|  |  |
| --- | --- |
| **Øvelse** | Den Lille Kagedyst |
| **Projekt** | DenLilleKagedyst |
| **Formål** | Repetition af flere emner fra 1.semester pensum |
| **Beskrivelse** | Den givne Visual Studio solution rummer et antal tomme eller ufærdige defini­tioner af flere klasser, der tilsammen skal modellere koncepter fra madlavning (f.eks. kager 😊). Din opgave er at udfylde disse klasse-definitioner, dels så de er i overensstemmelse med de beskrivelser der følger nedenfor, dels så den test-kode der er skrevet i metoden **Kør** i klassen **Test** (mere om den senere) kan køres (og give de forventede resultater). |
| **Trin** | 1. Download og unzip (VIGTIGT) filen **DenLilleKagedyst.zip** fra Moodle. 2. Åbn **DenLilleKagedyst** i Visual Studio. Vi starter i klassen **Ingrediens**. Denne klasse repræsenterer en "ingrediens", dvs. noget som indgår i en opskrift. Det kunne være sukker, æg, mel osv.. Klassen **Ingrediens** rummer fra start kun en **ToString**-metode. Du skal nu implementere tre properties **Navn**, **Enhed** og **PrisPrEnhed** samt en passende constructor i klassen, således at der ikke længere er nogen fejl i **ToString**. 3. Gå nu over i klassen **Test**, og ned i metoden **Kør**. Indkommentér (dvs. fjern **//**-symbolet fra starten af linjerne) de kode-linjer der står under **1)**. Det skal være sådan, at det ikke forekommer fejl i disse linjer. Hvis der forekommer fejl, må du gå tilbage i til **Ingrediens**-klassen og rette den til (det kan f.eks. være din definition af constructoren, der skal rettes). 4. Kør programmet. Du skulle gerne få udskrevet information om hver af de fire ingredienser. 5. Fortsæt til klassen **IngrediensRepository**. Denne klasse rummer en del elementer fra starten, men den er ikke færdig. Overvej disse spørgsmål:    1. Hvad er mon denne klasses formål? Dens navn antyder det nok…    2. Hvad hedder de enkelte elementer i klasse-definitionen (rummer klassen f.eks. instance fields, properties, constructor, metoder…)?    3. Hvad er det nu **List**-klassen kan gøre for os? 6. **IngrediensRepository** rummer tre metoder, som ikke er færdige. Start med at gøre metoderne **Tilføj** og **Udskriv** færdige, så de virker som beskrevet i kommentarerne for hver metode. Bemærk, at du selv må afgøre, om du vil lave ”ekstra”-delen for metoden **Tilføj**. 7. Indkommentér linjerne under **2)**, **3)** og **4)** i **Kør**-metoden i **Test**, og kør programmet. Du skulle gerne få udskrevet indholdet af **IngrediensReposi­tory** (fire **Ingrediens**-objekter) 8. Færdiggør nu metoden **Find** i **IngrediensRepository**, og indkommentér derefter linjerne under **5)** i **Kør**-metoden i **Test**. Du skulle gerne se at **Find** finder ingredienserne ”Mel” og ”Sukker”, men IKKE ”Salt”. 9. Fortsæt til klassen **Lager**. Denne klasse repræsenterer et ”lager” af ingredi­enser, f.eks. et køkkenskab. Klassen rummer nogle elementer fra starten, og minder lidt om **IngrediensRepository**, men tjener et andet formål. Overvej først, hvad den grundlæggende forskel mellem de to klasser er i.f.t. deres formål (Hint: Hvor repræsenteres information om egenskaberne for en ingrediens, og hvor repræsenteres information om konkrete mængder af ingredienser?). 10. Færdiggør nu metoderne **Tilføj** og **Udskriv**, så de virker som beskrevet i kom­mentarerne for hver metode. Indkommentér derefter linjerne under **6)**, **7)** og **8)** i **Kør**-metoden i **Test**, og kør programmet. Du skulle gerne få udskrevet information om alle ingredienserne på lageret. **NB**: Fik du den rigtige værdi for ”Mel” (skal være 3,7)? 11. Færdiggør derefter metoden **FindAntal**, og indkommentér linjerne under **9)** i **Kør**-metoden i **Test**. Du skulle gerne se at **FindAntal** finder de antal som blev vist i udskriften fra forrige trin. **NB**: Fik du den rigtige værdi for ”Peber” (skal være 0)? 12. Fortsæt til klassen **Opskrift**. Denne klasse repræsenterer en opskrift – f.eks. på en kage – i form af navnet på den madvare opskriften vedrører, samt de ingredienser – givet med navn og krævet antal – der skal bruges. 13. Start med at implementere de instance fields, properties og constructor der skal bruges i denne klasse. VIGTIGT HINT: En ”opskrift” minder vel i meget høj grad om et ”lager”, da de begge rummer oplysninger om navn og antal for en mængde af ingredienser…? Så måske kan vi lade os kraftigt inspirere af hvordan **Lager** er implementeret 😊. 14. Indkommentér linjerne under **10)** i **Kør**-metoden i **Test**, og tjek at du nu er i stand til at oprette **Opskrift**-objekter. 15. Færdiggør nu metoderne **Tilføj**, **FindAntal** og **Udskriv** i **Opskrift**. Igen burde det være muligt at lade sig kraftigt inspirere af implementationen af de tilsvarende metoder i **Lager**. 16. Indkommentér linjerne under **11)** og **12)** i **Kør**-metoden i **Test**, og tjek at du får de forventede resultater. 17. Fortsæt til klassen **OpskriftLogik**. Denne klasse rummer to metoder, som kan beskrives som eksempler på forretningslogik relateret til opskrifter. Færdiggør de to metoder, så de virker som beskrevet i kom­mentarerne for hver metode. Indkommentér derefter linjerne under 13**)**, **14)** og **15)** i **Kør**-metoden i **Test**, og kør programmet. Du skal naturligvis kontrollere, om programmet gav de resultater du forventede. 18. Nu har du løst opgaven… men der er faktisk en hel del ting der (måske) kunne forbedres. Her en er række af mulige forbedringer du kan overveje, og meget gerne prøve at implementere. Der er frit slag mht. at ændre på den kode du nåede frem til efter trin 17, men det skal helst være sådan at test-koden i **Kør**-metoden kan køre som før, og give de samme resultater.     1. Givet at vi i hele opgaven har antaget at navnet på en ingrediens er unikt, burde vi kunne forbedre implementationen af **Ingrediens-Repository** ved at bruge en **Dictionary** i stedet for en **List** til intern opbevaring af vores **Ingrediens**-objekter. Prøve at indføre denne ændring, således at ”interfacet” (properties og metoder) til **Ingredi­ensRepository** er bevaret som det var, men med en ændret imple­mentation.     2. Klasserne **Lager** og **Opskrift** har en hel del tilfælles. Måske kan vi bruge nedarvning til at undgå denne kode-duplikering? Der er måske en lille udfordring mht. metoden **Udskriv**, men kan vi ikke klare det med passende brug af *abstract* og *override*, og måske endda **Template Method** design-pattern?     3. De to metoder i **OpskriftLogik** ser ud til at få al den information de skal bruge via deres parametre, da der ikke er hverken instance fields eller properties i **OpskriftLogik**. Kunne vi gøre det lidt nem­mere at bruge disse metoder? Hint: Kunne vi ændre metoderne sådan at man ikke behøver lave et **OpskriftLogik-**objekt for at kalde metoderne? Her må man gerne ændre lidt på test-koden i **Kør** 😊.     4. Vi har i hele opgaven antaget, at navnet på en ingrediens er unikt, og repræsenteret et navn som en **string**. Dette valg gør dog, at programmet er sårbart ift. stavefejl, f.eks. hvis nogen skriver ”Suker” som navnet på en ingrediens, hvor de reelt mente ”Sukker”. Overvej, hvorvidt man kan bruge en anden data-type til at repræsentere lovlige navne på ingredienser (Hint. Hvad er en Enumeration-type for noget…?)     5. De to metoder i **OpskriftLogik** repræsenterer forretningslogik rela­te­ret til opskrifter. Andre eksempler på forretningslogik man kunne prøve at implementere er:        1. Givet et lager og en opskrift, hvor mange styk kan man så producere (altså et lidt mere komplekst spørgsmål end blot hvorvidt man kan producere eller ej)?        2. Givet et lager og en opskrift, og givet det tilfælde at vi ikke kan producere den givne opskrift, hvilke ingredienser (og antal af hver ingrediens) mangler vi?     6. Det kan virke lidt underligt at have forretningslogik for opskrifter liggende i en separat klasse, da det vel burde ligge i **Opskrift**-klassen selv (jf. *Information Expert* design pattern). Prøv at se om det kan lade sig gøre at flytte logikken ind i **Opskrift**-klassen. |