NTNU

Øving 12

TDAT1005 Databaser med videregående programmering
Institutt for datateknologi og informatikk (IDI), NTNU

Arv og polymorfi, del 3

I denne øvingen skal du jobbe både med arv, interface og polymorfi.

En av de viktigste hensiktene med å bruke interface-mekanismen i Java, er å skjule unødvendig informasjon fra klienter. En vanlig måte å gjøre dette på i praksis, er å tilby funksjonalitet til omverdenen gjennom et interface og å skjule hvordan objekter lages gjennom en egen **fabrikklasse** (se også Kap. 14.10 i Java-læreboka).

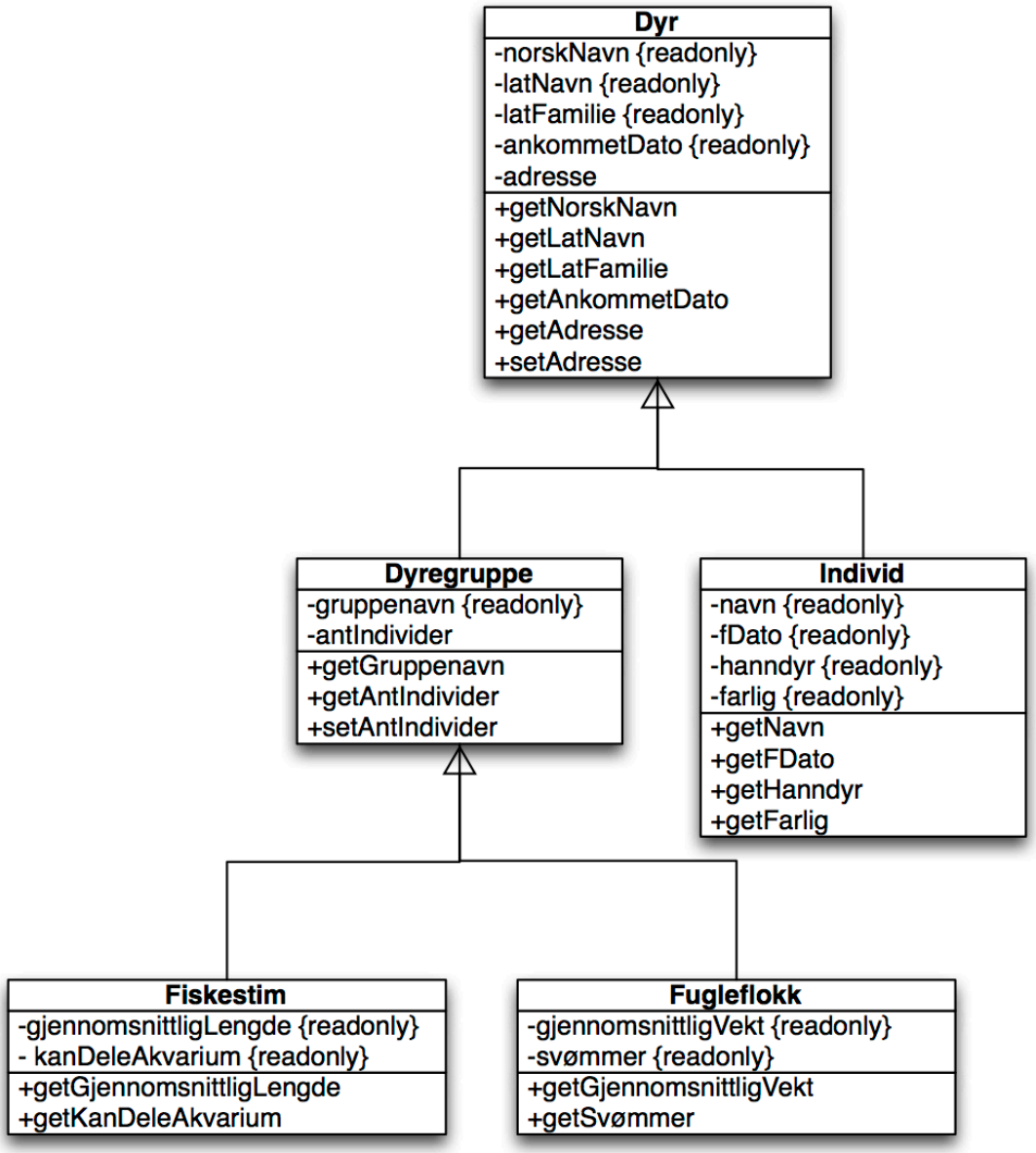
Hvorfor skjule hvordan objekter lages og brukes, lurer du kanskje på. Noen ganger krever det å lage riktig objekt god kjennskap til mange klasser, der vi både må velge rett klasse, rett konstruktør og senere velge riktige meldinger når vi skal bruke objektet. Hvis klienten skal bruke objektene på en bestemt måte der detaljene bare kompliserer ting, kan vi forenkle bruken ved å skjule dem.

Et typisk eksempel på en slik situasjon er objekter som skal håndtere kommunikasjon med databaser. Forskjellige databaser krever forskjellige typer "kommunikasjonsobjekter" og forskjellige typer meldinger, så å skjule dette for klienten og istedet tilby forenklede meldinger ved hjelp av et interface kan være en god måte å gjøre databasekommunikasjon enklere på. I tillegg, hvis vi på et tidspunkt bytter databasesystem trenger vi da kun å endre klassene bak interfacet samt fabrikklassen – klienten trenger ikke å endre noe som helst.

I denne oppgaven skal du jobbe med programvare som en dyrehage skal bruke til å holde oversikt over dyrene sine.

Oppgave 1

Du skal implementere følgende klassediagram:



Vi ser at hver dyr er registrert med navnet på arten sin (hhv. på norsk og latin) og familien arten tilhører. I tillegg registrerer vi datoen dyret ankom dyrehagen, og en "adresse" til hvor dyret holder til (for eksempel "Bur 13, Reptilbygget"). Vi ser også at dyrene er delt inn i to hovedgrupper:

- **Individ:** Dyr vi ønsker registrert som individ (eks. bjørn, gorilla, osv.). Om disse dyrene lagrer vi i tillegg navn, fødselsår, kjønn og hvorvidt dyret er farlig eller ikke.
- **Dyregrupper:** Dyr vi forholder oss til i grupper, som for eksempel fiskestimer, insektsvermer og fugleflokker. Om hver gruppe lagrer vi gruppenavn og omtrentlig antall individer i gruppen. I tillegg har vi spesielt tilpassede klasser for hhv. **fiskestimer** (der vi lagrer gjennomsnittlig lengde for et voksent eksemplar av arten og om den kan dele akvarium med en annen dyregruppe) og **fugleflokker** (der vi lagrer gjennomsnittlig vekt for et voksent eksemplar av arten og om det er en type svømmefugl) som du ser i klassediagrammet.

a. Koden for klassen `Dyr` er gitt her, resten skal du implementere selv (inkludert tilpassede versjoner av `toString`).

b. Kan eller bør noen av klassene og/eller metodene være abstrakte? Tenk på forskjellige måter klassene kan bli brukt på i framtiden og vurder ut fra det. Gjør så de klassene/metodene du mener bør være det abstrakte.

c. Overskriv metoden `getNorskNavn` i klassen `Dyregruppe` slik at den skriver ut "gruppe av "+ det norske navnet. Hva skjer når du gjør dette, og hvorfor? Hva må du eventuelt endre i Dyr-klassen for å få det til å fungere?

Oppgave 2

Du skal nå lage en fabrikklasse og et interface som håndterer dyreobjekter for den delen av dyrehagen som består av skandinaviske rovdyr (bjørn, ulv, jerv osv). Klassen skal senere brukes i flere applikasjoner som brukes i stell og overvåkning av dyrene, spesialtilpasset denne dyregruppen. Vi kommer i det som følger til å begrense oss til dyrene bjørn og ulv. Det du skal gjøre er følgende:

a. Legg alle klassene du har laget hittil i en pakke kalt `dyrehage`.

b. Gjør følgende interface til en del av pakken:

```
interface SkandinaviskeRovdyr{
  String getNavn();
  int getFdato();
  int getAlder();
  String getAdresse();
  void flytt(String nyAdresse);
  String skrivUtInfo();
}
```

c. La klassen `Individ` implementere `SkandinaviskeRovdyr`, og programmer de metodene som eventuelt mangler.

d. Lag en klasse ved navn `Rovdyrfabrikk` med følgende metoder:

```
public SkandinaviskeRovdyr nyBinne()
public SkandinaviskeRovdyr nyHannbjørn()
public SkandinaviskeRovdyr nyUlvetispe()
public SkandinaviskeRovdyr nyUlvehann()
```

Tips: Brunbjørnen har artsnavn "Ursus arctos" og familienavn "Ursidae", mens ulven har artsnavn "Canis lupus" og familienavn "Canidae". Vi antar at alle bjørner og ulver er farlige.

Legg merke til at du nå bruker polymorfi gjennom et **interface** (og ikke arv). Legg også merke til at klienten nå kan lage bjørne- og ulveobjekter uten å kjenne til artsnavn/familie, og uten å måtte forholde seg til om dyret er farlig eller ikke (dette ordner fabrikklassen).

Oppgave 3

Lag en enkel testklient som tester *hver* av metodene i interfacet `SkandinaviskeRovdyr`. Du skal altså lage dyreobjekter ved hjelp av et `RovdyrFabrikk`-objekt, og kun bruke metodene i `SkandinaviskeRovdyr`. Husk at du må importere pakken `dyrehage` for å kunne bruke den (klienten skal ikke være en del av pakken).

Oppgave 4

Etter en tids bruk finner dyrehagen ut at de vil lagre informasjon om hvor mange ungekull dyrene registrert som individer har fått. Siden det er ofte er vanskelig å vite hvem som er far til et kull samt at antall kull ikke er like viktig å holde styr på for hanner (det påvirker vanligvis ikke helsa til hanndyr like mye som for hunndyr), ønsker de kun å lagre antall kull for hunndyr. Du skal implementere denne endringen som følger:

a. Lag to nye klasser, `Hunnindivid` og `Hannindivid` som begge arver `Individ`. I `Hunnindivid`-klassen legger du så til variabelen

```
private int antKull
```

   og gjør nødvendige endringer (konstruktør, get/set-metoder).

b. For at Rovdyrfabrikk fortsatt skal fungere som før, må du gjøre noen endringer her. Finn ut av hvilke og gjør dem.

c. Interfacet `SkandinaviskeRovdyr` må nå utvides for at klienter skal kunne benytte seg av den nye funksjonaliteten. Utvid det med metodene

```
int getAntKull();
void leggTilKull(int antall);
void leggTilNyttKull();
```

   Den siste metoden legger kun til et enkelt kull (den vanligste operasjonen) i `Hunnindividklassen`. I `Hannindivid`-klassen skal `getAntKull()` returnere 0, og `leggTilKull`-meldingene ikke gjøre noe. Hvilke konsekvenser får dette for `Individ`-klassen?

d. Til slutt skal du utvide testklienten din til å dekke også disse metodene.

Lykke til!

Institutt for datateknologi og informatikk (IDI), NTNU