NTNU

Oblig 5 Del B

IDATx2001 Programmering 2
Institutt for datateknologi og informatikk (IDI)/
Institutt for IKT og Realfag (IIR), NTNU

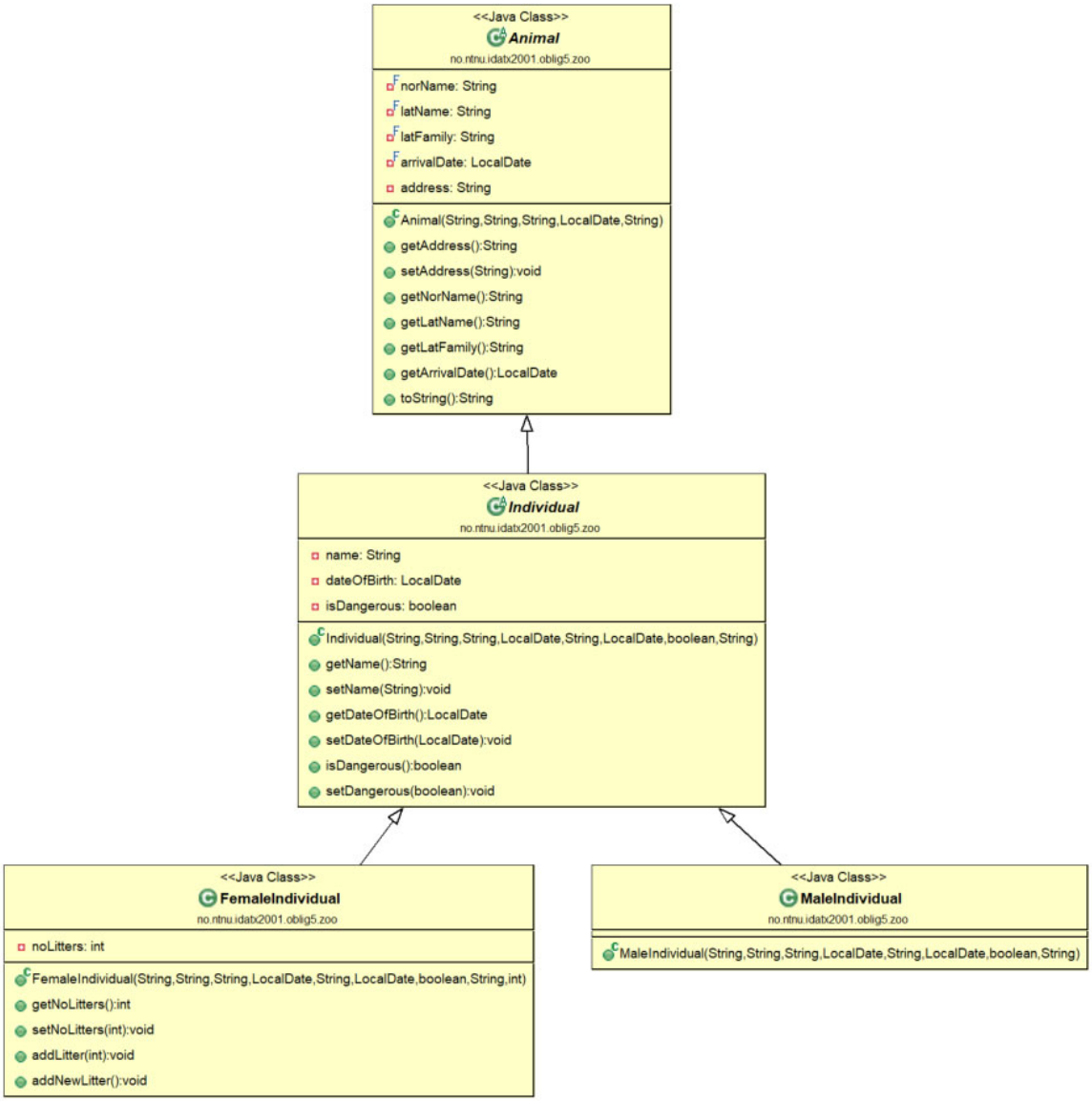
Grensesnitt og designmønster

En av de viktigste hensiktene med å bruke grensesnitt-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 grensesnitt og å skjule hvordan objekter lages gjennom en egen fabrikklasse.

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 i stedet tilby forenklede meldinger ved hjelp av et grensesnitt 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 grensesnitt 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.



Figur 1: Klassediagram for dyrehagen

Oppgave 1

Du skal nå lage en fabrikklasse og et grensesnitt 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. Lag følgende grensesnitt-klasse og la ***Individual*** implementere denne. Programmer metodene:

```
public interface ScandinavianWildAnimal {

    String getName();
    LocalDate getDateOfBirth();
    int getAge();
    String getAddress();
    void move(String newAddress);
    String printInfo();

}
```

b. Lag en klasse ved navn ***WildAnimalFactory*** med følgende metoder:

```
public class WildAnimalFactory {

    public ScandinavianWildAnimal newMaleBear(…){
       …
    }

    public ScandinavianWildAnimal newFemaleWolf(…){
        …
      }

    public ScandinavianWildAnimal newMaleWolf(…){
        …
    }
}
```

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 **grensesnitt** (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).

c. Lag en klient klasse som bruker ***WildAnimalFactory*** og dyr. Her er et eksempel på bruk av fabrikk-klassen:

```
ScandinavianWildAnimal ulla = wildAnimalFactory.newFemaleWolf("Ulla",
LocalDate.of(2015,2,26), LocalDate.of(2015,4,29), "Innhegning 2,
Skandinaviske rovdyr");
```

Oppgave 2

a. Vi ønsker å logge alle aktiviteter i klient-programmet du laget i oppgave 1c. Til dette kan vi bruke følgende klasse: java.util.logging.Logger. Vi definerer følgende objekt-variabel:

```
private static final Logger log = Logger.getLogger(ZooLogger.class.getName());
```

Vi kan nå bruke log til å logge ønsket informasjon på fil og/eller konsol.

Identifiser hva slags designmønster Logger er. Begrunn svaret.

b. Re-implementer klassen WildAnimalFactory fra oppgave 1 slik at den blir Singleton.