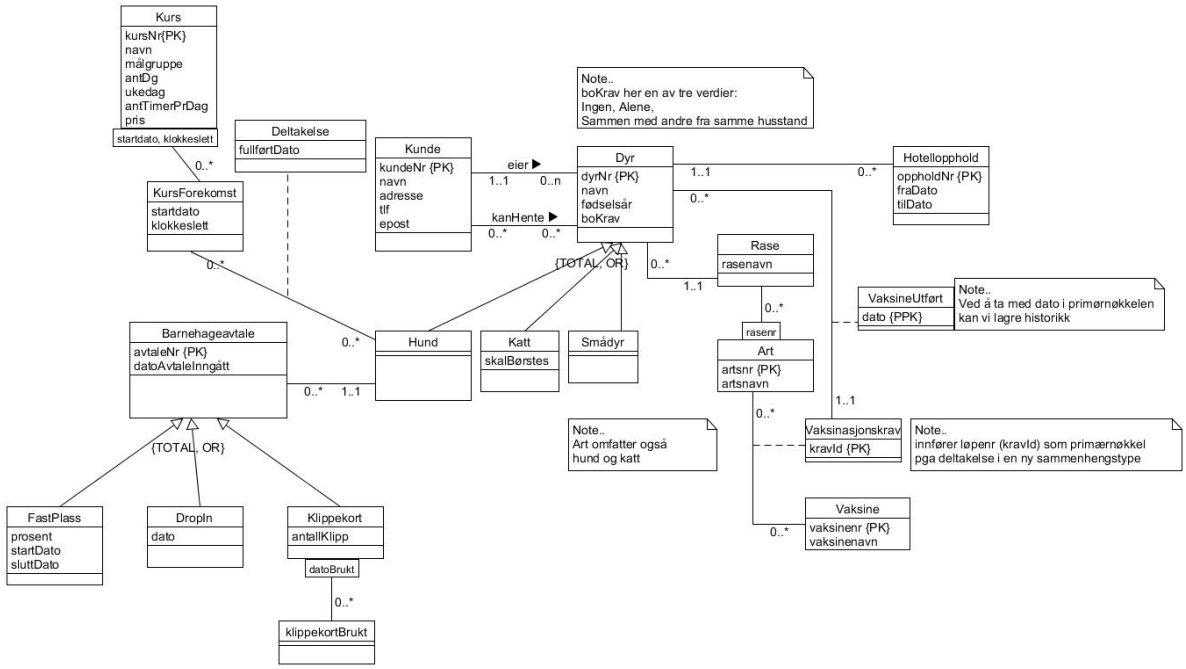
LØSNINGSFORSLAG EKSAMEN TDAT1005 VÅREN 2018

Oppgave 1



(Det eksisterer alternative modeller som er like gode.)

Relasjonsmodellen:

Kunde(kundeNr, navn, adresse, tlf, epost)
Dyr(dyrNr, navn, fødselsår, boKrav, eierKundenr*, (artsnr, rasenr)*)
KanHente(dyrNr*, kundeNr*)
Hund(hundDyrNr*) – se kommentar ang oversetting av arv nederst
Katt(kattDyrNr*, skalBørstes)
Smådyr(smådyrDyrNr*)
Hotellopphold(oppholdNr, fraDato, tilDato, dyrNr*)
Art(artsnr, artsnavn)
Rase(artsnr*, rasenr, rasenavn)
Vaksine(vaksinenr, vaksinenavn)
Vaksinasjonskrav(kravId, vaksinenr*, artsnr*)
VaksineUtført(kravId*, dyrNr*, dato)
Barnehageavtale(avtaleNr, datoAvtaleInngått, hundDyrNr*)
FastPlass(avtaleNr*, prosent, startdato, sluttdato)
DropIn(avtaleNr*, dato)

Klippekort(klippekortAvtaleNr*, antallKlipp)
KlippekortBrukt(klippekortAvtaleNr *, datoBrukt)

Kurs(kursNr, navn, målgruppe, antDg, ukedag, antTimerPrDag, pris
KursForekomst(kursNr*, startdato, klokkeslett)
Deltakelse((kursNr, startdato, klokkeslett)*, hundDyrNr*, fullførtDato)

Ang oversetting av arv (generalisering/spesialisering):

Det henvises til læreboka i databaseer side xxx

Det er spesielt to misforståelser som går igjen. Jeg viser det ved eksemplet med dyr, hund og katt. Vi antar at {TOTAL, OR} gjelder.

1. Man gjentar alle attributter i subentitetstypene. Da ser det slik ut:

   Dyr(dyrNr, navn, fødselsår, boKrav, eierKundenr*, (artsnr, rasenr)*)
   Hund(hundDyrNr*, navn, fødselsår, boKrav, eierKundenr*, (artsnr, rasenr)*)
   Katt(kattDyrNr*, skalBørstes, navn, fødselsår, boKrav, eierKundenr*, (artsnr, rasenr)*)

   Spørsmålet er hva som skal lagres i relasjonen Dyr. Ettersom hundDyrNr og kattDyrNr er fremmednøkler som refererer til Dyr, så må informasjon om alle dyr lagres to ganger, en gang i Dyr og en gang enten i Katt eller i Hund.

2. Som over, men relasjonen Dyr utelates.

   Dette er noe bedre på en måte, ettersom data lagres kun én gang. Men dette er ikke i samsvar med EER-modellen. Prøv og tegne modellen uten entitetstypen Dyr. Sammenhengstypene som Dyr inngår i blir nå ikke veldig elegante, de må gå både til hund og til Katt, det blir i det hele tatt en meget dårlig modell.

Oppgave 2

2a)

```
SELECT DISTINCT stedsnavn
FROM STED s, ETAPPE e, LOP_ETAPPE le
WHERE stedid = fra_stedid OR stedid = til_stedid
AND e.enr = le.enr
AND lnr = 1 AND aar = 2018
ORDER BY stedsnavn;
```

2b)

```
SELECT lopenr, s1.stedsnavn, s2.stedsnavn, distanse
FROM STED s1, STED s2, ETAPPE e, LOP_ETAPPE le
```

```
WHERE s1.stedid = fra_stedid AND s2.stedid = til_stedid
AND e.enr = le.enr
AND lnr = 1 AND aar = 2018
ORDER BY lopenr;
```

2c)

```
SELECT t.deltnr, navn
FROM TIDTAKING t, DELTAKER d
WHERE lnr = 1 AND aar = 2018 AND fullfort = 1
AND t.deltnr = d.deltnr
GROUP BY t.deltnr, navn
HAVING COUNT(enr) >= 4;
```

Alternative løsninger:

Det finnes (i alle fall) en mulig løsning uten bruk av HAVING som kan fungere hvis en antar at alle som har deltatt på løpenr > 3 (evt. lignende logiske uttrykk) har deltatt på minst 4 etapper blir det også riktig. En løsning da f.eks.:

```
SELECT t.deltnr, navn
FROM TIDTAKING t, DELTAKER d, LOP_ETAPPE le
WHERE t.lnr = 1 AND t.aar = 2018 AND fullfort = 1
AND t.deltnr = d.deltnr AND (t.lnr=le.lnr AND t.aar=le.aar AND
t.enr=le.enr)
AND lopenr > 3
GROUP BY t.deltnr, navn;
```

2d)

```
CREATE VIEW DELT_FINNMARK AS
SELECT deltnr, l.lnr, COUNT(*) AS ant_etapper, SUM(tid) AS tot_tid
FROM TIDTAKING t, LOP l
WHERE lopsnavn = 'Finnmarksløpet' AND l.aar = 2018
AND l.lnr = t.lnr AND l.aar = t.aar AND fullfort = 1
GROUP BY deltnr, l.lnr
HAVING COUNT(*) =
(SELECT COUNT(*)
```

```
 FROM LOP_ETAPPE le
 WHERE le.lnr = l.lnr AND le.aar = 2018);
```

Merknad: HAVING-delen av løsning over er med for å sjekke at deltakerne som er med i view-et har deltatt i alle etappene til løpet (forutsetter da at det finnes deltakere som har bestått alle etappene - hvis det ikke er tilfelle vil ikke løsningen fungere - da må en i stedet for ha en løsning som finner deltakerne som har flest antall etapper fullført).

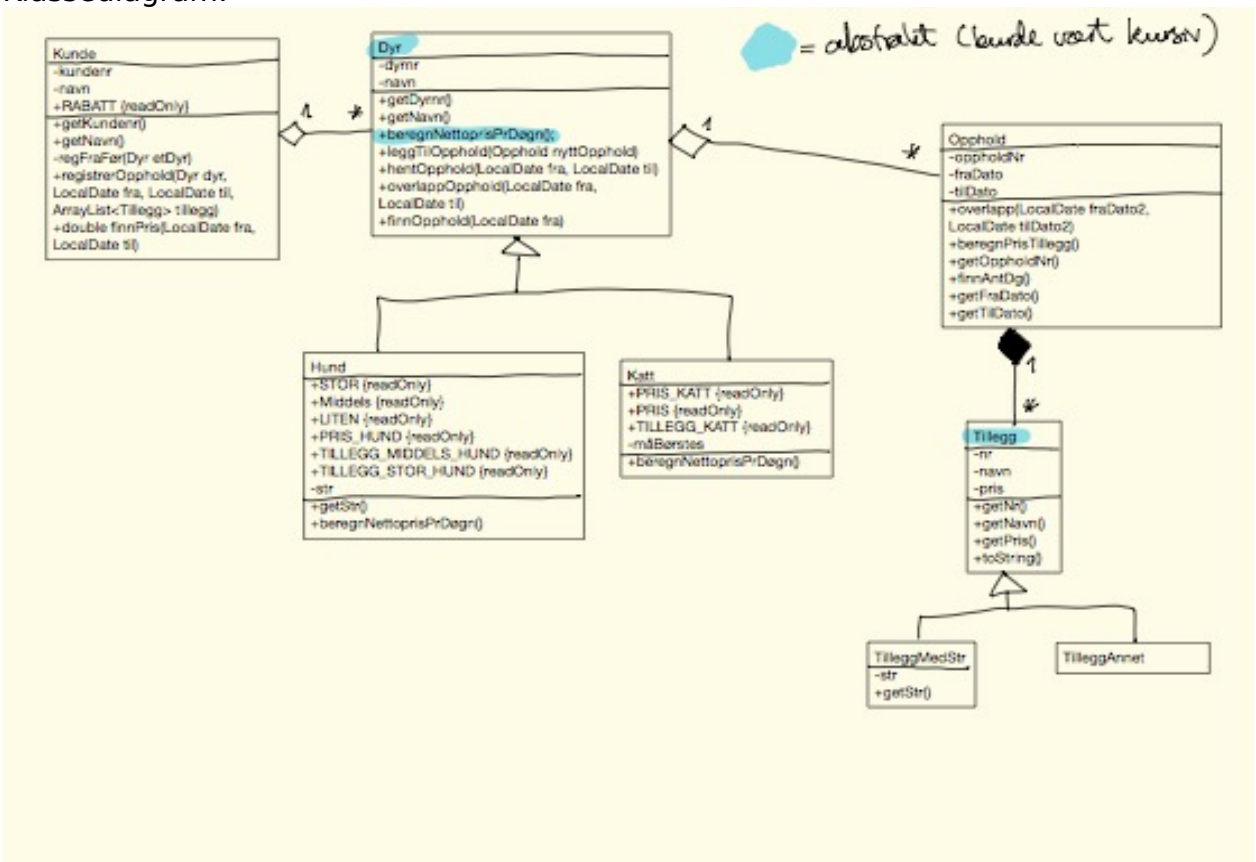
HAVING-delen er med her m.t.p. oppgave 2e), men det er ikke noe krav at denne delen av spørringa er med i oppgave 2d).

2e)

```
SELECT navn
FROM DELT_FINNMARK df, DELTAKER d
WHERE d.deltnr = df.deltnr
AND tot_tid = (SELECT MIN(tot_tid) FROM DELT_FINNMARK);
```

Oppgave 3

Klassediagram:



Kode:

```
import java.sql.*;
import java.util.ArrayList;
import java.time.LocalDate;

class TestDyrehotell {
  public static void main(String [] args) throws Exception {
    System.out.println("Test");
    Database db = new Database();
    java.util.ArrayList<Tillegg> tillegg = db.hentTillegg();
    System.out.println(tillegg);
    db.lukkConn();
  }
}

class Kunde  {
  private final int kundenr;
  private final String navn;  // entydig for denne kunden
  private ArrayList<Dyr> dyrene = new ArrayList<Dyr>();

  public static final double RABATT = 25.0;

  public Kunde(int kundenr, String navn) {
    this.kundenr = kundenr;
    this.navn = navn;
  }

  public int getKundenr() {
    return kundenr;
  }

  public String getNavn() {
    return navn;
  }

  private boolean regFraFør(Dyr etDyr) {
    for (Dyr d : dyrene) {
      if (d.getNavn().equals(etDyr.getNavn())) {
        return true;
      }
    }
    return false;
  }

  public boolean registrerOpphold(Dyr dyr, LocalDate fra, LocalDate til, ArrayList<Tillegg> tillegg) {
    if (!regFraFør(dyr)) {
      dyrene.add(dyr);
    }
    if (dyr.overlappOpphold(fra, til)) {
      return false;
    }
    dyr.leggTilOpphold(new Opphold(1000, fra, til, tillegg));
    return true;
  }

  public double finnPris(LocalDate fra, LocalDate til) {
    int antDøgn = (int) fra.until(til, java.time.temporal.ChronoUnit.DAYS);
    double pris = 0.0;
    boolean rabattHund = false;
    boolean rabattKatt = false;
```

```
    for (Dyr etDyr : dyrene) {
      Opphold opph = etDyr.hentOpphold(fra, til);
      if (opph != null) {
        double nettopris = etDyr.beregnNettoprisPrDøgn() * antDøgn;
        if (etDyr instanceof Hund) {
          if (!rabattHund) {
            pris += nettopris;
            rabattHund = true;
          } else {
            pris += nettopris * (1.0 - RABATT);
          }
        }
        if (etDyr instanceof Katt) {
          if (!rabattKatt) {
            pris += nettopris;
            rabattKatt = true;
          } else {
            pris += nettopris * (1.0 - RABATT);
          }
        }
        pris += opph.beregnPrisTillegg();
      }
    }
    return pris;
  }
}


abstract class Dyr {
  private final int dyrnr;
  private final String navn;
  private ArrayList<Opphold> oppholdene = new ArrayList<Opphold>();

  protected Dyr(int dyrnr, String navn) {
    this.dyrnr = dyrnr;
    this.navn = navn;
  }

   public int getDyrnr() {
     return dyrnr;
   }

   public String getNavn() {
     return navn;
  }

  abstract public double beregnNettoprisPrDøgn();

  public void leggTilOpphold(Opphold nyttOpphold) {
    oppholdene.add(nyttOpphold);
  }

  public Opphold hentOpphold(LocalDate fra, LocalDate til) {
    for (Opphold o : oppholdene) {
      if (o.getFraDato().equals(fra) && o.getTilDato().equals(til)) { // Kan også teste på den første, og
hvis denne stemmer så på den andre.
        return o;
      }
    }
    return null;
```

```
  }

  public boolean overlappOpphold(LocalDate fra, LocalDate til) {
    for (Opphold o : oppholdene) {
      if (o.overlapp(fra, til)) {
        return true;
      }
    }
    return false;
  }

  public Opphold finnOpphold(LocalDate fra) {
    for (Opphold etOpphold : oppholdene) {
      if (etOpphold.getFraDato().equals(fra)) {
        return etOpphold;
      }
    }
    return null;
  }
}

class Hund extends Dyr {
  public static int STOR = 3;
  public static int MIDDELS = 2;
  public static int LITEN = 1;
  public static final double PRIS_HUND = 360.0;
  public static final double TILLEGG_MIDDELS_HUND = 30.0;
  public static final double TILLEGG_STOR_HUND = 50.0;

  private final int str;

  public Hund(int dyrnr, String navn, int str) {
    super(dyrnr, navn);
            if (str<1 || str>3) {
              // Ugyldig størrelse
              throw new IllegalArgumentException("Feil i størrelsesinndata.");
            }
    this.str = str;
  }

  public double getStr() {
    return str;
  }

  public double beregnNettoprisPrDøgn() {
    if (str == STOR) {
      return PRIS_HUND + TILLEGG_STOR_HUND;
    } else if (str == MIDDELS) {
      return PRIS_HUND + TILLEGG_MIDDELS_HUND;
    } else {
      return PRIS_HUND;
    }
  }
}

class Katt extends Dyr {
  public static final double PRIS_KATT = 170.0;
  public static final double TILLEGG_KATT = 20.0;  // hvis den må børstes
  private final boolean måBørstes;
```

```
  public Katt(int dyrnr, String navn, boolean måBørstes) {
    super(dyrnr, navn);
    this.måBørstes = måBørstes;
  }

  public double beregnNettoprisPrDøgn() {
    return PRIS_KATT + (måBørstes ? TILLEGG_KATT : 0.0);
  }
}

class Opphold {
  private int oppholdNr;
  private LocalDate fraDato;
  private LocalDate tilDato;
  private ArrayList<Tillegg> tillegg;

  public Opphold(int oppholdNr, LocalDate fraDato,  LocalDate tilDato, ArrayList<Tillegg> tillegg) {
    this.oppholdNr = oppholdNr;
    this.fraDato = fraDato;
    this.tilDato = tilDato;
  }

  public boolean overlapp(LocalDate fraDato2,  LocalDate tilDato2) {
    // Hvis utsjekking på Opphold 2 er senest samtidig med starten på Opphold 1, eller hvis innsjekk på
Opphold 2 er tidligst samme dag som utsjekk på Opphold 1 er det ikke overlapp.
    if (tilDato2.isBefore(fraDato.plusDays(1)) || fraDato2.isAfter(tilDato.minusDays(1))) {
       return true;
      }
      return false;
  }

  public double beregnPrisTillegg() {
    double pris = 0.0;
    for (Tillegg t : tillegg) {
      pris += t.getPris();
    }
    return pris;
  }

 public int getOppholdr() {
   return oppholdNr;
 }

 public int finnAntDg() {
   return (int) fraDato.until(tilDato, java.time.temporal.ChronoUnit.DAYS);
 }

 public LocalDate getFraDato() {
   return fraDato;
 }

 public LocalDate getTilDato() {
    return tilDato;
 }
}

abstract class Tillegg {
  private final int nr;
  private final String navn;
  private final double pris;
```

```
  protected Tillegg(int nr, String navn, double pris) {
    this.nr = nr;
    this.navn = navn;
    this.pris = pris;
  }

  public int getNr() {
    return nr;
  }

  public String getNavn() {
    return navn;
  }

  public double getPris() {
    return pris;
  }

  public String toString() {
    return nr + ", " + navn + ", " + pris;
  }
}

class TilleggMedStr extends Tillegg {
  private final String str;
  public TilleggMedStr(int nr, String navn, double pris, String str) {
    super(nr, navn, pris);
    this.str = str;
  }
  public String getStr() {
    return str;
  }

  public String toString() {
    return super.toString() + ", " + str + "\n";
  }
}

class TilleggAnnet extends Tillegg {
  public TilleggAnnet(int nr, String navn, double pris) {
    super(nr, navn, pris);
  }
  public String toString() {
    return super.toString() + "\n";
  }
}


class Database {
  private final String DATABASENAVN = "jdbc:mysql://mysql.stud.iie.ntnu.no:3306/elsemal?
user=elsemal&password=UO7neaBv";
    private final String DATABASEDRIVER = "com.mysql.jdbc.Driver";
  private Connection conn;
  public Database()  throws Exception {
    try {
      Class.forName(DATABASEDRIVER);
      conn = DriverManager.getConnection(DATABASENAVN);
      } catch (Exception e) {
      Opprydder.skrivMelding(e, "Feil i konstruktøren Databaser()");
```

```
      throw e;
    }
  }

  public void lukkConn() throws Exception {
    Opprydder.lukkForbindelse(conn);
  }

  public java.util.ArrayList<Tillegg> hentTillegg() {
    java.util.ArrayList<Tillegg> liste = new java.util.ArrayList<Tillegg>();
    String sql = "select nr, navn, pris, str from tillegg left join strTillegg on tillegg.nr =
strTillegg.strTilleggNr";
    Statement stm = null;
    ResultSet res = null;
    try {
      stm = conn.createStatement();
      res = stm.executeQuery(sql);
      while (res.next()) {
        Tillegg sistLeste = null;
        int nr = res.getInt("nr");
        String navn = res.getString("navn");
        double pris = res.getDouble("pris");
        String str = res.getString("str");
        if (res.wasNull()) {
          sistLeste = new TilleggAnnet(nr, navn, pris);
        } else {
          sistLeste = new TilleggMedStr(nr, navn, pris, str);
        }
        liste.add(sistLeste);
      }
    } catch (SQLException e) {
      Opprydder.skrivMelding(e, "Feil i metoden hentTillegg()");
      liste = null;
    } finally {
      Opprydder.lukkResSet(res);
      Opprydder.lukkSetning(stm);
    }
    return liste;
  }
}


/*
drop table strTillegg;
drop table Tillegg;
drop table størrelse;

create table størrelse(
  str varchar(10) primary key);

insert into størrelse values('Liten');
insert into størrelse values('Medium');
insert into størrelse values('Stor');

create table tillegg(
  nr int not null primary key,
  navn varchar(20) not null,
  pris double not null);

create table strTillegg(
```

```
  strTilleggNr int not null primary key,
  str varchar(10) not null);

alter table strTillegg
 add constraint tillegg_fk foreign key(strTilleggNr) references tillegg(nr);
alter table strTillegg
 add constraint størrelse_fk foreign key(str) references størrelse(str);

insert into tillegg values(1, 'MMS', 50);
insert into tillegg values(2, 'Godtepose', 25);
insert into tillegg values(3, 'Bad', 300);
insert into tillegg values(4, 'Bad', 400);
insert into tillegg values(5, 'Bad', 500);

insert into strTillegg values(3, 'Liten');
insert into strTillegg values(4, 'Medium');
insert into strTillegg values(5, 'Stor');

select tillegg.nr, navn, pris, str from
tillegg left join strTillegg on tillegg.nr = strTillegg.strTilleggNr;

*/
```

Oppgave 4

a) Programmet skriver ut:

A 1

B 2

B 2

Begrunnelse:

MinTest test = eksamen.getTest();

Her vil test være en referanse til testObj som «eies av» eksamen-objektet (lages i konstruktor). Endringer på test vil derfor også endre testObj.

Merk imidlertid at wrapper-typer (som f.eks. String og Integer) alltid er immutable – mao. vil endringer på disse ikke endre eksisterende objekter, men vil opprette nye.

b)

*Tabellen er ikke på normalform da feltet **emne** ikke har atomiske verdier. 1NF oppnås ved ved å ha atomiske verdier der. Det er imidlertid fullt mulig å forbedre tabellen videre (ved å splitte den opp) til 3NF og BCNF. Kandidaten bør beskrive hvordan dette da kan bli seende ut.*

c)

Interface gir muligheten til å skille grensesnitt og implementasjon. Interfacet blir da som en kontrakt på hva klienten kan forvente seg. JDBC er et fint eksempel. Her vil klienten forholde seg til interfacet, mens databasetilbyder lager driveren/implementasjonen som sørger for databasekommunikasjonen (driveren

implementerer da også interfacet). For å skifte databasesystem kan man dermed bare skifte ut databasedriveren uten å endre klientkoden.

Kandidaten bør også gi eksempel på en situasjon hvor man kan ønske å lage egne interface.