PGR112 – Oppgaver 07 (2023-01-31)

I disse oppgavene er det mer åpent med tanke på hvordan de løses, og disse oppgavene vil utfordre deg litt til å komme på egne konsepter av objekter å lage.

Om disse oppgavene er vanskelig å starte på, vil jeg anbefale deg å lese tekstartiklene jeg har lagt ut på Canvas under «Sider» i emnet, eller bruk følgende link:

- https://kristiania.instructure.com/courses/10078/pages

Før du begynner i BlueJ (eller programmet du bruker), så trenger vi å planlegge litt først før oppgavene «angripes».

Du ønsker å lage en liste med minst 3 ulike klasser som kan representere noe enkelt som kan «objektiviseres», med dette så mener jeg for eksempel:

- Lyspære (LightBulb)
- Pizza
- Lommebok (Wallet)
- Bok (Book)
- Batteri (Battery)

Målet er å finne noe som du ønsker å kode, og er henholdsvis enkelt å definere ved hjelp av felter og metoder. Sjekk ut «eksempel»-mappen for å se eksempler på hva jeg mener.

Det som menes med «enkelt» er at det er forståelig og ikke-komplisert. Du ønsker noe som har identifiserbare kjennetegn (farge, navn, tittel, antall, osv).

Klasser i Java fungerer som en mal for hvordan et objekt skal opprettes. Objekter i Java har felter (hvor data lagres), metoder (hva kan objektet gjøre?) og en konstruktør (hva som gjøres for å klargjøre objektet som skal opprettes basert på klassen).

Eksempler på felter du kan benytte deg av, hvilken type som skal brukes (String, int, double, boolean osv) per felt er opp til deg:

- color
- size
- weight
- height
- temperature
- humidity
- quantity
- isOpen
- isClosed
- isCold
- isUnique
- isFragile

Og så videre; **Dette er bare eksempler**, kom gjerne på egne forslag til felter så lenge det gir mening for objektet du ønsker å ende opp med. Ikke lås deg til at det er disse feltene du må bruke, du som programmerer er den som har kontroll over din egen kode!

Metoder kan for eksempelvis være getter- og setter-metoder, og ellers annen funksjonalitet som er relevant for klassen. For eksempelvis en dør trenger gjerne metoder for å både lukke og åpne døren, en vannflask trenger gjerne metoder for å drikke (minske volumet) eller for å fylle på (øke volumet), og så videre.

- Oppgave #1
    - Her antas det at du har laget en liste med minst 3 klasser du ønsker å lage, og har planlagt hvilke felter og metoder denne trenger.
  - For hver av klassene du har planlagt:
    - Lag klassen med et engelsk navn, og:
      - **Opprett felter**
      - **Opprett getter/setter-metoder**
      - **Opprett** andre **metoder** relevant for hva objektet kan gjøre (endre data, printe ut informasjon om seg selv, gjøre et regnestykke basert på dataen lagret i objektet, osv.)
      - **Opprett** (minst én) **konstruktør**
      - **Opprett** statisk **main-metode** som:
        - Oppretter flere objekter ved bruk av **new** nøkkelordet, hvor du ta i bruk de ulike metodene for å endre på objektet.

          Print gjerne ut informasjon til konsollen for å ha kontroll over hva som endrer seg underveis og for å vise at metodene gjør som de skal.

- Oppgave #2
    - 1) Planlegg og opprett en klasse som andre klasser kan basere seg på ved hjelp av arv, sjekk slides fra forelesningen for å få en bedre forståelse av hva som menes med dette

    - 2) Etter du har opprettet denne klassen, opprett 2 nye klasser som arver fra klassen du har laget

    - 3) Opprett en ny klasse som heter JavaProgram
        - I denne klassen, opprett en main-metode som tar i bruk klassene som arver fra klassen du laget innledningsvis, og tar i bruk metoder for å gjøre noe med denne klassen.

- Oppgave #3
    - Velg en av klassene du lagde i oppgave 1, og gjør denne om til en klasse som andre klasser kan arve fra. Dette kan gjøres i en ny klasse med et litt annerledes navn eller ved å legge på «Generic» foran navnet på klassen.
    - Opprett en ny klasse som arver fra denne klassen, og test ut at alt fungerer som det skal (at du har tilgang til metoder og data (via getter-metoder) i en main-metode for å se om det funker som det skal.
    - **Ekstra:** Opprett en ny klasse som er litt annerledes fra klassen du opprettet i det forrige steget, og la denne klassen også arve fra klassen du lagde innledningvis i denne oppgaven.

Tips og triks:

Når du jobber med oppgaver eller koding generelt, still deg selv alltid spørsmål underveis:

- Hvordan gjør jeg akkurat denne spesifikke tingen som det spørres om?
- Har jeg gjort noe lignende før?
- Hva burde være tilgjengelig når? (spesifikt private vs. public)
    - *For eksempel*: Skal alle andre klasser kunne endre et passord som er lagret i et objekt, for eksempel en bankkonto-objekt opprettet av en type klasse ved navn BankAccount? Ikke alle felter trenger en setter-metode!

Det kan være veldig greit å ta notater i forbindelse med dette, noe som kan gjøres i et eget program eller ved hjelp av kommentarer i koden din. Det viktigste er å identifisere hva som brukes for å løse spesifikke problemer, for etterhvert vil problemene som dukker opp være problemer du har løst før!

Når du jobber med kode generelt, så er dette en veldig god strategi å følge: Identifiser problemene som skal løses. Om problemet er for stort der og da, bryt det opp i mindre problemer. Angrip de mindre problemene, hvor du til slutt vil ende opp med noe som løser det større problemet.

Når du arbeider med et spesifikt (mindre) problem, tenk **kun** på å løse det problemet, ikke overkompliser det ved å forsøke å tenke på hele problemstillingen på en gang.

Det som er beskrevet over kalles for en «Divide and Conquer»-strategi, og er en måte å angripe problemer på, spesielt i en kodekontekst:

Divide:
: Bryt problemet ned i mindre problemer

Conquer:
: Løs de mindre problemene helt til du løser det større problemet du hadde

Combine:
: Kombiner løsningene på de mindre problemene for å sette sammen en løsing som løser det større problemet du hadde innledningsvis.

Bruk gjerne litt tid på å se på tidligere oppgaver og spesielt løsningsforslagene, for å identifisere bruken av denne teknikken der i løsningsforslagene.

Å kode og det å programmere handler om å løse problemer, og dette er en ferdighet som må trenes over tid med mengdetrening. Å brukte teknikker og strategier er en del av dette.

I første utgangspunkt, lag kode som løser problemet uten å tenke over at koden ser stygg ut eller rotete ut, dette kan fikses etterpå!