|  |  |
| --- | --- |
| **Øvelse** | SmartRepository |
| **Projekt** | SmartRepository |
| **Formål** | Brug af lambda-udtryk ifm. en repository-implementation |
| **Beskrivelse** | Den givne Visual Studio solution rummer fra starten følgende klasser:   * **Produkt**: En rimeligt ligetil domæne-klasse. * **IProduktRepository**: En interface-definition af et repository der kan rumme **Produkt**-objekter. Det rummer tre ret specialiserede metoder til at hente produkter der opfylder visse betingelser. * **ProduktRepository**: En ufærdig implementation af interfacet **IProduktRepository**. * **SmartRepository**: En type-parameteriseret klasse, som er tænkt som en baseklasse for konkrete repository-klasser. * **ProduktRepository2**: En tom klasse-definition, der skal udfyldes sidst i opgaven. |
| **Trin** | 1. Download og unzip (VIGTIGT) filen **SmartRepository.zip** fra Moodle. 2. Åbn **SmartRepository** i Visual Studio. Start med at se nærmere på klassen **Produkt** og interfacet **IPro­dukt­Repository** og. De fire metoder i interfacet bliver testet i **Program.cs**, hvor det forventede resultat også er angivet for hver metode. 3. Kør nu programmet. Tre af metoderne virker ikke efter hensigten endnu… Det skyldes at klassen **ProduktRepository** (som implementerer interfacet **IProduktRepository**) ikke er gjort færdig (se de tre **// TODO** i klassen). 4. Implementér metoderne i **ProduktRepository**, så de returnerer de korrekte **Produkt**-objekter. Du kan enten bruge de statements du kender i forvejen (f.eks. ***foreach*** og ***if***), eller prøve at bruge metoden **FindAll**, som kan kaldes på et **List**-objekt (og dermed på ***\_produkter*** i **ProduktRepository**). 5. Hvis du løste Trin 4 ved brug af **FindAll**-metoden, er du færdig med dette trin 😊. Hvis ikke, så prøv at ændre din implementation til at bruge **FindAll** i stedet. Husk at **FindAll** tager et såkaldt lambda-udtryk (se noterne) af typen **Predicate<Produkt>** som parameter, hvis den kaldes på ***\_produkter***. 6. De tre specielle metoder fra interfacet **IProduktRepository** kan således alle implementeres ved at kalde metoden **FindAll** på ***\_produkter*** med forskellige lambda-udtryk. Vi tager denne idé lidt længere i klassen **SmartRepository**. Tag et kig på denne klasse nu. Denne klasse er tænkt som en baseklasse for konkrete repository-klasser. Selv om klassen er ret lille, burde den:    1. Kunne være baseklasse for repository-klasser for en vilkår­lig domæne-klasse (hvorfor det…? Hvad er det nu det betyder, når der står **<T>** i definitionen af **SmartRepository**…?), og    2. Kunne – ved hjælp af metoden **HentHvorDetGælderAt** – hente de objekter i repository der opfylder en vilkårlig betingelse (hvorfor det...? Hvordan er det vi kan tilføre metoden den betingelse den skal bruge til udvælgelse af objekter…?) 7. Fortsæt til klassen **ProduktRepository2**. Denne klasse skal du nu implemen­tere så den også implementerer **IProduktRepository**, men så den gør det ved hjælp af baseklassen **SmartRepository** (dvs. **ProduktRepository2** skal arve fra **SmartRepository** og implementere **IProduktRepository**). Mere specifikt skal du således implementere de tre specielle metoder ved at kalde **HentHvorDetGælderAt**. 8. Du kan teste din implementation af **ProduktRepository2** ved at gå tilbage til **Program.cs**, og skifte til at bruge **ProduktRepository2** (udkommentér linje 2, og indkommentér linje 3). Du skal selvfølgelig få præcis de samme resul­tater som ved brug af **ProduktRepository2** 😊. |