# Disaheim 4

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1:** anvende centrale metoder til at specificere og konstruere algoritmer [...] * **1Pf2:** anvende centrale faciliteter i programmeringssproget til realisering af algoritmer, designmønstre, […], datatyper, datastrukturer, designmodeller […] * **1Pf3**: anvende et i professionen udbredt, integreret udviklingsværktøj, herunder versionsstyringssystem […] til at designe og konstruere praksisnære applikationer baseret på en lagdelt arkitektur * **1Pk3**: i en struktureret sammenhæng tilegne sig ny viden, færdigheder og kompetencer inden for programmeringssprog, udviklingsværktøjer, programmeringsteknikker og programdesign * **1Pf6**: anvende moderne teknikker og værktøjer til afvikling af test […] |
| **Forventet læringsudbytte [SOLO]** | Princip- og mønsterssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn (Single Responsibility Principle) i **SOLID** * [Unistrukturel] Du kan huske enkelte kendetegn (Controller) i **GRASP** * [Unistrukturel] Du kan genkende enkelte kendetegn ved Repository-mønstret * [Unistrukturel] Du kan genkende enkelte kendetegn ved logisk arkitektur og lagdeling i forbindelse med **Arkitekturprincipper**   Programmeringssporet:   * [Unistrukturel] Du kan huske enkelte kendetegn ved ’is’- og ’as’-operatorerne til safe type casting |
| **Din forberedelse** | Programmeringssporet:   * Skimlæs [Larman] kapitel 13 (s. 197-212) Logical architecture and UML package Diagrams   + Bemærk, de tre øverste lag i figur 13.4 (s. 203) * [Larman] 17.3 (s. 302-314) Controller   + Hav fokus på, hvilket problem controlleren løser * [GRASP](https://en.wikipedia.org/wiki/GRASP_(object-oriented_design))   + Med fokus på Controller * [SOLID, GRASP, and Other Basic Principles of Object-Oriented Design](https://dzone.com/articles/solid-grasp-and-other-basic-principles-of-object-o)   + Læs kun om Single Responsibility Principle (SRP) og Controller for nu * [Repository pattern in C#](https://codecompiled.com/?s=Repository+pattern+in+c%23+2015)   + Fokusér på forklaringen af Repository-mønstret i starten af artiklen, og skimlæs det sidste med interfaces fra afsnittet ”How Repository Pattern is implemented in C#” * [Teste om et objekt er en bestemt type med ‘is’-operatoren](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/is) * [Safe Type Casting with IS and AS Operator](https://www.dotnettricks.com/learn/csharp/safe-type-casting-with-is-and-as-operator) |

Du er nu kommet så langt med Disaheim (og nok også dine andre projekter), at det bliver nødvendigt at introducere nye vigtige designelementer i arbejdet med at organisere et it-system med mere end nogle få klasser for at sikre overblik og minimere risikoen for et uhensigtsmæssig kollaps i designstrukturen.   
I denne opgave introduceres du til nogle få vigtige og centrale designprincipper og -mønstre fra både SOLID og GRASP, som skal hjælpe dig med at strukturere Disaheim-løsningen på en ny og bedre måde. Du kommer også til at arbejde med Repository-mønstret og begynder så småt at se på den logiske arkitektur af et it-system via en simpel lagdelt arkitektur.

# Dagens ord:

Når man designer klasserne i et it-system, skal man sikre sig, at klassernes ansvar bl.a. er nemme at forstå og vedligeholde og ikke mindst teste. Her er **Single Responsibility Principle** (SRP og S’et i SOLID) helt centralt, hvor man i sit design sigter mest muligt efter, at en klasse kun har ét ansvar. Hvis man ændrer i sin klasse, kan dette betyde, at eventuelle sideeffekter (og dermed ændringer) i relaterede klasser minimeres eller fjernes helt. Dette relaterer også til to andre principper i GRASP: Low Coupling og High Cohesion

Ud over anvendelse af principper, så har man gennem tiden observeret visse designmønstre (kan du huske mønstergenkendelse i Computational Thinking?), som ofte går igen i softwareudvikling. Brug af disse mønstre giver ikke kun gode løsninger til problemstillinger, man ofte støder på, men også større læsbarhed og færre fejl. Et mønster, som du allerede har stiftet overfladisk bekendtskab med, er **Controller-mønstret**, hvis ansvar (jf. GRASP) i en **lagdelt arkitektur** er at håndtere hændelser fra brugergrænsefladen og videreformidle disse hændelser til systemets forretningslogik, dvs. dets operationer, kommandoer, m.m., der opretter, ændrer, sletter og gemmer data. Et andet mønster, man ofte støder på, er **Repository-mønstret** fra Martin Fowler. Dette mønster skaber et abstraktionslag med en Repository-klasse, som abstraherer (indkapsler) din konkrete implementering af opbevaring og tilgang til systemets data.