MENU

Generel opgavevejledning

Abstract:

Den følgende vejledning er generel og skal ikke følges slavisk. Man skal dog altid vide med sig selv, hvorfor man evt. vælger at fravige den.

Objektorienterede konstruktions-opgaver drejer sig om at lave en eller flere klasser. Ud over disse klasser skal man lave en test-anvendelse, der illustrerer en forståelse af hvordan man anvender instanser af klasserne til at opbygge objektsystemer.

Når man løser opgaven er det godt at arbejde systematisk frem efter følgende spørgsmål, som skal besvares for, at man kan løse opgaven.

Hvilke klasser skal der laves?

Det er vigtigt for overblikket at forstå hvilke klasser der skal laves. Klasserne har en sammenhæng og uden at kende alle klasser kan man umuligt forstå sammenhængen. Gør dig det derfor helt klart hvilke klasser du skal lave.

Læg dig fast på hvad instansvariablene skal hedde. I nogle opgaver er det angivet hvad klasserne skal hedde - hold dig *præcist* til disse navne! Det er specielt vigtigt du gør det når du først laver og gemmer de filer som klasserne skrives i. Skal du til at ændre navnene er det irriterende, og bryder din koncentration. Hvis du i stedet vælger at holde dig konsekvent til forkerte navnene vil de distrahere dig igennem hele opgaveløsning - vær omhyggelig!

Når man har klassenavnene klar, er det en god idé at lave alle klasserne i hver sin file, uden indhold og få gemt dem med det samme. Så er de klar og man har løst et teknisk problem og kan koncentere sig om det følgende.

[Kode] Jeg bruger selv en sekvens i kawa der hedder: **Ctrl-N** - `class` <klassens navn> `{` <return> `}` - `Ctrl-S` - <return> - <return> - og forfra igen.

Når man først har fået denne sekvens ind på rygraden kan man hælde klasser ned på harddisken i et hæsblæsende tempo!

Hvilken datakerne skal objektet have?

Når man sidder med den enkelte klasse skal man først lave datakernen.

Det er utrolig vigtigt at man får lavet datakernen rigtigt fra starten af. Hvis man er det mindste i tvivl er det altid godt at få verificeret ens valg af datakerne hos underviseren. Under eksamen må man i stedet tænke sig godt om!

Dagligdagen er fuld af studerende der vælger en forkert datakerne og laver en masse spildt arbejde. Et dårligt valg af datakerne forplanter sig til alt andet i klassen og dårligdommen følger med.

Skal datakernen indeholde en reference til et andet objekt?

Det er vigtigt at du ikke går videre før du har fastlagt datakernen for alle objekter i opgaven.

[Kode] Når du har datakernen klar til alle objekt, skriver du den ind øverst i hver af klasserne.

Hvilke konstruktorer skal klassen have?

Konstruktorerne og datakernen er bundet *meget* stærkt sammen. Det er nemlig konstruktorernes opgave at initialisere datakernen.

Hvis det fremgår af opgaven hvilke konstruktorer du skal lave, så lav dem *og ikke andre*. Hvis det ikke fremgår, er det givet vis fordi opgavestilleren vil se om du kan træffe nogle fornuftige valg på dette punkt.

Tænk først i de tre former for konstruktorer: Default- set- og copy-konstruktorer. Der kan være behov for konstruktorer, der falder uden for disse tre kategorier, men det er sjældent.

Tænk på hvilke du kan få brug for. Hvis du er i tvivl om behovet for en bestemt konstruktor, så lad være med at lave den! En konstruktor er altid nem at lave senere.

[Kode] Tag én klasse af gangen når du laver konstruktorer. Det kan specielt i starten være en god idé at lave konstruktorerne for de forskellige klasser før du går videre. På den måde skal du kun tænke konstruktorer for en tid, og bliver ikke distraheret af metoder osv. Man skal dog gøre det der passer en selv bedst.

Lav *altid* `toString`

[Kode] Uanset om det er nævnt i opgaven eller ej, så lav *altid* en **`toString`** til *alle* klasser. Metoden er uundværlig i testsammenhæng. Får du ikke testet dine klasser ordenligt, kan du ikke være sikker på de virker.

Når du har lavet **`toString`** skal du teste de konstruktorer du har lavet. Initialiserer de datakernen rigtigt?

Hvilke andre metoder skal klassen have?

Her er billedet ofte mere broget, og opgaven vil derfor ofte være mere præcist beskrevet vedrørende metoderne.

Ofte er navnene givet i opgaven - hold dig til dem!

Det der skal med som parametre er det metoden skal bruge *midleritidigt* for at løse en opgave. Vælg ikke parameternavne der ligger alt for tæt op af navnene på instansvariablene i datakernen - det går der rod i, før eller siden!

Det er også godt at vælge navne på lokale variable så de ikke kolliderer med andre navne. Hvis man er lidt forsigtig behøver man ikke tænke på navnene-kollision og regler for det. Man får intet ud af have navnesammenfald - undlad det derfor, med mindre du er sikker i din sag!

Vær omhyggelig med returtypen. Retyrtypen er lille i billedet, sammenlignet med de andre elementer i metoden, men den kan give problemer der er mindst lige så store.

I objektorienterede konstruktionsopgaver er indholdet i metoderne sjældent særlig kompliceret, rent algoritmisk. Det skyldes, at man med en objektorienteret opgave primært ønsker at øve/teste det objektorienterede og ikke det algoritmiske.

[Kode] Er du allerede begyndt at kode metoden. Nej, vel! Du skal være *helt klar* på returtype, navn og parameterliste før du koder noget som helt. Når du har signaturen stående kan du begynde at tænke på hvordan du vil kode indholdet.

Opbygning af objektsystem og klienten

Normalt laver man ikke et særligt klient-objekt. Det skyldes at klienten normalt er meget simpel, og det derfor vil være overkill at lave den.

I stedet laver man både opbygning af objektsystem og klient i **`main`**. Typisk kan dette opdeles i to dele: første del der opbygger objektsystemet, og anden del der er "klienten". Andre gange er det vekslende dele, der afløser hinanden, så det skiftevis er ændring af objektsystem og klientens anvendelse af objektsystemet.

Hvis ikke andet er nævnt i opgaven, er det op til dig selv, om du gider lave et egentligt klient-objekt.

Alle de retningslinier vi her har gennemgået, er for intet at regne mod værdien af den regel, som følger nederst på siden. Husk den, eller du får nogle problemer, som ikke er helt ufortjente :-)

MENU

Husk:

Pen og papir er de vigtigste redskaber, når der skal løses en opgave