trainer)** der varetager svømmeundervisningen.

Klassen **Member**

Property	***Type***	***Beskrivelse***
Id	int	Unikt id der identificerer medlemmet, eks. **101**
Name	string	Persons navn eks. **”Peter”**
Phone	string	Telefonnummer eks. **”66778899”**
EnrollmentYear	int	Det år hvor medlemmets begyndte i klubben eks. **2009**

Klassen **Trainer**

Property	***Type***	***Beskrivelse***
Id	int	Unikt id der identificerer instruktøren, eks. **35**
Name	string	Persons navn eks. **”Jakob”**
Phone	string	Telefonnummer eks. **”44119933”**
Skill	string	Hvilke hold instruktøren kan undervise eks. **”øvede”**.

A. Udvid og opdater (design) klassediagrammet med klasserne **Member** og **Trainer**.
   - Overvej om det er muligt at anvende arv (hint det er tilladt at indføre en ny klasse).

B. Udvid implementationen af klassen **Team** således, at der til et Team objekt kan tilføjes et objekt af typen **Member**.
   - I klassen **Team** skal der implementeres en metode **EnrolMember**, der tilføjer et medlem til holdet.

   - Der er en max grænse (Limit) på et hold, så udvid EnrolMember metoden, så den kaster en passende exception, hvis der ikke er mere plads på holdet.

   - Test den opdaterede klasse ved at tilføje nogle instanser af **Member** til et Team objekt. Vis et eksempel hvor der er plads på holdet og et hvor der ikke er plads på holdet (Vink du kan sætte Limit til 2 eller 3, så bliver det nemmere)

Iteration 4

I denne iteration skal der laves en metode, der kan udskrive alle de hold, der er optaget. Samt tjekke at to hold ikke ligger samtidigt.
Bemærk. Alle hold (team) starter på hele klokkeslæt og varer 1 time.

A. Opdater klassen **TeamRepository** i klassediagrammet med en metode **GetAllFullBooked**. Metoden skal returnere en liste af de team der er fuldt optaget.
    - Metoden skal have følgende signatur:

      public List<Team> GetAllFullBooked ()

    - Test metoden ved at kalde den i Program.cs.

B. Opdater i klassen **TeamRepository** metoden **AddTeam**, så den tjekker at holdet ikke ligger på en dag og tidspunkt, hvor der allerede er et hold. Hvis der er forsøg dobbelt hold kastes en passende exception.

    - Test metoden ved at kalde den i Program.cs.