# Properties

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1:** anvende centrale metoder til at specificere og konstruere algoritmer [...] * **1Pf2:** anvende centrale faciliteter i programmeringssproget til realisering af algoritmer […] * **1Pf3**: anvende et i professionen udbredt, integreret udviklingsværktøj, herunder versionsstyringssystem […] til at designe og konstruere praksisnære applikationer […] * **1Pk3**: i en struktureret sammenhæng tilegne sig ny viden, færdigheder og kompetencer inden for programmeringssprog, udviklingsværktøjer, programmeringsteknikker og programdesign |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **C#** **klasse** – C# klasse, felt, metode, property (get, set), access modifiers (public, private), access keywords (this) * [Unistrukturel] Du kan genkende enkelte kendetegn ved **OOP-principper** – indkapsling (en: encapsulation)   Færdighedssporet:   * [Unistrukturel] Du kan genkende enkelte kendetegn ved **versionsstyring** – Git-kommandoer: init, status, add, commit og push, pull, merge, branch, checkout, log, config (aliases) |
| **Din forberedelse** | Programmeringssporet:   * Indkapsling og Properties (video: 32:06) * [Encapsulation In C#](https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/)   + I kodeeksempel 3 er der en mindre fejl: Der skulle have stået ”d.Department” frem for ”d.department” på linje 16 i kodeeksemplet (overvej hvorfor dette er en fejl)   Færdighedssporet:   * [2.7 Git Basics – Git Aliases [ProGit]](https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases) * [3.1 Git branching – Branches in a nutshell [ProGit]](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell) * [3.2 Git Branching – Basic Branching and merging [ProGit]](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) |

Du fortsætter i denne opgave med OrangeTree-casen fra forrige Ex09-opgaven, men nu med fokus på properties. **Hvis du ikke er helt færdig med Ex09-opgaven, da gør den færdig først til og med øvelse 2.5 som en del af forberedelsen.**

Derudover introduceres du til, hvordan man kan benytte et command-line interface (CLI) til at skifte fra en Git branch til en anden, benytte merging og benytte Visual Studio IDE til at tilgå jeres Git repository.

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug **Møde på midten** (beskrivelse følger i punktform) i hver gruppe til at diskutere begreberne ”indkapsling”, ”property”, ”git branch”, ”git merge”, ”git checkout”. Husk, at alle i gruppen skal have taletid:

1. Hvert enkelt medlem i gruppen skriver sine overvejelser om begreberne ned (2 minutter).
2. Hver gruppe starter **Ordet rundt**, idet det første gruppemedlem forklarer sine overvejelser   
   (2 minutter). Hele gruppen diskuterer bidraget (2 minutter). Hvis gruppen enes om, at det er en god forklaring, skriver det første gruppemedlem sit bidrag ind i gruppens dokument.
3. Det næste gruppemedlem forklarer sine overvejelser osv. Fortsæt med **Ordet rundt** indtil alle 4 gruppemedlemmer har bidraget, eller tiden er gået.

*Tidsramme: 20 minutter*

# Øvelse 2: OrangeTree-simulation, men nu med properties

Du skal fortsætte med din OrangeTree implementering fra forrige opgave Ex09-OrangeTree.

## Øvelse 2.1: Omdan felter til properties

Du har i Ex09-opgaven implementeret en OrangeTree klasse med følgende felter:

private int age;

private int height;

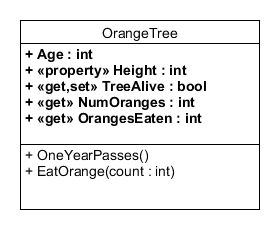
private bool treeAlive;

private int numOranges;

private int orangesEaten;

Med udgangspunkt i dagens forberedelse, da udfør følgende:

* Omdan hver af de 5 felter til tilsvarende public properties, så C#-klassen udtrykker nedenstående designklasse for OrangeTree:



Designklassen benytter sig af nogle lokalt udvalgte konventioner, som du skal være opmærksom på:

* De 5 felter er ikke angivet i designklassen, da der her er valgt konventionen, at de er der implicit via deres property. Dvs. der skal både være et privat age-felt og en public Age-property i C#-klassen, selvom det kun er angivet som en Age-property i designklassen. Det sparer en del plads. Men man skal være enig i at følge denne konvention, hvor felter ikke vises, men er der implicit.
* Navngivningskonvention: en property anvender Pascal-case (UpperCamelCase), og det tilsvarende felt lowerCamelCase.
* Indkapsling: properties er altid public og felter altid private.
* Der er anvendt forskellige stereotype-notationer for at illustrere forskellige muligheder; De tre første properties Age, Height og TreeAlive skal implementeres helt ens i C# med både en getter og setter. Det er bare angivet på tre forskellige måder efter smag og budskab.
* Felterne numOranges og orangesEaten skal kun kunne læses, så der er ikke behov for nogen setter i deres tilsvarende property. Dette er angivet i designklassen med get-stereotypen. Dvs. du skal tilsvarende kun implementere en getter i deres respektive C#-properties.
* Der er ikke længere brug for SetXXX og GetXXX-metoderne (dem kan du slette, når dine properties er på plads).

## Øvelse 2.2: Kontrollér ændring af Age-property

Udfør følgende:

* Sørg for, at man aldrig kan sætte et appelsintræ’s alder til et negativt tal via dens Age-property.

## Øvelse 2.3: Opdatér unit-test af OrangeTreeSim

Du er nu parat til at teste din nye implementering af OrangeTree-klassen.

Under normale omstændigheder må du **aldrig ændre i en unit-test, du ikke selv har lavet**. Testen udleveret i Ex09-opgaven har GUD lavet (I må selv gætte, hvem det er) og er selvfølgelig helt korrekt og uden fejl ☺ Almindelige dødelige kan godt lave fejl, så i de tilfælde skal udviklerne give besked til personen, der har lavet testen, og ikke selv ændre den; det kunne jo være, at det ikke var en fejl.

Så forhåbentligt har du ikke ændret i testen til Ex09 for at få den til at fungere.

Når det nu er sagt, skal du i denne øvelse faktisk rette i unit-testen.

Udfør følgende:

* Erstat alle GetXXX og SetXXX-metoder i testen til tilsvarende properties i din nye OrangeTree-klasse.
* Kør testen, og tilret, indtil den kører uden problemer (er grøn)

# Øvelse 3: Git Branching and Merging

Du skal i denne øvelse benytte dig af Git Branching til at oprette en ny branch baseret på din master branch, for herefter at lave en ændring i denne nye branch, for endeligt at tilføje/flette disse ændringer med den oprindelige master.

## Øvelse 3.1: Git branch

Gå ind i din hjemmeprojekt-mappe, som du har lagt op på Git (som du oprettede i Ex03-CSharpData i øvelse 5.2.2 – 5.2.4).

Bekræft, at du er på master branchen ved at benytte følgende git kommando:

|  |
| --- |
| git status |

Du skulle gerne kunne se følgende besked:



Herefter skal du oprette din nye branch, lad os give den navnet ”test”:

|  |
| --- |
| git branch test |

Nu når du har oprettet en ny branch, skal du gå over i den med følgende kommando:

|  |
| --- |
| git checkout <branch name> |

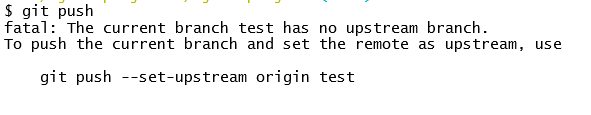
Benyt ”git status” igen og bekræft, at du befinder dig i din nye test-branch.

Nu er du klar til næste øvelse.

## Øvelse 3.2: Ændring i ny branch

Udfør følgende steps:

* Se, hvad du har i den mappe, som du står i med: ”ls” eller ”dir”
* Opret en ny mappe med ”mkdir testfolder”
  + Benyt ”ls” eller ”dir” igen, og bekræft, at du nu kan se mappen
* Gå ind i mappen med ”cd testfolder”
  + Benyt ”ls” eller ”dir” igen, og bekræft, at du er inde i mappen
* Skab en ny fil inde i mappen med: ”touch test.txt”
  + Bekræft, at du kan se filen med ”ls” eller ”dir”
* Naviger op til rod-mappen igen med ”cd ..”
  + Bekræft, at du er på det øverste niveau igen med ”ls” eller ”dir”
* Stage dine filer med: git add .
* Opret herefter dit commit: git commit -m “Testing branching”
* Push din nye branch med ”git push”  
  Du vil sandsynligvis se følgende fejl:



* Git brokker sig over, at din lokale branch ikke har nogen upstream (remote) branch endnu. Så git kan ikke automatisk give dig lov til at benytte "git push", da git ikke ved, hvor dit commit skal hen.
* Du kan fikse problemet ved at benytte det hint, git bash giver dig:
  + git push --set-upstream origin test

## Øvelse 3.3: Se ændringerne, skift branch

Benyt nu ”git log” til at se de commits din nuværende branch indeholder. Registrer, at du kan se dit nyeste commit med navnet ”Testing branching”. (Bemærk, du skal benytte dig af ’q’ for at lukke din git log).

Herefter skal du skifte tilbage til din master branch og se, hvilke commits den indeholder.

|  |
| --- |
| git checkout master  git log |

Som du kan se, så fremgår ”Testing branching” ikke af listen.

Lav en ændring på master branchen, og commit den.

## Øvelse 3.4: Git merge

Nu skal du tilføje/flette dine ændringer/dit commit fra test-branchen ind i master branchen. Vi går først tilbage til vores test-branch, hvorefter vi benytter os af ”git merge” til at merge ændringerne til masteren.

|  |
| --- |
| git checkout test  git merge master |

Nu skal du benytte git log, for at se (visuelt), hvordan dine branches er blevet flettet sammen:

|  |
| --- |
| git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short |

**Note**: For at forlade ”git loggen”, skal du trykke på ”q”.

Du kan tilføje et alias (en kommando genvej) for denne kommando, ved hjælp af:

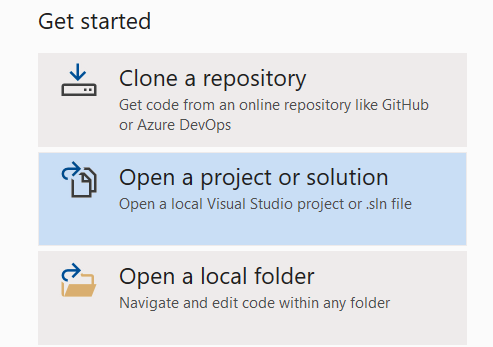
|  |
| --- |
| git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short" |

# Øvelse 4: Tilgå jeres repository inde i Visual Studio IDE

I denne øvelse, skal du åbne et af dine hjemmeprojekter, som du har lagt op på GitLab, inde fra Visual Studio.

## Øvelse 4.1: Åbn eksisterende projekt

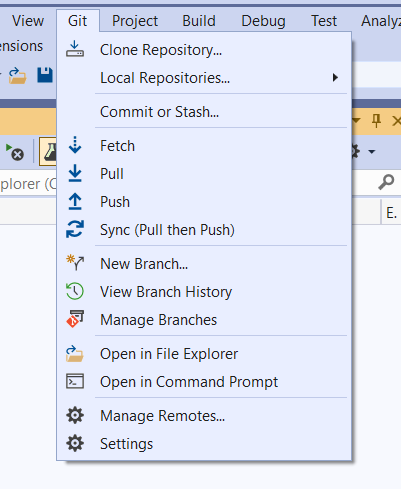
Åbn Visual Studio. Vælg ”Open a project or solution”, og vælg dit et hjemmeprojekt, du har lagt op på GitLab.



## Øvelse 4.2: Inspektion af Git menu og Visual Studio bundpanel

Gennem inspektion skal du og dit team se, om I kan gennemskue, hvordan:

* I ser historikken for den branch, som I befinder jer i
* Laver et commit med jeres ændringer
* Opretter en ny branch
* Laver en ændring i en ny branch
* Merger jeres ændringer fra jeres nye branch til masteren



# Øvelse 5: Review spørgsmål

* Hvad er en property, og hvorfor giver det mening at anvende den?
* Hvad betyder det at have en konvention i forhold til navngivning, designklasser og C#-klasser?
* Hvad er et Git alias? Og hvorfor er det smart?
* Hvad sker der, når du trykker på "Sync" inde i Visual Studio IDE?
* Hvad er en branch?
* Hvad er "merging", når vi snakker om Git?
* Hvordan kan vi se historikken af den branch vi befinder os i via Git Bash?
* Hvordan kan vi se historikken af den branch vi befinder os i via Visual Studio IDE?