Clean Code och skalbarhet

Sebastian Lindgren

Välkomna till dagens föreläsning!

Clean Code - skalbarhet!

Dagens agenda

- Repetition Clean Code
- OOPs centrala delar
- Skalbarhet

Repetition 1/3: clean code

Vi ser vad ni minns om clean code ifrån senaste tillfället!

- Varför blir kod “dirty over time”?
- Vad segas ned av dirty code?
- Varför är clean code en “matter of professional survival” (relaterat till: code dept).

Repetition 2/3: refactoring

Vi ser vad ni minns om refactoring ifrån senaste tillfället!

- Vad gör man när man refaktoriserar kod?
- Vad handlar refaktorisering inte om?

Repetition 3/3: code-smells

Vi ser vad ni minns om code-smells ifrån senaste tillfället!

- Vilka olika typer av code-smells kommer ni ihåg?

Repetition 3/3: code-smells

Vi ser vad ni minns om code-smells ifrån senaste tillfället!

- Vilka olika typer av code-smells kommer ni ihåg [3]?
  - Naming: *Non-informative Name*, *Disinformation* . . .
  - Comments: *Redundant Comment*, *Obsolete Comment* . . .
  - Formatting: *Non-aligned grouping*, *Magic numbers*. . .
  - Code structure: *functions/methods*, *classes*, *interface*, *objects: Too many arguments*, *Implementation Dependency*, *Feature Envy* . . .
  - Error handling: *Misplaced Responsibility*. . .
  - Layering/boundaries: *Artificial Coupling*. . .
  - Testing/testability: *Slow Test* . . .

Objektorienterad programmering

Innan vi går djupare in på *skalbarhet* och hur vi kan uppnå detta med OOP så kör vi en liten genomgång av OOP och dess centrala delar.

Objektorienterad programmering

Vad är OOP?

I objektorienterad programmering så vill man komma undan situationen att data och funktioner ligger utspridda. Genom att gruppera data och funktionalitet kan man förbättra strukturen i programmet och dess delar, öka återanvändbarheten och slippa repetition.

Clean Code och OOP

Med hjälp av väl använda OOP koncept och program där vi har väl genomtänkta och uppdelade entiteter i form av klasser och objekt som var och en är anvarig för ett visst område uppnår vi mycket enklare clean code.

Clean Code och OOP

Clean Code och OOP går hand i hand. Om man kan och använder sina grunder i objektorienterad C# väl, planerar och refaktorerar sina program, och till detta ser till att följa clean code principerna så uppnår vi kod av hög standard som är enkel att skala.

OOPs centrala delar

Så vilka är de centrala delarna i OOP [2]?

- Abstraktion
  - Separationen av och fokuset på interfaces hos en klass och inte implementationen
  - Vi delar upp vår logik i interface och faktiskt utförande i klasser
- Inkapsling
  - Ha data och funktioner kopplade till en entitet
  - Bara visa de detaljer som är viktiga att interagera med
- Arv
  - Hierarkier där klasser bygger på andra klasser
- Polymorfism
  - Enhetlig behandling av klasser i en hierarki. Så länge man kan hantera objekt i roten av hierarkin kan man hantera objekt som kommer senare också.

Abstraktion

Vi bygger våra klasser på mer abstrakta begrepp eller s.k. kontrakt.

- Abstrakta klasser
- Interfaces

Med dessa så visar vi vilka Properties och publika metoder som måste finnas. I varje ärvande klass så kommer dessa att overrideas/implementeras.

Inkapsling

Vi vill kunna bygga våra entiteter med flera olika fält och hjälpande metoder som inte behöver synas utåt för användaren av klassen och objekten.

- nyckelord för begränsning av tillkomst
  - public
  - private
  - protected
  - (internal)

Arv

Vi vill kunna ärva metoder ifrån olika klasser för att kunna bygga på tidigare kodbaser och på så sätt undvika att skriva mer kod än nödvändigt eller att skriva koddubletter.

- ":"

Ofta används arv för att implementera ett eller flera interfaces.

Polymorfism

Det finns många olika scenarion där vi vill kunna använda oss av polymorfism. T.ex. när vi vill kunna köra olika funktioner i en lista av objekt som ärver ifrån samma abstraktion/interface.

- virtual
- override

Polymorfism exempel


```
List<Player> players = new List<Player>();

players.Add(new Player());
players.Add(new Mage());
players.Add(new Warrior());

foreach (Player player in players)
{
    player.Attack();
}
```

Modellera världen

Med hjälp av de entiteter vi skapar med OOP samt dess centralbegrepp kan vi s.k. *modellera världen*. Det finns många olika patterns inom programmering som går att använda för att öka hur väl man kan använda och återanvända kod.

Bra resurser

Ett allmänt tips för egna självstudier inom OOP är att titta lite närmare här:

- C# console app + debug: https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-console?view=vs-2022
- OOP: https://csharpskolan.se/article/oop-arv/
- Klasser och objekt: https://csharpskolan.se/article/oop-objektens-liv/
- Inkapsling: https://csharpskolan.se/oop-inkapsling/
- Polymorfism: https://csharpskolan.se/article/oop-polymorfism/
- Videolektioner: https://dotnet.microsoft.com/en-us/learn/csharp

Vad är skalbarhet?

Med skalbarhet i kod pratar vi om hur enkelt vi kan *anpassa oss till nya behov/förändring*.

Skalbarhet

1. En fråga man kan ställa sig är: *om vi skulle få ett nytt behov, hur snabbt kan vi förändra koden så att det nya behovet går att tillfredställa?*
   - Ju färre nya rader kod som skulle behövas. Desto bättre!

Skalbarhet

2. En faktor som också är bra att ta i hänsyn är hur många *repeating parts* man har.
   - Vi vill hellre bygga strukturer som hanterar repetition än att repetera kod.

Skalbarhet

Ett tankeexperiment. Tänk dig att du har en applikation för en studentportal som har 3 stakeholders: Student, Teacher och Support. Efter att ha haft igång applikationen i 6 månader så kommer ett behov av att lägga till “Parent” som stakeholder. Hur enkelt skulle det vara för dig att lägga till denna nya entitet?

Skalbarhet med OOP

OOP är en paradigm som ger dig verktyg att få mycket högre skalbarhet! Men det krävs också att man arbetar med de verktyg man har.

I praktiken

I grunden handlar det om att använda **abstraktion** och fiffiga användningar av inkapsling, arv och polymorfism.

Patterns

Vi vill också använda patterns. Något vi kommer att lära oss mer om i nästa föreläsning [4] [1].

Övning

Låt oss göra ännu en liten övning!

[1] *Design patterns*. URL: https://refactoring.guru/design-patterns.

[2] Kyle Herrity. *What is object-oriented programming? 4 basic concepts of OOP*. URL: https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming.

[3] Robert C. Martin and James O. Coplien. *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ [etc.]: Prentice Hall, 2009. ISBN: 9780132350884 0132350882. URL: https://www.amazon.de/gp/product/0132350882/ref=oh_details_o00_s00_i00.

[4] *What's a design pattern?* URL: https://refactoring.guru/design-patterns/what-is-pattern.